Verification

Lecture 23

Bernd Finkbeiner
Peter Faymonville
Michael Gerke

UNIVERSITÄT
DES
SAARLANDES

REVIEW: Decidability of first-order theories

Theory		full	QFF
T_E	Equality	no	yes
T_{PA}	Peano arithmetic	no	no
$T_{\mathbb{N}}$	Presburger arithmetic	yes	yes
$T_{\mathbb{Z}}$	integers	yes	yes
$T_{\mathbb{R}}$	reals	yes	yes
$T_{\mathbb{Q}}$	rationals	yes	yes
T_{cons}	lists	no	yes
T_{A}	arrays	no	yes
$T_{\mathsf{A}}^{=}$	arrays with extensionality	no	yes

REVIEW: Quantifier Elimination (QE)

Algorithm for elimination of all quantifiers of formula F until quantifier-free formula G that is equivalent to F remains

Note: Could be enough to require that F is equisatisfiable to F', that is F is satisfiable iff F' is satisfiable

A theory T admits quantifier elimination if there is an algorithm that given Σ-formula F returns a quantifier-free Σ-formula G that is T-equivalent to F.

REVIEW: $\widehat{T_{\mathbb{Z}}}$ admits QE (Cooper's method)

Algorithm: Given $\widehat{\Sigma_{\mathbb{Z}}}$-formula $\exists x.\ F[x]$, where F is quantifier-free, construct quantifier-free $\widehat{\Sigma_{\mathbb{Z}}}$-formula that is equivalent to $\exists x.\ F[x]$.

1. Put $F[x]$ into Negation Normal Form (NNF).
2. Normalize literals: $s < t$, $k|t$, or $\neg(k|t)$
3. Put x in $s < t$ on one side: $hx < t$ or $s < hx$
4. Replace hx with x' without a factor
5. Replace $F[x']$ by $\bigvee F[j]$ for finitely many j.

Decision Procedures for Quantifier-free Fragments

For theory T with signature Σ and axioms Σ-formulae of form

$$\forall x_1, \ldots, x_n.\ F[x_1, \ldots, x_n]$$

Decide if

$$F[x_1, \ldots, x_n] \text{ or } \exists x_1, \ldots, x_n.\ F[x_1, \ldots, x_n] \text{ is } T\text{-satisfiable}$$

[Decide if

$$F[x_1, \ldots, x_n] \text{ or } \forall x_1, \ldots, x_n.\ F[x_1, \ldots, x_n] \text{ is } T\text{-valid}$$]

where F is quantifier-free and $\text{free}(F) = \{x_1, \ldots, x_n\}$

Note: no quantifier alternations

We consider only conjunctive quantifier-free Σ-formulae, i.e., conjunctions of Σ-literals (Σ-atoms or negations of Σ-atoms).

For given arbitrary quantifier-free Σ-formula F, convert it into DNF Σ-formula

$$F_1 \lor \ldots \lor F_k$$

where each F_i conjunctive.

F is T-satisfiable iff at least one F_i is T-satisfiable.

Preliminary Concepts

Vector

variable n-vector

$$\bar{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

n-vector $\bar{a} \in \mathbb{Q}^n$

$$\bar{a} = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix}$$

transpose

$$\bar{a}^{\mathsf{T}} = \begin{bmatrix} a_1 & \cdots & a_n \end{bmatrix}$$

Matrix

$m \times n$-matrix

$A \in \mathbb{Q}^{m \times n}$

$$A = \begin{bmatrix} a_{11} \cdots a_{1n} \\ \vdots \\ a_{m1} \cdots a_{mn} \end{bmatrix}$$

transpose

$$A^{\mathsf{T}} = \begin{bmatrix} a_{11} \cdots a_{m1} \\ \vdots \\ a_{1n} \cdots a_{mn} \end{bmatrix}$$

column, row

$$\begin{bmatrix} a_{1j} \\ \vdots \\ a_{i1} \cdots a_{ij} \cdots a_{in} \\ \vdots \\ a_{mj} \end{bmatrix}$$

Multiplication

vector-vector

$$\bar{a}^\mathsf{T}\bar{b} = [a_1 \cdots a_n] \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} = \sum_{i=1}^{n} a_i b_i$$

matrix-vector

$$A\bar{x} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ & \vdots & \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} a_{1i} x_i \\ \vdots \\ \sum_{i=1}^{n} a_{mi} x_i \end{bmatrix}$$

matrix-matrix

$$\underset{A}{\begin{bmatrix} & & \\ & a_{ik} & \\ & & \end{bmatrix}} \underset{B}{\begin{bmatrix} & & \\ & b_{kj} & \\ & & \end{bmatrix}} = \underset{P}{\begin{bmatrix} & & \\ & p_{ij} & \\ & & \end{bmatrix}}$$

where $p_{ij} = \bar{a}_i \bar{b}_j = \begin{bmatrix} a_{i1} & \cdots & a_{in} \end{bmatrix} \begin{bmatrix} b_{1j} \\ \vdots \\ b_{nj} \end{bmatrix} = \sum_{k=1}^{n} a_{ik} b_{kj}$

Special Vectors and Matrices

$\overline{0}$ - vector (column) of 0s

$\overline{1}$ - vector of 1s

Thus $\overline{1}^{\mathsf{T}}\overline{x} = \sum_{i=1}^{n} x_i$

$$I = \begin{bmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{bmatrix}$$ identity matrix ($n \times n$)

Thus $IA = AI = A$

unit vector $e_i = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \leftarrow i$

Vector Space - set S of vectors closed under addition and scaling of vectors. That is,

$$\text{if } \overline{v}_1, \dots, \overline{v}_k \in S \quad \text{then} \quad \lambda_1\overline{v}_1 + \cdots + \lambda_k\overline{v}_k \in S$$

for $\lambda_1, \dots, \lambda_n \in \mathbb{R}$.

