Interpolation Tutorial @IITB 2015

Lecture 1: Proof generation and Interpolation

Instructor: Ashutosh Gupta

TIFR, India

Compile date: 2015-11-07

Where are we and where are we going?

I assume, you have seen

- CDCL (or DPLL)
- CDCL($\mathcal{T}$) (or DPLL($\mathcal{T}$))
- Theory of equality with uninterpreted functions($\mathcal{T}_{EUF}$)
- Theory of linear rational arithmetic ($\mathcal{T}_{LRA}$)

Where are we and where are we going?

I assume, you have seen

- CDCL (or DPLL)
- CDCL($\mathcal{T}$) (or DPLL($\mathcal{T}$))
- Theory of equality with uninterpreted functions($\mathcal{T}_{EUF}$)
- Theory of linear rational arithmetic ($\mathcal{T}_{LRA}$)

We will see

- proof generation in CDCL($\mathcal{T}$) with $\mathcal{T}_{LRA}/\mathcal{T}_{EUF}$
- interpolation in boolean, LRA, EUF
- extension to Horn clauses (if time permits!)

Topic 1.1

Proof generation in CDCL

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 **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.*
- $x > 0 \vee y + x \approx 3.5z$ *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

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

do

stands for decision level

Recall: CDCL(T)

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

Input: CNF F, boolean encoder e

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

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

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

stands for decision level

dstack records history for backtracking

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

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 := $\textsc{UnitPropagation}()$;

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**

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

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

returns a clause set and a decision level

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 $\ell = \bot$.

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 $F = (x = y \vee y = z) \wedge (y \neq z \vee z = u) \wedge (z = x)$

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

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

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

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

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

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

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

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

Topic 1.2

Resolution proofs for propositional logic

Resolution Proofs

A resolution proof rule is

$$\frac{p \vee C \qquad \neg q \vee D}{C \vee D}$$

Resolution Proofs

A resolution proof rule is

$$\frac{p \vee C \qquad \neg q \vee D}{C \vee D}$$

Example 1.3

Suppose $F = (p \vee q) \wedge (\neg p \vee q) \wedge (\neg q \vee r) \wedge \neg r$

Resolution Proofs

A resolution proof rule is

$$\frac{p \vee C \qquad \neg q \vee D}{C \vee D}$$

Example 1.3

Suppose $F = (p \vee q) \wedge (\neg p \vee q) \wedge (\neg q \vee r) \wedge \neg r$

$$p \vee q \qquad \neg p \vee q$$

Resolution Proofs

A resolution proof rule is

$$\frac{p \vee C \qquad \neg q \vee D}{C \vee D}$$

Example 1.3

Suppose $F = (p \vee q) \wedge (\neg p \vee q) \wedge (\neg q \vee r) \wedge \neg r$

$$\frac{p \vee q \qquad \neg p \vee q}{q}$$

Resolution Proofs

A resolution proof rule is

$$\frac{p \vee C \qquad \neg q \vee D}{C \vee D}$$

Example 1.3

Suppose $F = (p \vee q) \wedge (\neg p \vee q) \wedge (\neg q \vee r) \wedge \neg r$

$$\dfrac{p \vee q \qquad \neg p \vee q}{q} \qquad \neg q \vee r$$

Resolution Proofs

A resolution proof rule is

$$\frac{p \vee C \qquad \neg q \vee D}{C \vee D}$$

Example 1.3

Suppose $F = (p \vee q) \wedge (\neg p \vee q) \wedge (\neg q \vee r) \wedge \neg r$

$$\frac{\dfrac{p \vee q \qquad \neg p \vee q}{q} \qquad \neg q \vee r}{r}$$

Resolution Proofs

A resolution proof rule is

$$\frac{p \vee C \qquad \neg q \vee D}{C \vee D}$$

Example 1.3

Suppose $F = (p \vee q) \wedge (\neg p \vee q) \wedge (\neg q \vee r) \wedge \neg r$

$$\cfrac{\cfrac{\cfrac{p \vee q \qquad \neg p \vee q}{q} \qquad \neg q \vee r}{r} \qquad \neg r}{\perp}$$

Implication graph for conflict graph

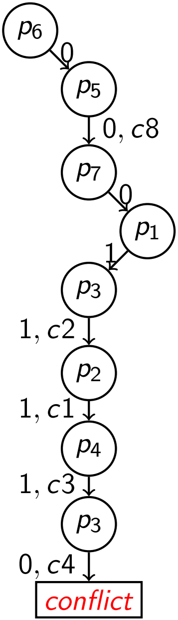
Example 1.4

$$
\begin{aligned}
c_1 &= (\neg p_1 \vee p_2) \\
c_2 &= (\neg p_1 \vee p_3 \vee p_5) \\
c_3 &= (\neg p_2 \vee p_4) \\
c_4 &= (\neg p_3 \vee \neg p_4) \\
c_5 &= (p_1 \vee p_5 \vee \neg p_2) \\
c_6 &= (p_2 \vee p_3) \\
c_7 &= (p_2 \vee \neg p_3 \vee p_7) \\
c_8 &= (p_6 \vee \neg p_5)
\end{aligned}
$$

Implication graph for conflict graph

Example 1.4

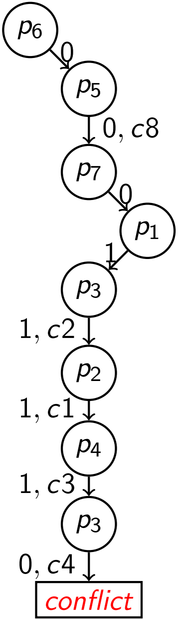
$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$



Implication graph for conflict graph

Example 1.4

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$



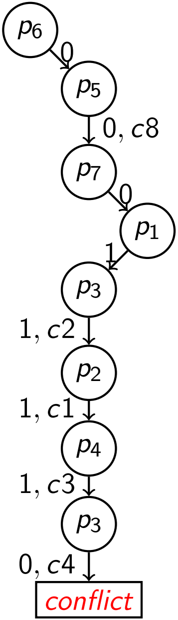
Implication graph

$\boxed{\neg p_6 @ 1}$

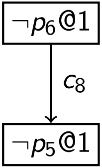
Implication graph for conflict graph

Example 1.4

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$



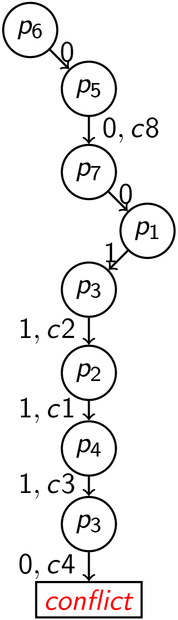
Implication graph



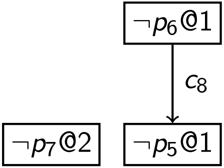
Implication graph for conflict graph

Example 1.4

$$
\begin{aligned}
c_1 &= (\neg p_1 \vee p_2)\\
c_2 &= (\neg p_1 \vee p_3 \vee p_5)\\
c_3 &= (\neg p_2 \vee p_4)\\
c_4 &= (\neg p_3 \vee \neg p_4)\\
c_5 &= (p_1 \vee p_5 \vee \neg p_2)\\
c_6 &= (p_2 \vee p_3)\\
c_7 &= (p_2 \vee \neg p_3 \vee p_7)\\
c_8 &= (p_6 \vee \neg p_5)
\end{aligned}
$$



