# SMT SOLVERS

## Spring School on Logic and Verification 2016

Martina Seidl
Institute for Formal Models and Verification

JOHANNES KEPLER
UNIVERSITY LINZ

RiSE
Rigorous Systems Engineering

# The Program “Middle”

```
int middle (int x, int y, int z) {
  int m = z;
  if (y < z) {
    if (x < y)
      m = y;
    else if (x < z)
       m = y;
  } else {
    if (x > y)
      m = y;
    else if (x > z)
      m = x;
  }
  return m;
}
```

This program is supposed to return the middle (median) of three numbers.

# The Program “Middle”

```
int middle (int x, int y, int z) {
  int m = z;
  if (y < z) {
    if (x < y)
      m = y;
    else if (x < z)
       m = y;
  } else {
    if (x > y)
      m = y;
    else if (x > z)
      m = x;
  }
  return m;
}
```

**Some test cases:**

```
middle (1, 2, 3) = 2
middle (1, 3, 2) = 2
middle (2, 3, 1) = 2
middle (3, 1, 2) = 2
middle (3, 2, 1) = 2
middle (1, 1, 1) = 1
middle (1, 1, 2) = 1
middle (1, 2, 1) = 1
middle (2, 1, 1) = 1
middle (1, 2, 2) = 2
middle (2, 1, 2) = 2
middle (2, 2, 1) = 2
```

# The Program “Middle”

```
int middle (int x, int y, int z) {
  int m = z;
  if (y < z) {
    if (x < y)
      m = y;
    else if (x < z)
       m = y;
  } else {
    if (x > y)
      m = y;
    else if (x > z)
      m = x;
  }
  return m;
}
```

**Missed test case:**

`middle (2, 1, 3) =` 1

# The Program “Middle”

```
int middle (int x, int y, int z) {
  int m = z;
  if (y < z) {
    if (x < y)
      m = y;
    else if (x < z)
→     m = y;
  } else {
    if (x > y)
      m = y;
    else if (x > z)
      m = x;
  }
  return m;
}
```

**Missed test case:**

`middle (2, 1, 3) = 1`

**BUG !**

# Specification for Middle

Let $a$ be an array of size $3$ indexed from $0$ to $2$.

$$
\begin{aligned}
& a[i] = x \wedge a[j] = y \wedge a[k] = z \\
\wedge \\
& a[0] \leq a[1] \wedge a[1] \leq a[2] \\
\wedge \\
& i \neq j \wedge i \neq k \wedge j \neq k \\
\rightarrow \\
& m = a[1]
\end{aligned}
$$

Note: coming up with this specification is a manual process

# Encoding of Middle in Logic

```
int m = z;
if (y < z) {
  if (x < y)
    m = y;
  else if (x < z)
    m = y;
} else {
  if (x > y)
    m = y;
  else if (x > z)
    m = x;
}
return m;
```

# Encoding of Middle in Logic

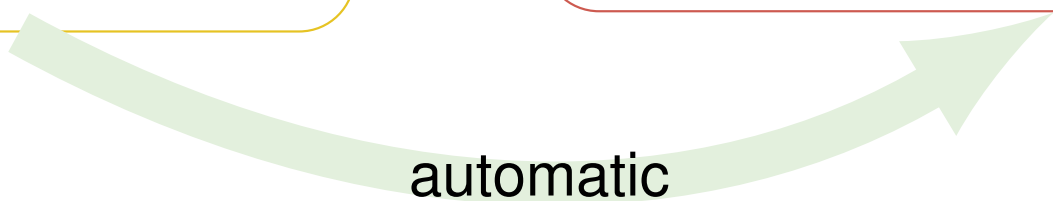
```
int m = z;
if (y < z) {
  if (x < y)
    m = y;
  else if (x < z)
    m = y;
} else {
  if (x > y)
    m = y;
  else if (x > z)
    m = x;
}
return m;
```

$$
\begin{aligned}
&(y < z \wedge x < y \rightarrow m = y) \\
&\wedge \\
&(y < z \wedge x \geq y \wedge x < z \rightarrow m = y) \\
&\wedge \\
&(y < z \wedge x \geq y \wedge x \geq z \rightarrow m = z) \\
&\wedge \\
&(y \geq z \wedge x > y \rightarrow m = y) \\
&\wedge \\
&(y \geq z \wedge x \leq y \wedge x > z \rightarrow m = x) \\
&\wedge \\
&(y \geq z \wedge x \leq y \wedge x \leq z \rightarrow m = z)
\end{aligned}
$$

# Encoding of Middle in Logic

```
int m = z;
if (y < z) {
  if (x < y)
    m = y;
  else if (x < z)
    m = y;
} else {
  if (x > y)
    m = y;
  else if (x > z)
    m = x;
}
return m;
```

$$
\begin{aligned}
&(y < z \wedge x < y \rightarrow m = y) \\
&\wedge \\
&(y < z \wedge x \geq y \wedge x < z \rightarrow m = y) \\
&\wedge \\
&(y < z \wedge x \geq y \wedge x \geq z \rightarrow m = z) \\
&\wedge \\
&(y \geq z \wedge x > y \rightarrow m = y) \\
&\wedge \\
&(y \geq z \wedge x \leq y \wedge x > z \rightarrow m = x) \\
&\wedge \\
&(y \geq z \wedge x \leq y \wedge x \leq z \rightarrow m = z)
\end{aligned}
$$



# Checking Specification as SMT Problem

Let $P$ be the encoding of the program, and $S$ of the specification

- program is correct if "$P \rightarrow S$" is valid

# Checking Specification as SMT Problem

Let $P$ be the encoding of the program, and $S$ of the specification

- program is correct if "$P \rightarrow S$" is valid
- program has a bug if "$P \rightarrow S$" is invalid

# Checking Specification as SMT Problem

Let $P$ be the encoding of the program, and $S$ of the specification

- program is correct if "$P \rightarrow S$" is valid
- program has a bug if "$P \rightarrow S$" is invalid
- program has a bug if negation of "$P \rightarrow S$" is satisfiable

# Checking Specification as SMT Problem

Let $P$ be the encoding of the program, and $S$ of the specification

- program is correct if "$P \rightarrow S$" is valid
- program has a bug if "$P \rightarrow S$" is invalid
- program has a bug if negation of "$P \rightarrow S$" is satisfiable
- program has a bug if "$P \wedge \neg S$" is satisfiable (has a model)

# Checking Specification as SMT Problem

Let $P$ be the encoding of the program, and $S$ of the specification

- program has a bug if "$P \wedge \neg S$" is satisfiable (has a model)

$$\left.\begin{array}{ll}
(y < z \wedge x < y \rightarrow m = y) & \wedge \\
(y < z \wedge x \geq y \wedge x < z \rightarrow m = y) & \wedge \\
(y < z \wedge x \geq y \wedge x \geq z \rightarrow m = z) & \wedge \\
(y \geq z \wedge x > y \rightarrow m = y) & \wedge \\
(y \geq z \wedge x \leq y \wedge x > z \rightarrow m = x) & \wedge \\
(y \geq z \wedge x \leq y \wedge x \leq z \rightarrow m = z) & \wedge
\end{array}\right\} P$$

# Checking Specification as SMT Problem

Let $P$ be the encoding of the program, and $S$ of the specification

- program has a bug if "$P \wedge \neg S$" is satisfiable (has a model)

$$\underbrace{\begin{array}{ll}
(y < z \wedge x < y \rightarrow m = y) & \wedge \\
(y < z \wedge x \geq y \wedge x < z \rightarrow m = y) & \wedge \\
(y < z \wedge x \geq y \wedge x \geq z \rightarrow m = z) & \wedge \\
(y \geq z \wedge x > y \rightarrow m = y) & \wedge \\
(y \geq z \wedge x \leq y \wedge x > z \rightarrow m = x) & \wedge \\
(y \geq z \wedge x \leq y \wedge x \leq z \rightarrow m = z) & \wedge
\end{array}}_{P}$$

$$\underbrace{\begin{array}{ll}
a[i] = x \wedge a[j] = y \wedge a[k] = z & \wedge \\
a[0] \leq a[1] \wedge a[1] \leq a[2] & \wedge \\
i \neq j \wedge i \neq k \wedge j \neq k & \wedge \\
m \neq a[1] &
\end{array}}_{\neg S}$$

# Encoding with LIA in SMTLIB2

```
(set-logic QF_AUFLIA)
(declare-fun x () Int) (declare-fun y () Int) (declare-fun z () Int)
(declare-fun i () Int) (declare-fun j () Int) (declare-fun k () Int)
(declare-fun m () Int) (declare-fun a () (Array Int Int))
(assert (=> (and (< y z) (< x y) ) (= m y)))
(assert (=> (and (< y z) (>= x y) (< x z)) (= m y)))
(assert (=> (and (< y z) (>= x y) (>= x z)) (= m z)))
(assert (=> (and (>= y z) (> x y) ) (= m y)))
(assert (=> (and (>= y z) (<= x y) (> x z) ) (= m x)))
(assert (=> (and (>= y z) (<= x y) (<= x z)) (= m z)))
(assert (and (<= 0 i) (<= i 2) (<= 0 j) (<= j 2) (<= 0 k) (<= k 2)))
(assert (and (= (select a i) x) (= (select a j) y) (= (select a k) z)))
(assert (<= (select a 0) (select a 1) (select a 2)))
(assert (distinct i j k))
(assert (distinct m (select a 1)))
(check-sat) (get-model) (exit)
```

# Encoding with LIA in SMTLIB2

```
(set-logic QF_AUFLIA)
(declare-fun x () Int) (declare-fun y () Int) (declare-fun z () Int)
(declare-fun i () Int) (declare-fun j () Int) (declare-fun k () Int)
(declare-fun m () Int) (declare-fun a () (Array Int Int))
(assert (=> (and (< y z) (< x y) ) (= m y)))
(assert (=> (and (< y z) (>= x y) (< x z)) (= m y)))
(assert (=> (and (< y z) (>= x y) (>= x z)) (= m z)))
(assert (=> (and (>= y z) (> x y) ) (= m y)))
(assert (=> (and (>= y z) (<= x y) (> x z) ) (= m x)))
(assert (=> (and (>= y z) (<= x y) (<= x z)) (= m z)))
(assert (and (<= 0 i) (<= i 2) (<= 0 j) (<= j 2) (<= 0 k) (<= k 2)))
(assert (and (= (select a i) x) (= (select a j) y) (= (select a k) z)))
(assert (<= (select a 0) (select a 1) (select a 2)))
(assert (distinct i j k))
(assert (distinct m (select a 1)))
(check-sat) (get-model) (exit)
```

# Encoding with LIA in SMTLIB2

```
(set-logic QF_AUFLIA)
(declare-fun x () Int) (declare-fun y () Int) (declare-fun z () Int)
(declare-fun i () Int) (declare-fun j () Int) (declare-fun k () Int)
(declare-fun m () Int) (declare-fun a () (Array Int Int))
(assert (=> (and (< y z) (< x y) ) (= m y)))
(assert (=> (and (< y z) (>= x y) (< x z)) (= m y)))
(assert (=> (and (< y z) (>= x y) (>= x z)) (= m z)))
(assert (=> (and (>= y z) (> x y) ) (= m y)))
(assert (=> (and (>= y z) (<= x y) (> x z) ) (= m x)))
(assert (=> (and (>= y z) (<= x y) (<= x z)) (= m z)))
(assert (and (<= 0 i) (<= i 2) (<= 0 j) (<= j 2) (<= 0 k) (<= k 2)))
(assert (and (= (select a i) x) (= (select a j) y) (= (select a k) z)))
(assert (<= (select a 0) (select a 1) (select a 2)))
(assert (distinct i j k))
(assert (distinct m (select a 1)))
(check-sat) (get-model) (exit)
```

# Encoding with LIA in SMTLIB2

```
(set-logic QF_AUFLIA)
(declare-fun x () Int) (declare-fun y () Int) (declare-fun z () Int)
(declare-fun i () Int) (declare-fun j () Int) (declare-fun k () Int)
(declare-fun m () Int) (declare-fun a () (Array Int Int))
(assert (=> (and (< y z) (< x y) ) (= m y)))
(assert (=> (and (< y z) (>= x y) (< x z)) (= m y)))
(assert (=> (and (< y z) (>= x y) (>= x z)) (= m z)))
(assert (=> (and (>= y z) (> x y) ) (= m y)))
(assert (=> (and (>= y z) (<= x y) (> x z) ) (= m x)))
(assert (=> (and (>= y z) (<= x y) (<= x z)) (= m z)))
(assert (and (<= 0 i) (<= i 2) (<= 0 j) (<= j 2) (<= 0 k) (<= k 2)))
(assert (and (= (select a i) x) (= (select a j) y) (= (select a k) z)))
(assert (<= (select a 0) (select a 1) (select a 2)))
(assert (distinct i j k))
(assert (distinct m (select a 1)))
(check-sat) (get-model) (exit)
```

# Encoding with LIA in SMTLIB2

```
(set-logic QF_AUFLIA)
(declare-fun x () Int) (declare-fun y () Int) (declare-fun z () Int)
(declare-fun i () Int) (declare-fun j () Int) (declare-fun k () Int)
(declare-fun m () Int) (declare-fun a () (Array Int Int))
(assert (=> (and (< y z) (< x y) ) (= m y)))
(assert (=> (and (< y z) (>= x y) (< x z)) (= m y)))
(assert (=> (and (< y z) (>= x y) (>= x z)) (= m z)))
(assert (=> (and (>= y z) (> x y) ) (= m y)))
(assert (=> (and (>= y z) (<= x y) (> x z) ) (= m x)))
(assert (=> (and (>= y z) (<= x y) (<= x z)) (= m z)))
(assert (and (<= 0 i) (<= i 2) (<= 0 j) (<= j 2) (<= 0 k) (<= k 2)))
(assert (and (= (select a i) x) (= (select a j) y) (= (select a k) z)))
(assert (<= (select a 0) (select a 1) (select a 2)))
(assert (distinct i j k))
(assert (distinct m (select a 1)))
(check-sat) (get-model) (exit)
```

# Encoding with LIA in SMTLIB2

```
(set-logic QF_AUFLIA)
(declare-fun x () Int) (declare-fun y () Int) (declare-fun z () Int)
(declare-fun i () Int) (declare-fun j () Int) (declare-fun k () Int)
(declare-fun m () Int) (declare-fun a () (Array Int Int))
(assert (=> (and (< y z) (< x y) ) (= m y)))
(assert (=> (and (< y z) (>= x y) (< x z)) (= m y)))
(assert (=> (and (< y z) (>= x y) (>= x z)) (= m z)))
(assert (=> (and (>= y z) (> x y) ) (= m y)))
(assert (=> (and (>= y z) (<= x y) (> x z) ) (= m x)))
(assert (=> (and (>= y z) (<= x y) (<= x z)) (= m z)))
(assert (and (<= 0 i) (<= i 2) (<= 0 j) (<= j 2) (<= 0 k) (<= k 2)))
(assert (and (= (select a i) x) (= (select a j) y) (= (select a k) z)))
(assert (<= (select a 0) (select a 1) (select a 2)))
(assert (distinct i j k))
(assert (distinct m (select a 1)))
(check-sat) (get-model) (exit)
```

# Outline

- Short recap of SAT
- From SAT to SMT
- Decision procedures for selected theories
- Eager and lazy SMT solving
- Combination of theories
- Further topics

# Propositional Logic

- historical origins: ancient Greeks

# Propositional Logic

- historical origins: ancient Greeks
- topic in various areas of science: for example, in philosophy, mathematics, and computer science

# Propositional Logic

- historical origins: ancient Greeks
- topic in various areas of science: for example, in philosophy, mathematics, and computer science
- very simple language
  - no arguments to propositions
  - no functions, no terms
  - no quantifiers

# Propositional Logic

- historical origins: ancient Greeks
- topic in various areas of science: for example, in philosophy, mathematics, and computer science
- very simple language
  - no arguments to propositions
  - no functions, no terms
  - no quantifiers
- deciding satisfiability (SAT) is easy (relative to other logics)

# Propositional Logic

- historical origins: ancient Greeks
- topic in various areas of science: for example, in philosophy, mathematics, and computer science
- very simple language
  - no arguments to propositions
  - no functions, no terms
  - no quantifiers
- deciding satisfiability (SAT) is easy (relative to other logics)
  - archetypical NP-complete problem

# SAT: A Success Story

- used for practical problems (millions of variables)
- used by many different research communities and industry
- some application areas:
  - description of digital circuits
  - automated verification
  - planning, scheduling, configuration problems
  - large research area in theoretical computer science
- huge research community
- annual competition, conference
- standard input format: CNF
- proof checking facilities
- ...

