Automated reasoning 2016

Lecture 1: SMT Solvers

Instructor: Ashutosh Gupta

TIFR, India

Compile date: 2016-02-16

Moving away from verification

We have learned some methods for verification.

It is clear that they depend on a logical solver.

Moving away from verification

We have learned some methods for verification.

It is clear that they depend on a logical solver.

Now we will move away from verification and learn the technology of the solvers

Where are we and where are we going?

We have seen in previous course

- DPLL($\mathcal{T}$) for solving quantifier free(QF) formulas in some theory $\mathcal{T}$
- theory of equality and function symbols(EUF)

Where are we and where are we going?

We have seen in previous course

- DPLL($\mathcal{T}$) for solving quantifier free(QF) formulas in some theory $\mathcal{T}$
- theory of equality and function symbols(EUF)

We will see in this lecture

- reminder of DPLL($\mathcal{T}$) and EUF
- efficient algorithms for implementing DPLL(EUF)
- Z3 implementation of EUF

Install tools

Let us install a few tools.

- linux
- emacs
- git
- python
- g++
- make
- gdb
- z3 source `https://github.com/Z3Prover/z3`
  - compile in debug and trace mode

Topic 1.1

SMT solver reminder

DPLL($\mathcal{T}$) - some notation

Let $\mathcal{T}$ be a FO-theory with signature $\mathbf{S}$.

DPLL($\mathcal{T}$) - some notation

Let $\mathcal{T}$ be a FO-theory with signature $\mathbf{S}$.

We assume input formulas are from $\mathcal{T}$, QF, and in CNF.

DPLL($\mathcal{T}$) - some notation

Let $\mathcal{T}$ be a FO-theory with signature **S**.

We assume input formulas are from $\mathcal{T}$, QF, and in CNF.

Definition 1.1

For a QF $\mathcal{T}$ formula F, let atoms(F) denote the set of atoms appearing in F.

DPLL($\mathcal{T}$) - some notation

Let $\mathcal{T}$ be a FO-theory with signature $\mathbf{S}$.

We assume input formulas are from $\mathcal{T}$, QF, and in CNF.

Definition 1.1

For a QF $\mathcal{T}$ formula F, let $atoms(F)$ denote the set of atoms appearing in F.

Example 1.1

- $f(x) \approx g(h(x, y))$ *is a formula in QF_EUF.*
- $x > 0 \lor y + x \approx 3.5z$ *is a formula in QF_LRA.*

Boolean encoder

For a formula F, let boolean encoder e be a partial map from $atoms(F)$ to fresh boolean variables.

Boolean encoder

For a formula F, let boolean encoder e be a partial map from $atoms(F)$ to fresh boolean variables.

For a term t, let $e(t)$ denote the term obtained by replacing each atom a by $e(a)$ if $e(a)$ is defined.

Boolean encoder

For a formula F, let boolean encoder e be a partial map from $atoms(F)$ to fresh boolean variables.

For a term t, let $e(t)$ denote the term obtained by replacing each atom a by $e(a)$ if $e(a)$ is defined.

Example 1.2

Let $F = x < 2 \vee (y > 0 \vee x \geq 2)$
and $e = \{x_1 \mapsto x < 2, x_2 \mapsto y > 0\}$
$e(F) = x_1 \vee (x_2 \vee \neg x_1)$

Recall: CDCL(T)

Algorithm 1.1: CDCL($\mathcal{T}$)

Input: CNF F, boolean encoder e

$\textsc{AddClauses}(e(F))$; m := $\textsc{UnitPropagation}()$; $dl := 0$; $dstack := \lambda x.0$;

Recall: CDCL(T)

Algorithm 1.1: CDCL($\mathcal{T}$)

Input: CNF F, boolean encoder e

AddClauses($e(F)$); m := UnitPropagation(); dl := 0; $dstack := \lambda x.0$;

do

stands for decision level

Recall: CDCL(T)

Algorithm 1.1: CDCL($\mathcal{T}$)

Input: CNF F, boolean encoder e

AddClauses($e(F)$); m := UnitPropagation(); $dl := 0$; $dstack := \lambda x.0$;

stands for decision level

do

// backtracking

// Boolean decision

// Theory propagation

Recall: CDCL(T)

Algorithm 1.1: CDCL($\mathcal{T}$)

Input: CNF F, boolean encoder e

$\textsc{AddClauses}(e(F))$; m := $\textsc{UnitPropagation}()$; $dl := 0$; $dstack := \lambda x.0$;

stands for decision level

do

// backtracking

// Boolean decision

if *m is partial* **then**

$dstack(dl) := m.size()$;

$dl := dl + 1$; m := $\textsc{Decide}()$; m := $\textsc{UnitPropagation}()$;

// Theory propagation

Recall: CDCL(T)

Algorithm 1.1: CDCL($\mathcal{T}$)

Input: CNF F, boolean encoder e

$\textsc{AddClauses}(e(F))$; m := $\textsc{UnitPropagation}()$; $dl := 0$; $dstack := \lambda x.0$;

stands for decision level

do

// backtracking

// Boolean decision

if *m is partial* **then**

$dstack(dl) := m.size()$;

dstack records history for backtracking

$dl := dl + 1$; m := $\textsc{Decide}()$; m := $\textsc{UnitPropagation}()$;

// Theory propagation

Recall: CDCL(T)

Algorithm 1.1: CDCL($\mathcal{T}$)

Input: CNF F, boolean encoder e

$\textsc{AddClauses}(e(F))$; m $:=$ $\textsc{UnitPropagation}()$; $dl := 0$; $dstack := \lambda x.0$;

stands for decision level

do

// backtracking

while $\exists x\ \{x \mapsto 0, x \mapsto 1\} \subseteq m$ **do**

  if $dl = 0$ **then return** *unsat*;

  (C, dl) $:=$ $\textsc{AnalyzeConflict}(m)$; // clause learning

  $m.resize(dstack(dl))$; $\textsc{AddClauses}(\{C\})$; m $:=$ $\textsc{UnitPropagation}()$;

// Boolean decision

if *m is partial* **then**

  $dstack(dl) := m.size()$;

  $dl := dl + 1$; m $:=$ $\textsc{Decide}()$; m $:=$ $\textsc{UnitPropagation}()$;

dstack records history for backtracking

// Theory propagation

Recall: CDCL(T)

Algorithm 1.1: CDCL($\mathcal{T}$)

Input: CNF F, boolean encoder e

$\textsc{AddClauses}(e(F))$; m := $\textsc{UnitPropagation}()$; $dl := 0$; $dstack := \lambda x.0$;

stands for decision level

do

// backtracking

while $\exists x\ \{x \mapsto 0, x \mapsto 1\} \subseteq m$ **do**

  if $dl = 0$ **then return** *unsat*;

  (C, dl) := $\textsc{AnalyzeConflict}(m)$; // clause learning

  $m.resize(dstack(dl))$; $\textsc{AddClauses}(\{C\})$; m := $\textsc{UnitPropagation}()$;

// Boolean decision

if *m is partial* **then**

  $dstack(dl) := m.size()$;

  $dl := dl + 1$; m := $\textsc{Decide}()$; m := $\textsc{UnitPropagation}()$;

dstack records history for backtracking

// Theory propagation

if $\forall x\ \{x \mapsto 0, x \mapsto 1\} \not\subseteq m$ **then**

  (Cs, dl') := $\textsc{TheoryDeduction}(\bigwedge e^{-1}(m))$;

  if $dl' < dl$ **then** $\{dl = dl'; m.resize(dstack(dl)); \}$;

  $\textsc{AddClauses}(e(Cs))$; m := $\textsc{UnitPropagation}()$;

Recall: CDCL(T)

Algorithm 1.1: CDCL($\mathcal{T}$)

Input: CNF F, boolean encoder e

$\textsc{AddClauses}(e(F))$; m $:=$ $\textsc{UnitPropagation}()$; $dl := 0$; $dstack := \lambda x.0$;

do

// backtracking

while $\exists x\ \{x \mapsto 0, x \mapsto 1\} \subseteq m$ **do**

  if $dl = 0$ **then return** *unsat*;