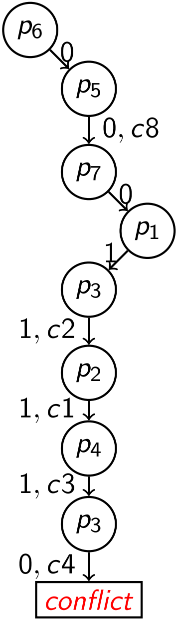
Implication graph



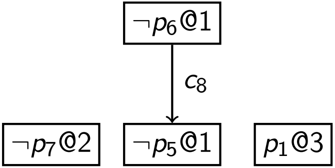
Implication graph for conflict graph

Example 1.4

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$



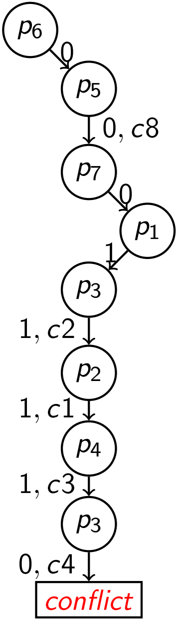
Implication graph



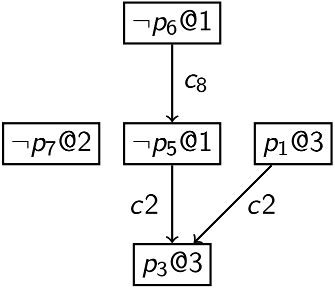
Implication graph for conflict graph

Example 1.4

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$



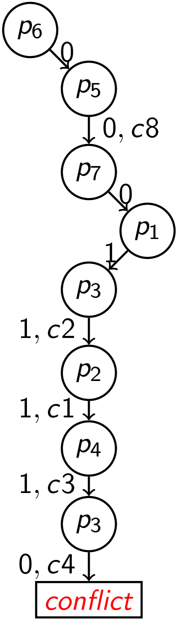
Implication graph



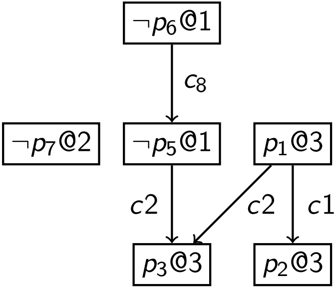
Implication graph for conflict graph

Example 1.4

$$
\begin{aligned}
c_1 &= (\neg p_1 \vee p_2) \\
c_2 &= (\neg p_1 \vee p_3 \vee p_5) \\
c_3 &= (\neg p_2 \vee p_4) \\
c_4 &= (\neg p_3 \vee \neg p_4) \\
c_5 &= (p_1 \vee p_5 \vee \neg p_2) \\
c_6 &= (p_2 \vee p_3) \\
c_7 &= (p_2 \vee \neg p_3 \vee p_7) \\
c_8 &= (p_6 \vee \neg p_5)
\end{aligned}
$$



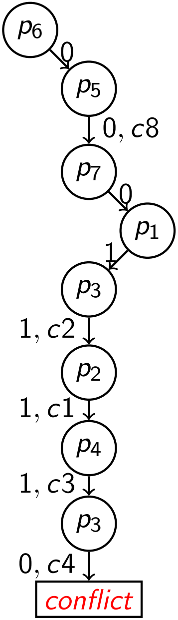
Implication graph



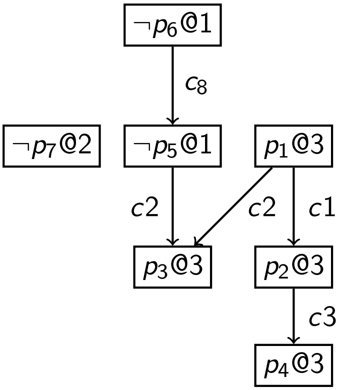
Implication graph for conflict graph

Example 1.4

$$c_1 = (\neg p_1 \vee p_2)$$
$$c_2 = (\neg p_1 \vee p_3 \vee p_5)$$
$$c_3 = (\neg p_2 \vee p_4)$$
$$c_4 = (\neg p_3 \vee \neg p_4)$$
$$c_5 = (p_1 \vee p_5 \vee \neg p_2)$$
$$c_6 = (p_2 \vee p_3)$$
$$c_7 = (p_2 \vee \neg p_3 \vee p_7)$$
$$c_8 = (p_6 \vee \neg p_5)$$



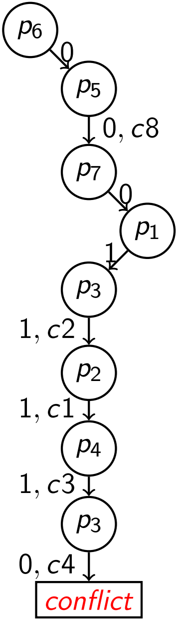
Implication graph



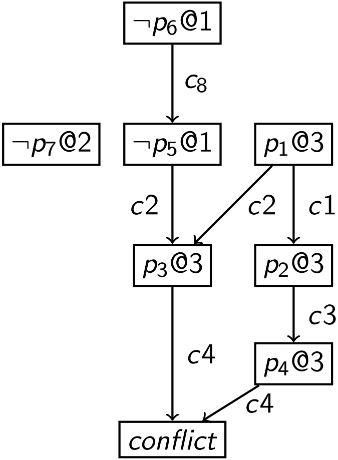
Implication graph for conflict graph

Example 1.4

$$
\begin{aligned}
c_1 &= (\neg p_1 \lor p_2)\\
c_2 &= (\neg p_1 \lor p_3 \lor p_5)\\
c_3 &= (\neg p_2 \lor p_4)\\
c_4 &= (\neg p_3 \lor \neg p_4)\\
c_5 &= (p_1 \lor p_5 \lor \neg p_2)\\
c_6 &= (p_2 \lor p_3)\\
c_7 &= (p_2 \lor \neg p_3 \lor p_7)\\
c_8 &= (p_6 \lor \neg p_5)
\end{aligned}
$$



Implication graph

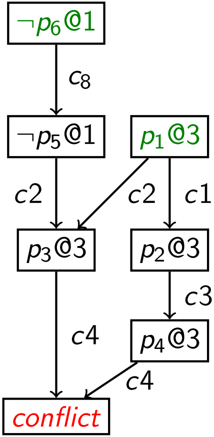


Reading proofs from implication graphs

- For each learned clause we assign a resolution proof that proves that the learned clause is implied by the clauses in the solver so far.