Linear Equation

$$F : A\overline{x} = \overline{b}$$

$m \times n$-matrix (A), variable n-vector ($\overline{x}$), m-vector ($\overline{b}$)

represents the $\Sigma_{\mathbb{Q}}$-formula

$$F : (a_{11}x_1 + \cdots + a_{1n}x_n = b_1) \land \cdots \land (a_{m1}x_1 + \cdots + a_{mn}x_n = b_m)$$

Gaussian Elimination

Find $\overline{x}$ s.t. $A\overline{x} = \overline{b}$ by elementary row operations

- Swap two rows.
- Multiply a row by a nonzero scalar.
- Add one row to another.

Example:

Solve

$$\begin{bmatrix} 3 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 1 \\ 2 \end{bmatrix}$$

Construct the augmented matrix

$$\left[\begin{array}{ccc|c} 3 & 1 & 2 & 6 \\ 1 & 0 & 1 & 1 \\ 2 & 2 & 1 & 2 \end{array}\right]$$

Apply the row operations as follows:

1. Add $-2\bar{a}_1 + 4\bar{a}_2$ to $\bar{a}_3$

$$\left[\begin{array}{ccc|c} 3 & 1 & 2 & 6 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & -6 \end{array}\right]$$

2. Add $-\overline{a}_1 + 2\overline{a}_2$ to $\overline{a}_2$

$$\left[\begin{array}{ccc|c} 3 & 1 & 2 & 6 \\ 0 & -1 & 1 & -3 \\ 0 & 0 & 1 & -6 \end{array}\right]$$

This augmented matrix is in triangular form.

Solving

$$x_3 = -6$$

$$-x_2 - x_3 = -3 \quad \Rightarrow \quad x_2 = -3$$

$$3x_1 + x_2 + 2x_3 = 6 \quad \Rightarrow \quad x_1 = 7$$

The solution is $\overline{x} = \begin{bmatrix} 7 & -3 & -6 \end{bmatrix}^T$

Inverse Matrix

A^{-1} is the inverse matrix of square matrix A if

$$AA^{-1} = A^{-1}A = I$$

Square matrix A is nonsingular (invertible) if its inverse A^{-1} exists.

How to compute A^{-1} of A?

$$[A \mid I] \xrightarrow[\text{row operations}]{\text{elementary}} [I \mid A^{-1}]$$

How to compute kth column of A^{-1}?
Solve $A\bar{y} = e_k$, i.e.

$$\left[\begin{array}{c|c} & 0 \\ & \vdots \\ A & 1 \\ & \vdots \\ & 0 \end{array}\right] \xrightarrow[\text{row operations}]{\text{elementary}} \begin{array}{c} \bar{y} = \ldots \\ (k\text{th column of } A^{-1}) \end{array}$$

Linear Inequality

$$G: \; A\overline{x} \leq b$$

represents the $\Sigma_{\mathbb{Q}}$-formula

$$G: \; (a_{11}x_1 + \cdots + a_{1n}x_n \leq b_1) \;\wedge\; \cdots \;\wedge\; (a_{m1}x_1 + \cdots + a_{mn}x_n \leq b_m)$$

The inequality describes a polyhedron in $\mathbb{R}^n$.

For $m \times n$-matrix A, m-vector b, variable n-vector $\overline{x}$ where $m \geq n$:

An n-vector $\overline{v}$ is a vertex of $A\overline{x} \leq b$ if there is nonsingular $n \times n$-submatrix A_0 and corresponding n-subvector b_0 s.t.

$$A_0\overline{v} = b_0$$

Optimization Problem

$$\begin{array}{llll} \textbf{max} & \bar{c}^\mathsf{T}\bar{x} & & \ldots\text{objective function} \\ \textbf{subject to} & & & \\ & A\bar{x} \le & \bar{b} & \ldots\text{constraints} \end{array}$$

Solution: vertex $\bar{v}^*$ satisfying $A\bar{x} \le \bar{b}$ and maximize $\bar{c}^\mathsf{T}\bar{x}$. That is,

$A\bar{v}^* \le \bar{b}$ and
$\bar{c}^\mathsf{T}\bar{v}^*$ is maximal: $\bar{c}^\mathsf{T}\bar{v}^* \ge \bar{c}^\mathsf{T}\bar{u}$ for all $\bar{u}$ satisfying $A\bar{u} \le \bar{b}$

- If $A\bar{x} \le \bar{b}$ is unsatisfiable $\Rightarrow$ maximum is $-\infty$
- It's possible that the maximum is unbounded
  $\Rightarrow$ maximum is ∞

Example: Consider optimization problem:

$$\textbf{max} \quad \underbrace{\begin{bmatrix} 1 & 1 & -1 & -1 \end{bmatrix}}_{\bar{c}^{\mathsf{T}}} \underbrace{\begin{bmatrix} x \\ y \\ z_1 \\ z_2 \end{bmatrix}}_{\bar{x}}$$

subject to

$$\underbrace{\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x \\ y \\ z_1 \\ z_2 \end{bmatrix}}_{\bar{x}} \leq \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 3 \\ 2 \\ 2 \end{bmatrix}}_{\bar{b}}$$

A is a 7×4-matrix, $\bar{b}$ is a 7-vector, and
$\bar{x}$ is a variable 4-vector representing the four variables $\{x, y, z_1, z_2\}$.