  (C, dl) $:=$ $\textsc{AnalyzeConflict}(m)$; // clause learning

  $m.resize(dstack(dl))$; $\textsc{AddClauses}(\{C\})$; m $:=$ $\textsc{UnitPropagation}()$;

// Boolean decision

if *m is partial* **then**

  $dstack(dl) := m.size()$;

  $dl := dl + 1$; m $:=$ $\textsc{Decide}()$; m $:=$ $\textsc{UnitPropagation}()$;

// Theory propagation

if $\forall x\ \{x \mapsto 0, x \mapsto 1\} \not\subseteq m$ **then**

  (Cs, dl') $:=$ $\textsc{TheoryDeduction}(\bigwedge e^{-1}(m))$;

  if $dl' < dl$ **then** $\{dl = dl'; m.resize(dstack(dl)); \}$;

  $\textsc{AddClauses}(e(Cs))$; m $:=$ $\textsc{UnitPropagation}()$;

stands for decision level

dstack records history for backtracking

returns a clause set and a decision level

Recall: CDCL(T)

Algorithm 1.1: CDCL($\mathcal{T}$)

Input: CNF F, boolean encoder e

$\textsc{AddClauses}(e(F))$; m $:=$ $\textsc{UnitPropagation}()$; $dl := 0$; $dstack := \lambda x.0$;

stands for decision level

do

// backtracking

while $\exists x\ \{x \mapsto 0, x \mapsto 1\} \subseteq m$ **do**

  if $dl = 0$ **then return** *unsat*;

  (C, dl) $:=$ $\textsc{AnalyzeConflict}(m)$; // `clause learning`

  $m.resize(dstack(dl))$; $\textsc{AddClauses}(\{C\})$; m $:=$ $\textsc{UnitPropagation}()$;

// Boolean decision

if *m is partial* **then**

dstack records history for backtracking

  $dstack(dl) := m.size()$;

  $dl := dl + 1$; m $:=$ $\textsc{Decide}()$; m $:=$ $\textsc{UnitPropagation}()$;

// Theory propagation

if $\forall x\ \{x \mapsto 0, x \mapsto 1\} \not\subseteq m$ **then**

returns a clause set and a decision level

  (Cs, dl') $:=$ $\textsc{TheoryDeduction}(\bigwedge e^{-1}(m))$;

  if $dl' < dl$ **then** $\{dl = dl'; m.resize(dstack(dl)); \}$;

  $\textsc{AddClauses}(e(Cs))$; m $:=$ $\textsc{UnitPropagation}()$;

while *m is partial or* $\exists x\ \{x \mapsto 0, x \mapsto 1\} \subseteq m$;

return *sat*

Theory propagation

THEORYDEDUCTION looks at the atoms assigned so far and checks

- if they are mutually unsatisfiable
- if not, are there other literals from F that are implied by the current assignment

Theory propagation

THEORYDEDUCTION looks at the atoms assigned so far and checks

- if they are mutually unsatisfiable
- if not, are there other literals from F that are implied by the current assignment

Any implementation must comply with the following goals

- Correctness: boolean model is consistent with $\mathcal{T}$
- Termination: unsat partial models are never repeated

TheoryDeduction

TheoryDeduction solves conjunction of literals and returns a set of clauses and a decision level.

$$(Cs, dl') := \textsc{TheoryDeduction}(\bigwedge e^{-1}(m))$$

TheoryDeduction

TheoryDeduction solves conjunction of literals and returns a set of clauses and a decision level.

$$(Cs, dl') := \textsc{TheoryDeduction}(\bigwedge e^{-1}(m))$$

Cs may contain the clauses of the form

$$(\bigwedge L) \Rightarrow \ell$$

where $\ell \in lits(F) \cup \{\bot\}$ and $L \subseteq e^{-1}(m)$.

Note: The RHS need not be a single literal

Requirement form TheoryDeduction

The output of TheoryDeduction must satisfy the following conditions

Requirement form THEORYDEDUCTION

The output of THEORYDEDUCTION must satisfy the following conditions

- If $\bigwedge e^{-1}(m)$ is unsat in $\mathcal{T}$ then Cs must contain a clause with $\ell = \perp$.

Requirement form TheoryDeduction

The output of TheoryDeduction must satisfy the following conditions

- If $\bigwedge e^{-1}(m)$ is unsat in $\mathcal{T}$ then Cs must contain a clause with $\ell = \bot$.
- if $\bigwedge e^{-1}(m)$ is sat then $dl' = dl$.
  Otherwise, dl' is the decision level immediately after which the unsatisfiablity occurred (clearly stated shortly).

Example : DPLL($\mathcal{T}$)

Consider $F = (x = y \lor y = z) \land (y \neq z \lor z = u) \land (z = x)$

Example : DPLL($\mathcal{T}$)

Consider $F = (x = y \vee y = z) \wedge (y \neq z \vee z = u) \wedge (z = x)$
$e(F) = (x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge x_4$

Example : DPLL($\mathcal{T}$)

Consider $F = (x = y \lor y = z) \land (y \neq z \lor z = u) \land (z = x)$
$e(F) = (x_1 \lor x_2) \land (\neg x_2 \lor x_3) \land x_4$

After $\textsc{AddClauses}(e(F)); m := \textsc{UnitPropagation}()$
$m = \{x_4 \mapsto 1\}$

Example : DPLL($\mathcal{T}$)

Consider $F = (x = y \lor y = z) \land (y \neq z \lor z = u) \land (z = x)$
$e(F) = (x_1 \lor x_2) \land (\neg x_2 \lor x_3) \land x_4$

After $\textsc{AddClauses}(e(F)); m := \textsc{UnitPropagation}()$
$m = \{x_4 \mapsto 1\}$

After $m := \textsc{Decide}();$
$m = \{x_4 \mapsto 1, x_2 \mapsto 0\}$

Example : DPLL($\mathcal{T}$)

Consider $F = (x = y \lor y = z) \land (y \neq z \lor z = u) \land (z = x)$
$e(F) = (x_1 \lor x_2) \land (\neg x_2 \lor x_3) \land x_4$

After $\textsc{AddClauses}(e(F)); m := \textsc{UnitPropagation}()$
$m = \{x_4 \mapsto 1\}$

After $m := \textsc{Decide}();$
$m = \{x_4 \mapsto 1, x_2 \mapsto 0\}$

After $m := \textsc{UnitPropagation}()$
$m = \{x_4 \mapsto 1, x_2 \mapsto 0, x_1 \mapsto 1\}$

Example : DPLL($\mathcal{T}$)

Consider $F = (x = y \vee y = z) \wedge (y \neq z \vee z = u) \wedge (z = x)$
$e(F) = (x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge x_4$

After $\textsc{AddClauses}(e(F)); m := \textsc{UnitPropagation}()$
$m = \{x_4 \mapsto 1\}$

After $m := \textsc{Decide}();$
$m = \{x_4 \mapsto 1, x_2 \mapsto 0\}$

After $m := \textsc{UnitPropagation}()$
$m = \{x_4 \mapsto 1, x_2 \mapsto 0, x_1 \mapsto 1\}$

After $(Cs, dl') :=\ \textsc{TheoryDeduction}(x = y \wedge y \neq z \wedge z = x)$
$Cs = \{x \neq y \vee y = z \vee z \neq x\}, dl' = 1, e(Cs) = \{\neg x_1 \vee x_2 \vee \neg x_4\}$

Example : DPLL($\mathcal{T}$)

Consider $F = (x = y \vee y = z) \wedge (y \neq z \vee z = u) \wedge (z = x)$
$e(F) = (x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge x_4$

After $\textsc{AddClauses}(e(F)); m := \textsc{UnitPropagation}()$
$m = \{x_4 \mapsto 1\}$