We demonstrate the process using an example.

Example 1.5



Input clauses:

$c_8 = (p_6 \vee \neg p_5)$ $\quad c_2 = (\neg p_1 \vee p_3 \vee p_5)$

$c_1 = (\neg p_1 \vee p_2)$ $\quad c_3 = (\neg p_2 \vee p_4)$ $\quad c_4 = (\neg p_3 \vee \neg p_4)$

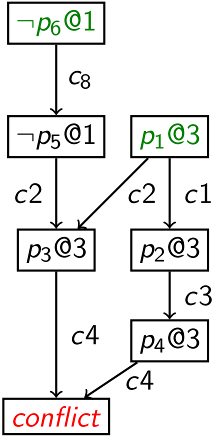
Conflict clause : $p_6 \vee \neg p_1$

Reading proofs from implication graphs

- For each learned clause we assign a resolution proof that proves that the learned clause is implied by the clauses in the solver so far.

We demonstrate the process using an example.

Example 1.5



Input clauses:

$c_8 = (p_6 \vee \neg p_5)$ $\quad c_2 = (\neg p_1 \vee p_3 \vee p_5)$

$c_1 = (\neg p_1 \vee p_2)$ $\quad c_3 = (\neg p_2 \vee p_4)$ $\quad c_4 = (\neg p_3 \vee \neg p_4)$

Conflict clause : $p_6 \vee \neg p_1$

Conflict as a resolution proof:

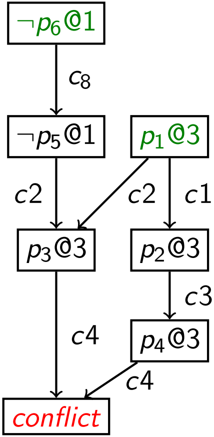
$$\frac{p_6 \vee \neg p_5 \qquad \neg p_6}{\neg p_5}$$

Reading proofs from implication graphs

- For each learned clause we assign a resolution proof that proves that the learned clause is implied by the clauses in the solver so far.

We demonstrate the process using an example.

Example 1.5



Input clauses:

$c_8 = (p_6 \vee \neg p_5)$ $\quad c_2 = (\neg p_1 \vee p_3 \vee p_5)$

$c_1 = (\neg p_1 \vee p_2)$ $\quad c_3 = (\neg p_2 \vee p_4)$ $\quad c_4 = (\neg p_3 \vee \neg p_4)$

Conflict clause : $p_6 \vee \neg p_1$

Conflict as a resolution proof:

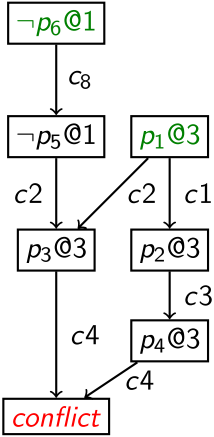
$$\dfrac{\dfrac{\neg p_1 \vee p_3 \vee p_5 \qquad \dfrac{p_6 \vee \neg p_5 \qquad \neg p_6}{\neg p_5}}{\neg p_1 \vee p_3} \qquad p_1}{p_3}$$

Reading proofs from implication graphs

- For each learned clause we assign a resolution proof that proves that the learned clause is implied by the clauses in the solver so far.

We demonstrate the process using an example.

Example 1.5



Input clauses:

$c_8 = (p_6 \vee \neg p_5)$ $\qquad c_2 = (\neg p_1 \vee p_3 \vee p_5)$

$c_1 = (\neg p_1 \vee p_2)$ $\qquad c_3 = (\neg p_2 \vee p_4)$ $\quad c_4 = (\neg p_3 \vee \neg p_4)$

Conflict clause : $p_6 \vee \neg p_1$

Conflict as a resolution proof:

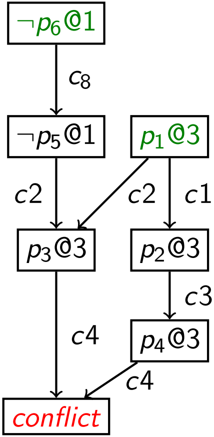
$$\dfrac{\dfrac{\neg p_1 \vee p_3 \vee p_5 \qquad \dfrac{p_6 \vee \neg p_5 \qquad \neg p_6}{\neg p_5}}{\neg p_1 \vee p_3} \qquad p_1}{p_3} \qquad\qquad \dfrac{\neg p_1 \vee p_2 \qquad p_1}{p_2}$$

Reading proofs from implication graphs

- For each learned clause we assign a resolution proof that proves that the learned clause is implied by the clauses in the solver so far.

We demonstrate the process using an example.

Example 1.5



Input clauses:

$c_8 = (p_6 \vee \neg p_5)$ $\quad c_2 = (\neg p_1 \vee p_3 \vee p_5)$

$c_1 = (\neg p_1 \vee p_2)$ $\quad c_3 = (\neg p_2 \vee p_4)$ $\quad c_4 = (\neg p_3 \vee \neg p_4)$

Conflict clause : $p_6 \vee \neg p_1$

Conflict as a resolution proof:

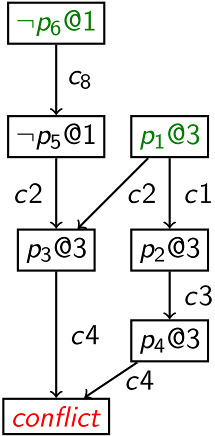
$$\dfrac{\dfrac{\neg p_1 \vee p_3 \vee p_5 \qquad \dfrac{p_6 \vee \neg p_5 \qquad \neg p_6}{\neg p_5}}{\neg p_1 \vee p_3} \qquad p_1}{p_3} \qquad\qquad \dfrac{\neg p_2 \vee p_4 \qquad \dfrac{\neg p_1 \vee p_2 \qquad p_1}{p_2}}{p_4}$$

Reading proofs from implication graphs

- For each learned clause we assign a resolution proof that proves that the learned clause is implied by the clauses in the solver so far.

We demonstrate the process using an example.