# Propositional Logic: Syntax

The set $\mathcal{L}_{prop}$ of well-formed propositional formulas is the smallest set such that

1. $\top, \bot \in \mathcal{L}_{prop}$;
2. $\mathcal{P}_0 \subseteq \mathcal{L}_{prop}$ where $\mathcal{P}_0$ is the set of atomic propositions (atoms, variables);
3. if $\phi \in \mathcal{L}_{prop}$ then $(\neg\phi) \in \mathcal{L}_{prop}$;
4. if $\phi, \psi \in \mathcal{L}_{prop}$ then $(\phi \circ \psi) \in \mathcal{L}_{prop}$ with $\circ \in \{\vee, \wedge, \leftrightarrow, \rightarrow\}$.

- $\mathcal{L}_{prop}$ is the language of propositional logic.
- The elements of $\mathcal{L}_{prop}$ are propositional formulas.

# Propositional Logic: Semantics

Given an assignment $\nu : \mathcal{P}_0 \to \mathbb{B}$,
the interpretation $[.]_\nu : \mathcal{L}_{prop} \to \mathbb{B}$ is defined by:

- $[\top]_\nu = \mathbf{1}$, $[\bot]_\nu = \mathbf{0}$
- if $x \in \mathcal{P}_0$ then $[x]_\nu = \nu(x)$
- $[\neg\phi]_\nu = \mathbf{1}$ iff $[\phi]_\nu = \mathbf{0}$
- $[\phi \vee \psi]_\nu = \mathbf{1}$ iff $[\phi]_\nu = \mathbf{1}$ or $[\psi]_\nu = \mathbf{1}$

# Propositional Logic: Semantics

Given an assignment $\nu : \mathcal{P}_0 \to \mathbb{B}$,
the interpretation $[.]_\nu : \mathcal{L}_{prop} \to \mathbb{B}$ is defined by:

- $[\top]_\nu = \mathbf{1}$, $[\bot]_\nu = \mathbf{0}$
- if $x \in \mathcal{P}_0$ then $[x]_\nu = \nu(x)$
- $[\neg\phi]_\nu = \mathbf{1}$ iff $[\phi]_\nu = \mathbf{0}$
- $[\phi \vee \psi]_\nu = \mathbf{1}$ iff $[\phi]_\nu = \mathbf{1}$ or $[\psi]_\nu = \mathbf{1}$

- An assignment
  - **satisfies** a formula $\phi$ iff $[\phi]_\nu = \mathbf{1}$.
  - **falsifies** a formula $\phi$ iff $[\phi]_\nu = \mathbf{0}$.

# Propositional Logic: Semantics

Given an assignment $\nu : \mathcal{P}_0 \to \mathbb{B}$,
the interpretation $[.]_\nu : \mathcal{L}_{prop} \to \mathbb{B}$ is defined by:

- $[\top]_\nu = \mathbf{1}$, $[\bot]_\nu = \mathbf{0}$
- if $x \in \mathcal{P}_0$ then $[x]_\nu = \nu(x)$
- $[\neg\phi]_\nu = \mathbf{1}$ iff $[\phi]_\nu = \mathbf{0}$
- $[\phi \vee \psi]_\nu = \mathbf{1}$ iff $[\phi]_\nu = \mathbf{1}$ or $[\psi]_\nu = \mathbf{1}$

- An assignment
  - **satisfies** a formula $\phi$ iff $[\phi]_\nu = \mathbf{1}$.
  - **falsifies** a formula $\phi$ iff $[\phi]_\nu = \mathbf{0}$.
- A satisfying assignment is called **model**.

# Propositional Logic: Semantics

Given an assignment $\nu : \mathcal{P}_0 \to \mathbb{B}$,
the interpretation $[.]_\nu : \mathcal{L}_{prop} \to \mathbb{B}$ is defined by:

- $[\top]_\nu = \mathbf{1}$, $[\bot]_\nu = \mathbf{0}$
- if $x \in \mathcal{P}_0$ then $[x]_\nu = \nu(x)$
- $[\neg\phi]_\nu = \mathbf{1}$ iff $[\phi]_\nu = \mathbf{0}$
- $[\phi \vee \psi]_\nu = \mathbf{1}$ iff $[\phi]_\nu = \mathbf{1}$ or $[\psi]_\nu = \mathbf{1}$

- An assignment
  - **satisfies** a formula $\phi$ iff $[\phi]_\nu = \mathbf{1}$.
  - **falsifies** a formula $\phi$ iff $[\phi]_\nu = \mathbf{0}$.
- A satisfying assignment is called **model**.
- A falsifying assignment is called **counter-model**.

# Properties of (Propositional) Formulas (1/2)

- formula $\phi$ is **satisfiable** iff there exists $[.]_\nu$ with $[\phi]_\nu = \mathbf{1}$

# Properties of (Propositional) Formulas (1/2)

- formula $\phi$ is **satisfiable** iff there exists $[.]_\nu$ with $[\phi]_\nu = \mathbf{1}$
- formula $\phi$ is **valid** iff for all $[.]_\nu$ it holds that $[\phi]_\nu = \mathbf{1}$

# Properties of (Propositional) Formulas (1/2)

- formula $\phi$ is **satisfiable** iff there exists $[.]_\nu$ with $[\phi]_\nu = \mathbf{1}$
- formula $\phi$ is **valid** iff for all $[.]_\nu$ it holds that $[\phi]_\nu = \mathbf{1}$
- a valid formula is called **tautology**

# Properties of (Propositional) Formulas (1/2)

- formula $\phi$ is **satisfiable** iff there exists $[.]_\nu$ with $[\phi]_\nu = \mathbf{1}$
- formula $\phi$ is **valid** iff for all $[.]_\nu$ it holds that $[\phi]_\nu = \mathbf{1}$
- a valid formula is called **tautology**
- formula $\phi$ is **refutable** iff there exists $[.]_\nu$ with $[\phi]_\nu = \mathbf{0}$

# Properties of (Propositional) Formulas (1/2)

- formula $\phi$ is **satisfiable** iff there exists $[.]_\nu$ with $[\phi]_\nu = \mathbf{1}$
- formula $\phi$ is **valid** iff for all $[.]_\nu$ it holds that $[\phi]_\nu = \mathbf{1}$
- a valid formula is called **tautology**
- formula $\phi$ is **refutable** iff there exists $[.]_\nu$ with $[\phi]_\nu = \mathbf{0}$
- formula $\phi$ is **unsatisfiable** iff $[\phi]_\nu = \mathbf{0}$ for all $[.]_\nu$

# Properties of (Propositional) Formulas (1/2)

- formula $\phi$ is **satisfiable** iff there exists $[.]_\nu$ with $[\phi]_\nu = \mathbf{1}$
- formula $\phi$ is **valid** iff for all $[.]_\nu$ it holds that $[\phi]_\nu = \mathbf{1}$
- a valid formula is called **tautology**
- formula $\phi$ is **refutable** iff there exists $[.]_\nu$ with $[\phi]_\nu = \mathbf{0}$
- formula $\phi$ is **unsatisfiable** iff $[\phi]_\nu = \mathbf{0}$ for all $[.]_\nu$
- an unsatisfiable formula is called **contradiction**

# Properties of Propositional Formulas (2/2)

A **satisfiable** formula is

- possibly valid
- possibly refutable
- not unsatisfiable.

# Properties of Propositional Formulas (2/2)

A **satisfiable** formula is

- possibly valid
- possibly refutable
- not unsatisfiable.

A **valid** formula is

- satisfiable
- not refutable
- not unsatisfiable.

# Properties of Propositional Formulas (2/2)

A **satisfiable** formula is

- possibly valid
- possibly refutable
- not unsatisfiable.

A **refutable** formula is

- possibly satisfiable
- possibly unsatisfiable
- not valid.

A **valid** formula is

- satisfiable
- not refutable
- not unsatisfiable.

# Properties of Propositional Formulas (2/2)

A **satisfiable** formula is

- possibly valid
- possibly refutable
- not unsatisfiable.

A **refutable** formula is

- possibly satisfiable
- possibly unsatisfiable
- not valid.

A **valid** formula is

- satisfiable
- not refutable
- not unsatisfiable.

An **unsatisfiable** formula is

- refutable
- not valid
- not satisfiable.

# Special Formula Structures

- **literal**: variable or a negated variable (also (negated) truth constants)
  - examples of literals: $x, \neg x, y, \neg y$

# Special Formula Structures

- **literal**: variable or a negated variable (also (negated) truth constants)
  - examples of literals: $x, \neg x, y, \neg y$
- **clause**: disjunction of literals
  - unary clause (clause of size one): $l$ where $l$ is a literal
  - empty clause (clause of size zero): $\bot$
  - examples of clauses: $(x \vee y)$, $(\neg x \vee x' \vee \neg x'')$, $x, \neg y$

# Special Formula Structures

- **literal**: variable or a negated variable (also (negated) truth constants)
  - examples of literals: $x, \neg x, y, \neg y$
- **clause**: disjunction of literals
  - unary clause (clause of size one): $l$ where $l$ is a literal
  - empty clause (clause of size zero): $\bot$
  - examples of clauses: $(x \vee y), (\neg x \vee x' \vee \neg x''), x, \neg y$
- **cube**: conjunction of literals
  - unary cube (cube of size one): $l$ where $l$ is a literal
  - empty cube (cube of size zero): $\top$
  - examples of cubes: $(x \wedge y), (\neg x \wedge x' \wedge \neg x''), x, \neg y$

# Negation Normal Form

Negation Normal Form (NNF) is defined as follows:

- Literals and truth constants are in NNF;
- $\phi \circ \psi$ ($\circ \in \{\vee, \wedge\}$) is in NNF iff $\phi$ and $\psi$ are in NNF;
- no other formulas are in NNF.

**In other words:**

A formula in NNF contains only conjunctions, disjunctions, and negations and negations only occur in front of variables and constants.

# Special Formula Structures: Conjunctive Normal Form

A propositional formula is in conjunctive normal form (CNF) iff it is a conjunction of clauses.

A formula in conjunctive normal form is

- in negation normal form.
- $\top$ if it contains no clauses.
- easy to check whether it can be refuted (can be set to false).

remark: CNF is the input of most SAT-solvers (DIMACS format).

# Transformation to CNF

1. transform formula to NNF
2. for complex subformula $\psi$ generate new variable $x_\psi$
3. replace $\psi$ by $x_\psi$
4. add definition $\psi \leftrightarrow x_\psi$ as clauses
5. collect all constraints in a big conjunction

# Transformation to CNF

1. transform formula to NNF
2. for complex subformula $\psi$ generate new variable $x_\psi$
3. replace $\psi$ by $x_\psi$
4. add definition $\psi \leftrightarrow x_\psi$ as clauses
5. collect all constraints in a big conjunction

- the transformation is **satisfiability equivalent**:
  the result is satisfiable iff the original formula is satisfiable

# Transformation to CNF

1. transform formula to NNF
2. for complex subformula $\psi$ generate new variable $x_\psi$
3. replace $\psi$ by $x_\psi$
4. add definition $\psi \leftrightarrow x_\psi$ as clauses
5. collect all constraints in a big conjunction

- the transformation is **satisfiability equivalent**:
  the result is satisfiable iff the original formula is satisfiable
- not logically equivalent: **new variables**

# Transformation to CNF

1. transform formula to NNF
2. for complex subformula $\psi$ generate new variable $x_\psi$
3. replace $\psi$ by $x_\psi$
4. add definition $\psi \leftrightarrow x_\psi$ as clauses
5. collect all constraints in a big conjunction

- the transformation is **satisfiability equivalent**:
  the result is satisfiable iff the original formula is satisfiable
- not logically equivalent: **new variables**
- extract satisfying assignment for original formula from one assignment satisfying the CNF

# Tseitin Transformation: Definitions

## Negation

$$
\begin{aligned}
x \leftrightarrow \overline{y} \quad &\Leftrightarrow \quad (x \rightarrow \overline{y}) \wedge (\overline{y} \rightarrow x) \\
&\Leftrightarrow \quad (\overline{x} \vee \overline{y}) \wedge (y \vee x)
\end{aligned}
$$

# Tseitin Transformation: Definitions

## Disjunction

$$\begin{aligned} x \leftrightarrow (y \vee z) \quad &\Leftrightarrow \quad (y \rightarrow x) \wedge (z \rightarrow x) \wedge (x \rightarrow (y \vee z)) \\ &\Leftrightarrow \quad (\overline{y} \vee x) \wedge (\overline{z} \vee x) \wedge (\overline{x} \vee y \vee z) \end{aligned}$$

# Tseitin Transformation: Definitions

## Conjunction

$$
\begin{aligned}
x \leftrightarrow (y \wedge z) \quad &\Leftrightarrow \quad (x \rightarrow y) \wedge (x \rightarrow z) \wedge ((y \wedge z) \rightarrow x) \\
&\Leftrightarrow \quad (\overline{x} \vee y) \wedge (\overline{x} \vee z) \wedge (\overline{(y \wedge z)} \vee x) \\
&\Leftrightarrow \quad (\overline{x} \vee y) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z} \vee x)
\end{aligned}
$$

# Tseitin Transformation: Definitions

## Equivalence

$$
\begin{aligned}
x \leftrightarrow (y \leftrightarrow z) \quad &\Leftrightarrow \quad (x \rightarrow (y \leftrightarrow z)) \wedge ((y \leftrightarrow z) \rightarrow x) \\
&\Leftrightarrow \quad (x \rightarrow ((y \rightarrow z) \wedge (z \rightarrow y)) \wedge ((y \leftrightarrow z) \rightarrow x) \\
&\Leftrightarrow \quad (x \rightarrow (y \rightarrow z)) \wedge (x \rightarrow (z \rightarrow y)) \wedge ((y \leftrightarrow z) \rightarrow x) \\
&\Leftrightarrow \quad (\overline{x} \vee \overline{y} \vee z) \wedge (\overline{x} \vee \overline{z} \vee y) \wedge ((y \leftrightarrow z) \rightarrow x) \\
&\Leftrightarrow \quad (\overline{x} \vee \overline{y} \vee z) \wedge (\overline{x} \vee \overline{z} \vee y) \wedge (((y \wedge z) \vee (\overline{y} \wedge \overline{z})) \rightarrow x) \\
&\Leftrightarrow \quad (\overline{x} \vee \overline{y} \vee z) \wedge (\overline{x} \vee \overline{z} \vee y) \wedge ((y \wedge z) \rightarrow x) \wedge ((\overline{y} \wedge \overline{z}) \rightarrow x) \\
&\Leftrightarrow \quad (\overline{x} \vee \overline{y} \vee z) \wedge (\overline{x} \vee \overline{z} \vee y) \wedge (\overline{y} \vee \overline{z} \vee x) \wedge (y \vee z \vee x)
\end{aligned}
$$

# Resolution

- the resolution calculus consists of the single resolution rule

$$\frac{x \vee C \qquad \neg x \vee D}{C \vee D}$$

  - $C$ and $D$ are (possibly empty) clauses
  - the clause $C \vee D$ is called resolvent
  - variable $x$ is called pivot
  - usually antecedent clauses $x \vee C$ and $\neg x \vee D$ are assumed not to be tautological, that is $x \notin C$ and $x \notin D$.