After $m := \textsc{Decide}();$
$m = \{x_4 \mapsto 1, x_2 \mapsto 0\}$

After $m := \textsc{UnitPropagation}()$
$m = \{x_4 \mapsto 1, x_2 \mapsto 0, x_1 \mapsto 1\}$

After $(Cs, dl') := \textsc{TheoryDeduction}(x = y \wedge y \neq z \wedge z = x)$
$Cs = \{x \neq y \vee y = z \vee z \neq x\}, dl' = 1, e(Cs) = \{\neg x_1 \vee x_2 \vee \neg x_4\}$

After $\textsc{AddClauses}(e(Cs)); m := \textsc{UnitPropagation}()$
$m = \{x_4 \mapsto 1, x_2 \mapsto 0, x_1 \mapsto 1, x_1 \mapsto 0\}$

Example : DPLL($\mathcal{T}$)

Consider $F = (x = y \vee y = z) \wedge (y \neq z \vee z = u) \wedge (z = x)$
$e(F) = (x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge x_4$

After $\textsc{AddClauses}(e(F)); m := \textsc{UnitPropagation}()$
$m = \{x_4 \mapsto 1\}$

After $m := \textsc{Decide}();$
$m = \{x_4 \mapsto 1, x_2 \mapsto 0\}$

After $m := \textsc{UnitPropagation}()$
$m = \{x_4 \mapsto 1, x_2 \mapsto 0, x_1 \mapsto 1\}$

After $(Cs, dl') := \textsc{TheoryDeduction}(x = y \wedge y \neq z \wedge z = x)$
$Cs = \{x \neq y \vee y = z \vee z \neq x\}, dl' = 1, e(Cs) = \{\neg x_1 \vee x_2 \vee \neg x_4\}$

After $\textsc{AddClauses}(e(Cs)); m := \textsc{UnitPropagation}()$
$m = \{x_4 \mapsto 1, x_2 \mapsto 0, x_1 \mapsto 1, x_1 \mapsto 0\} \leftarrow$ conflict

Reminder: theory propagation implementation

Algorithm 1.2: THEORYDEDUCTION

Input: Set of literals *Ls*

Read only input: *m* partial model, *dstack* decision depths, *dl* current decision level

Reminder: theory propagation implementation

Algorithm 1.2: TheoryDeduction

Input: Set of literals *Ls*
Read only input: *m* partial model, *dstack* decision depths, *dl* current decision level
foreach $\ell \in Ls$ **do**
    $DP_{\mathcal{T}}.push(\ell)$

Reminder: theory propagation implementation

Algorithm 1.2: THEORYDEDUCTION

Input: Set of literals Ls

Read only input: m partial model, $dstack$ decision depths, dl current decision level

foreach $\ell \in Ls$ **do**

  $DP_{\mathcal{T}}.push(\ell)$

if $DP_{\mathcal{T}}.checkSat() == unsat$ **then**

  $Ls' := DP_{\mathcal{T}}.unsatCore()$; // minimize clause

  $dl' := \max\{dl'' | \exists \ell \in Ls', i.\ dstack(dl'') < i \land m[i] = e(\ell) \mapsto \_\}$;

  return $(\{\neg \bigwedge Ls'\}, dl')$

else

Reminder: theory propagation implementation

Algorithm 1.2: THEORYDEDUCTION

Input: Set of literals Ls
Read only input: m partial model, $dstack$ decision depths, dl current decision level

foreach $\ell \in Ls$ **do**
    $DP_{\mathcal{T}}.push(\ell)$

if $DP_{\mathcal{T}}.checkSat() == unsat$ **then**
    $Ls' := DP_{\mathcal{T}}.unsatCore()$; // minimize clause
    $dl' := \max\{dl'' | \exists \ell \in Ls', i.\ dstack(dl'') < i \wedge m[i] = e(\ell) \mapsto \_\}$;
    return $(\{\neg \bigwedge Ls'\}, dl')$
else
    //implied clauses
    $Cs := \emptyset$;
    foreach $\ell \in Lits(F)$ **do**
        $DP_{\mathcal{T}}.push(\neg\ell)$;
        if $DP_{\mathcal{T}}.checkSat() == unsat$ **then**
            $Ls' := DP_{\mathcal{T}}.unsatCore()$; // ℓ is called implied model and $\neg\ell \in Ls'$
            $Cs := Cs \cup \{\neg \bigwedge Ls'\}$;
        $DP_{\mathcal{T}}.pop()$;
    return (Cs, dl)

Topic 1.2

Theory of equality and function symbols (EUF)

Reminder: Theory of equality and function symbols (EUF)

EUF syntax: quantifier-free first order formulas with signature $\mathbf{S} = (\mathbf{F}, \emptyset)$, i.e., countably many function symbols and no predicates.

Reminder: Theory of equality and function symbols (EUF)

EUF syntax: quantifier-free first order formulas with signature $\mathbf{S} = (\mathbf{F}, \emptyset)$, i.e., countably many function symbols and no predicates.

The theory axioms include

1. $\forall x.\ x \approx x$
2. $\forall x, y.\ x \approx y \Rightarrow y \approx x$
3. $\forall x, y, z.\ x \approx y \wedge y \approx z \Rightarrow x \approx z$
4. for each $f/n \in \mathbf{F}$,

$$\forall x_1, .., x_n, y_1, .., y_n.\ x_1 \approx y_1 \wedge .. \wedge x_n \approx y_n \Rightarrow f(x_1, .., x_n) \approx f(y_1, .., y_n)$$

Reminder: Theory of equality and function symbols (EUF)

EUF syntax: quantifier-free first order formulas with signature $\mathbf{S} = (\mathbf{F}, \emptyset)$, i.e., countably many function symbols and no predicates.

The theory axioms include

1. $\forall x.\ x \approx x$
2. $\forall x, y.\ x \approx y \Rightarrow y \approx x$
3. $\forall x, y, z.\ x \approx y \wedge y \approx z \Rightarrow x \approx z$
4. for each $f/n \in \mathbf{F}$,

$$\forall x_1, .., x_n, y_1, .., y_n.\ x_1 \approx y_1 \wedge .. \wedge x_n \approx y_n \Rightarrow f(x_1, .., x_n) \approx f(y_1, .., y_n)$$

Since the axioms are valid in FOL with equality, the theory is sometimes referred as the base theory.

Reminder: Theory of equality and function symbols (EUF)

EUF syntax: quantifier-free first order formulas with signature $\mathbf{S} = (\mathbf{F}, \emptyset)$, i.e., countably many function symbols and no predicates.

The theory axioms include

1. $\forall x.\ x \approx x$
2. $\forall x, y.\ x \approx y \Rightarrow y \approx x$
3. $\forall x, y, z.\ x \approx y \wedge y \approx z \Rightarrow x \approx z$
4. for each $f/n \in \mathbf{F}$,

$$\forall x_1, .., x_n, y_1, .., y_n.\ x_1 \approx y_1 \wedge .. \wedge x_n \approx y_n \Rightarrow f(x_1, .., x_n) \approx f(y_1, .., y_n)$$

Since the axioms are valid in FOL with equality, the theory is sometimes referred as the base theory.