Example (cont):

The objective function is

$$(x - z_1) + (y - z_2) \ .$$

The constraints are equivalent to the $\Sigma_{\mathbb{Q}}$-formula

$$\begin{aligned} & x \geq 0 \ \wedge \ y \geq 0 \ \wedge \ z_1 \geq 0 \ \wedge \ z_2 \geq 0 \\ & \wedge \ x + y \leq 3 \ \wedge \ x - z_1 \leq 2 \ \wedge \ y - z_2 \leq 2 \end{aligned}$$

$\overline{v} = \begin{bmatrix} 2 & 1 & 0 & 0 \end{bmatrix}^{\mathsf{T}}$ is a vertex of the constraints. For the nonsingular submatrix A_0 (rows 3, 4, 5, 6 of A), we have

$$\underbrace{\begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & -1 & 0 \end{bmatrix}}_{A_0} \underbrace{\begin{bmatrix} 2 \\ 1 \\ 0 \\ 0 \end{bmatrix}}_{\overline{v}} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ 3 \\ 2 \end{bmatrix}}_{b_0}$$

Duality Theorem

For $A \in \mathbb{Z}^{m\times n}, \overline{b} \in \mathbb{Z}^m, \overline{c} \in \mathbb{Z}^n$,

$$\max\{\overline{c}^\mathsf{T}\overline{x} \mid A\overline{x} \le \overline{b}\} = \min\{\overline{y}^\mathsf{T}\overline{b} \mid \overline{y} \ge \overline{0} \ \wedge \ \overline{y}^\mathsf{T}A = \overline{c}^\mathsf{T}\}$$

if the constraints are satisfiable.

That is,

maximizing the function $c^\mathsf{T}\overline{x}$ over $A\overline{x} \le \overline{b}$
(the primal form of the optimization problem)

is equivalent to

minimizing the function $\overline{y}^\mathsf{T}\overline{b}$ over all the nonnegative $\overline{y}$
s.t. $\overline{y}^\mathsf{T}A = \overline{c}^\mathsf{T}$
(the dual form of the optimization problem)

Outline of Algorithm

Given $\Sigma_{\mathbb{Q}}$-formula

$$F: \; a_{11}x_1 + \cdots + a_{1n}x_n \leq b_1 \;\wedge\; \cdots \;\wedge\; a_{m1}x_1 + \cdots + a_{mn}x_n \leq b_m$$

or in matrix notation

$$F: \; A\overline{x} \leq \overline{b}$$

Note: • equations

$$a_{i1}x_1 + \ldots + a_{in}x_n = b_i$$

are allowed --- break into two inequalities

$$a_{i1}x_1 + \ldots + a_{in}x_n \leq b_i \;\wedge\; -a_{i1}x_1 - \ldots - a_{in}x_n \leq -b_i.$$

• Strict inequalities

$$a_{i1}x_1 + \cdots + a_{in}x_n < b_i \; .$$

excluded from our discussion - but can be added.

Outline of Algorithm (cont)

To determine the satisfiability of F,

Step 0: reformulate the satisfiability of F as an optimization problem

$$M_F : \max\{\overline{c}^{\mathsf{T}}\overline{x}' \mid A'\overline{x}' \leq \overline{b}'\}$$

s.t. F is $T_{\mathbb{Q}}$-satisfiable iff the optimal value of M_F is a particular value v_F (derived from the structure of F)

Step 1, Step 2, . . . (until termination) execute the simplex method

Outline of Algorithm (cont)

The simplex method traverses the vertices of $A'\overline{x}' \leq \overline{b}'$ searching for the maximum of the objective function $\overline{c}^{\mathsf{T}}\overline{x}'$:
if $\overline{v}_1, \overline{v}_2, \ldots$ are the traversed vertices in Step 1, Step 2, . . ., then

$$\overline{c}^{\mathsf{T}}\overline{v}_1 < \overline{c}^{\mathsf{T}}\overline{v}_2 < \cdots .$$

The simplex method terminates at some vertex $\overline{v}_{i^*}$ where $\overline{c}^{\mathsf{T}}\overline{v}_{i^*}$ is the global optimum

Final step: Compare the discovered optimal value $\overline{c}^{\mathsf{T}}\overline{v}_{i^*}$ to the desired value v_F.

- if equal, then F is $T_{\mathbb{Q}}$-satisfiable
- otherwise, F is $T_{\mathbb{Q}}$-unsatisfiable

$T_{\mathbb{Q}}$-Satisfiability

For a generic $\Sigma_{\mathbb{Q}}$-formula

$$F : \bigwedge_{i=1}^{m} a_{i1}x_1 + \cdots + a_{in}x_n \leq b_i$$

the corresponding optimization problem is

$$\begin{array}{ll} \textbf{max} & 1 \\ \textbf{subject to} & \\ & \bigwedge_{i=1}^{m} a_{i1}x_1 + \cdots + a_{in}x_n \leq b_i \end{array}$$

The optimum is $-\infty$ iff the constraints are $T_{\mathbb{Q}}$-unsatisfiable and 1 otherwise.