- in other words:
  $(\neg x \to C), (x \to D) \Rightarrow C \vee D$

- resolution is sound and complete.

# Resolution

one application of resolution

$$\frac{x \vee y \vee \neg z \qquad \neg x \vee y' \vee \neg z}{y \vee \neg z \vee y'}$$

derivation of empty clause:

$$\frac{y \qquad \neg y}{\bot}$$

derivation of tautology:

$$\frac{x \vee a \qquad \neg x \vee \neg a}{a \vee \neg a}$$

- the resolution calculus works only on formulas in CNF
- if the empty clause can be derived then the formula is unsatisfiable
- if no new clause can be generated by application of the resolution rule then the formula is satisfiable

# Binary Constraint Propagation

Let $\phi$ be a formula in CNF containing a unit clause $C$, i.e., $\phi$ has a clause $C = (l)$ which consists only of literal $l$. Then $BCP(\phi, l)$ is obtained from $\phi$ by

- removing all clauses with $l$
- removing all occurrences of $\bar{l}$

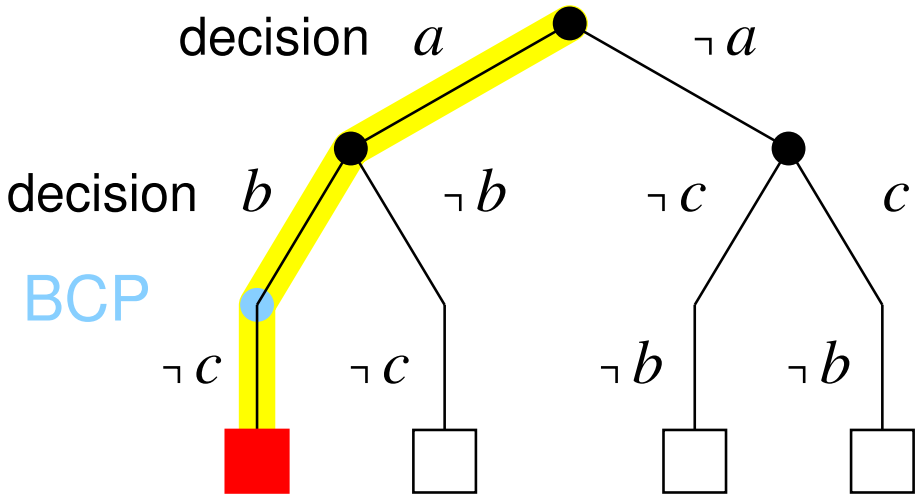
- BCP can trigger other applications of BCP
- if BCP results in empty clause, then formula is unsatisfiable
- if BCP produces the empty CNF, then formula satisfiable

# DPLL

$a = 1$

$b = 1$

$c = 0$




clauses

$\neg a \vee \neg b \vee \neg c$

$\neg a \vee \neg b \vee c$

$\neg a \vee b \vee \neg c$

$\neg a \vee b \vee c$

$a \vee \neg b \vee \neg c$

$a \vee \neg b \vee c$

$a \vee b \vee \neg c$

$a \vee b \vee c$

# Sometimes SAT

# is not enough !!!

# Expressiveness against Efficiency

- **SAT**: efficient, involved encodings
- **FO**: first-order logic): often too powerful
- satisfiability with respect to some theory is required
  (non-standard interpretations are not of interest)
  **Example**: $x + y < z \vee \neg(x + 1 \leq y \rightarrow x < z)$
- theory needs not be first-order axiomatizable
- specialized inference method for each theory
- **SMT**: sweetspot between SAT and FO
  - propositional logic + domain specific reasoning
  - in general more efficient than with general-purpose solvers with incorporated theory axioms

# Satisfiability Modulo Theories (SMT)

$$f(x) \neq f(y) \wedge x+u = 3 \wedge v+y = 3 \wedge u = a[z] \wedge v = a[w] \wedge z = w$$

- formulas in first-order logic
  - usually without quantifiers, variables implicitly existentially quantified
  - but with sorted / typed symbols and
  - functions / constants / predicates are interpreted
  - SMT quantifier reasoning weaker than in first-order theorem proving (FO)
  - much richer language compared to propositional logic (SAT)
- many (industrial) applications
  - standardized language SMTLIB used in applications and competitions

# SMT Solvers

- **roots**: late 1970s and early 1980s
- **modern research**: 1990s - exploit new SAT technology
- **today**: solvers from industry and academia
- **mainly two solving approaches**
  - eager
  - lazy

# Theories in SMT

- no need to axiomatize “theories” using axioms with quantifiers
  - important theories are “built-in”: **uninterpreted functions, equality, arithmetic, arrays, bit-vectors** . . .
  - focus is on decidable theories, thus fully automatic procedures
- state-of-the-art SMT solvers essentially rely on SAT solvers
  - direct encoding to SAT
  - SAT solver enumerates sets of literals to be checked
  - propositional and theory conflicts recorded as propositional clauses
  - DPLL(T), CDCL (T), read DPLL modulo theory T or CDCL modulo T

# Syntax of SMT Formulas: Signature

A signature $\Sigma$ is a pair $(\mathcal{F}, \mathcal{P})$ where

- $\mathcal{F}$ is a set of function symbols (with arities)
  - $\mathcal{F}_0$ denotes the set of constants ($\mathcal{F}_0 \subseteq \mathcal{F}$)
- $\mathcal{P}$ is a set of predicate symbols (with arities)
  - $\mathcal{P}_0$ denotes the set of propositional atoms ($\mathcal{P}_0 \subseteq \mathcal{P}$)

- the elements of $\Sigma$ are the building blocks of formulas
- example: $\Sigma = (\{x/0, y/0, 0/0, +/2\}, \{< /2\})$
- remarks:
  1. we consider only ground (variable-free) formulas
  2. we assume the considered expressions to be typed

# Syntax of SMT Formulas: Terms

Let $\Sigma = (\mathcal{F}, \mathcal{P})$ be a signature. Then the set of terms $\mathsf{Terms}_\Sigma$ over $\Sigma$ is the smallest set such that

1. if $c \in \mathcal{F}_0$ then $c \in \mathsf{Terms}_\Sigma$;
2. if $f/n \in \mathcal{F}$, $n > 0$, and $t_1, \ldots, t_n \in \mathsf{Terms}_\Sigma$ then $f(t_1, \ldots, t_n) \in \mathsf{Terms}_\Sigma$.

Example

- Given $\Sigma = (\{x/0, y/0, 0/0, +/2\}, \{< /2\})$, then terms are $x$, $x + y$, $x + 0, \ldots$
- note: the number of possible terms is infinite

# Syntax of SMT Formula: Atoms and Literals

Let $\Sigma = (\mathcal{F}, \mathcal{P})$ be a signature, $p/n \in \mathcal{P}$ with $n \geq 0$, and let $t_1, \ldots, t_n$ be terms.

- $p(t_1, \ldots, t_n)$ is an atomic formula (also atom).

**Example**: Given $\Sigma = (\{x/0, y/0, 0/0, +/2\}, \{< /2\})$:

- atoms: $x < 0, x + y < 0 + x, \ldots$
- literals: $x < 0, x + y < 0 + x, \neg(x < 0), \ldots$

**Important predicates**: $=, \neq$

# Syntax of SMT Formula: Atoms and Literals

Let $\Sigma = (\mathcal{F}, \mathcal{P})$ be a signature, $p/n \in \mathcal{P}$ with $n \geq 0$, and let $t_1, \ldots, t_n$ be terms.

- $p(t_1, \ldots, t_n)$ is an atomic formula (also atom).
- A literal is an atom $p(t_1, \ldots, t_n)$ or its negation $\neg p(t_1, \ldots, t_n)$.

**Example**: Given $\Sigma = (\{x/0, y/0, 0/0, +/2\}, \{< /2\})$:

- atoms: $x < 0, x + y < 0 + x, \ldots$
- literals: $x < 0, x + y < 0 + x, \neg(x < 0), \ldots$

**Important predicates**: $=, \neq$

# Syntax of SMT Formulas

The set $\mathcal{L}$ of well-formed SMT formulas over signature $\Sigma$ is the smallest set such that

1. $\top, \bot \in \mathcal{L}$;
2. $p(t_1, \ldots t_n) \in \mathcal{L}$ where $p(t_1, \ldots, t_n)$ is an atom over $\Sigma$;
3. if $\phi \in \mathcal{L}$ then $(\neg\phi) \in \mathcal{L}$;
4. if $\phi, \psi \in \mathcal{L}$ then $(\phi \circ \psi) \in \mathcal{L}$ with $\circ \in \{\vee, \wedge, \leftrightarrow, \rightarrow\}$.

# Syntax of SMT Formulas

The set $\mathcal{L}$ of well-formed SMT formulas over signature $\Sigma$ is the smallest set such that

1. $\top, \bot \in \mathcal{L}$;
2. $p(t_1, \ldots t_n) \in \mathcal{L}$ where $p(t_1, \ldots, t_n)$ is an atom over $\Sigma$;
3. if $\phi \in \mathcal{L}$ then $(\neg\phi) \in \mathcal{L}$;
4. if $\phi, \psi \in \mathcal{L}$ then $(\phi \circ \psi) \in \mathcal{L}$ with $\circ \in \{\vee, \wedge, \leftrightarrow, \rightarrow\}$.

- Note: clauses, cubes, negation normal form, conjunctive normal form as before

# Syntax of SMT Formulas

The set $\mathcal{L}$ of well-formed SMT formulas over signature $\Sigma$ is the smallest set such that

1. $\top, \bot \in \mathcal{L}$;
2. $p(t_1, \ldots t_n) \in \mathcal{L}$ where $p(t_1, \ldots, t_n)$ is an atom over $\Sigma$;
3. if $\phi \in \mathcal{L}$ then $(\neg\phi) \in \mathcal{L}$;
4. if $\phi, \psi \in \mathcal{L}$ then $(\phi \circ \psi) \in \mathcal{L}$ with $\circ \in \{\vee, \wedge, \leftrightarrow, \rightarrow\}$.

- Note: clauses, cubes, negation normal form, conjunctive normal form as before

**Where are the theories ???**

# Definition of a Theory

A first-order theory $\mathcal{T} = (\Sigma, \mathcal{A})$ is defined by

1. its signature $\Sigma$,
2. its axiomms $\mathcal{A}$ (often first-order formulas over $\Sigma$, with quantifiers).

**Example**: Axioms in theory of equality are

- reflexivity
- symmetry
- transitivity
- congruence

A theory literal is a literal over $\Sigma$.

# Semantics of SMT Formulas: Interpretation

An interpretation w.r.t. a signature $\Sigma$ is is a pair $(\mathcal{U}, \_^{\mathcal{I}})$ where

1. the domain $\mathcal{U}$ is a non-empty set of symbols;
2. the mapping $\_^{\mathcal{I}}$: is defined as follows:
   - $x^{\mathcal{I}} \in \{\mathbf{1}, \mathbf{0}\}$ for $x \in \mathcal{P}_0$;
   - $c^{\mathcal{I}} \in \mathcal{U}$ for $c \in \mathcal{F}_0$;
   - $f^{\mathcal{I}} : \mathcal{U}^n \mapsto \mathcal{U}$ for $f/n \in \mathcal{F}$ ;
   - $p^{\mathcal{I}} : \mathcal{U}^n \mapsto \{\mathbf{1}, \mathbf{0}\}$ for $p/n \in \mathcal{P}$.

# Semantics of SMT Formulas: Interpretation

An interpretation w.r.t. a signature $\Sigma$ is is a pair $(\mathcal{U}, \_^{\mathcal{I}})$ where

1. the domain $\mathcal{U}$ is a non-empty set of symbols;
2. the mapping $\_^{\mathcal{I}}$: is defined as follows:
   - $x^{\mathcal{I}} \in \{\mathbf{1}, \mathbf{0}\}$ for $x \in \mathcal{P}_0$;
   - $c^{\mathcal{I}} \in \mathcal{U}$ for $c \in \mathcal{F}_0$;
   - $f^{\mathcal{I}} : \mathcal{U}^n \mapsto \mathcal{U}$ for $f/n \in \mathcal{F}$ ;
   - $p^{\mathcal{I}} : \mathcal{U}^n \mapsto \{\mathbf{1}, \mathbf{0}\}$ for $p/n \in \mathcal{P}$.

The interpretation of a term $f(t_1, \ldots, t_n)$ over $\Sigma$ with function symbol $f/n \in \mathcal{F}$ and terms $t_1, \ldots t_n$ defined by

$$(f(t_1, \ldots, t_n))^{\mathcal{I}} = f^{\mathcal{I}}(t_1^{\mathcal{I}}, \ldots, t_n^{\mathcal{I}}).$$

# Semantics of SMT Formulas: Interpretation

An interpretation w.r.t. a signature $\Sigma$ is is a pair $(\mathcal{U}, \_^{\mathcal{I}})$ where

1. the domain $\mathcal{U}$ is a non-empty set of symbols;
2. the mapping $\_^{\mathcal{I}}$: is defined as follows:
   - $x^{\mathcal{I}} \in \{\mathbf{1}, \mathbf{0}\}$ for $x \in \mathcal{P}_0$;
   - $c^{\mathcal{I}} \in \mathcal{U}$ for $c \in \mathcal{F}_0$;
   - $f^{\mathcal{I}} : \mathcal{U}^n \mapsto \mathcal{U}$ for $f/n \in \mathcal{F}$ ;
   - $p^{\mathcal{I}} : \mathcal{U}^n \mapsto \{\mathbf{1}, \mathbf{0}\}$ for $p/n \in \mathcal{P}$.