Note: Predicates can be easily added if desired

Proofs in $\mathcal{T}_{EUF}$

Proof rules of $\mathcal{T}_{EUF}$

$$\frac{x = y}{y = x}\,Symmetry$$

$$\frac{x = y \quad y = z}{x = z}\,Transitivity$$

$$\frac{x_1 = y_1 \quad .. \quad x_n = y_n}{f(x_1, .., x_n) = f(y_1, .., y_n)}\,Congruence$$

Example 1.3

Consider: $y = z \wedge y = z \wedge f(x, u) \neq f(z, u)$

Proofs in $\mathcal{T}_{EUF}$

Proof rules of $\mathcal{T}_{EUF}$

$$\frac{x = y}{y = x}\,Symmetry$$

$$\frac{x = y \quad y = z}{x = z}\,Transitivity$$

$$\frac{x_1 = y_1 \quad .. \quad x_n = y_n}{f(x_1, .., x_n) = f(y_1, .., y_n)}\,Congruence$$

Example 1.3

Consider: $y = z \wedge y = z \wedge f(x, u) \neq f(z, u)$

$$\frac{y = z}{x = y}$$

Proofs in $\mathcal{T}_{EUF}$

Proof rules of $\mathcal{T}_{EUF}$

$$\frac{x = y}{y = x}\,\mathit{Symmetry}$$

$$\frac{x = y \quad y = z}{x = z}\,\mathit{Transitivity}$$

$$\frac{x_1 = y_1 \quad .. \quad x_n = y_n}{f(x_1, .., x_n) = f(y_1, .., y_n)}\,\mathit{Congruence}$$

Example 1.3

Consider: $y = z \wedge y = z \wedge f(x, u) \neq f(z, u)$

$$\frac{\dfrac{y = z}{x = y} \quad y = z}{x = z}$$

Proofs in $\mathcal{T}_{EUF}$

Proof rules of $\mathcal{T}_{EUF}$

$$\frac{x = y}{y = x}\,Symmetry$$

$$\frac{x = y \quad y = z}{x = z}\,Transitivity$$

$$\frac{x_1 = y_1 \quad .. \quad x_n = y_n}{f(x_1, .., x_n) = f(y_1, .., y_n)}\,Congruence$$

Example 1.3

Consider: $y = z \wedge y = z \wedge f(x, u) \neq f(z, u)$

$$\cfrac{\cfrac{\cfrac{y = z}{x = y} \quad y = z}{x = z}}{f(x, u) = f(z, u)}$$

Proofs in $\mathcal{T}_{EUF}$

Proof rules of $\mathcal{T}_{EUF}$

$$\frac{x = y}{y = x}\, Symmetry$$

$$\frac{x = y \quad y = z}{x = z}\, Transitivity$$

$$\frac{x_1 = y_1 \quad .. \quad x_n = y_n}{f(x_1, .., x_n) = f(y_1, .., y_n)}\, Congruence$$

Example 1.3

Consider: $y = z \land y = z \land f(x, u) \neq f(z, u)$

$$\dfrac{\dfrac{\dfrac{\dfrac{y = z}{x = y} \quad y = z}{x = z}}{f(x,u) = f(z,u)} \quad f(x,u) \neq f(z,u)}{\perp}$$

DP_{EUF}

Decides conjunction of literals with interface push,pop,checkSat and unsatCore

General idea: maintain equivalence classes among terms

DP_{EUF}

Decides conjunction of literals with interface push,pop,checkSat and unsatCore

General idea: maintain equivalence classes among terms

Algorithm 1.3: $DP_{EUF}.push(t_1 \bowtie t_2)$

globals: set of terms $Ts := \emptyset$, set of pair of classes $DisEq := \emptyset$, bool $conflictFound := 0$

DP_{EUF}

Decides conjunction of literals with interface push,pop,checkSat and unsatCore

General idea: maintain equivalence classes among terms

Algorithm 1.3: $DP_{EUF}.push(t_1 \bowtie t_2)$

---

globals: set of terms $Ts := \emptyset$, set of pair of classes $DisEq := \emptyset$, bool $conflictFound := 0$

$Ts := Ts \cup subTerms(t_1) \cup subTerms(t_2)$;

DP_{EUF}

Decides conjunction of literals with interface push,pop,checkSat and unsatCore

General idea: maintain equivalence classes among terms

Algorithm 1.3: $DP_{EUF}.push(t_1 \bowtie t_2)$

globals: set of terms $Ts := \emptyset$, set of pair of classes $DisEq := \emptyset$, bool $conflictFound := 0$

$Ts := Ts \cup subTerms(t_1) \cup subTerms(t_2)$;

$C_1 := getClass(t_1)$; $C_2 := getClass(t_2)$; // if t_1 and t_2 are seen first time, create new classes

DP_{EUF}

Decides conjunction of literals with interface push,pop,checkSat and unsatCore

General idea: maintain equivalence classes among terms

Algorithm 1.3: $DP_{EUF}.push(t_1 \bowtie t_2)$

globals: set of terms $Ts := \emptyset$, set of pair of classes $DisEq := \emptyset$, bool $conflictFound := 0$

$Ts := Ts \cup subTerms(t_1) \cup subTerms(t_2)$;

$C_1 := getClass(t_1)$; $C_2 := getClass(t_2)$; // if t_1 and t_2 are seen first time, create new classes

if $\bowtie$ = "$\approx$" **then**

  if $C_1 = C_2$ **then** **return** ;

DP_{EUF}

Decides conjunction of literals with interface push,pop,checkSat and unsatCore

General idea: maintain equivalence classes among terms

Algorithm 1.3: $DP_{EUF}.push(t_1 \bowtie t_2)$

globals: set of terms $Ts := \emptyset$, set of pair of classes $DisEq := \emptyset$, bool $conflictFound := 0$

$Ts := Ts \cup subTerms(t_1) \cup subTerms(t_2)$;

$C_1 := getClass(t_1)$; $C_2 := getClass(t_2)$; // if t_1 and t_2 are seen first time, create new classes

if $\bowtie$ = "$\approx$" **then**

  if $C_1 = C_2$ **then** **return** ;

  $C = mergeClasses(C_1, C_2)$; $parent(C) := (C_1, C_2, t_1 \approx t_2)$;

DP_{EUF}

Decides conjunction of literals with interface push,pop,checkSat and unsatCore

General idea: maintain equivalence classes among terms

Algorithm 1.3: $DP_{EUF}.push(t_1 \bowtie t_2)$

---

globals: set of terms $Ts := \emptyset$, set of pair of classes $DisEq := \emptyset$, bool $conflictFound := 0$

$Ts := Ts \cup subTerms(t_1) \cup subTerms(t_2)$;

$C_1 := getClass(t_1)$; $C_2 := getClass(t_2)$; // if t_1 and t_2 are seen first time, create new classes

if $\bowtie$ = "$\approx$" **then**

    if $C_1 = C_2$ **then** **return** ;

    $C = mergeClasses(C_1, C_2)$; $parent(C) := (C_1, C_2, t_1 \approx t_2)$;

    if $(C_1, C_2) \in DisEq$ **then** { $conflictFound := 1$; **return**; } ;

DP_{EUF}

Decides conjunction of literals with interface push,pop,checkSat and unsatCore

General idea: maintain equivalence classes among terms

Algorithm 1.3: $DP_{EUF}.push(t_1 \bowtie t_2)$

globals: set of terms $Ts := \emptyset$, set of pair of classes $DisEq := \emptyset$, bool $conflictFound := 0$

$Ts := Ts \cup subTerms(t_1) \cup subTerms(t_2)$;

$C_1 := getClass(t_1)$; $C_2 := getClass(t_2)$; // if t_1 and t_2 are seen first time, create new classes

if $\bowtie$ = "$\approx$" **then**

  if $C_1 = C_2$ **then** **return** ;

  $C = mergeClasses(C_1, C_2)$; $parent(C) := (C_1, C_2, t_1 \approx t_2)$;

  if $(C_1, C_2) \in DisEq$ **then** { $conflictFound := 1$; **return**; } ;

DP_{EUF}

Decides conjunction of literals with interface push,pop,checkSat and unsatCore

General idea: maintain equivalence classes among terms

Algorithm 1.3: $DP_{EUF}.push(t_1 \bowtie t_2)$

globals: set of terms $Ts := \emptyset$, set of pair of classes $DisEq := \emptyset$, bool $conflictFound := 0$

$Ts := Ts \cup subTerms(t_1) \cup subTerms(t_2)$;

$C_1 := getClass(t_1)$; $C_2 := getClass(t_2)$; // if t_1 and t_2 are seen first time, create new classes

if $\bowtie$ = "$\approx$" **then**

  if $C_1 = C_2$ **then** **return** ;