Example 1.5



Input clauses:

$c_8 = (p_6 \vee \neg p_5) \qquad c_2 = (\neg p_1 \vee p_3 \vee p_5)$

$c_1 = (\neg p_1 \vee p_2) \qquad c_3 = (\neg p_2 \vee p_4) \quad c_4 = (\neg p_3 \vee \neg p_4)$

Conflict clause : $p_6 \vee \neg p_1$

Conflict as a resolution proof:

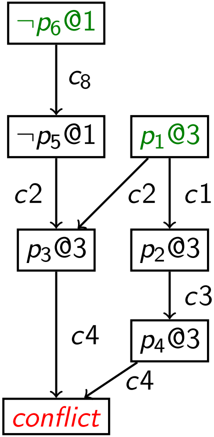
$$\dfrac{\dfrac{\neg p_1 \vee p_3 \vee p_5 \qquad \dfrac{p_6 \vee \neg p_5 \qquad \neg p_6}{\neg p_5}}{\neg p_1 \vee p_3} \qquad p_1}{p_3} \qquad \dfrac{\neg p_3 \vee \neg p_4 \qquad \dfrac{\neg p_2 \vee p_4 \qquad \dfrac{\neg p_1 \vee p_2 \qquad p_1}{p_2}}{p_4}}{\neg p_3}$$

Reading proofs from implication graphs

- For each learned clause we assign a resolution proof that proves that the learned clause is implied by the clauses in the solver so far.

We demonstrate the process using an example.

Example 1.5



Input clauses:

$c_8 = (p_6 \lor \neg p_5)$ $\quad$ $c_2 = (\neg p_1 \lor p_3 \lor p_5)$

$c_1 = (\neg p_1 \lor p_2)$ $\quad$ $c_3 = (\neg p_2 \lor p_4)$ $\quad$ $c_4 = (\neg p_3 \lor \neg p_4)$

Conflict clause : $p_6 \lor \neg p_1$

Conflict as a resolution proof:

$$\dfrac{\dfrac{\dfrac{\neg p_1 \lor p_3 \lor p_5 \qquad \dfrac{p_6 \lor \neg p_5 \qquad \neg p_6}{\neg p_5}}{\neg p_1 \lor p_3} \qquad p_1}{p_3} \qquad \dfrac{\neg p_3 \lor \neg p_4 \qquad \dfrac{\neg p_2 \lor p_4 \qquad \dfrac{\neg p_1 \lor p_2 \qquad p_1}{p_2}}{p_4}}{\neg p_3}}{\bot}$$

Resolution proofs for conflict clauses

Example 1.6 (contd.)

$$\dfrac{\dfrac{\dfrac{\neg p_1 \vee p_3 \vee p_5 \qquad \dfrac{p_6 \vee \neg p_5 \quad \neg p_6}{\neg p_5}}{\neg p_1 \vee p_3} \qquad p_1}{p_3} \qquad \dfrac{\neg p_3 \vee \neg p_4 \qquad \dfrac{\neg p_2 \vee p_4 \qquad \dfrac{\neg p_1 \vee p_2 \quad p_1}{p_2}}{p_4}}{\neg p_3}}{\bot}$$

Resolution proofs for conflict clauses

Example 1.6 (contd.)

$$\dfrac{\dfrac{\dfrac{\neg p_1 \vee p_3 \vee p_5 \qquad \dfrac{p_6 \vee \neg p_5}{\neg p_5}}{\neg p_1 \vee p_3}}{p_3} \qquad \dfrac{\neg p_3 \vee \neg p_4 \qquad \dfrac{\neg p_2 \vee p_4 \qquad \dfrac{\neg p_1 \vee p_2}{p_2}}{p_4}}{\neg p_3}}{\bot}$$

Resolution proofs for conflict clauses

Example 1.6 (contd.)

$$\dfrac{\dfrac{\dfrac{\neg p_1 \vee p_3 \vee p_5 \qquad \dfrac{p_6 \vee \neg p_5}{p_6 \vee \neg p_5}}{p_6 \vee \neg p_1 \vee p_3}}{p_6 \vee \neg p_1 \vee p_3} \qquad \dfrac{\neg p_3 \vee \neg p_4 \qquad \dfrac{\neg p_2 \vee p_4 \qquad \dfrac{\neg p_1 \vee p_2}{\neg p_1 \vee p_2}}{\neg p_1 \vee p_4}}{\neg p_1 \vee \neg p_3}}{p_6 \vee \neg p_1 \vee \bot}$$

The above is a resolution proof of the conflict clause.

Resolution proofs for conflict clauses

Example 1.6 (contd.)

$$\dfrac{\dfrac{\dfrac{\neg p_1 \lor p_3 \lor p_5 \qquad \dfrac{p_6 \lor \neg p_5}{p_6 \lor \neg p_5}}{p_6 \lor \neg p_1 \lor p_3}}{p_6 \lor \neg p_1 \lor p_3} \qquad \dfrac{\neg p_3 \lor \neg p_4 \qquad \dfrac{\neg p_2 \lor p_4 \qquad \dfrac{\neg p_1 \lor p_2}{\neg p_1 \lor p_2}}{\neg p_1 \lor p_4}}{\neg p_1 \lor \neg p_3}}{p_6 \lor \neg p_1 \lor \bot}$$

The above is a resolution proof of the conflict clause.

One more issue:

There may be a leaf of the above proof that is a conflict clause in itself.

- In the case, there must be a resolution proof for the conflict clause.
- We “stitch” that proof with the above proof .

CDCL($\mathcal{T}$) with proof generation

Algorithm 1.2: CDCL($\mathcal{T}$)

Input: CNF F, boolean encoder e

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

do

 // backtracking

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

  $(C, dl, proof) := \textsc{AnalyzeConflict}(m, proofs)$; $proofs(C) := proof$;

  if $C = \emptyset$ **then return** *unsat(proof)*;

  $m.resize(dstack(dl))$; $\textsc{AddClauses}(\{C\})$; 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**

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

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

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

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

return *sat*

CDCL($\mathcal{T}$) with proof generation

Algorithm 1.2: CDCL($\mathcal{T}$)

Input: CNF F, boolean encoder e

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

do

// backtracking

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

$(C, dl, proof) := \textsc{AnalyzeConflict}(m,proofs)$; $proofs(C) := proof$;

if $C = \emptyset$ **then return** *unsat(proof)*;

$m.resize(dstack(dl))$; $\textsc{AddClauses}(\{C\})$; 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**

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

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

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

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

return *sat*

We also need proofs of Cs from the theory solvers

Topic 1.3

Proofs from theory solvers

Theory solvers

Each theory needs to have its own proof rules and instrumentation of the employed decision procedure to obtain proofs.

Theory solvers

Each theory needs to have its own proof rules and instrumentation of the employed decision procedure to obtain proofs.

Here, we will look at two examples

- Theory of linear rational arithmetic ($\mathcal{T}_{LRA}$)
- Theory of equality with uninterpreted functions($\mathcal{T}_{EUF}$)

Proof generation in $\mathcal{T}_{LRA}$

In the theory of LRA, atoms are linear constraints over rational variables.