The interpretation of an atom $p(t_1, \ldots, t_n)$ over $\Sigma$ with predicate symbol $p/n \in \mathcal{P}$ and terms $t_1, \ldots t_n$ defined by

$$(p(t_1, \ldots, t_n))^{\mathcal{I}} = p^{\mathcal{I}}(t_1^{\mathcal{I}}, \ldots, t_n^{\mathcal{I}}).$$

# SMT Formulas: Semantics

Given an interpretation $(\mathcal{U}, \_^{\mathcal{I}})$ w.r.t. signature $\Sigma$.
The evaluation function $[]_{\mathcal{I}} : \mathcal{L} \mapsto \mathbb{B}$ for a formula over $\Sigma$ is defined by:

- $[\top]_{\mathcal{I}} = \mathbf{1}$, $[\bot]_{\mathcal{I}} = \mathbf{0}$
- $[p(t_1, \ldots, t_n)]_{\mathcal{I}} = \mathbf{1}$ iff $(p(t_1, \ldots, t_n))^{\mathcal{I}} = \mathbf{1}$
- $[\neg\phi]_{\mathcal{I}} = \mathbf{1}$ iff $[\phi]_{\mathcal{I}} = \mathbf{0}$
- $[\phi \vee \psi]_{\mathcal{I}} = \mathbf{1}$ iff $[\phi]_{\mathcal{I}} = \mathbf{1}$ or $[\psi]_{\mathcal{I}} = \mathbf{1}$
- $[\phi \wedge \psi]_{\mathcal{I}} = \mathbf{1}$ iff $[\phi]_{\mathcal{I}} = \mathbf{1}$ and $[\psi]_{\mathcal{I}} = \mathbf{1}$

# Semantics under a Theory

Let $\mathcal{T} = (\Sigma, \mathcal{A})$ be a theory, let $\Gamma$ be a set of formulas over $\Sigma$, and let $\phi$ be a formula over $\Sigma$.

- We are interested only in interpretations respecting $\mathcal{A}$.
  - An interpretation respecting $\mathcal{A}$ is called **$\mathcal{T}$-interpretation**.
- A $\mathcal{T}$-model of formula $\phi$ is a $\mathcal{T}$-interpretation satisfying $\phi$.
- A formula is $\mathcal{T}$-satisfiable iff it has a $\mathcal{T}$-model.
- $\Gamma$ $\mathcal{T}$-entails a formula $\phi$ ($\Gamma \models_{\mathcal{T}} \phi$) iff every $\mathcal{T}$-model of $\Gamma$ is a $\mathcal{T}$-model of $\phi$.
- $\Gamma$ is $\mathcal{T}$-consistent iff $\Gamma \not\models_{\mathcal{T}} \bot$.
- $\phi$ is $\mathcal{T}$-valid iff $\top \models_{\mathcal{T}} \phi$.

# Combination of Theories

- Often one theory is not enough.
- Practical applications need combinations of arithmetic, arrays, ...
- Challenge: Combine decision procedures of different theories
- The union $\mathcal{T}_1 \cup \mathcal{T}_2$ of theories $\mathcal{T}_1$ and $\mathcal{T}_2$ with $\mathcal{T}_1 = (\Sigma_1, \mathcal{A}_1)$ and $\mathcal{T}_2 = (\Sigma_2, \mathcal{A}_2)$ is given by
  1. signature $\Sigma_1 \cup \Sigma_2$,
  2. axioms $\mathcal{A}_1 \cup \mathcal{A}_2$.
- 

# Equality with Uninterpreted Functions (EUF)

- functions as in first-order (FO): sorted / typed without interpretation
- equality as single interpreted predicate
  - congruence axiom $\forall x, y \colon x = y \rightarrow f(x) = f(y)$
  - similar variants for functions with multiple arguments
- uninterpreted functions allow to abstract from concrete implementations

# Equality with Uninterpreted Functions (EUF)

- functions as in first-order (FO): sorted / typed without interpretation
- equality as single interpreted predicate
  - congruence axiom $\forall x, y \colon x = y \rightarrow f(x) = f(y)$
  - similar variants for functions with multiple arguments
- uninterpreted functions allow to abstract from concrete implementations

Example: $a * (f(b) + f(c)) = d, b * (f(a) + f(c)) \neq d, a = b$

# Equality with Uninterpreted Functions (EUF)

- functions as in first-order (FO): sorted / typed without interpretation
- equality as single interpreted predicate
  - congruence axiom $\forall x, y \colon x = y \rightarrow f(x) = f(y)$
  - similar variants for functions with multiple arguments
- uninterpreted functions allow to abstract from concrete implementations

Example: $a * (f(b) + f(c)) = d, b * (f(a) + f(c)) \neq d, a = b$

Abstraction:

$(h(a, g(f(b), f(c)) = d, h(b, g(f(a), f(c))) \neq d, a = b)$

# Congruence Closure By Example (1/2)

assume flattened structure where all sub-terms are identified by variables

$$[x \mid y \mid t \mid u \mid v]$$

$$\underbrace{x = y}\wedge\, x = g(y) \wedge t = g(x) \wedge u = f(x,t) \wedge v = f(y,x) \wedge u \neq v$$

asserted literal $x = y$ puts $x$ and $y$ in to the same equivalence class

$$[x\ y \mid t \mid u \mid v]$$

# Congruence Closure By Example (1/2)

assume flattened structure where all sub-terms are identified by variables

$$[x \mid y \mid t \mid u \mid v]$$

$$\underbrace{x = y}\wedge\, x = g(y) \wedge t = g(x) \wedge u = f(x,t) \wedge v = f(y,x) \wedge u \neq v$$

asserted literal $x = y$ puts $x$ and $y$ in to the same equivalence class

$$[x\ y \mid t \mid u \mid v]$$

$$x = y \wedge\ \underbrace{x = g(y) \wedge t = g(x)} \wedge u = f(x,t) \wedge v = f(y,x) \wedge u \neq v$$

apply congruence axiom since $x$ and $y$ in same equivalence class

$$[x\ y\ t \mid u \mid v]$$

# Congruence Closure By Example (2/2)

$$[x\ y\ t \mid u \mid v]$$

$$x = y \wedge x = g(y) \wedge t = g(x) \wedge \underbrace{u = f(x,t) \wedge v = f(y,x)}_{\text{apply congruence axiom since } y,\ x \text{ and } t \text{ are all in same equivalence class}} \wedge u \neq v$$

# Congruence Closure By Example (2/2)

$$[x\ y\ t \mid u \mid v]$$

$$x = y \wedge x = g(y) \wedge t = g(x) \wedge \underbrace{u = f(x,t) \wedge v = f(y,x)}_{\text{apply congruence axiom since } y,\ x \text{ and } t \text{ are all in same equivalence class}} \wedge u \neq v$$

$$[x\ y\ t \mid u\ v]$$

$$x = y \wedge x = g(y) \wedge t = g(x) \wedge u = f(x,t) \wedge v = f(y,x) \wedge u \neq v$$

# Congruence Closure By Example (2/2)

$$[x\ y\ t \mid u \mid v]$$

$$x = y \wedge x = g(y) \wedge t = g(x) \wedge \underbrace{u = f(x,t) \wedge v = f(y,x)}_{\text{apply congruence axiom since } y,\ x \text{ and } t \text{ are all in same equivalence class}} \wedge u \neq v$$

$$[x\ y\ t \mid u\ v]$$

$$x = y \wedge x = g(y) \wedge t = g(x) \wedge u = f(x,t) \wedge v = f(y,x) \wedge u \neq v$$

$u$ and $v$ in the same equivalence class but $u \neq v$ asserted

UNSATISFIABLE

# Theory of Linear Real Arithmetic (LRA)

Example: $z \leq x - y \ \wedge \ x + 2 \cdot y \leq 5 \ \wedge \ 4 \cdot z - 2 \cdot x \geq y$

- **constants**: integers, rationals, etc.
- **predicates**: equality $=$, disequality $\neq$, inequality $\leq$, $<$, ...
- **functions**: addition $+$, subtraction $-$, multiplication $\cdot$ by constant only

- related to optimization problems solved in operation research (OR)

# Fourier-Motzkin Elimination by Example (1/2)

$$z \leq x - y \quad \wedge \quad x + 2 \cdot y \leq 5 \quad \wedge \quad 4 \cdot z - 2 \cdot x \geq y$$

# Fourier-Motzkin Elimination by Example (1/2)

$$z \leq x - y \quad \wedge \quad x + 2 \cdot y \leq 5 \quad \wedge \quad 4 \cdot z - 2 \cdot x \geq y$$

pick pivot variable, e.g. $x$, and isolate it on one side

# Fourier-Motzkin Elimination by Example (1/2)

$$z \leq x - y \quad \wedge \quad x + 2 \cdot y \leq 5 \quad \wedge \quad 4 \cdot z - 2 \cdot x \geq y$$

pick pivot variable, e.g. $x$, and isolate it on one side

$$z + y \leq x \quad \wedge \quad x \leq 5 - 2 \cdot y \quad \wedge \quad 4 \cdot z - y \geq 2 \cdot x$$
$$z + y \leq x \quad \wedge \quad x \leq 5 - 2 \cdot y \quad \wedge \quad 2 \cdot z - 0.5 \cdot y \geq x$$
$$z + y \leq x \quad \wedge \quad x \leq 5 - 2 \cdot y \quad \wedge \quad x \leq 2 \cdot z - 0.5 \cdot y$$

# Fourier-Motzkin Elimination by Example (1/2)

$$z \leq x - y \quad \wedge \quad x + 2 \cdot y \leq 5 \quad \wedge \quad 4 \cdot z - 2 \cdot x \geq y$$

pick pivot variable, e.g. $x$, and isolate it on one side

$$z + y \leq x \quad \wedge \quad x \leq 5 - 2 \cdot y \quad \wedge \quad 4 \cdot z - y \geq 2 \cdot x$$
$$z + y \leq x \quad \wedge \quad x \leq 5 - 2 \cdot y \quad \wedge \quad 2 \cdot z - 0.5 \cdot y \geq x$$
$$z + y \leq x \quad \wedge \quad x \leq 5 - 2 \cdot y \quad \wedge \quad x \leq 2 \cdot z - 0.5 \cdot y$$

eliminate $x$ by adding $A \leq B$ for all $A \leq x$ and $x \leq B$

# Fourier-Motzkin Elimination by Example (1/2)

$$z \leq x - y \quad \wedge \quad x + 2 \cdot y \leq 5 \quad \wedge \quad 4 \cdot z - 2 \cdot x \geq y$$

pick pivot variable, e.g. $x$, and isolate it on one side

$$z + y \leq x \quad \wedge \quad x \leq 5 - 2 \cdot y \quad \wedge \quad 4 \cdot z - y \geq 2 \cdot x$$
$$z + y \leq x \quad \wedge \quad x \leq 5 - 2 \cdot y \quad \wedge \quad 2 \cdot z - 0.5 \cdot y \geq x$$
$$z + y \leq x \quad \wedge \quad x \leq 5 - 2 \cdot y \quad \wedge \quad x \leq 2 \cdot z - 0.5 \cdot y$$

eliminate $x$ by adding $A \leq B$ for all $A \leq x$ and $x \leq B$

$$z + y \leq 5 - 2 \cdot y \quad \wedge \quad z + y \leq 2 \cdot z - 0.5 \cdot y$$
$$z \leq 5 - 3 \cdot y \quad \wedge \quad z + y \leq 2 \cdot z - 0.5 \cdot y$$
$$z \leq 5 - 3 \cdot y \quad \wedge \quad 1.5 \cdot y \leq z$$

# Fourier-Motzkin Elimination by Example (1/2)

$$z \leq x - y \quad \wedge \quad x + 2 \cdot y \leq 5 \quad \wedge \quad 4 \cdot z - 2 \cdot x \geq y$$

pick pivot variable, e.g. $x$, and isolate it on one side

$$z + y \leq x \quad \wedge \quad x \leq 5 - 2 \cdot y \quad \wedge \quad 4 \cdot z - y \geq 2 \cdot x$$
$$z + y \leq x \quad \wedge \quad x \leq 5 - 2 \cdot y \quad \wedge \quad 2 \cdot z - 0.5 \cdot y \geq x$$
$$z + y \leq x \quad \wedge \quad x \leq 5 - 2 \cdot y \quad \wedge \quad x \leq 2 \cdot z - 0.5 \cdot y$$

eliminate $x$ by adding $A \leq B$ for all $A \leq x$ and $x \leq B$

$$z + y \leq 5 - 2 \cdot y \quad \wedge \quad z + y \leq 2 \cdot z - 0.5 \cdot y$$
$$z \leq 5 - 3 \cdot y \quad \wedge \quad z + y \leq 2 \cdot z - 0.5 \cdot y$$
$$z \leq 5 - 3 \cdot y \quad \wedge \quad 1.5 \cdot y \leq z$$

repeat with new pivot variable, e.g. $z$

# Fourier-Motzkin Elimination by Example (1/2)

$$z \leq x - y \quad \wedge \quad x + 2 \cdot y \leq 5 \quad \wedge \quad 4 \cdot z - 2 \cdot x \geq y$$

pick pivot variable, e.g. $x$, and isolate it on one side

$$\begin{aligned} z + y \leq x \quad &\wedge \quad x \leq 5 - 2 \cdot y \quad \wedge \quad 4 \cdot z - y \geq 2 \cdot x \\ z + y \leq x \quad &\wedge \quad x \leq 5 - 2 \cdot y \quad \wedge \quad 2 \cdot z - 0.5 \cdot y \geq x \\ z + y \leq x \quad &\wedge \quad x \leq 5 - 2 \cdot y \quad \wedge \quad x \leq 2 \cdot z - 0.5 \cdot y \end{aligned}$$

eliminate $x$ by adding $A \leq B$ for all $A \leq x$ and $x \leq B$

$$\begin{aligned} z + y \leq 5 - 2 \cdot y \quad &\wedge \quad z + y \leq 2 \cdot z - 0.5 \cdot y \\ z \leq 5 - 3 \cdot y \quad &\wedge \quad z + y \leq 2 \cdot z - 0.5 \cdot y \\ z \leq 5 - 3 \cdot y \quad &\wedge \quad 1.5 \cdot y \leq z \end{aligned}$$

repeat with new pivot variable, e.g. $z$

$$\begin{aligned} 1.5 \cdot y &\leq 5 - 3 \cdot y \\ y &\leq 10/9 \end{aligned}$$