  $C = mergeClasses(C_1, C_2)$; $parent(C) := (C_1, C_2, t_1 \approx t_2)$;

  if $(C_1, C_2) \in DisEq$ **then** { $conflictFound := 1$; **return**; } ;

else

  // $\bowtie$ = "$\not\approx$"

  $DisEq := DisEq \cup (C_1, C_2)$;

  if $C_1 = C_2$ **then** $conflictFound := 1$; **return** ;

DP_{EUF}

Decides conjunction of literals with interface push,pop,checkSat and unsatCore

General idea: maintain equivalence classes among terms

Algorithm 1.3: $DP_{EUF}.push(t_1 \bowtie t_2)$

globals: set of terms $Ts := \emptyset$, set of pair of classes $DisEq := \emptyset$, bool $conflictFound := 0$

$Ts := Ts \cup subTerms(t_1) \cup subTerms(t_2)$;

$C_1 := getClass(t_1)$; $C_2 := getClass(t_2)$; // if t_1 and t_2 are seen first time, create new classes

if $\bowtie$ = "$\approx$" **then**

  if $C_1 = C_2$ **then** **return** ;

  $C = mergeClasses(C_1, C_2)$; $parent(C) := (C_1, C_2, t_1 \approx t_2)$;

  if $(C_1, C_2) \in DisEq$ **then** { $conflictFound := 1$; **return**; } ;

else

  // $\bowtie$ = "$\not\approx$"

  $DisEq := DisEq \cup (C_1, C_2)$;

  if $C_1 = C_2$ **then** $conflictFound := 1$; **return** ;

foreach $f(r_1, \ldots, r_n), f(s_1, \ldots, s_n) \in Ts \wedge \forall i \in 1..n.\ \exists C.\ r_i, s_i \in C$ **do**

  $DP_{EUF}.push(f(r_1, \ldots, r_n) \approx f(s_1, \ldots, s_n))$;

DP_{EUF}

Decides conjunction of literals with interface push,pop,checkSat and unsatCore

General idea: maintain equivalence classes among terms

Algorithm 1.3: $DP_{EUF}.push(t_1 \bowtie t_2)$

globals: set of terms $Ts := \emptyset$, set of pair of classes $DisEq := \emptyset$, bool $conflictFound := 0$

$Ts := Ts \cup subTerms(t_1) \cup subTerms(t_2)$;

$C_1 := getClass(t_1)$; $C_2 := getClass(t_2)$; // if t_1 and t_2 are seen first time, create new classes

if $\bowtie\ =$ "$\approx$" **then**

  if $C_1 = C_2$ **then** **return** ;

  $C = mergeClasses(C_1, C_2)$; $parent(C) := (C_1, C_2, t_1 \approx t_2)$;

  if $(C_1, C_2) \in DisEq$ **then** { $conflictFound := 1$; **return**; } ;

else

  // $\bowtie\ =$ "$\not\approx$"

  $DisEq := DisEq \cup (C_1, C_2)$;

  if $C_1 = C_2$ **then** $conflictFound := 1$; **return** ;

foreach $f(r_1, \dots, r_n), f(s_1, \dots, s_n) \in Ts \wedge \forall i \in 1..n.\ \exists C.\ r_i, s_i \in C$ **do**

  $DP_{EUF}.push(f(r_1, \dots, r_n) \approx f(s_1, \dots, s_n))$;

Exercise 1.1

Run DP_{EUF}.push on $x \approx f(x) \wedge f(f(f(x))) \not\approx f(f(x))$.

checkSat and pop

- $DP_{EUF}.checkSat()$ { **return** *conflictFound*; }
- $DP_{EUF}.pop()$ is implemented by recording the time stamp of pushes and undoing all the mergers happened in after last push.

Unsat core

Definition 1.2

For an unsat set of formulas Σ, an unsat core of Σ is a subset (preferably minimal) of Σ that is unsat.

Unsat core

Definition 1.2

For an unsat set of formulas Σ, an unsat core of Σ is a subset (preferably minimal) of Σ that is unsat.

Algorithm 1.4: $DP_{EUF}.unsatCore()$

---

assume($conflictFound = 1$);

Unsat core

Definition 1.2

For an unsat set of formulas Σ, an unsat core of Σ is a subset (preferably minimal) of Σ that is unsat.

Algorithm 1.4: $DP_{EUF}.unsatCore()$

---

assume($conflictFound = 1$);
Let $(t_1 \not\approx t_2)$ be the dis-equality that was violated;

Unsat core

Definition 1.2

For an unsat set of formulas Σ, an unsat core of Σ is a subset (preferably minimal) of Σ that is unsat.

Algorithm 1.4: $DP_{EUF}.unsatCore()$

---

assume($conflictFound = 1$);
Let $(t_1 \not\approx t_2)$ be the dis-equality that was violated;
return $\{t_1 \not\approx t_2\} \cup getReason(t_1, t_2)$;

Unsat core

Definition 1.2

For an unsat set of formulas Σ, an unsat core of Σ is a subset (preferably minimal) of Σ that is unsat.

Algorithm 1.4: $DP_{EUF}.unsatCore()$

---

assume($conflictFound = 1$);
Let $(t_1 \not\approx t_2)$ be the dis-equality that was violated;
return $\{t_1 \not\approx t_2\} \cup getReason(t_1, t_2)$;

Algorithm 1.5: $getReason(t_1, t_2)$

Unsat core

Definition 1.2

For an unsat set of formulas Σ, an unsat core of Σ is a subset (preferably minimal) of Σ that is unsat.

Algorithm 1.4: $DP_{EUF}.unsatCore()$

---

assume($conflictFound = 1$);
Let $(t_1 \not\approx t_2)$ be the dis-equality that was violated;
return $\{t_1 \not\approx t_2\} \cup getReason(t_1, t_2)$;

Algorithm 1.5: $getReason(t_1, t_2)$

Let $(t'_1 \approx t'_2)$ be the merge operation that placed t_1 and t_2 in same class;

Unsat core

Definition 1.2

For an unsat set of formulas Σ, an unsat core of Σ is a subset (preferably minimal) of Σ that is unsat.

Algorithm 1.4: $DP_{EUF}.unsatCore()$

assume($conflictFound = 1$);
Let $(t_1 \not\approx t_2)$ be the dis-equality that was violated;
return $\{t_1 \not\approx t_2\} \cup getReason(t_1, t_2)$;

Algorithm 1.5: $getReason(t_1, t_2)$

Let $(t'_1 \approx t'_2)$ be the merge operation that placed t_1 and t_2 in same class;
if $t'_1 = f(s_1, .., s_k) \approx f(u_1, ...u_k) = t'_2$ *was derived due to congruence* **then**
  $reason := \bigcup_i getReason(s_i, u_i)$
else
  $reason := \{t'_1 \approx t'_2\}$

Unsat core

Definition 1.2

For an unsat set of formulas Σ, an unsat core of Σ is a subset (preferably minimal) of Σ that is unsat.