Proof generation in $\mathcal{T}_{LRA}$

In the theory of LRA, atoms are linear constraints over rational variables.

The following is the only proof rule for the theory.

$$\frac{a_1 x \leq b_1 \quad a_1 x \leq b_1}{(\lambda_1 a_1 + \lambda_2 a_2)x \leq (\lambda_1 b_1 + \lambda_2 b_2)} \lambda_1, \lambda_2 \geq 0$$

Proof generation in $\mathcal{T}_{LRA}$

In the theory of LRA, atoms are linear constraints over rational variables.

The following is the only proof rule for the theory.

$$\frac{a_1 x \leq b_1 \quad a_1 x \leq b_1}{(\lambda_1 a_1 + \lambda_2 a_2)x \leq (\lambda_1 b_1 + \lambda_2 b_2)} \lambda_1, \lambda_2 \geq 0$$

Example 1.7

Consider: $3x_1 \leq -6 \wedge x_1 - 3x_2 \leq 1 \wedge x_1 + x_2 \leq 2$

Proof generation in $\mathcal{T}_{LRA}$

In the theory of LRA, atoms are linear constraints over rational variables.

The following is the only proof rule for the theory.

$$\frac{a_1 x \leq b_1 \quad a_1 x \leq b_1}{(\lambda_1 a_1 + \lambda_2 a_2)x \leq (\lambda_1 b_1 + \lambda_2 b_2)} \lambda_1, \lambda_2 \geq 0$$

Example 1.7

Consider: $3x_1 \leq -6 \wedge x_1 - 3x_2 \leq 1 \wedge x_1 + x_2 \leq 2$

$$\frac{x_1 - 3x_2 \leq 1 \quad x_1 + x_2 \leq 2}{4x_1 \leq 7} \lambda_1 = 1, \lambda_2 = 3$$

Proof generation in $\mathcal{T}_{LRA}$

In the theory of LRA, atoms are linear constraints over rational variables.

The following is the only proof rule for the theory.

$$\frac{a_1 x \leq b_1 \quad a_1 x \leq b_1}{(\lambda_1 a_1 + \lambda_2 a_2)x \leq (\lambda_1 b_1 + \lambda_2 b_2)} \lambda_1, \lambda_2 \geq 0$$

Example 1.7

Consider: $3x_1 \leq -6 \wedge x_1 - 3x_2 \leq 1 \wedge x_1 + x_2 \leq 2$

$$\frac{3x_1 \leq -6 \qquad \dfrac{x_1 - 3x_2 \leq 1 \quad x_1 + x_2 \leq 2}{4x_1 \leq 7} \lambda_1 = 1, \lambda_2 = 3}{0 \leq -1} \lambda_1 = 4/3, \lambda_2 = 1$$

LRA solver

There are many decision procedures for solving LRA.

LRA solver

There are many decision procedures for solving LRA.

We will present Fourier-Motzkin algorithm for solving LRA.

LRA solver

There are many decision procedures for solving LRA.

We will present Fourier-Motzkin algorithm for solving LRA.

The algorithm is not the most efficient one, but it will help us illustrate the proof extraction.

Fourier-Motzkin

Algorithm proceeds by eliminating variables one by one. After eliminating all the variables, if input reduces to $\top$ then only the input is sat. Otherwise, unsat.

Fourier-Motzkin

Algorithm proceeds by eliminating variables one by one. After eliminating all the variables, if input reduces to $\top$ then only the input is sat. Otherwise, unsat.

For each variable x, any conjunction of linear inequalities can be converted to the following form.

$$\bigwedge_{j=1}^{m} s_j \leq x \wedge \bigwedge_{i=1}^{l} x \leq t_i \wedge \bigwedge_{k=1}^{n} 0 \leq u_k$$

Fourier-Motzkin

Algorithm proceeds by eliminating variables one by one. After eliminating all the variables, if input reduces to $\top$ then only the input is sat. Otherwise, unsat.

For each variable x, any conjunction of linear inequalities can be converted to the following form.

$$\bigwedge_{j=1}^{m} s_j \leq x \wedge \bigwedge_{i=1}^{l} x \leq t_i \wedge \bigwedge_{k=1}^{n} 0 \leq u_k$$

$$\updownarrow$$

$$\bigwedge_{j=1}^{m} \bigwedge_{i=1}^{l} s_j \leq t_i \wedge \bigwedge_{k=1}^{n} 0 \leq u_k$$

Fourier-Motzkin

Algorithm proceeds by eliminating variables one by one. After eliminating all the variables, if input reduces to $\top$ then only the input is sat. Otherwise, unsat.

For each variable x, any conjunction of linear inequalities can be converted to the following form.

$$\bigwedge_{j=1}^{m} s_j \leq x \wedge \bigwedge_{i=1}^{l} x \leq t_i \wedge \bigwedge_{k=1}^{n} 0 \leq u_k$$

$$\updownarrow$$

$$\bigwedge_{j=1}^{m} \bigwedge_{i=1}^{l} s_j \leq t_i \wedge \bigwedge_{k=1}^{n} 0 \leq u_k$$

equisatisfiable but without x

Fourier-Motzkin

Algorithm proceeds by eliminating variables one by one. After eliminating all the variables, if input reduces to $\top$ then only the input is sat. Otherwise, unsat.

For each variable x, any conjunction of linear inequalities can be converted to the following form.

$$\bigwedge_{j=1}^{m} s_j \leq x \wedge \bigwedge_{i=1}^{l} x \leq t_i \wedge \bigwedge_{k=1}^{n} 0 \leq u_k$$

$$\updownarrow$$

$$\bigwedge_{j=1}^{m} \bigwedge_{i=1}^{l} s_j \leq t_i \wedge \bigwedge_{k=1}^{n} 0 \leq u_k$$

equisatisfiable but without x

Exercise 1.1

a. Add support for equality, dis-equality, and strict inequalities.

b. What is the complexity?

Example: Fourier-Motzkin

Consider: $-x_1 + x_2 + 2x_3 \leq 0 \wedge x_1 - x_2 \leq 0 \wedge x_1 - x_3 \leq 0 \wedge 1 \leq x_3$

Example: Fourier-Motzkin

Consider: $-x_1 + x_2 + 2x_3 \leq 0 \wedge x_1 - x_2 \leq 0 \wedge x_1 - x_3 \leq 0 \wedge 1 \leq x_3$

Suppose we eliminate x_1 first. We transform the constraints in our format.

Example: Fourier-Motzkin

Consider: $-x_1 + x_2 + 2x_3 \leq 0 \wedge x_1 - x_2 \leq 0 \wedge x_1 - x_3 \leq 0 \wedge 1 \leq x_3$

Suppose we eliminate x_1 first. We transform the constraints in our format.