# Fourier-Motzkin by Example (2/2)

$$y \quad \leq \quad 10/9 \tag{3}$$

$(3)$ has (as one) solution $y = 0 \in (-\infty, 10/9]$ or
$y = 1 \in (-\infty, 10/9]$

# Fourier-Motzkin by Example (2/2)

$$y \leq 10/9 \tag{3}$$

(3) has (as one) solution $y = 0 \in (-\infty, 10/9]$ or $y = 1 \in (-\infty, 10/9]$

$$z \leq 5 - 3 \cdot y \quad \wedge \quad 1.5 \cdot y \leq z \tag{2}$$

(2) then allows $z = 0 \in [0, 5]$ $z = 2 \in [1.5, 2]$

# Fourier-Motzkin by Example (2/2)

$$y \leq 10/9 \tag{3}$$

(3) has (as one) solution $y = 0 \in (-\infty, 10/9]$ or $y = 1 \in (-\infty, 10/9]$

$$z \leq 5 - 3 \cdot y \quad \wedge \quad 1.5 \cdot y \leq z \tag{2}$$

(2) then allows $z = 0 \in [0, 5]$ $z = 2 \in [1.5, 2]$

$$z + y \leq x \quad \wedge \quad x \leq 5 - 2 \cdot y \quad \wedge \quad x \leq 2 \cdot z - 0.5 \cdot y \tag{1}$$

(1) then forces $x = 0$ forces $x = 3$

# Fourier-Motzkin by Example (2/2)

$$y \leq 10/9 \tag{3}$$

(3) has (as one) solution $y = 0 \in (-\infty, 10/9]$ or $y = 1 \in (-\infty, 10/9]$

$$z \leq 5 - 3 \cdot y \quad \wedge \quad 1.5 \cdot y \leq z \tag{2}$$

(2) then allows $z = 0 \in [0, 5]$ $z = 2 \in [1.5, 2]$

$$z + y \leq x \quad \wedge \quad x \leq 5 - 2 \cdot y \quad \wedge \quad x \leq 2 \cdot z - 0.5 \cdot y \tag{1}$$

(1) then forces $x = 0$ forces $x = 3$

**SATISFIABLE**

# Arrays

- functions “read” and “write”: $\mathsf{read}(a, i)$, $\mathsf{write}(a, i, v)$
- axioms

  array congruence

  $$\forall a, i, j : i = j \rightarrow \mathsf{read}(a, i) = \mathsf{read}(a, j)$$

  read over write 1

  $$\forall a, v, i, j : i = j \rightarrow \mathsf{read}(\mathsf{write}(a, i, v), j) = v$$

  read over write 2

  $$\forall a, v, i, j : i \neq j \rightarrow \mathsf{read}(\mathsf{write}(a, i, v), j) = \mathsf{read}(a, j)$$

- used to model memory (HW and SW)

# Array to EUF Example

- eagerly reduce arrays to uninterpreted functions:

  $\mathsf{read}(\mathsf{write}(a, i, v), j)$ replaced by $(i = j \ ? \ v : \mathsf{read}(a, j))$

# Array to EUF Example

- eagerly reduce arrays to uninterpreted functions:

  $\mathsf{read}(\mathsf{write}(a, i, v), j)$ replaced by $(i = j \ ? \ v : \mathsf{read}(a, j))$

- Example:

$$i \neq j \ \wedge \ u = \mathsf{read}(\mathsf{write}(a, i, v), j) \ \wedge \ v = \mathsf{read}(a, j) \ \wedge \ u \neq v$$

# Array to EUF Example

- eagerly reduce arrays to uninterpreted functions:

  $\mathsf{read}(\mathsf{write}(a, i, v), j)$ replaced by $(i = j \ ? \ v : \mathsf{read}(a, j))$

- Example:

$$i \neq j \ \wedge \ u = \mathsf{read}(\mathsf{write}(a, i, v), j) \ \wedge \ v = \mathsf{read}(a, j) \ \wedge \ u \neq v$$

$$i \neq j \ \wedge \ u = (i = j \ ? \ v : \mathsf{read}(a, j)) \ \wedge \ v = \mathsf{read}(a, j) \ \wedge \ u \neq v$$

# Array to EUF Example

- eagerly reduce arrays to uninterpreted functions:

  $\mathsf{read}(\mathsf{write}(a, i, v), j)$ replaced by $(i = j \ ? \ v : \mathsf{read}(a, j))$

- Example:

$$i \neq j \ \wedge \ u = \mathsf{read}(\mathsf{write}(a, i, v), j) \ \wedge \ v = \mathsf{read}(a, j) \ \wedge \ u \neq v$$

$$i \neq j \ \wedge \ u = (i = j \ ? \ v : \mathsf{read}(a, j)) \ \wedge \ v = \mathsf{read}(a, j) \ \wedge \ u \neq v$$

$$i \neq j \ \wedge \ u = \mathsf{read}(a, j) \ \wedge \ v = \mathsf{read}(a, j) \ \wedge \ u \neq v$$

# Array to EUF Example

- eagerly reduce arrays to uninterpreted functions:

  $\mathsf{read}(\mathsf{write}(a, i, v), j)$ replaced by $(i = j \ ? \ v : \mathsf{read}(a, j))$

- Example:

$$i \neq j \ \wedge \ u = \mathsf{read}(\mathsf{write}(a, i, v), j) \ \wedge \ v = \mathsf{read}(a, j) \ \wedge \ u \neq v$$

$$i \neq j \ \wedge \ u = (i = j \ ? \ v : \mathsf{read}(a, j)) \ \wedge \ v = \mathsf{read}(a, j) \ \wedge \ u \neq v$$

$$i \neq j \ \wedge \ u = \mathsf{read}(a, j) \ \wedge \ v = \mathsf{read}(a, j) \ \wedge \ u \neq v$$

$$i \neq j \ \wedge \ u = \mathsf{read}(a, j) = \mathsf{read}(a, j) = v \ \wedge \ u \neq v$$

# Array to EUF Example

- eagerly reduce arrays to uninterpreted functions:

  $\mathsf{read}(\mathsf{write}(a, i, v), j)$ replaced by $(i = j \ ? \ v : \mathsf{read}(a, j))$

- Example:

$$i \neq j \ \wedge \ u = \mathsf{read}(\mathsf{write}(a, i, v), j) \ \wedge \ v = \mathsf{read}(a, j) \ \wedge \ u \neq v$$

$$i \neq j \ \wedge \ u = (i = j \ ? \ v : \mathsf{read}(a, j)) \ \wedge \ v = \mathsf{read}(a, j) \ \wedge \ u \neq v$$

$$i \neq j \ \wedge \ u = \mathsf{read}(a, j) \ \wedge \ v = \mathsf{read}(a, j) \ \wedge \ u \neq v$$

$$i \neq j \ \wedge \ u = \mathsf{read}(a, j) = \mathsf{read}(a, j) = v \ \wedge \ u \neq v$$

**UNSATISFIABLE**

# Theory of Bit-Vectors

- allows “bit-precise” reasoning
  - captures semantics of low-level languages like assembler, C, C++, . . .
  - Java / C# also use two-complement representations for `int`
  - modelling of hardware / circuits on the word-level (RTL)
  - important for security applications and precise test case generation
- many operations
  - logical operations, bit-wise operations (and, or)
  - equalities, inequalities, disequalities
  - shift, concatenation, slicing
  - addition, multiplication, division, modulo, . . .
- main approach is reduction to SAT through bit-blasting
  - reduction of bit-vector operations similar to circuit synthesis
  - Ackermann’s Reduction only needs equality and disequality

# SMT-LIB

- international initiative
  - rigorous descriptions of background theories
  - common input/output format
  - library of benchmarks
- SMT-Lib 2.5 provides languages for
  - terms and formulas
  - background theories
  - logics
  - commands

# Theories in SMT-LIB

- ArraysEx Functional arrays with extensionality
- FixedSizeBitVectors Bit vectors with arbitrary size
- Core Core theory, defining the basic Boolean operators
- FloatingPoint Floating point numbers
- Ints Integer numbers
- Reals Real numbers
- Reals_Ints Real and integer numbers

# Exampe: Core Theory

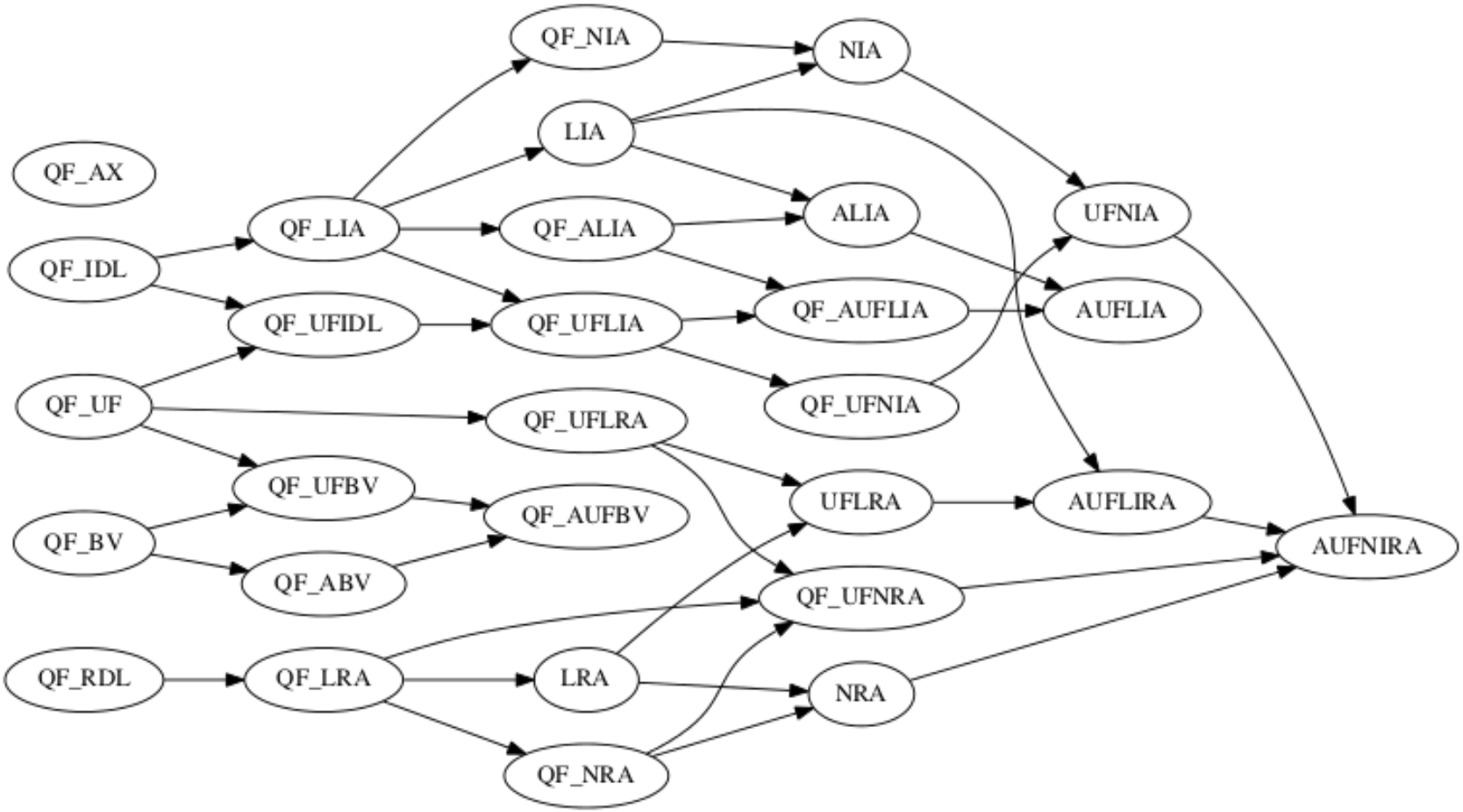
```
(theory Core

 :sorts ((Bool 0))

 :funs ((true Bool)
        (false Bool)
        (not Bool Bool)
        (=> Bool Bool Bool :right-assoc)
        (and Bool Bool Bool :left-assoc)
        (or Bool Bool Bool :left-assoc)
        (xor Bool Bool Bool :left-assoc)
        (par (A) (= A A Bool :chainable))
        (par (A) (distinct A A Bool :pairwise))
        (par (A) (ite Bool A A A))
       )

 :values
 "The set of values for the sort Bool is {true, false}."
)
```

# Logics in SMT-LIB




from http://smtlib.cs.uiowa.edu/logics.shtml

# Eager SMT Solving

- **prerequisite**: decidable ground satisfiability problem

# Eager SMT Solving

- **prerequisite**: decidable ground satisfiability problem
- **basic idea**:
  1. translate SMT problem into equisatisfiable propositional formula
  2. use off-the-shelf SAT solver

# Eager SMT Solving

- **prerequisite**: decidable ground satisfiability problem
- **basic idea**:
  1. translate SMT problem into equisatisfiable propositional formula
  2. use off-the-shelf SAT solver
- **advantage**: SAT solver is used out of the box

# Eager SMT Solving

- **prerequisite**: decidable ground satisfiability problem
- **basic idea**:
  1. translate SMT problem into equisatisfiable propositional formula
  2. use off-the-shelf SAT solver
- **advantage**: SAT solver is used out of the box
- **problem**: blow-up

# Ackermann's Reduction

**Given**: EUF formula

1. **flatten terms by introducing new variables**
   - remove nested function applications
   - (dis)equalities have at least one variable on one side

# Ackermann's Reduction

**Given**: EUF formula

1. **flatten terms by introducing new variables**
   - remove nested function applications
   - (dis)equalities have at least one variable on one side
2. **replace function instances with new symbol**:
   - replace all function applications $f(u)$ by new variable $f^u$
   - replace all function applications $f(u, v)$ by new variable $f^{u,v}$

# Ackermann's Reduction

**Given**: EUF formula

1. **flatten terms by introducing new variables**
   - remove nested function applications
   - (dis)equalities have at least one variable on one side
2. **replace function instances with new symbol**:
   - replace all function applications $f(u)$ by new variable $f^u$
   - replace all function applications $f(u, v)$ by new variable $f^{u,v}$
3. **add constraints**:
   - if formula contains $f^u$ and $f^v$ add $u = v \rightarrow f^u = f^v$

# Ackermann's Reduction

**Given**: EUF formula

1. **flatten terms by introducing new variables**
   - remove nested function applications
   - (dis)equalities have at least one variable on one side
2. **replace function instances with new symbol**:
   - replace all function applications $f(u)$ by new variable $f^u$
   - replace all function applications $f(u, v)$ by new variable $f^{u,v}$
3. **add constraints**:
   - if formula contains $f^u$ and $f^v$ add $u = v \rightarrow f^u = f^v$
4. **solve formulas with (dis)equalities**
   - if formula after the first two steps contains $n$ variables then domains of size $n$ are enough
   - or bit-vectors of length $\lceil \log_2 n \rceil$ bits
   - eager encoding into SAT