Algorithm 1.4: $DP_{EUF}.unsatCore()$

---

assume($conflictFound = 1$);
Let $(t_1 \not\approx t_2)$ be the dis-equality that was violated;
return $\{t_1 \not\approx t_2\} \cup getReason(t_1, t_2)$;

Algorithm 1.5: $getReason(t_1, t_2)$

Let $(t_1' \approx t_2')$ be the merge operation that placed t_1 and t_2 in same class;
if $t_1' = f(s_1, .., s_k) \approx f(u_1, ...u_k) = t_2'$ *was derived due to congruence* **then**
    $reason := \bigcup_i getReason(s_i, u_i)$
else
    $reason := \{t_1' \approx t_2'\}$
$Set = getReason(t_1, t_1') \cup reason \cup getReason(t_2', t_2)$

Topic 1.3

Algorithms for EUF

DP_{EUF} implementation - union-find

Equivalence classes are usually implemented using union-find data structure

- each class is represented using a tree over its member terms
- root of the tree represents the class

DP_{EUF} implementation - union-find

Equivalence classes are usually implemented using union-find data structure

- each class is represented using a tree over its member terms
- root of the tree represents the class
- getClass() returns root of the tree, which involves traversing to the root

DP_{EUF} implementation - union-find

Equivalence classes are usually implemented using union-find data structure

- each class is represented using a tree over its member terms
- root of the tree represents the class
- getClass() returns root of the tree, which involves traversing to the root
- mergeClasses() simply adds the root of smaller tree as a child of the root of larger class

DP_{EUF} implementation - union-find

Equivalence classes are usually implemented using union-find data structure

- each class is represented using a tree over its member terms
- root of the tree represents the class
- getClass() returns root of the tree, which involves traversing to the root
- mergeClasses() simply adds the root of smaller tree as a child of the root of larger class

Efficient data-structure: for n pushes, run time is $O(n \log n)$

DP_{EUF} implementation - union-find

Equivalence classes are usually implemented using union-find data structure

- each class is represented using a tree over its member terms
- root of the tree represents the class
- getClass() returns root of the tree, which involves traversing to the root
- mergeClasses() simply adds the root of smaller tree as a child of the root of larger class

Efficient data-structure: for n pushes, run time is $O(n \log n)$

Exercise 1.2

Prove the above complexity

Example: union-find

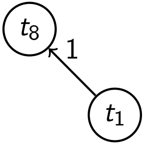
Consider:

$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$

Example: union-find

Consider:

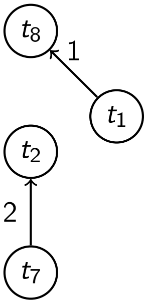
$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



Example: union-find

Consider:

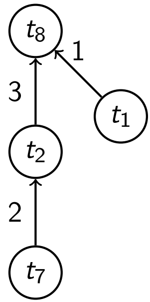
$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



Example: union-find

Consider:

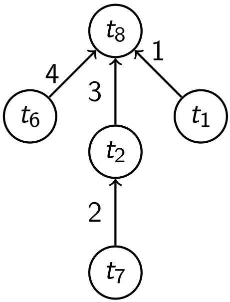
$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



Example: union-find

Consider:

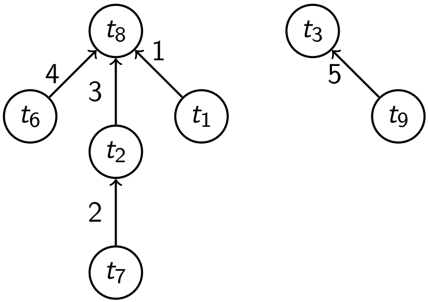
$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



Example: union-find

Consider:

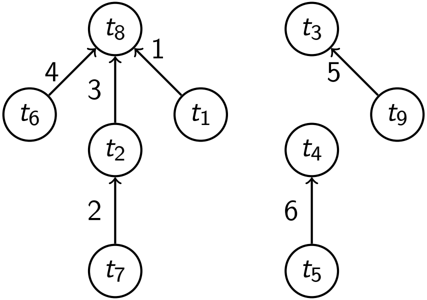
$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



Example: union-find

Consider:

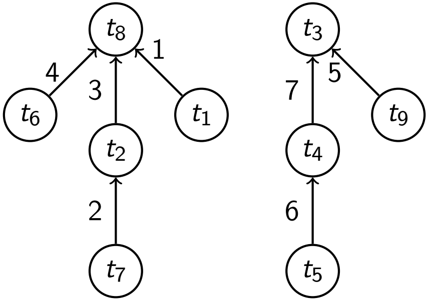
$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



Example: union-find

Consider:

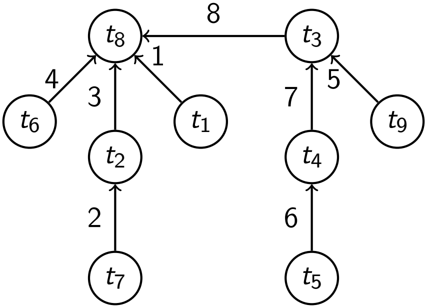
$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



Example: union-find

Consider:

$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



unsatCore using union find

- generate proof of unsatisfiablity using union find
- collect leaves of the proof, which can serve as an unsat core

Proof generation in union-find

Proof generation from union find data structure for an unsat input.

The proof is constructed bottom up.

1. There must be a dis-equality $s \neq v$ that was violated.
   We need to find the proof for $s = v$.

Proof generation in union-find

Proof generation from union find data structure for an unsat input.

The proof is constructed bottom up.

1. There must be a dis-equality $s \neq v$ that was violated.
   We need to find the proof for $s = v$.

2. Find the latest edge in the path between s and v.

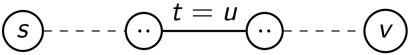
Proof generation in union-find

Proof generation from union find data structure for an unsat input.

The proof is constructed bottom up.

1. There must be a dis-equality $s \neq v$ that was violated.
   We need to find the proof for $s = v$.

2. Find the latest edge in the path between s and v. Let us say it is due to t and u.



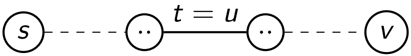
Proof generation in union-find

Proof generation from union find data structure for an unsat input.

The proof is constructed bottom up.

1. There must be a dis-equality $s \neq v$ that was violated.
   We need to find the proof for $s = v$.

2. Find the latest edge in the path between s and v. Let us say it is due to t and u.



   2.1 Recursively, find the proof of $s = t$ and $u = v$.

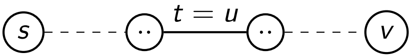
Proof generation in union-find

Proof generation from union find data structure for an unsat input.

The proof is constructed bottom up.

1. There must be a dis-equality $s \neq v$ that was violated.
   We need to find the proof for $s = v$.

2. Find the latest edge in the path between s and v. Let us say it is due to t and u.



   2.1 Recursively, find the proof of $s = t$ and $u = v$.
   2.2 Construct a proof for $s = v$, which is either
      - in input or
      - due to congruence.
   2.3 In the congruence case, let us suppose $t = f(t_1, .., t_n) = f(u_1, .., u_n) = u$.
   Then, we also recursively search for proofs of $t_i = u_i$.

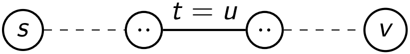
Proof generation in union-find

Proof generation from union find data structure for an unsat input.

The proof is constructed bottom up.

1. There must be a dis-equality $s \neq v$ that was violated.
   We need to find the proof for $s = v$.

2. Find the latest edge in the path between s and v. Let us say it is due to t and u.



   2.1 Recursively, find the proof of $s = t$ and $u = v$.
   2.2 Construct a proof for $s = v$, which is either
      - in input or
      - due to congruence.
   2.3 In the congruence case, let us suppose $t = f(t_1, .., t_n) = f(u_1, .., u_n) = u$. Then, we also recursively search for proofs of $t_i = u_i$.

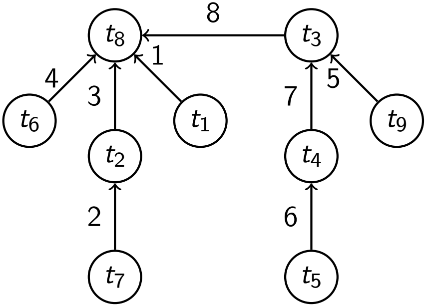
   Stitch the proofs appropriately.