$\underbrace{x_2 + 2x_3 \leq x_1}_{\text{lower bounding}}$

Example: Fourier-Motzkin

Consider: $-x_1 + x_2 + 2x_3 \leq 0 \wedge x_1 - x_2 \leq 0 \wedge x_1 - x_3 \leq 0 \wedge 1 \leq x_3$

Suppose we eliminate x_1 first. We transform the constraints in our format.

$\underbrace{x_2 + 2x_3 \leq x_1}_{\text{lower bounding}} \wedge \underbrace{(x_1 \leq x_2 \wedge x_1 \leq x_3)}_{\text{upper bounding}}$

Example: Fourier-Motzkin

Consider: $-x_1 + x_2 + 2x_3 \leq 0 \wedge x_1 - x_2 \leq 0 \wedge x_1 - x_3 \leq 0 \wedge 1 \leq x_3$

Suppose we eliminate x_1 first. We transform the constraints in our format.

$$\underbrace{x_2 + 2x_3 \leq x_1}_{\text{lower bounding}} \wedge \underbrace{(x_1 \leq x_2 \wedge x_1 \leq x_3)}_{\text{upper bounding}} \wedge \underbrace{1 \leq x_3}_{x_1 \text{ does not occur}}$$

Example: Fourier-Motzkin

Consider: $-x_1 + x_2 + 2x_3 \leq 0 \wedge x_1 - x_2 \leq 0 \wedge x_1 - x_3 \leq 0 \wedge 1 \leq x_3$

Suppose we eliminate x_1 first. We transform the constraints in our format.

$\underbrace{x_2 + 2x_3 \leq x_1}_{\text{lower bounding}} \wedge \underbrace{(x_1 \leq x_2 \wedge x_1 \leq x_3)}_{\text{upper bounding}} \wedge \underbrace{1 \leq x_3}_{x_1 \text{ does not occur}}$

Eliminated constraints:

$x_2 + 2x_3 \leq x_2 \wedge x_2 + 2x_3 \leq x_3 \wedge 1 \leq x_3$

Example: Fourier-Motzkin

Consider: $-x_1 + x_2 + 2x_3 \leq 0 \wedge x_1 - x_2 \leq 0 \wedge x_1 - x_3 \leq 0 \wedge 1 \leq x_3$

Suppose we eliminate x_1 first. We transform the constraints in our format.

$\underbrace{x_2 + 2x_3 \leq x_1}_{\text{lower bounding}} \wedge \underbrace{(x_1 \leq x_2 \wedge x_1 \leq x_3)}_{\text{upper bounding}} \wedge \underbrace{1 \leq x_3}_{x_1 \text{ does not occur}}$

Eliminated constraints:

$x_2 + 2x_3 \leq x_2 \wedge x_2 + 2x_3 \leq x_3 \wedge 1 \leq x_3$

After simplification:

$x_3 \leq 0 \wedge x_2 + x_3 \leq 0 \wedge 1 \leq x_3$

Example: Fourier-Motzkin

Consider: $-x_1 + x_2 + 2x_3 \leq 0 \wedge x_1 - x_2 \leq 0 \wedge x_1 - x_3 \leq 0 \wedge 1 \leq x_3$

Suppose we eliminate x_1 first. We transform the constraints in our format.

$$\underbrace{x_2 + 2x_3 \leq x_1}_{\text{lower bounding}} \wedge \underbrace{(x_1 \leq x_2 \wedge x_1 \leq x_3)}_{\text{upper bounding}} \wedge \underbrace{1 \leq x_3}_{x_1 \text{ does not occur}}$$

Eliminated constraints:

$x_2 + 2x_3 \leq x_2 \wedge x_2 + 2x_3 \leq x_3 \wedge 1 \leq x_3$

After simplification:

$x_3 \leq 0 \wedge x_2 + x_3 \leq 0 \wedge 1 \leq x_3$

Since x_2 is bounded only from one side, we can trivially eliminate x_2

$x_3 \leq 0 \wedge 1 \leq x_3$

Example: Fourier-Motzkin

Consider: $-x_1 + x_2 + 2x_3 \leq 0 \wedge x_1 - x_2 \leq 0 \wedge x_1 - x_3 \leq 0 \wedge 1 \leq x_3$

Suppose we eliminate x_1 first. We transform the constraints in our format.

$\underbrace{x_2 + 2x_3 \leq x_1}_{\text{lower bounding}} \wedge \underbrace{(x_1 \leq x_2 \wedge x_1 \leq x_3)}_{\text{upper bounding}} \wedge \underbrace{1 \leq x_3}_{x_1 \text{ does not occur}}$

Eliminated constraints:

$x_2 + 2x_3 \leq x_2 \wedge x_2 + 2x_3 \leq x_3 \wedge 1 \leq x_3$

After simplification:

$x_3 \leq 0 \wedge x_2 + x_3 \leq 0 \wedge 1 \leq x_3$

Since x_2 is bounded only from one side, we can trivially eliminate x_2

$x_3 \leq 0 \wedge 1 \leq x_3$

Eliminating x_3

$1 \leq 0$

Example: Fourier-Motzkin

Consider: $-x_1 + x_2 + 2x_3 \leq 0 \wedge x_1 - x_2 \leq 0 \wedge x_1 - x_3 \leq 0 \wedge 1 \leq x_3$

Suppose we eliminate x_1 first. We transform the constraints in our format.

$$\underbrace{x_2 + 2x_3 \leq x_1}_{\text{lower bounding}} \wedge \underbrace{(x_1 \leq x_2 \wedge x_1 \leq x_3)}_{\text{upper bounding}} \wedge \underbrace{1 \leq x_3}_{x_1 \text{ does not occur}}$$

Eliminated constraints:

$x_2 + 2x_3 \leq x_2 \wedge x_2 + 2x_3 \leq x_3 \wedge 1 \leq x_3$

After simplification:

$x_3 \leq 0 \wedge x_2 + x_3 \leq 0 \wedge 1 \leq x_3$

Since x_2 is bounded only from one side, we can trivially eliminate x_2

$x_3 \leq 0 \wedge 1 \leq x_3$

Eliminating x_3

$1 \leq 0 \leftarrow$ false formula therefore unsat

Proof generation from Fourier-Motzkin

Observation:

- Fourier-Motzkin proceeds by replacing inequalities by other inequalities

Proof generation from Fourier-Motzkin

Observation:

- Fourier-Motzkin proceeds by replacing inequalities by other inequalities
- incoming inequalities are positive linear combination of old inequalities

Proof generation from Fourier-Motzkin

Observation:

- Fourier-Motzkin proceeds by replacing inequalities by other inequalities
- incoming inequalities are positive linear combination of old inequalities
- We may instrument Fourier-Motzkin to keep the record and produce proof if input is found to be unsat

Proof generation from Fourier-Motzkin

Observation:

- Fourier-Motzkin proceeds by replacing inequalities by other inequalities
- incoming inequalities are positive linear combination of old inequalities
- We may instrument Fourier-Motzkin to keep the record and produce proof if input is found to be unsat

Example 1.8

In previous example,

$$\frac{-x_1 + x_2 + 2x_3 \leq 0 \quad x_1 - x_3 \leq 0}{x_2 + x_3 \leq 0} \quad \frac{-x_1 + x_2 + 2x_3 \leq 0 \quad x_1 - x_2 \leq 0}{x_3 \leq 0}$$

Proof generation from Fourier-Motzkin

Observation:

- Fourier-Motzkin proceeds by replacing inequalities by other inequalities
- incoming inequalities are positive linear combination of old inequalities
- We may instrument Fourier-Motzkin to keep the record and produce proof if input is found to be unsat

Example 1.8

In previous example,

$$\dfrac{-x_1 + x_2 + 2x_3 \leq 0 \quad x_1 - x_3 \leq 0}{x_2 + x_3 \leq 0} \quad \dfrac{-x_1 + x_2 + 2x_3 \leq 0 \quad x_1 - x_2 \leq 0}{x_3 \leq 0} \; -x_3 \leq -1$$

Proof generation from Fourier-Motzkin

Observation:

- Fourier-Motzkin proceeds by replacing inequalities by other inequalities
- incoming inequalities are positive linear combination of old inequalities
- We may instrument Fourier-Motzkin to keep the record and produce proof if input is found to be unsat

Example 1.8

In previous example,

$$\frac{-x_1 + x_2 + 2x_3 \leq 0 \quad x_1 - x_3 \leq 0}{x_2 + x_3 \leq 0} \qquad \frac{\dfrac{-x_1 + x_2 + 2x_3 \leq 0 \quad x_1 - x_2 \leq 0}{x_3 \leq 0} \quad -x_3 \leq -1}{0 \leq -1}$$

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 1.9

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

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

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

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

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

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

Example 1.9

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

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

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 1.9

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

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

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 1.9

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

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

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

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

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

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

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

Example 1.9

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

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

DP_{EUF}

General idea: maintain equivalence classes among terms

DP_{EUF}

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}

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}

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}

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}

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}

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}

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

DP_{EUF}

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**; } ;

  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**

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

else

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

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

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

DP_{EUF}

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**; } ;
    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**
        $DP_{EUF}.push(f(r_1, \dots, r_n) = f(s_1, \dots, s_n))$;
else
    // $\bowtie$ = "$\not\approx$"
    $DisEq := DisEq \cup (C_1, C_2)$;
    if $C_1 = C_2$ **then** $conflictFound := 1$; **return** ;

Exercise 1.2

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

DP_{EUF}

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**; } ;
    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**
        $DP_{EUF}.push(f(r_1, \dots, r_n) = f(s_1, \dots, s_n))$;
else
    // $\bowtie$ = "$\not\approx$"
    $DisEq := DisEq \cup (C_1, C_2)$;
    if $C_1 = C_2$ **then** $conflictFound := 1$; **return** ;

Exercise 1.2

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

b. Give a fix in push such that it works on the above example.

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 1.3

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:

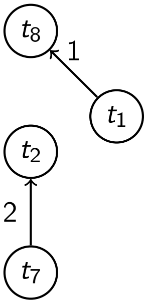
$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$

$$t_1 \xrightarrow{1} t_8$$

Example: union-find

Consider:

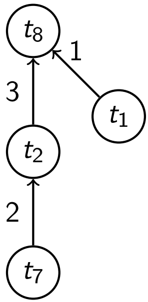
$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



Example: union-find

Consider:

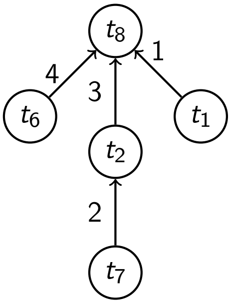
$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



Example: union-find

Consider:

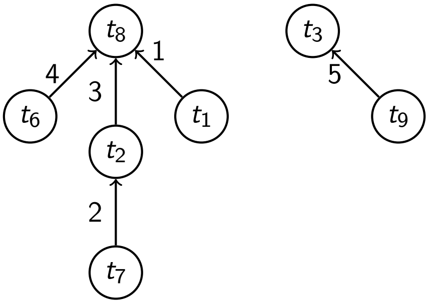
$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



Example: union-find

Consider:

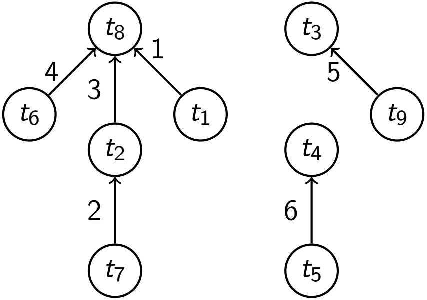
$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



Example: union-find

Consider:

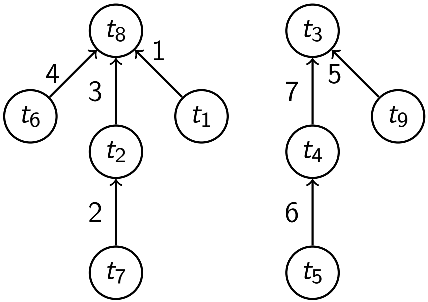
$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



Example: union-find

Consider:

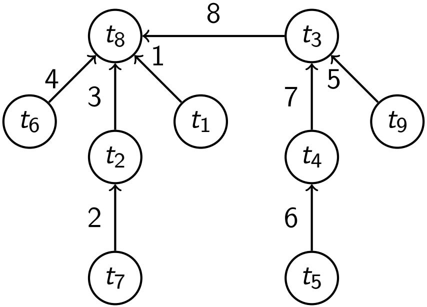
$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



Example: union-find

Consider:

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



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 $s = v$ in the path between t and u.

   t - - - - - - s ——— v - - - - - - u

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

   $$t \text{ - - - - - - } s \text{ ——— } v \text{ - - - - - - } u$$

   2.1 Recursively, find the proof of $t = s$ and $u = v$.
   2.2 Construct a proof for $s = v$, which is either in input or due to congruence. In the later case, let us suppose $s = f(s_1, .., s_n) = f(v_1, .., v_n) = v$. Then, we also recursively search for proofs of $s_i = v_i$.