# Example of Ackermann's Reduction

we start with an already flattened formula

$$x = f(y) \wedge y = f(x) \wedge x \neq y$$

# Example of Ackermann's Reduction

we start with an already flattened formula

$$x = f(y) \wedge y = f(x) \wedge x \neq y$$

after introduction of variable

$$x = f^y \wedge y = f^x \wedge x \neq y$$

# Example of Ackermann's Reduction

we start with an already flattened formula

$$x = f(y) \wedge y = f(x) \wedge x \neq y$$

after introduction of variable

$$x = f^y \wedge y = f^x \wedge x \neq y$$

after adding lemmas

$$x = f^y \wedge y = f^x \wedge x \neq y \wedge (x = y \rightarrow f^x = f^y)$$

# Example of Ackermann's Reduction

we start with an already flattened formula

$$x = f(y) \wedge y = f(x) \wedge x \neq y$$

after introduction of variable

$$x = f^y \wedge y = f^x \wedge x \neq y$$

after adding lemmas

$$x = f^y \wedge y = f^x \wedge x \neq y \wedge (x = y \rightarrow f^x = f^y)$$

resulting formula has $4$ variables thus needs bit-vectors of length 2

# Bit-Blasting Bit-Vector Equality

for each bit-vector equality $u = v$ with $u$ and $v$ bit-vectors of width $w$ introduce new propositional variables for individual bits

$$u_1, \ldots, u_w \qquad v_1, \ldots, v_w$$

# Bit-Blasting Bit-Vector Equality

for each bit-vector equality $u = v$ with $u$ and $v$ bit-vectors of width $w$ introduce new propositional variables for individual bits

$$u_1, \ldots, u_w \qquad v_1, \ldots, v_w$$

- replace $u = v$ by new propositional variable $e_{u=v}$
- disequality $u \neq v$ is replaced by $\neg e_{u=v}$

# Bit-Blasting Bit-Vector Equality

for each bit-vector equality $u = v$ with $u$ and $v$ bit-vectors of width $w$ introduce new propositional variables for individual bits

$$u_1, \ldots, u_w \qquad v_1, \ldots, v_w$$

- replace $u = v$ by new propositional variable $e_{u=v}$
- disequality $u \neq v$ is replaced by $\neg e_{u=v}$

add the propositional constraint

$$e_{u=v} \leftrightarrow \bigwedge_{i=1}^{w} (u_i \leftrightarrow v_i)$$

# Bit-Blasting Bit-Vector Equality

for each bit-vector equality $u = v$ with $u$ and $v$ bit-vectors of width $w$ introduce new propositional variables for individual bits

$$u_1, \ldots, u_w \qquad v_1, \ldots, v_w$$

- replace $u = v$ by new propositional variable $e_{u=v}$
- disequality $u \neq v$ is replaced by $\neg e_{u=v}$

add the propositional constraint

$$e_{u=v} \leftrightarrow \bigwedge_{i=1}^{w} (u_i \leftrightarrow v_i)$$

resulting formula satisfiable iff original formula satisfiable

# Bit-Blasting Ackermann Example

$$x = f^y \wedge y = f^x \wedge x \neq y \wedge (x = y \rightarrow f^x = f^y)$$

# Bit-Blasting Ackermann Example

$$x = f^y \wedge y = f^x \wedge x \neq y \wedge (x = y \rightarrow f^x = f^y)$$

now replacing the bit-vector equalities and the disequality by new $e$ variables

$$e_{x=f^y} \wedge e_{y=f^x} \wedge \neg e_{x=y} \wedge (e_{x=y} \rightarrow e_{f^x=f^y})$$

# Bit-Blasting Ackermann Example

$$x = f^y \wedge y = f^x \wedge x \neq y \wedge (x = y \rightarrow f^x = f^y)$$

now replacing the bit-vector equalities and the disequality by new $e$ variables

$$e_{x=f^y} \wedge e_{y=f^x} \wedge \neg e_{x=y} \wedge (e_{x=y} \rightarrow e_{f^x=f^y})$$

and adding the equality constraints

$$\begin{aligned}
e_{x=f^y} &\leftrightarrow (x_1 \leftrightarrow f_1^y) \wedge (x_2 \leftrightarrow f_2^y) \\
e_{y=f^x} &\leftrightarrow (y_1 \leftrightarrow f_1^x) \wedge (y_2 \leftrightarrow f_2^x) \\
e_{x=y} &\leftrightarrow (x_1 \leftrightarrow y_1) \wedge (x_2 \leftrightarrow y_2) \\
e_{f^x=f^y} &\leftrightarrow (f_1^x \leftrightarrow f_1^y) \wedge (f_2^x \leftrightarrow f_2^y)
\end{aligned}$$

# Bit-Blasting Ackermann Example

$$x = f^y \wedge y = f^x \wedge x \neq y \wedge (x = y \rightarrow f^x = f^y)$$

now replacing the bit-vector equalities and the disequality by new $e$ variables

$$e_{x=f^y} \wedge e_{y=f^x} \wedge \neg e_{x=y} \wedge (e_{x=y} \rightarrow e_{f^x=f^y})$$

and adding the equality constraints

$$\begin{aligned}
e_{x=f^y} &\leftrightarrow (x_1 \leftrightarrow f_1^y) \wedge (x_2 \leftrightarrow f_2^y) \\
e_{y=f^x} &\leftrightarrow (y_1 \leftrightarrow f_1^x) \wedge (y_2 \leftrightarrow f_2^x) \\
e_{x=y} &\leftrightarrow (x_1 \leftrightarrow y_1) \wedge (x_2 \leftrightarrow y_2) \\
e_{f^x=f^y} &\leftrightarrow (f_1^x \leftrightarrow f_1^y) \wedge (f_2^x \leftrightarrow f_2^y)
\end{aligned}$$

gives an “equi-satisfiable” formula which can be checked by SAT solver

# Bit-Blasting Ackermann Example in Limboole Syntax

```
$ cat ackbitblasted.limboole
exfy & eyfx & !exy & (exy -> efxfy) &
(exfy <-> (x1 <-> fy1) & (x2 <-> fy2)) &
(eyfx <-> (y1 <-> fx1) & (y2 <-> fx2)) &
(exy <-> (x1 <-> y1) & (x2 <-> y2)) &
(efxfy <-> (fx1 <-> fy1) & (fx2 <-> fy2))
$ limboole ackbitblasted.limboole -s|grep -v SAT|sort
```

# Bit-Blasting Ackermann Example in Limboole Syntax

```
$ cat ackbitblasted.limboole
exfy & eyfx & !exy & (exy -> efxfy) &
(exfy <-> (x1 <-> fy1) & (x2 <-> fy2)) &
(eyfx <-> (y1 <-> fx1) & (y2 <-> fx2)) &
(exy <-> (x1 <-> y1) & (x2 <-> y2)) &
(efxfy <-> (fx1 <-> fy1) & (fx2 <-> fy2))
$ limboole ackbitblasted.limboole -s|grep -v SAT|sort
efxfy = 0
exfy = 1
exy = 0
eyfx = 1
fx1 = 0
fx2 = 1
fy1 = 1
fy2 = 1
x1 = 1
x2 = 1
y1 = 0
y2 = 1
```

# Lazy SMT Solving

- for deciding sets of $\mathcal{T}$-literals efficient decision procedures are available
- naive approach: generate DNF and check each cube
  - not so good
  - how can we do it better?
- basic idea:
  1. search for propositional models by solver
  2. theory information is used lazily for checking $\mathcal{T}$-consistency of propositional models
- very modular
- very flexible

# Propositional Skeleton

$$x \neq y \ \wedge \ (2 * x \leq z \quad \vee \quad \neg (x - y \geq z \ \wedge \ z \leq y))$$

eliminate $\neq$ by disjunction

$$(\underbrace{x < y}_{a} \vee \underbrace{x > y}_{b}) \ \wedge \ (\underbrace{2 * x \leq z}_{c} \vee \neg(\underbrace{x - y \geq z}_{d} \wedge \underbrace{z \leq y}_{e}))$$

**propositional skeleton**

$(a \vee b) \wedge (c \vee \neg(d \wedge e))$ with $\alpha(x < y) = a, \quad \alpha(x > y) = b, \ldots$

# Propositional Skeleton

$$x \neq y \ \wedge \ (2 * x \leq z \quad \vee \quad \neg (x - y \geq z \ \wedge \ z \leq y))$$

eliminate $\neq$ by disjunction

$$(\underbrace{x < y}_{a} \vee \underbrace{x > y}_{b}) \ \wedge \ (\underbrace{2 * x \leq z}_{c} \vee \neg (\underbrace{x - y \geq z}_{d} \ \wedge \ \underbrace{z \leq y}_{e}))$$

**propositional skeleton**

$(a \vee b) \wedge (c \vee \neg (d \wedge e))$ with $\alpha(x < y) = a, \quad \alpha(x > y) = b, \ldots$

- SAT solver enumerates solutions, $a = b = c = d = e = 1$

# Propositional Skeleton

$$x \neq y \ \wedge \ (2 * x \leq z \quad \vee \quad \neg (x - y \geq z \ \wedge \ z \leq y))$$

eliminate $\neq$ by disjunction

$$(\underbrace{x < y}_{a} \vee \underbrace{x > y}_{b}) \ \wedge \ (\underbrace{2 * x \leq z}_{c} \vee \neg(\underbrace{x - y \geq z}_{d} \ \wedge \ \underbrace{z \leq y}_{e}))$$

**propositional skeleton**

$(a \vee b) \wedge (c \vee \neg(d \wedge e))$ with $\alpha(x < y) = a, \quad \alpha(x > y) = b, \ldots$

- SAT solver enumerates solutions, $a = b = c = d = e = 1$
- check solution literals with theory solver

# Propositional Skeleton

$$x \neq y \ \wedge \ (2 * x \leq z \quad \vee \quad \neg(x - y \geq z \ \wedge \ z \leq y))$$

eliminate $\neq$ by disjunction

$$(\underbrace{x < y}_{a} \vee \underbrace{x > y}_{b}) \ \wedge \ (\underbrace{2 * x \leq z}_{c} \vee \neg(\underbrace{x - y \geq z}_{d} \ \wedge \ \underbrace{z \leq y}_{e}))$$

**propositional skeleton**

$(a \vee b) \wedge (c \vee \neg(d \wedge e))$ with $\alpha(x < y) = a, \quad \alpha(x > y) = b, \ldots$

- SAT solver enumerates solutions, $a = b = c = d = e = 1$
- check solution literals with theory solver
- spurious solutions (disproven by theory solver) added as "lemma", e.g. $\neg(a \wedge b \wedge c \wedge c \wedge d \wedge e)$ or just $\neg(a \wedge b)$ after minimization

# Lemmas on Demand

```
Result LemmasOnDemand (EUF E)
  F = propSkeleton (E)
  Result R = UNSATISFIABLE
  Assignment A = {}

  while (true)
    (R, A) = solve (F)
    if (R == UNSATISFIABALE)
      return UNSATISFIABLE

    if (tSolve (E, A))
      return SATISFIABLE

     /* use minimial unsat. core of A for refining F */
     F = F + not (mus (A))
```

# Lemmas on Demand

```
Result LemmasOnDemand (EUF E)
  F = propSkeleton (E)
  Result R = UNSATISFIABLE
  Assignment A = {}

  while (true)
    (R, A) = solve (F)
    if (R == UNSATISFIABALE)
      return UNSATISFIABLE

    if (tSolve (E, A))
      return SATISFIABLE

     /* use minimial unsat. core of A for refining F */
     F = F + not (mus (A))
```

**Minimal Unsatisfiable Core (MUS)**

- given an unsatisfiable set of "constraints" $S$ (set of literals, or clauses)
- an MUS $M$ is a sub-set $M \subseteq S$ such that
  - $M$ is still unsatisfiable
  - any $M' \subset M$ (with $M' \neq M$) is satisfiable
- note that "being inconsistent" is a monotone property

# Lemmas on Demand: Example

| EUF formula | propositional clauses |
|---|---|
| $g(a) = b$ | $1$ |
| $b \neq c$ | $\bar{2}$ |
| $g(a) = c \vee f(g(a)) \neq f(b)$ | $3 \vee \bar{4}$ |
| $d = e$ | $5$ |

# Lemmas on Demand: Example

| EUF formula | propositional clauses |
|---|---|
| $g(a) = b$ | $1$ |
| $b \neq c$ | $\bar{2}$ |
| $g(a) = c \vee f(g(a)) \neq f(b)$ | $3 \vee \bar{4}$ |
| $d = e$ | $5$ |

# Lemmas on Demand: Example

| EUF formula | propositional clauses |
|---|---|
| $g(a) = b$ | $1$ |
| $b \neq c$ | $\bar{2}$ |
| $g(a) = c \vee f(g(a)) \neq f(b)$ | $3 \vee \bar{4}$ |
| $d = e$ | $5$ |

# Lemmas on Demand: Example

| EUF formula | propositional clauses |
|---|---|
| $g(a) = b$ | $1$ |
| $b \neq c$ | $\bar{2}$ |
| $g(a) = c \vee f(g(a)) \neq f(b)$ | $3 \vee \bar{4}$ |
| $d = e$ | $5$ |

# Lemmas on Demand: Example

| EUF formula | propositional clauses |
|---|---|
| $g(a) = b$ | $1$ |
| $b \neq c$ | $\bar{2}$ |
| $g(a) = c \vee f(g(a)) \neq f(b)$ | $3 \vee \bar{4}$ |
| $d = e$ | $5$ |

# Lemmas on Demand: Example

| EUF formula | propositional clauses |
|---|---|
| $g(a) = b$ | $1$ |
| $b \neq c$ | $\bar{2}$ |
| $g(a) = c \vee f(g(a)) \neq f(b)$ | $3 \vee \bar{4}$ |
| $d = e$ | $5$ |

# Lemmas on Demand: Example

| EUF formula | propositional clauses |
|---|---|
| $g(a) = b$ | $1$ |
| $b \neq c$ | $\bar{2}$ |
| $g(a) = c \vee f(g(a)) \neq f(b)$ | $3 \vee \bar{4}$ |
| $d = e$ | $5$ |
| | SAT |
| | Model: $1, \bar{2}, 3, 5$ |

# Lemmas on Demand: Example

| EUF formula | propositional clauses |
|---|---|
| $g(a) = b$ | $1$ |
| $b \neq c$ | $\bar{2}$ |
| $g(a) = c \vee f(g(a)) \neq f(b)$ | $3 \vee \bar{4}$ |
| $d = e$ | $5$ |
| | SAT |
| | Model: $1, \bar{2}, 3, 5$ |

# Lemmas on Demand: Example

| EUF formula | propositional clauses |
|---|---|
| $g(a) = b$ | $1$ |
| $b \neq c$ | $\bar{2}$ |
| $g(a) = c \vee f(g(a)) \neq f(b)$ | $3 \vee \bar{4}$ |
| $d = e$ | $5$ |
| UNSAT | SAT<br>Model: $1, \bar{2}, 3, 5$ |