$T_{\mathbb{Q}}$-Satisfiability (cont.)

For a generic $\Sigma_{\mathbb{Q}}$-formula

$$\begin{aligned} F:\ & \textstyle\bigwedge_{i=1}^{m} a_{i1}x_1 + \cdots + a_{in}x_n \le b_i \\ & \wedge \textstyle\bigwedge_{i=1}^{l} a_{i1}x_1 + \cdots + a_{in}x_n < \beta_i \end{aligned}$$

the corresponding optimization problem is

$$\begin{aligned} &\textbf{max} \quad x_n + 1 \\ &\textbf{subject to} \\ &\qquad \textstyle\bigwedge_{i=1}^{m} a_{i1}x_1 + \cdots + a_{in}x_n \le b_i \\ &\qquad \textstyle\bigwedge_{i=1}^{l} a_{i1}x_1 + \cdots + a_{in}x_n + x_{n+1} \le \beta_i \end{aligned}$$

The optimum is positive iff the constraints are $T_{\mathbb{Q}}$-satisfiable.

The Theory of Equality T_E

$$\Sigma_E : \{=,\ a,\ b,\ c,\ \ldots,\ f,\ g,\ h,\ \ldots,\ p,\ q,\ r,\ \ldots\}$$

uninterpreted symbols:

- constants $a, b, c, \ldots$
- functions $f, g, h, \ldots$
- predicates $p, q, r, \ldots$

Example:

$x = y \ \wedge\ f(x) \neq f(y)$ T_E-unsatisfiable

$f(x) = f(y) \ \wedge\ x \neq y$ T_E-satisfiable

$f(f(f(a))) = a \ \wedge\ f(f(f(f(f(a))))) = a \ \wedge\ f(a) \neq a$

T_E-unsatisfiable

Axioms of T_E

1. $\forall x.\ x = x$ (reflexivity)
2. $\forall x, y.\ x = y \rightarrow y = x$ (symmetry)
3. $\forall x, y, z.\ x = y \wedge y = z \rightarrow x = z$ (transitivity)

define $=$ to be an equivalence relation.

Axiom schema

4. for each positive integer n and n-ary function symbol f,

$$\forall x_1, \ldots, x_n, y_1, \ldots, y_n.\ \bigwedge_i x_i = y_i \rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \quad \text{(congruence)}$$

For example,

$$\forall x, y.\ x = y \rightarrow f(x) = f(y)$$

Then

$$x = g(y, z) \rightarrow f(x) = f(g(y, z))$$

is T_E-valid.

Axiom schema

5. for each positive integer n and n-ary predicate symbol p,

$$\forall x_1, \ldots, x_n, y_1, \ldots, y_n.\ \bigwedge_i x_i = y_i \ \rightarrow \ (p(x_1, \ldots, x_n) \ \leftrightarrow \ p(y_1, \ldots, y_n)) \qquad \text{(equivalence)}$$

Thus,

$$x = y \ \rightarrow \ (p(x) \ \leftrightarrow \ p(y))$$

is T_E-valid.

We discuss T_E-formulae without predicates

For example, for Σ_E-formula

$$F: \ p(x) \ \wedge \ q(x,y) \ \wedge \ q(y,z) \ \rightarrow \ \neg q(x,z)$$

introduce fresh constant $\bullet$, and fresh functions f_p and f_q, and transform F to

$$G: \ f_p(x) = \bullet \ \wedge \ f_q(x,y) = \bullet \ \wedge \ f_q(y,z) = \bullet \ \rightarrow \ f_q(x,z) \neq \bullet \, .$$

Equivalence and Congruence Relations: Basics

Binary relation R over set S

- is an equivalence relation if
  - reflexive: $\forall s \in S.\ sRs$;
  - symmetric: $\forall s_1, s_2 \in S.\ s_1 R s_2 \ \rightarrow\ s_2 R s_1$;
  - transitive: $\forall s_1, s_2, s_3 \in S.\ s_1 R s_2 \ \wedge\ s_2 R s_3 \ \rightarrow\ s_1 R s_3$.

Example:
Define the binary relation $\equiv_2$ over the set $\mathbb{Z}$ of integers

$$m \equiv_2 n \quad \text{iff} \quad (m \bmod 2) = (n \bmod 2)$$

That is, $m, n \in \mathbb{Z}$ are related iff they are both even or both odd.
$\equiv_2$ is an equivalence relation

- is a congruence relation if in addition

$$\forall \bar{s}, \bar{t}.\ \bigwedge_{i=1}^{n} s_i R t_i \ \rightarrow\ f(\bar{s}) R f(\bar{t})\ .$$

Classes

For $\left\{ \begin{array}{c} \text{equivalence} \\ \text{congruence} \end{array} \right\}$ relation R over set S,

The $\left\{ \begin{array}{c} \text{equivalence} \\ \text{congruence} \end{array} \right\}$ class of $s \in S$ under R is

$$[s]_R \stackrel{\text{def}}{=} \{s' \in S \ : \ sRs'\} \ .$$

Example:

The equivalence class of 3 under $\equiv_2$ over $\mathbb{Z}$ is

$$[3]_{\equiv_2} = \{n \in \mathbb{Z} \ : \ n \text{ is odd}\} \ .$$

Partitions

A partition P of S is a set of subsets of S that is

- total $\left(\bigcup_{S' \in P} S' \right) = S$
- disjoint $\forall S_1, S_2 \in P.\ S_1 \cap S_2 = \varnothing$

Quotient

The quotient S/R of S by $\left\{\begin{array}{c}\text{equivalence}\\ \text{congruence}\end{array}\right\}$relation R is the set of $\left\{\begin{array}{c}\text{equivalence}\\ \text{congruence}\end{array}\right\}$classes

$$S/R \;=\; \{[s]_R \;:\; s \in S\}\,.$$

It is a partition

Example: The quotient $\mathbb{Z}/\equiv_2$ is a partition of $\mathbb{Z}$. The set of equivalence classes

$$\{\{n \in \mathbb{Z} \;:\; n \text{ is odd}\},\; \{n \in \mathbb{Z} \;:\; n \text{ is even}\}\}$$

Note duality between relations and classes

Refinements

Two binary relations R_1 and R_2 over set S.
R_1 is refinement of R_2, $R_1 \prec R_2$, if

$$\forall s_1, s_2 \in S.\ s_1 R_1 s_2 \ \rightarrow\ s_1 R_2 s_2\ .$$

R_1 refines R_2.

Examples:

- For $S = \{a, b\}$,
  $R_1 : \{aR_1 b\}$ $\qquad$ $R_2 : \{aR_2 b,\ bR_2 b\}$
  Then $R_1 \prec R_2$
- For set S,
  R_1 induced by the partition $P_1 : \{\{s\} \ : \ s \in S\}$
  R_2 induced by the partition $P_2 : \{S\}$
  Then $R_1 \prec R_2$.
- For set $\mathbb{Z}$
  $R_1 : \{xR_1 y \ : \ x \bmod 2 = y \bmod 2\}$
  $R_2 : \{xR_2 y \ : \ x \bmod 4 = y \bmod 4\}$
  Then $R_2 \prec R_1$.

Closures

Given binary relation R over S.

The equivalence closure R^E of R is the equivalence relation s.t.

- R refines R^E, i.e. $R \prec R^E$;
- for all other equivalence relations R' s.t. $R \prec R'$,
  either $R' = R^E$ or $R^E \prec R'$

That is, R^E is the "smallest" equivalence relation that "covers" R.

Example: If $S = \{a, b, c, d\}$ and $R = \{aRb, bRc, dRd\}$, then

- $aRb, bRc, dRd \in R^E$ since $R \subseteq R^E$;
- $aRa, bRb, cRc \in R^E$ by reflexivity;
- $bRa, cRb \in R^E$ by symmetry;
- $aRc \in R^E$ by transitivity;
- $cRa \in R^E$ by symmetry.

Hence,

$$R^E = \{aRb, bRa, aRa, bRb, bRc, cRb, cRc, aRc, cRa, dRd\} .$$

Similarly, the congruence closure R^C of R is the "smallest" congruence relation that "covers" R.

Congruence Closure Algorithm

Given Σ_E-formula

$$F: \; s_1 = t_1 \;\wedge\; \cdots \;\wedge\; s_m = t_m \;\wedge\; s_{m+1} \neq t_{m+1} \;\wedge\; \cdots \;\wedge\; s_n \neq t_n$$

decide if F is Σ_E-satisfiable.

Definition: For Σ_E-formula F,
the subterm set S_F of F is the set that contains precisely the subterms of F.

Example: The subterm set of

$$F: \; f(a,b) = a \;\wedge\; f(f(a,b),b) \neq a$$

is

$$S_F = \{a,\; b,\; f(a,b),\; f(f(a,b),b)\} \;.$$

The Algorithm

Given Σ_E-formula F

$$F : s_1 = t_1 \wedge \cdots \wedge s_m = t_m \wedge s_{m+1} \neq t_{m+1} \wedge \cdots \wedge s_n \neq t_n$$

with subterm set S_F, F is T_E-satisfiable iff there exists a congruence relation $\sim$ over S_F such that

- for each $i \in \{1, \ldots, m\}$, $s_i \sim t_i$;
- for each $i \in \{m+1, \ldots, n\}$, $s_i \not\sim t_i$.

Such congruence relation $\sim$ defines T_E-interpretation $I : (D_I, \alpha_I)$ of F. D_I consists of $|S_F/\sim|$ elements, one for each congruence class of S_F under $\sim$.

Instead of writing $I \models F$ for this T_E-interpretation, we abbreviate

$$\sim \models F$$

The goal of the algorithm is to construct the congruence relation of S_F, or to prove that no congruence relation exists.

$$F: \quad \underbrace{s_1 = t_1 \wedge \cdots \wedge s_m = t_m}_{\text{generate congruence closure}} \quad \wedge \underbrace{s_{m+1} \neq t_{m+1} \wedge \cdots \wedge s_n \neq t_n}_{\text{search for contradiction}}$$

The algorithm performs the following steps:

1. Construct the congruence closure $\sim$ of

$$\{s_1 = t_1, \ldots, s_m = t_m\}$$

   over the subterm set S_F. Then

$$\sim \models s_1 = t_1 \wedge \cdots \wedge s_m = t_m .$$

2. If for any $i \in \{m+1, \ldots, n\}$, $s_i \sim t_i$, return unsatisfiable.
3. Otherwise, $\sim \models F$, so return satisfiable.