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 $s = v$ in the path between t and u.

   $$t \text{ - - - - - - } s \text{ ——— } v \text{ - - - - - - } u$$

   2.1 Recursively, find the proof of $t = s$ and $u = v$.
   2.2 Construct a proof for $s = v$, which is either in input or due to congruence. In the later case, let us suppose $s = f(s_1, .., s_n) = f(v_1, .., v_n) = v$. Then, we also recursively search for proofs of $s_i = v_i$.

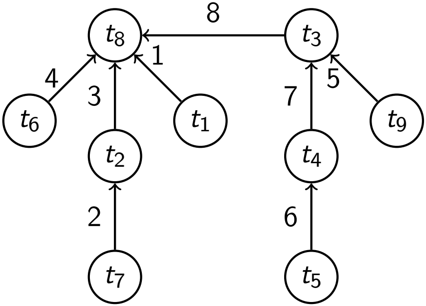
   Stitch the proofs appropriately.

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

Example: union-find proof generation

Consider:

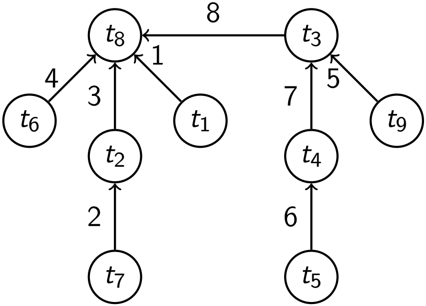
$$\underbrace{t_1 = t_8}_{1} \wedge \underbrace{t_7 = t_2}_{2} \wedge \underbrace{t_7 = t_1}_{3} \wedge \underbrace{t_6 = t_7}_{4} \wedge \underbrace{t_9 = t_3}_{5} \wedge \underbrace{t_5 = t_4}_{6} \wedge \underbrace{t_4 = t_3}_{7} \wedge \underbrace{t_5 = t_7}_{8} \wedge \underbrace{t_1 \neq t_4}_{9}$$



Example: union-find proof generation

Consider:

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



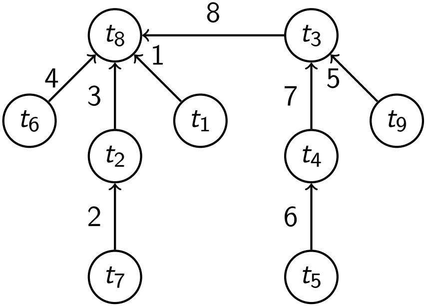
1. $t_1 \neq t_4$ is violated.

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

Example: union-find proof generation

Consider:

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



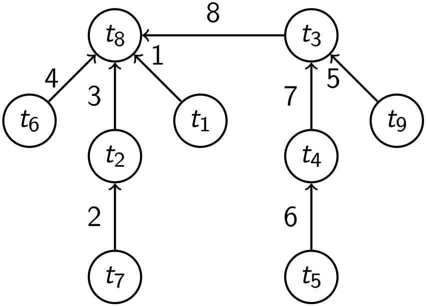
1. $t_1 \neq t_4$ is violated.
2. 8 is the latest edge in the path between t_1 and t_4

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

Example: union-find proof generation

Consider:

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



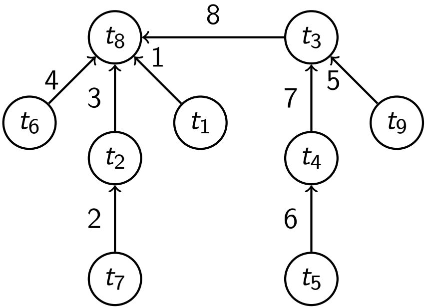
1. $t_1 \neq t_4$ is violated.
2. 8 is the latest edge in the path between t_1 and t_4
3. 8 corresponds to $t_5 = t_7$

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

Example: union-find proof generation

Consider:

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



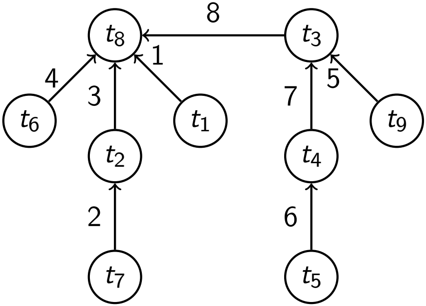
1. $t_1 \neq t_4$ is violated.
2. 8 is the latest edge in the path between t_1 and t_4
3. 8 corresponds to $t_5 = t_7$
4. Now look for proof of $t_1 = t_7$ and $t_4 = t_5$

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

Example: union-find proof generation

Consider:

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



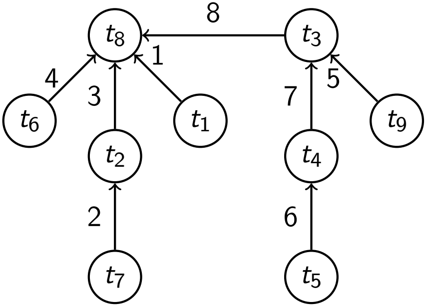
$$\dfrac{t_1 \neq t_4 \qquad \dfrac{\dfrac{\dfrac{t_7 = t_1}{t_1 = t_7} \qquad \dfrac{t_5 = t_7}{t_7 = t_5}}{}}{t_1 = t_4}}{\perp}$$

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 corresponds to $t_5 = t_7$
4. Now look for proof of $t_1 = t_7$ and $t_4 = t_5$
5. 3 is the latest edge between t_1 and t_7, which corresponds to $t_1 = t_7$.

Example: union-find proof generation

Consider:

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



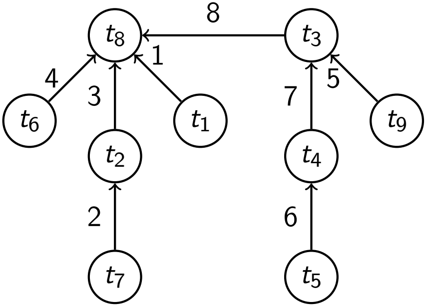
$$\dfrac{t_1 \neq t_4 \qquad \dfrac{\dfrac{t_7 = t_1}{t_1 = t_7} \quad \dfrac{t_5 = t_7}{t_7 = t_5} \quad t_5 = t_4}{t_1 = t_4}}{\bot}$$

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 corresponds to $t_5 = t_7$
4. Now look for proof of $t_1 = t_7$ and $t_4 = t_5$
5. 3 is the latest edge between t_1 and t_7, which corresponds to $t_1 = t_7$.
6. Similarly, $t_4 = t_5$ is edge 6

Example: union-find proof generation

Consider:

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



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

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 corresponds to $t_5 = t_7$
4. Now look for proof of $t_1 = t_7$ and $t_4 = t_5$
5. 3 is the latest edge between t_1 and t_7, which corresponds to $t_1 = t_7$.
6. Similarly, $t_4 = t_5$ is edge 6

End of Lecture 1