For improved algorithm: R. Nieuwenhuis and A. Oliveras. Proof-producing congruence closure. RTA'05, LNCS 3467

Example: union-find proof generation

Consider:

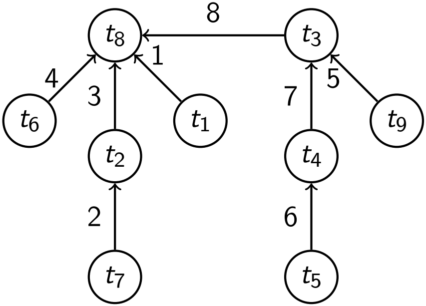
$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



Example: union-find proof generation

Consider:

$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



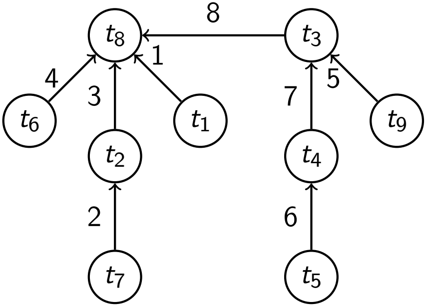
1. $t_1 \neq t_4$ is violated.

$$\frac{t_1 \neq t_4}{\bot}$$

Example: union-find proof generation

Consider:

$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



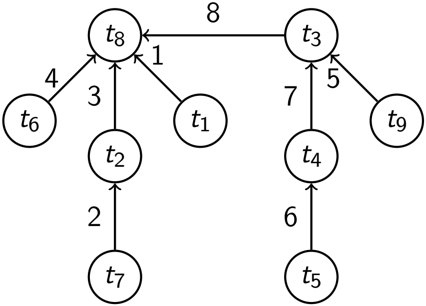
1. $t_1 \neq t_4$ is violated.
2. 8 is the latest edge in the path between t_1 and t_4

$$\dfrac{t_1 \neq t_4 \qquad \dfrac{\overline{\qquad\qquad}}{t_1 = t_4}}{\bot}$$

Example: union-find proof generation

Consider:

$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



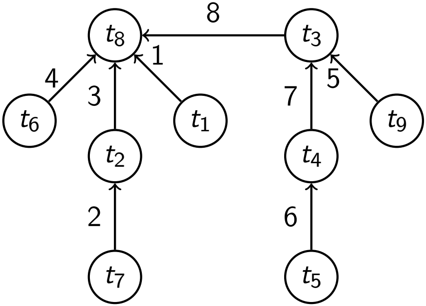
1. $t_1 \neq t_4$ is violated.
2. 8 is the latest edge in the path between t_1 and t_4
3. 8 is due to $t_5 = t_7$

$$\dfrac{t_1 \neq t_4 \qquad \dfrac{\dfrac{\dfrac{t_5 = t_7}{t_7 = t_5}}{\qquad\qquad}}{t_1 = t_4}}{\bot}$$

Example: union-find proof generation

Consider:

$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



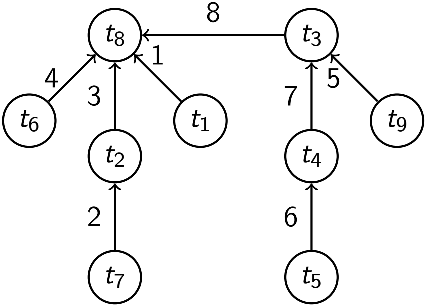
1. $t_1 \neq t_4$ is violated.
2. 8 is the latest edge in the path between t_1 and t_4
3. 8 is due to $t_5 = t_7$
4. Now look for proof of $t_1 = t_7$ and $t_5 = t_4$

$$\dfrac{t_1 \neq t_4 \qquad \dfrac{\dfrac{\dfrac{t_5 = t_7}{t_7 = t_5}}{}}{t_1 = t_4}}{\bot}$$

Example: union-find proof generation

Consider:

$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



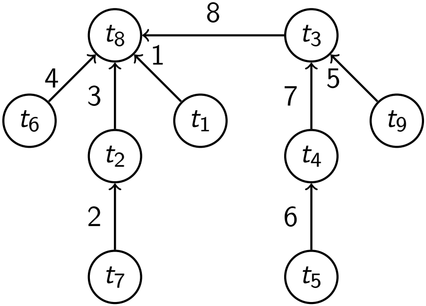
$$\dfrac{t_1 \neq t_4 \qquad \dfrac{\dfrac{\dfrac{t_7 = t_1}{t_1 = t_7} \qquad \dfrac{t_5 = t_7}{t_7 = t_5}}{}}{t_1 = t_4}}{\bot}$$

1. $t_1 \neq t_4$ is violated.
2. 8 is the latest edge in the path between t_1 and t_4
3. 8 is due to $t_5 = t_7$
4. Now look for proof of $t_1 = t_7$ and $t_5 = t_4$
5. 3 is the latest edge between t_1 and t_7, which is due to $t_1 = t_7$.

Example: union-find proof generation

Consider:

$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



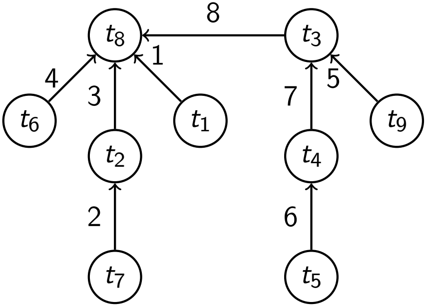
$$\dfrac{t_1 \neq t_4 \qquad \dfrac{\dfrac{\dfrac{t_7 = t_1}{t_1 = t_7} \qquad \dfrac{t_5 = t_7}{t_7 = t_5}}{} \qquad t_5 = t_4}{t_1 = t_4}}{\bot}$$

1. $t_1 \neq t_4$ is violated.
2. 8 is the latest edge in the path between t_1 and t_4
3. 8 is due to $t_5 = t_7$
4. Now look for proof of $t_1 = t_7$ and $t_5 = t_4$
5. 3 is the latest edge between t_1 and t_7, which is due to $t_1 = t_7$.
6. Similarly, $t_5 = t_4$ is edge 6

Example: union-find proof generation

Consider:

$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



$$\dfrac{t_1 \neq t_4 \qquad \dfrac{\dfrac{\dfrac{t_7 = t_1}{t_1 = t_7} \qquad \dfrac{t_5 = t_7}{t_7 = t_5}}{t_1 = t_5} \qquad t_5 = t_4}{t_1 = t_4}}{\bot}$$

1. $t_1 \neq t_4$ is violated.
2. 8 is the latest edge in the path between t_1 and t_4
3. 8 is due to $t_5 = t_7$
4. Now look for proof of $t_1 = t_7$ and $t_5 = t_4$
5. 3 is the latest edge between t_1 and t_7, which is due to $t_1 = t_7$.
6. Similarly, $t_5 = t_4$ is edge 6

Another example proof generation

Example 1.4

Run union find on $\underbrace{f^5(a) = a}_{1} \wedge \underbrace{f^3(a) = a}_{2} \wedge \underbrace{f(a) \neq a}_{3}$

Another example proof generation

Example 1.4

Run union find on $\underbrace{f^5(a) = a}_{1} \wedge \underbrace{f^3(a) = a}_{2} \wedge \underbrace{f(a) \neq a}_{3}$

$\dashrightarrow$ Parent relation

Another example proof generation

Example 1.4

Run union find on $\underbrace{f^5(a) = a}_{1} \wedge \underbrace{f^3(a) = a}_{2} \wedge \underbrace{f(a) \neq a}_{3}$

$\dashrightarrow$ Parent relation

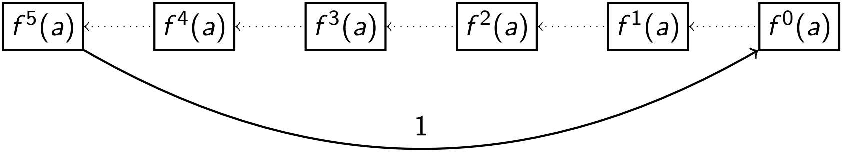
$$\boxed{f^5(a)} \dashleftarrow \boxed{f^4(a)} \dashleftarrow \boxed{f^3(a)} \dashleftarrow \boxed{f^2(a)} \dashleftarrow \boxed{f^1(a)} \dashleftarrow \boxed{f^0(a)}$$

Another example proof generation

Example 1.4

Run union find on $\underbrace{f^5(a) = a}_{1} \wedge \underbrace{f^3(a) = a}_{2} \wedge \underbrace{f(a) \neq a}_{3}$