How do we actually construct the congruence closure in Step 1?

Initially, begin with the finest congruence relation $\sim_0$ given by the partition

$$\{\{s\} \ : \ s \in S_F\} \, .$$

That is, let each term of S_F be its own congruence class.

Then, for each $i \in \{1, \ldots, m\}$, impose $s_i = t_i$ by merging the congruence classes

$$[s_i]_{\sim_{i-1}} \quad \text{and} \quad [t_i]_{\sim_{i-1}}$$

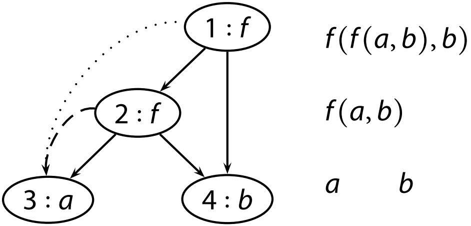
to form a new congruence relation $\sim_i$. To accomplish this merging,

- form the union of $[s_i]_{\sim_{i-1}}$ and $[t_i]_{\sim_{i-1}}$
- propagate any new congruences that arise within this union.

The new relation $\sim_i$ is a congruence relation in which $s_i \sim t_i$.

Directed Acyclic Graph (DAG)

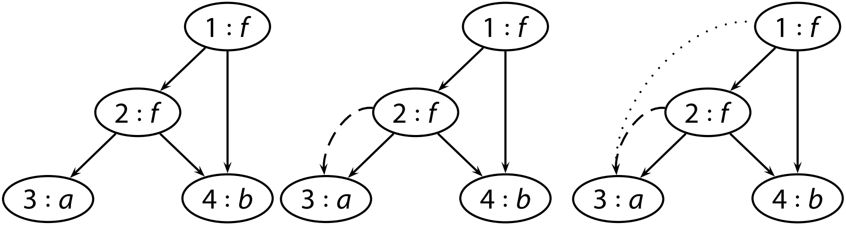
For Σ_E-formula F, graph-based data structure for representing the subterms of S_F (and congruence relation between them).



Efficient way for computing the congruence closure algorithm.

T_E-Satisfiability (Summary of idea)

$$f(a,b) = a \land f(f(a,b),b) \neq a$$



Initial DAG

$f(a,b) = a \Rightarrow$
merge $f(a,b)$ a

$f(a,b) \sim a,\ b \sim b \Rightarrow$
$f(f(a,b),b) \sim f(a,b)$
merge $f(f(a,b),b)$
$f(a,b)$

_ _ explicit equation ..... by congruence

$$\left.\begin{array}{r} \text{find } f(f(a,b),b) = a = \text{find } a \\ f(f(a,b),b) \neq a \end{array}\right\} \Rightarrow \textbf{Unsatisfiable}$$

DAG representation

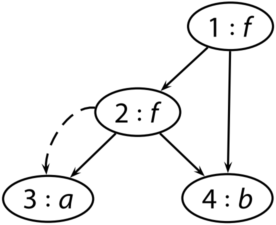
type **node** = {

`id`	:	**id** node's unique identification number
`fn`	:	**string** constant or function name
`args`	:	**id list** list of function arguments
`mutable find`	:	**id** the representative of the congruence class
`mutable ccpar`	:	**id set** if the node is the representative for its congruence class, then its `ccpar` (congruence closure parents) are all parents of nodes in its congruence class

}

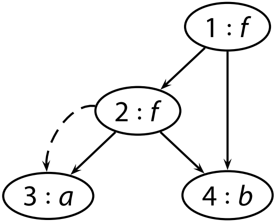
DAG Representation of node 2

$$
\begin{array}{llll}
\texttt{type}\ \textbf{node} = \{ & & & \\
\quad \texttt{id} & : & \textbf{id} & \ldots 2 \\
\quad \texttt{fn} & : & \textbf{string} & \ldots f \\
\quad \texttt{args} & : & \textbf{idlist} & \ldots [3,4] \\
\quad \texttt{mutable find} & : & \textbf{id} & \ldots 3 \\
\quad \texttt{mutable ccpar} & : & \textbf{idset} & \ldots \emptyset \\
\} & & &
\end{array}
$$



DAG Representation of node 3

```
type node = {
    id             : id       ... 3
    fn             : string   ... a
    args           : idlist   ... []
    mutable find   : id       ... 3
    mutable ccpar  : idset    ... {1,2}
  }
```

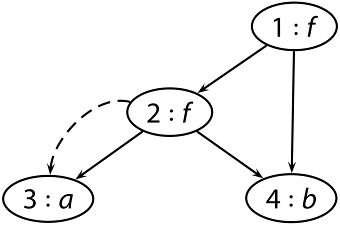


The Implementation

find function

returns the representative of node's congruence class

```
let rec find i =
  let n = node i in
  if n.find = i then i else find n.find
```