# Lemmas on Demand: Example

| EUF formula | propositional clauses |
|---|---|
| $g(a) = b$ | $1$ |
| $b \neq c$ | $\bar{2}$ |
| $g(a) = c \vee f(g(a)) \neq f(b)$ | $3 \vee \bar{4}$ |
| $d = e$ | $5$ |
| | $\bar{1} \vee 2 \vee \bar{3} \vee \bar{5}$ |

# Lemmas on Demand: Example

| EUF formula | propositional clauses |
|---|---|
| $g(a) = b$ | $1$ |
| $b \neq c$ | $\bar{2}$ |
| $g(a) = c \vee f(g(a)) \neq f(b)$ | $3 \vee \bar{4}$ |
| $d = e$ | $5$ |
| | $\bar{1} \vee 2 \vee \bar{3} \vee \bar{5}$ |
| | SAT |
| | Model: $1, \bar{2}, \bar{4}, 5$ |

# Lemmas on Demand: Example

| EUF formula | propositional clauses |
|---|---|
| $g(a) = b$ | $1$ |
| $b \neq c$ | $\bar{2}$ |
| $g(a) = c \vee f(g(a)) \neq f(b)$ | $3 \vee \bar{4}$ |
| $d = e$ | $5$ |
| | $\bar{1} \vee 2 \vee \bar{3} \vee \bar{5}$ |
| | SAT |
| | Model: $1, \bar{2}, \bar{4}, 5$ |

# Lemmas on Demand: Example

| EUF formula | propositional clauses |
|---|---|
| $g(a) = b$ | $1$ |
| $b \neq c$ | $\bar{2}$ |
| $g(a) = c \vee f(g(a)) \neq f(b)$ | $3 \vee \bar{4}$ |
| $d = e$ | $5$ |
| | $\bar{1} \vee 2 \vee \bar{3} \vee \bar{5}$ |
| UNSAT | SAT |
| | Model: $1, \bar{2}, \bar{4}, 5$ |

# Lemmas on Demand: Example

| EUF formula | propositional clauses |
|---|---|
| $g(a) = b$ | $1$ |
| $b \neq c$ | $\bar{2}$ |
| $g(a) = c \vee f(g(a)) \neq f(b)$ | $3 \vee \bar{4}$ |
| $d = e$ | $5$ |
| | $\bar{1} \vee 2 \vee \bar{3} \vee \bar{5}$ |
| | $\bar{1} \vee 2 \vee 4 \vee \bar{5}$ |

# Lemmas on Demand: Example

| EUF formula | propositional clauses |
|---|---|
| $g(a) = b$ | $1$ |
| $b \neq c$ | $\bar{2}$ |
| $g(a) = c \vee f(g(a)) \neq f(b)$ | $3 \vee \bar{4}$ |
| $d = e$ | $5$ |
| | $\bar{1} \vee 2 \vee \bar{3} \vee \bar{5}$ |
| | $\bar{1} \vee 2 \vee 4 \vee \bar{5}$ |
| | UNSAT |

**UNSATISFIABLE**

# Integration of SAT and Theory Solver

- the lemmas-on-demand approach is extremly lazy
  - theory consistency is only checked if the SAT solver found a model
  - integration of SAT solver and theory solver is not tight

# Integration of SAT and Theory Solver

- the lemmas-on-demand approach is extremly lazy
  - theory consistency is only checked if the SAT solver found a model
  - integration of SAT solver and theory solver is not tight

**How can we do better?**

# Integration of SAT and Theory Solver

- the lemmas-on-demand approach is extremly lazy
  - theory consistency is only checked if the SAT solver found a model
  - integration of SAT solver and theory solver is not tight

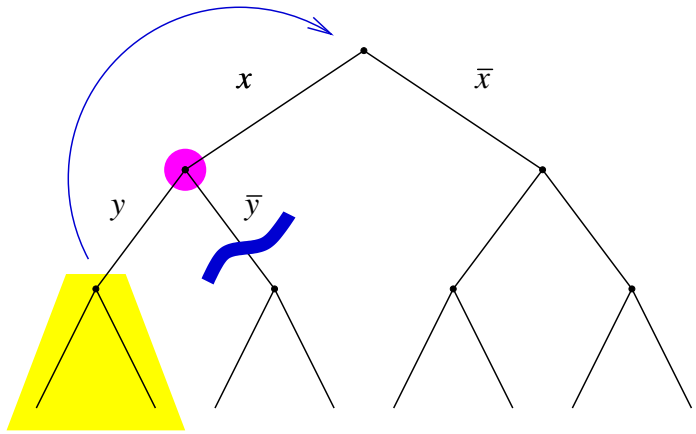
**How can we do better?**

Let's reconsider CDCL-based SAT solving.

# CDCL for SAT

```
Result cdcl (CNF F)
  Result R = UNDEF;
  Assignment A = {};
  while (true)
    /* Simplify under A. */
    (R,A) = bcp(F, A);
    if (R == UNDET)
      /* Decision making. */
      A = assign_dec_var(F,A);
    else
      if (R == SAT)
        return SATISFIABLE;
      else
        /* Backtracking. */
        dl = analyze(R,A);
        if (dl == 0)
          return UNSATISFIABLE;
        A = backtrack(dl);
```

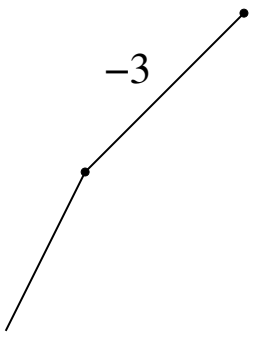
# Backjumping




- If $y$ has never been used to derive a conflict, then skip $\overline{y}$ case.
- Immediately jump back to the $\overline{x}$ case – assuming $x$ was used.

# Backjumping Example

Split on $-3$ first (bad decision).




~~(−3 1)~~

~~(−3 2)~~

(−1 −2 ~~3~~)

(−1 −2)

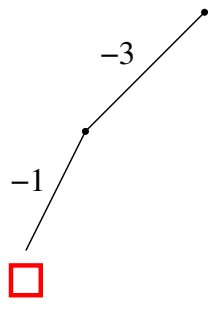
(−1 2)

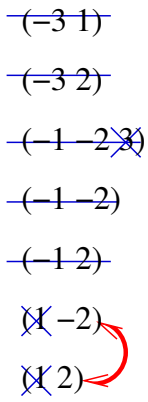
(1 −2)

(1 2)

# Backjumping Example

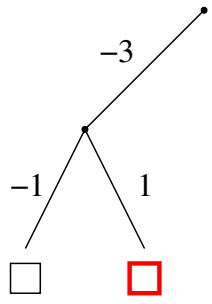
Split on $-1$ and get first conflict.

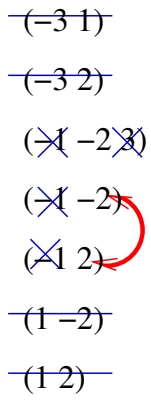


# Backjumping Example

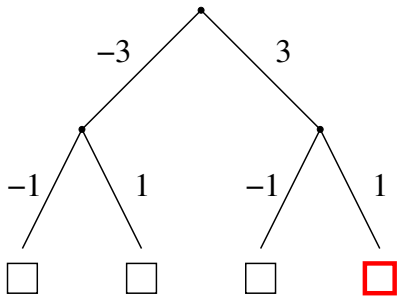
Regularly backtrack and assign $1$ to get second conflict.



# Backjumping Example

Backtrack to root, assign $3$ and derive same conflicts.




~~(−3 1)~~

~~(−3 2)~~

~~(−1 −2 3)~~

~~(−1 −2)~~

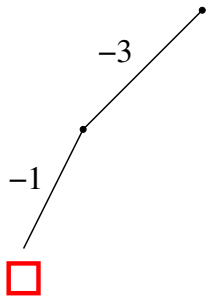
~~(−1 2)~~

~~(1 −2)~~

~~(1 2)~~

# Backjumping Example

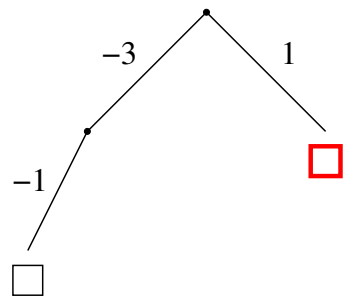
Assignment $-3$ does not contribute to conflict.




~~(−3 1)~~
~~(−3 2)~~
~~(−1 −2 3)~~
~~(−1 −2)~~
~~(−1 2)~~
~~(1~~ −2)
~~(1~~ 2)

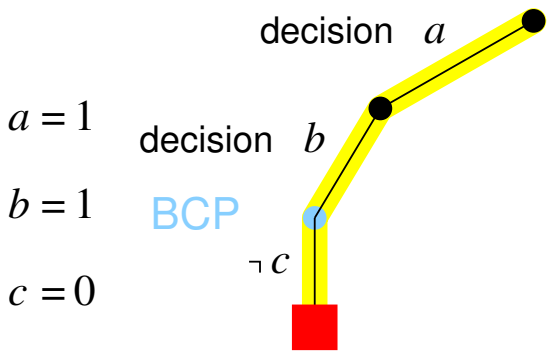
# Backjumping Example

So just backjump to root before assigning $1$.




~~(−3 1)~~
(−3 2)
~~(1 −2 3)~~
~~(1 −2)~~
~~(1 2)~~
~~(1 −2)~~
~~(1 2)~~

# Conflict Driven Clause Learning (CDCL)




clauses

$\neg a \vee \neg b \vee \neg c$
$\neg a \vee \neg b \vee c$
$\neg a \vee b \vee \neg c$
$\neg a \vee b \vee c$
$a \vee \neg b \vee \neg c$
$a \vee \neg b \vee c$
$a \vee b \vee \neg c$
$a \vee b \vee c$

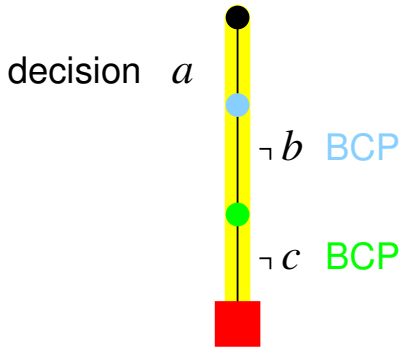
learn $\neg a \vee \neg b$

# Conflict Driven Clause Learning (CDCL)

$a = 1$

$b = 0$

$c = 0$




clauses

$\neg a \vee \neg b \vee \neg c$

$\neg a \vee \neg b \vee c$

$\neg a \vee b \vee \neg c$

$\neg a \vee b \vee c$

$a \vee \neg b \vee \neg c$

$a \vee \neg b \vee c$

$a \vee b \vee \neg c$

$a \vee b \vee c$

$\neg a \vee \neg b$

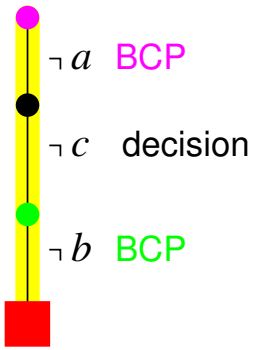
learn $\neg a$

# Conflict Driven Clause Learning (CDCL)

$a = 1$

$b = 0$

$c = 0$




clauses

$\neg a \vee \neg b \vee \neg c$

$\neg a \vee \neg b \vee c$

$\neg a \vee b \vee \neg c$

$\neg a \vee b \vee c$

$a \vee \neg b \vee \neg c$

$a \vee \neg b \vee c$

$a \vee b \vee \neg c$

$a \vee b \vee c$

$\neg a \vee \neg b$

$\neg a$

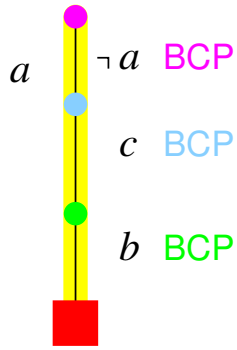
learn $c$

# Conflict Driven Clause Learning (CDCL)

$a = 1$

$b = 0$

$c = 0$




clauses

$\neg a \vee \neg b \vee \neg c$

$\neg a \vee \neg b \vee c$

$\neg a \vee b \vee \neg c$

$\neg a \vee b \vee c$

$a \vee \neg b \vee \neg c$

$a \vee \neg b \vee c$

$a \vee b \vee \neg c$

$a \vee b \vee c$

$\neg a \vee \neg b$

$\neg a$

$c$

learn $\bot$

empty clause

# DPLL(T)

- tight integration between CDCL-based SAT solver and theory solvers
  - $\mathcal{T}$-backjumping
  - $\mathcal{T}$-learning
  - $\mathcal{T}$-propagation
  - early pruning
  - filtering
  - ...
- also online approach of lazy SMT

# $\mathcal{T}$-Learning, $\mathcal{T}$-Backjumping

- If theory solver returns UNSAT, it returns an MUS as before (an inconsistent subset of theory literals).
- idea: use information from inconsistent subsect for CDCL conflict analysis.
- hope: wider backjumps, learn smaller clauses

# Example: $\mathcal{T}$-Learning, $\mathcal{T}$-Backjumping

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

# Example: $\mathcal{T}$-Learning, $\mathcal{T}$-Backjumping

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $\neg A_4$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

# Example: $\mathcal{T}$-Learning, $\mathcal{T}$-Backjumping

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $\neg A_4$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_6$ |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

# Example: $\mathcal{T}$-Learning, $\mathcal{T}$-Backjumping

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $\neg A_4$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_6$ |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | $A_5$ |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

# Example: $\mathcal{T}$-Learning, $\mathcal{T}$-Backjumping

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $\neg A_4$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_6$ |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | $A_5$ |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | $P_2$ |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

# Example: $\mathcal{T}$-Learning, $\mathcal{T}$-Backjumping

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $\neg A_4$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_6$ |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | $A_5$ |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | $P_2, A_2$ |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

# Example: $\mathcal{T}$-Learning, $\mathcal{T}$-Backjumping

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $\neg A_4$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_6$ |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | $A_5$ |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | $P_2, A_2, A_3$ |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

# Example: $\mathcal{T}$-Learning, $\mathcal{T}$-Backjumping

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $\neg A_4$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_6$ |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | $A_5$ |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | $P_2, A_2, A_3, P_1$ |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

# Example: $\mathcal{T}$-Learning, $\mathcal{T}$-Backjumping

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $\neg A_4$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_6$ |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | $A_5$ |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | $P_2$, $A_2$, $A_3$, $P_1$ |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | SATISFIABLE |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

# Example: $\mathcal{T}$-Learning, $\mathcal{T}$-Backjumping

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $\neg A_4$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_6$ |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | $A_5$ |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | $P_2, A_2, A_3, P_1$ |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | SATISFIABLE |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

- theory literals to be checked: $\neg A_4, A_6, A_5, A_2, A_3$

# Example: $\mathcal{T}$-Learning, $\mathcal{T}$-Backjumping

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $\neg A_4$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_6$ |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | $A_5$ |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | $P_2, A_2, A_3, P_1$ |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | SATISFIABLE |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

- theory literals to be checked: $\neg A_4, A_6, A_5, A_2, A_3$
- MUS: $\neg A_4, A_2, A_3$