$\dashrightarrow$ Parent relation

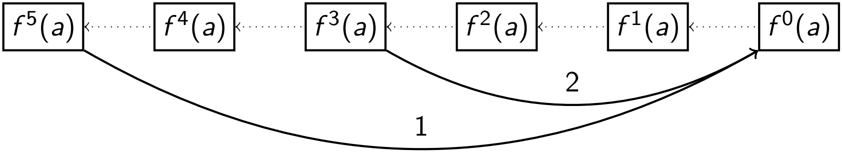


Another example proof generation

Example 1.4

Run union find on $\underbrace{f^5(a) = a}_{1} \wedge \underbrace{f^3(a) = a}_{2} \wedge \underbrace{f(a) \neq a}_{3}$

Parent relation

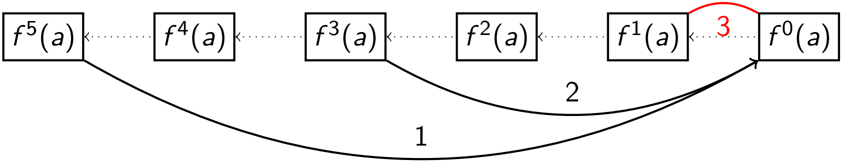


Another example proof generation

Example 1.4

Run union find on $\underbrace{f^5(a) = a}_{1} \wedge \underbrace{f^3(a) = a}_{2} \wedge \underbrace{f(a) \neq a}_{3}$

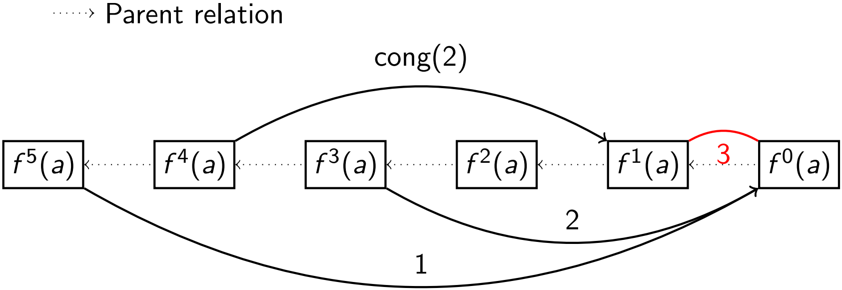
$\cdots\!\!\rightarrow$ Parent relation



Another example proof generation

Example 1.4

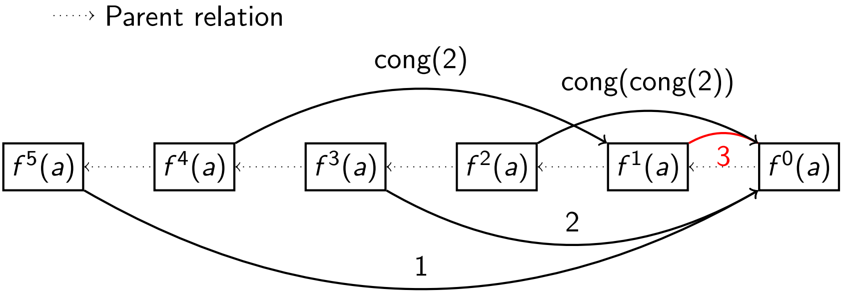
Run union find on $\underbrace{f^5(a) = a}_{1} \wedge \underbrace{f^3(a) = a}_{2} \wedge \underbrace{f(a) \neq a}_{3}$



Another example proof generation

Example 1.4

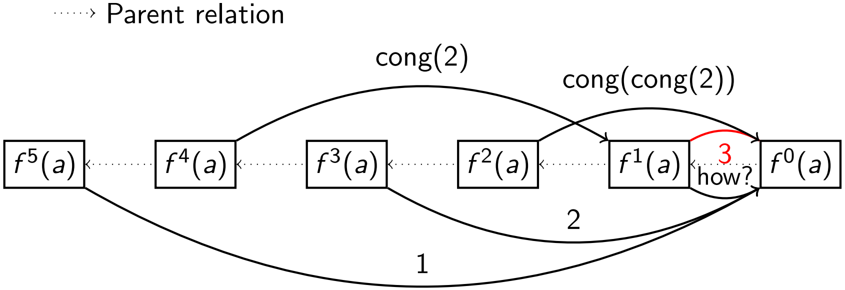
Run union find on $\underbrace{f^5(a) = a}_{1} \wedge \underbrace{f^3(a) = a}_{2} \wedge \underbrace{f(a) \neq a}_{3}$



Another example proof generation

Example 1.4

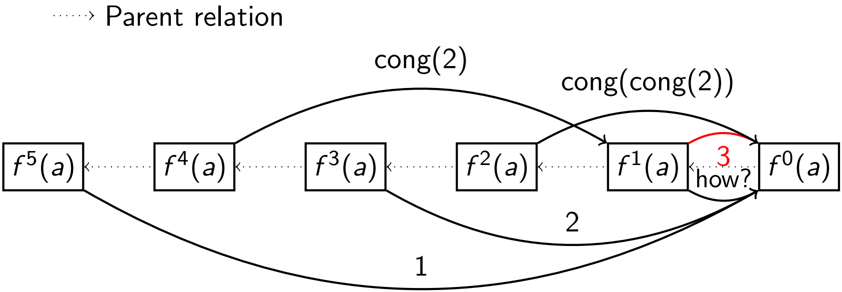
Run union find on $\underbrace{f^5(a) = a}_{1} \wedge \underbrace{f^3(a) = a}_{2} \wedge \underbrace{f(a) \neq a}_{3}$



Another example proof generation

Example 1.4

Run union find on $\underbrace{f^5(a) = a}_{1} \wedge \underbrace{f^3(a) = a}_{2} \wedge \underbrace{f(a) \neq a}_{3}$



Extract proof from the above graph?

Topic 1.4

Engineering for EUF

Engineering DPLL(QF_EUF)

Now we will look into the internals of Z3.

Engineering DPLL(QF_EUF)

Now we will look into the internals of Z3.

Key ideas to learn in implementation design

- term management in Z3
- tactics layer
- DPLL implementation
- base theory(QF_EUF) implementation

Engineering DPLL(QF_EUF)

Now we will look into the internals of Z3.

Key ideas to learn in implementation design

- term management in Z3
- tactics layer
- DPLL implementation
- base theory(QF_EUF) implementation

Key ideas to learn in software design

- be comfortable with large code base
- memory management
- customized library support

emacs and gdb

We need to learn to use an ide and a debugger before start looking inside Z3

emacs and gdb

We need to learn to use an ide and a debugger before start looking inside Z3

Here, we will use emacs23 and gdb on linux.

emacs and gdb

We need to learn to use an ide and a debugger before start looking inside Z3

Here, we will use emacs23 and gdb on linux.

I know you love some other tool chain. Please cooperate.

Topic 1.5

Problems

Problem

Exercise 1.3 (1.5 points)

Prove/Disprove that the following formula is unsat.

$$(f^4(a) \approx a \vee f^6(a) \approx a) \wedge f^3(a) \approx a \wedge f(a) \not\approx a$$

If unsat give a proof otherwise give a satisfying assignment.

Please show a run of DPLL($\mathcal{T}$) and union-find on the above example.

Problem

Exercise 1.4 (2.5 points)

Read Z3 code for maintenance of parent relation in EUF solving.
Write a short explanation of optimizations implemented to achieve efficient operations on the parent relation.
Files of interest are:

- *src/smt/smt_enode.cpp // class for nodes in union-find*
- *src/smt/smt_enode.h*
- *src/smt/smt_cg_table.cpp // class for parent relation*
- *src/smt/smt_cg_table.h*
- *src/smt/smt_context.cpp // class for smt solver*
- *src/smt/smt_context.h*
- *src/smt/smt_internalizer.cpp:902*
- *src/smt/smt_internalizer.cpp:968*

SMT solving begins at src/smt/smt_context.cpp:3100
context object in smt_context.h has many display functions. Use them to print current state during debugging

End of Lecture 1