Example: find 2 = 3
find 3 = 3

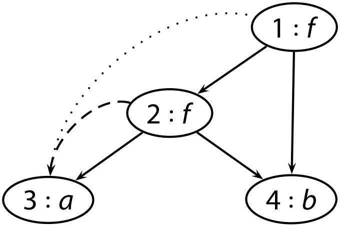
3 is the representative of 2.

union function

$$
\begin{aligned}
&\texttt{let}\ \text{union}\ i_1\ i_2 = \\
&\quad \texttt{let}\ n_1 \ = \ \text{node}\,(\text{find}\ i_1)\ \texttt{in} \\
&\quad \texttt{let}\ n_2 \ = \ \text{node}\,(\text{find}\ i_2)\ \texttt{in} \\
&\quad n_1.\texttt{find} \leftarrow n_2.\texttt{find}; \\
&\quad n_2.\texttt{ccpar} \leftarrow n_1.\texttt{ccpar} \cup n_2.\texttt{ccpar}; \\
&\quad n_1.\texttt{ccpar} \leftarrow \varnothing
\end{aligned}
$$

n_2 is the representative of the union class

Example



union 1 2 $\quad n_1 = 1 \quad n_2 = 3$

$1.\texttt{find} \leftarrow 3$

$3.\texttt{ccpar} \leftarrow \{1, 2\}$

$1.\texttt{ccpar} \leftarrow \emptyset$

ccpar function

Returns parents of all nodes in i's congruence class

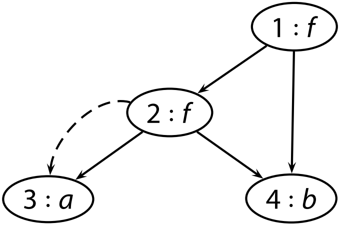
$$\texttt{let}\ \text{ccpar}\ i =$$
$$(\text{node}\ (\text{find}\ i)).\texttt{ccpar}$$

congruent predicate

Test whether i_1 and i_2 are congruent

$$\begin{aligned}
&\texttt{let}\ \text{congruent}\ i_1\ i_2 = \\
&\quad \texttt{let}\ n_1 = \text{node}\ i_1\ \texttt{in} \\
&\quad \texttt{let}\ n_2 = \text{node}\ i_2\ \texttt{in} \\
&\quad n_1.\texttt{fn} = n_2.\texttt{fn} \\
&\qquad \wedge |n_1.\texttt{args}| = |n_2.\texttt{args}| \\
&\qquad \wedge \forall i \in \{1, \ldots, |n_1.\texttt{args}|\}.\ \text{find}\ n_1.\texttt{args}[i] = \text{find}\ n_2.\texttt{args}[i]
\end{aligned}$$

Example:



Are 1 and 2 congruent?

`fn` fields	--- both f
# of arguments	--- same
left arguments $f(a, b)$ and a	--- both congruent to 3
right arguments b and b	--- both 4 (congruent)

Therefore 1 and 2 are congruent.

merge function

```
let rec merge i₁ i₂ =
  if find i₁ ≠ find i₂ then begin
    let P_{i₁} = ccpar i₁ in
    let P_{i₂} = ccpar i₂ in
    union i₁ i₂;
    foreach t₁, t₂ ∈ P_{i₁} × P_{i₂} do
      if find t₁ ≠ find t₂ ∧ congruent t₁ t₂
      then merge t₁ t₂
    done
  end
```

P_{i_1} and P_{i_2} store the current values of ccpar i_1 and ccpar i_2.

Decision Procedure: T_E-satisfiability

Given Σ_E-formula

$$F: \; s_1 = t_1 \;\wedge\; \cdots \;\wedge\; s_m = t_m \;\wedge\; s_{m+1} \neq t_{m+1} \;\wedge\; \cdots \;\wedge\; s_n \neq t_n \;,$$

with subterm set S_F, perform the following steps:

1. Construct the initial DAG for the subterm set S_F.
2. For $i \in \{1, \ldots, m\}$, merge s_i t_i.
3. If find s_i = find t_i for some $i \in \{m+1, \ldots, n\}$, return unsatisfiable.
4. Otherwise (if find $s_i \neq$ find t_i for all $i \in \{m+1, \ldots, n\}$) return satisfiable.

Theorem (Sound and Complete)

Quantifier-free conjunctive Σ_E-formula F is T_E-satisfiable iff the congruence closure algorithm returns satisfiable.

Recursive Data Structures

Quantifier-free Theory of Lists T_{cons}

$$\Sigma_{\text{cons}} : \ \{\text{cons},\ \text{car},\ \text{cdr},\ \text{atom},\ =\}$$

- constructor cons : $\text{cons}(a, b)$ list constructed by prepending a to b
- left projector car : $\text{car}(\text{cons}(a, b)) = a$
- right projector cdr : $\text{cdr}(\text{cons}(a, b)) = b$
- atom : unary predicate

Axioms of T_{cons}

- reflexivity, symmetry, transitivity
- congruence axioms:

$$\begin{array}{l}\forall x_1, x_2, y_1, y_2.\, x_1 = x_2 \wedge y_1 = y_2 \rightarrow \text{cons}(x_1, y_1) = \text{cons}(x_2, y_2) \\ \forall x, y.\, x = y \rightarrow \text{car}(x) = \text{car}(y) \\ \forall x, y.\, x = y \rightarrow \text{cdr}(x) = \text{cdr}(y)\end{array}$$