$\Rightarrow$ lemma: $A_4 \vee \neg A_2 \neg A_3$

# Example: $\mathcal{T}$-Learning, $\mathcal{T}$-Backjumping

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $\neg A_4$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_6$ |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | $A_5$ |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | $P_2, A_2, A_3, P_1$ |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | SATISFIABLE |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |
| $c_8$ | $x = z \vee \neg P_2$ | $A_4 \vee \neg P_2$ | |

- theory literals to be checked: $\neg A_4, A_6, A_5, A_2, A_3$
- MUS: $\neg A_4, A_2, A_3$

  $\Rightarrow$ lemma: $A_4 \vee \neg A_2 \neg A_3$
- learned clause: $A_4, \neg P_2$

# Example: $\mathcal{T}$-Learning, $\mathcal{T}$-Backjumping

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $\neg A_4$ , $\neg P_2$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | ... |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |
| $c_8$ | $x = z \vee \neg P_2$ | $A_4 \vee \neg P_2$ | |

- theory literals to be checked: $\neg A_4, A_6, A_5, A_2, A_3$
- MUS: $\neg A_4, A_2, A_3$

  $\Rightarrow$ lemma: $A_4 \vee \neg A_2 \neg A_3$
- learned clause: $A_4, \neg P_2$

$\Rightarrow$ backjump to $\neg A_4$

# Early Pruning

- idea: perform $\mathcal{T}$-reasoning durch search for SAT solver
  - call SMT solver for newly assigned propositional variable
  - call SMT solver after BCP
  - heuristics
- theory solver needs to support incremental reasoning
  - allow additions of literals without resetting the solver's state
- approximative reasoning is often adequate
  - tradeoff efficiency and effectiveness

# Early Pruning

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

- theory check: $A_2$

# Early Pruning

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $A_2$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

- theory check: $A_2$

# Early Pruning

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $A_2$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_3$ |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

- theory check: $A_2$, $A_3$

# Early Pruning

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $A_2$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_3$ |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | $\neg A_4$ |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

- theory check: $A_2$, $A_3$, $\neg A_4$

# Early Pruning

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $A_2$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_3$ |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | $\neg A_4$ |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

- theory check: $A_2$, $A_3$, $\neg A_4$
- **CONFLICT on theory level!**

# Propagation in DPLL (T)

- idea: deduce values of unassigned literals by theory reasoning
- in combination with early pruning
- propagate value of literal in SAT solver

# Propagation in DPLL (T)

- idea: deduce values of unassigned literals by theory reasoning
- in combination with early pruning
- propagate value of literal in SAT solver

- **two types of propagations**:
  1. BCP in the SAT solver
  2. theory propagation in the theory solver

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $A_2$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $A_2$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_3$ |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $A_2$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_3, A_4$ |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

- from $A_2$ and $A_3$ we can infer $A_4$

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $A_2$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_3$, $A_4$, $\neg P_1$ |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

- from $A_2$ and $A_3$ we can infer $A_4$
- propagate $A_4$ in the abstraction

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $A_2$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_3, A_4, \neg P_1, A_1$ |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

- from $A_2$ and $A_3$ we can infer $A_4$
- propagate $A_4$ in the abstraction

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $A_2$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_3, A_4, \neg P_1, A_1$ |
| $c_3$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | $A_6$ |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

**SATSIFIABLE !**

# Filtering Assignments

- idea: removal of unnecessary literals from current assignments
  1. consider original formula structure
  2. identify literals in clauses satisfied by other literals
  3. pure literals (only one polarity in original formula)
- problem: can reduce the effect of early pruning

# Example: Filtering

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | |
| $c_4$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

# Example: Filtering

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $A_1$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | |
| $c_4$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

# Example: Filtering

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $A_1$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_6$ |
| $c_4$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

# Example: Filtering

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $A_1$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_6$ |
| $c_4$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | $A_5$ |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

# Example: Filtering

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $A_1$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_6$ |
| $c_4$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | $A_5$ |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | $A_2$ |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

# Example: Filtering

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $A_1$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_6$ |
| $c_4$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | $A_5$ |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | $A_2$ |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | $\neg P_2$ |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

# Example: Filtering

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $A_1$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_6$ |
| $c_4$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | $A_5$ |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | $A_2$ |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | $\neg P_2$ |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | $\neg P_1$ |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

# Example: Filtering

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $A_1$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_6$ |
| $c_4$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | $A_5$ |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | $A_2$ |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | $\neg P_2$ |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | $\neg P_1, A_3$ |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

# Example: Filtering

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $A_1$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_6$ |
| $c_4$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | $A_5$ |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | $A_2$ |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | $\neg P_2$ |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | $\neg P_1, A_3$ |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

- theory literals to be checked: $A_1, A_2, A_3, A_5, A_6$

# Example: Filtering

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $A_1$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_6$ |
| $c_4$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | $A_5$ |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | $A_2$ |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | $\neg P_2$ |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | $\neg P_1, A_3$ |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

- theory literals to be checked: $A_1, A_2, A_3, A_5, A_6$

**UNSATISFIABLE !**

# Example: Filtering

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $A_1$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_6$ |
| $c_4$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | $A_5$ |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | $A_2$ |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | $\neg P_2$ |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | $\neg P_1, A_3$ |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

- theory literals to be checked: $A_1, A_2, A_3, A_5, A_6$
- check only theory literals which satisfy clauses alone: $A_1, A_3$

# Example: Filtering

| | EUF Formula | Abstraction | Assignment |
|---|---|---|---|
| $c_1$ | $r = g(x) \vee P_1$ | $A_1 \vee P_1$ | $A_1$ |
| $c_2$ | $\neg P_2 \vee x = y$ | $\neg P_2 \vee A_2$ | $A_6$ |
| $c_4$ | $y = z \vee \neg P_2$ | $A_3 \vee \neg P_2$ | $A_5$ |
| $c_4$ | $\neg(x = z) \vee \neg P_1$ | $\neg A_4 \vee \neg P_1$ | $A_2$ |
| $c_5$ | $P_1 \vee y = z$ | $P_1 \vee A_3$ | $\neg P_2$ |
| $c_6$ | $s = g(y) \vee \neg P_1$ | $A_5 \vee \neg P_1$ | $\neg P_1, A_3$ |
| $c_7$ | $P_1 \vee r \neq s \vee \neg P_2$ | $P_1 \vee A_6 \vee \neg P_2$ | |

- theory literals to be checked: $A_1, A_2, A_3, A_5, A_6$
- check only theory literals which satisfy clauses alone: $A_1, A_3$

**SATISFIABLE !**

# CDCL for SMT

```
Result cdcl (CNF F)
  Result R = UNDEF;
  Assignment A = {};
  while (true)
    /* Simplify under A. */
    (R,A) = T-bcp(F, A);
    if (R == UNDET)
      /* Decision making. */
      A = assign_dec_var(F,A);
    else
      if (R == SAT)
        if (T-consistent (F, A))
          return SATISFIABLE;
      /* Backtracking. */
      dl = T-analyze(R,A);
      if (dl == 0)
        return UNSATISFIABLE;
      A = backtrack(dl);
```

# Combining Theories

- what we have so far:

# Combining Theories

- **what we have so far:**
  - DPLL (EUF)

# Combining Theories

- **what we have so far:**
  - DPLL (EUF)
  - DPLL (LRA)

# Combining Theories

- **what we have so far:**
  - DPLL (EUF)
  - DPLL (LRA)

# Combining Theories

- **what we have so far:**
  - DPLL (EUF)
  - DPLL (LRA)
  - ...

# Combining Theories

- **what we have so far:**
  - DPLL (EUF)
  - DPLL (LRA)
  - ...
- what we want: **DPLL (EUF + LRA + ... )**

# Combining Theories

- **what we have so far:**
  - DPLL (EUF)
  - DPLL (LRA)
  - ...
- what we want: **DPLL (EUF + LRA + ... )**
- **Question**: Can theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$ with decidable satisfiability problem be combined such that the combined satisfiability problem is decidable?

# Combining Theories

- **what we have so far:**
  - DPLL (EUF)
  - DPLL (LRA)
  - ...
- what we want: **DPLL (EUF + LRA + ... )**
- **Question**: Can theories $\mathcal{T}_1, \ldots, \mathcal{T}_n$ with decidable satisfiability problem be combined such that the combined satisfiability problem is decidable?
- **Answer**: Yes (if theories have certain properties).

# Combining Theories: Preliminaries

Given two theories $\mathcal{T}_1$ and $\mathcal{T}_2$ with signatures $\Sigma_1$ and $\Sigma_2$, we require that

- the signatures are disjoint (except $=$)

# Combining Theories: Preliminaries

Given two theories $\mathcal{T}_1$ and $\mathcal{T}_2$ with signatures $\Sigma_1$ and $\Sigma_2$, we require that

- the signatures are disjoint (except $=$)
- the theories are stably-infinite

# Combining Theories: Preliminaries

Given two theories $\mathcal{T}_1$ and $\mathcal{T}_2$ with signatures $\Sigma_1$ and $\Sigma_2$, we require that

- the signatures are disjoint (except $=$)
- the theories are stably-infinite
  A theory is stably-infinite if every satisfiable quantifier free formula is satisfiable by an infinite $\mathcal{T}$-interpretation.

# Combining Theories: Preliminaries

Given two theories $\mathcal{T}_1$ and $\mathcal{T}_2$ with signatures $\Sigma_1$ and $\Sigma_2$, we require that

- the signatures are disjoint (except $=$)
- the theories are stably-infinite
  A theory is stably-infinite if every satisfiable quantifier free formula is satisfiable by an infinite $\mathcal{T}$-interpretation.
- the theories are convex

# Combining Theories: Preliminaries

Given two theories $\mathcal{T}_1$ and $\mathcal{T}_2$ with signatures $\Sigma_1$ and $\Sigma_2$, we require that

- the signatures are disjoint (except $=$)
- the theories are stably-infinite
  A theory is stably-infinite if every satisfiable quantifier free formula is satisfiable by an infinite $\mathcal{T}$-interpretation.
- the theories are convex
  A theory is convex iff for each entailed disjunction $C_1 \vee \ldots \vee C_n$ there exists at least one $i$ such that $C_i$ is entailed.

# Nelson-Open Approach

- Given theories $T_1$, $T_2$ with disjoint signatures $\Sigma_1, \Sigma_2$ and a set of constant symbols $C$ not in $\Sigma_1, \Sigma_2$
- Idea: Construct equisatisfiable formula $\phi_1 \wedge \phi_2$ with
  - $\phi_1$ has signature $\Sigma_1 \cup C$
  - $\phi_2$ has signature $\Sigma_2 \cup C$

  by purification
- Purification:
  1. Abstract alien subterms: replace alien subterm $t$ with fresh constant $c$ and add $c = t$ to the formula
  2. Separate: give the new equations to the corresponding theory solver

# Example (Purification)

- We want to solve the formula

$$f(f(x) - f(y)) = a$$
$$f(0) > a + 2$$
$$x = y$$

| Uninterpreted functions | Integer Arithmetic |
|---|---|
| $f(e_1) = a$ | $e_2 - e_3 = e_1$ |
| $f(x) = e_2$ | $e_4 = 0$ |
| $f(y) = e_3$ | $e_5 > a + 2$ |
| $f(e_4) = e_5$ | |
| $x = y$ | |

# Deterministic Nelson-Oppen

```
Result decideMixed (S)
  S' = purify (S)
  (S1, S2) = split (S')
  E = getInterfaceEqualities (S1, S2)

  while (true)
    if (decideT1 (S1) == UNSAT) return UNSAT
    if (decideT2 (S2) == UNSAT) return UNSAT

    S1' = deriveEqT2 (S2)
    S2' = deriveEqT1 (S1)

    if ((S1' == empty) && (S2' == empty))
      return SAT

    S1 += S1'
    S2 += S2'
```

# Example (Theory Propagation)

- We want to solve the formula

$$f(f(x) - f(y)) = a$$
$$f(0) > a + 2$$
$$x = y$$

| Uninterpreted functions | Integer Arithmetic |
|---|---|
| $f(e_1) = a$ | $e_2 - e_3 = e_1$ |
| $f(x) = e_2$ | $e_4 = 0$ |
| $f(y) = e_3$ | $e_5 > a + 2$ |
| $f(e_4) = e_5$ | |
| $x = y$ | |

# Example (Theory Propagation)

- We want to solve the formula

$$f(f(x) - f(y)) = a$$
$$f(0) > a + 2$$
$$x = y$$

| Uninterpreted functions | Integer Arithmetic |
|---|---|
| $f(e_1) = a$ | $e_2 - e_3 = e_1$ |
| $f(x) = e_2$ | $e_4 = 0$ |
| $f(y) = e_3$ | $e_5 > a + 2$ |
| $f(e_4) = e_5$ | $e_2 = e_3$ |
| $x = y$ | |

# Example (Theory Propagation)

- We want to solve the formula

$$f(f(x) - f(y)) = a$$
$$f(0) > a + 2$$
$$x = y$$

| Uninterpreted functions | Integer Arithmetic |
|---|---|
| $f(e_1) = a$ | $e_2 - e_3 = e_1$ |
| $f(x) = e_2$ | $e_4 = 0$ |
| $f(y) = e_3$ | $e_5 > a + 2$ |
| $f(e_4) = e_5$ | $e_2 = e_3$ |
| $x = y$ | |
| $e_1 = e_4$ | |

# Example (Theory Propagation)

- We want to solve the formula

$$f(f(x) - f(y)) = a$$
$$f(0) > a + 2$$
$$x = y$$

| Uninterpreted functions | Integer Arithmetic |
|---|---|
| $f(e_1) = a$ | $e_2 - e_3 = e_1$ |
| $f(x) = e_2$ | $e_4 = 0$ |
| $f(y) = e_3$ | $e_5 > a + 2$ |
| $f(e_4) = e_5$ | $e_2 = e_3$ |
| $x = y$ | $a = e_5$ |
| $e_1 = e_4$ | |

# Further Topics

- quantifiers
- rewritings, preprocessing
- parallel solving
- certificates
- SLS
- ...

# Applications

- interactive theorem provers (e.g., Isabelle, PVS)
- static analysis (e.g., Boogie, ESC/Java 2)
- automatic test generators (e.g., PEX)
- model checkers (e.g., BLAST, SLAM)

# Take-Home Messages

- **SMT is the sweetspot between SAT and FO**
  - expressiveness
  - efficiency
- **SMT is a family of logics**
  - modularity
  - user-friendliness
- **SMT technology is closely entangled with SAT technology**

# References


- `http://smtlib.cs.uiowa.edu/`
- Robert Nieuwenhuis, Albert Oliveras: Fast congruence closure and extensions. Inf. Comp. 205(4): 557-580, 2007
- Aaron R. Bradley, Zohar Manna: The calculus of computation. Springer 2007
- Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, Cesare Tinelli: Satisfiability Modulo Theories. Handbook of SAT, 2009
- Daniel Kroening, Ofer Strichman: Decision Procedures. Springer, 2008