- equivalence axiom:

$$\forall x, y.\, x = y \rightarrow (\text{atom}(x) \leftrightarrow \text{atom}(y))$$

-

$$\begin{array}{lr}(A1)\ \forall x, y.\, \text{car}(\text{cons}(x, y)) = x & \text{(left projection)} \\ (A2)\ \forall x, y.\, \text{cdr}(\text{cons}(x, y)) = y & \text{(right projection)} \\ (A3)\ \forall x.\, \neg\text{atom}(x) \rightarrow \text{cons}(\text{car}(x), \text{cdr}(x)) = x & \text{(construction)} \\ (A4)\ \forall x, y.\, \neg\text{atom}(\text{cons}(x, y)) & \text{(atom)}\end{array}$$

Simplifications

- Consider only quantifier-free conjunctive Σ_{cons}-formulae. Convert non-conjunctive formula to DNF and check each disjunct.
- $\neg\text{atom}(u_i)$ literals are removed:

$$\boxed{\text{replace} \quad \neg\text{atom}(u_i) \quad \text{with} \quad u_i = \text{cons}(u_i^1, u_i^2)}$$

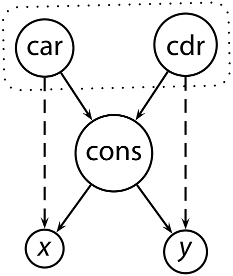
  by the (construction) axiom.
- Because of similarity to Σ_{E}, we sometimes combine $\Sigma_{\text{cons}} \cup \Sigma_{\text{E}}$.

Algorithm: T_{cons}-Satisfiability (the idea)

$$
\begin{array}{rl}
F: & \underbrace{s_1 = t_1 \ \wedge \ \cdots \ \wedge \ s_m = t_m}_{\text{generate congruence closure}} \\
\wedge & \underbrace{s_{m+1} \neq t_{m+1} \ \wedge \ \cdots \ \wedge \ s_n \neq t_n}_{\text{search for contradiction}} \\
\wedge & \underbrace{\text{atom}(u_1) \ \wedge \ \cdots \ \wedge \ \text{atom}(u_l)}_{\text{search for contradiction}}
\end{array}
$$

where s_i, t_i, and u_i are T_{cons}-terms

Algorithm: T_{cons}-Satisfiability



1. Construct the initial DAG for S_F
2. for each node n with $n.\mathtt{fn} = \text{cons}$
   - add $\text{car}(n)$ and merge $\text{car}(n)$ $n.\mathtt{args}[1]$
   - add $\text{cdr}(n)$ and merge $\text{cdr}(n)$ $n.\mathtt{args}[2]$

   by axioms (A1), (A2)
3. for $1 \leq i \leq m$, merge s_i t_i
4. for $m + 1 \leq i \leq n$, if find s_i = find t_i, return **unsatisfiable**
5. for $1 \leq i \leq l$, if $\exists v.$ find v = find $u_i \wedge v.\mathtt{fn} = \text{cons}$, return **unsatisfiable**
6. Otherwise, return **satisfiable**

Example:

Given $(\Sigma_{\text{cons}} \cup \Sigma_E)$-formula

$$F: \quad \begin{array}{l} \text{car}(x) = \text{car}(y) \ \land \ \text{cdr}(x) = \text{cdr}(y) \\ \land \neg\text{atom}(x) \ \land \ \neg\text{atom}(y) \ \land \ f(x) \neq f(y) \end{array}$$

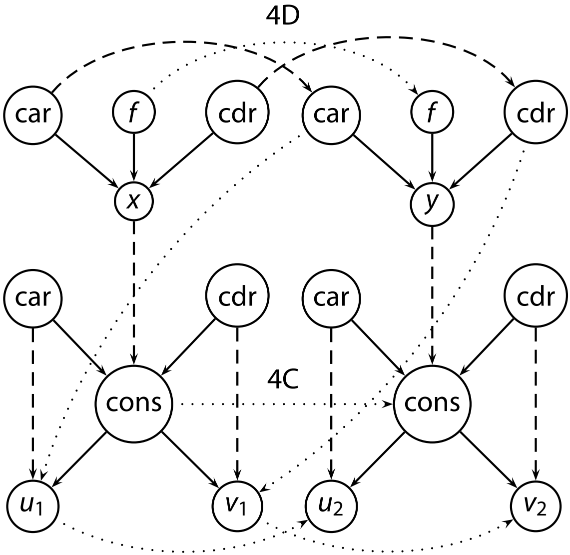
where the function symbol f is in Σ_E

$$F': \quad \begin{array}{lll} \text{car}(x) = \text{car}(y) & \land & (1) \\ \text{cdr}(x) = \text{cdr}(y) & \land & (2) \\ x = \text{cons}(u_1, v_1) & \land & (3) \\ y = \text{cons}(u_2, v_2) & \land & (4) \\ f(x) \neq f(y) & & (5) \end{array}$$

Recall the projection axioms:

$$(A1) \quad \forall x, y.\ \text{car}(\text{cons}(x, y)) = x$$
$$(A2) \quad \forall x, y.\ \text{cdr}(\text{cons}(x, y)) = y$$

Example(cont): congruence



F is **unsatisfiable**