# Seminar on Automated Reasoning 2010

# Lecture 5:

# Reasoning Modulo Theories - Overview, Decision Procedure for Equality

Swen Jacobs

22. October 2010

## First-order Theories

A **theory** $\mathcal{T}$ is defined by

- a signature $\Sigma_{\mathcal{T}}$, and
- a set of closed $\Sigma_{\mathcal{T}}$-formulas $A_{\mathcal{T}}$, the **axioms** of $\mathcal{T}$.

**Differences** to full FOL:

- restricted signature
- some or all symbols have a predefined meaning, given by axioms

## Theory of Equality

The **theory of equality** $\mathcal{T}_E$ is given by

$\Sigma_E = (\{a/0, b/0, \dots, f/1, \dots g/2, \dots\}, \{=/2, p/0, \dots q/1, \dots\})$,

and axioms

$$
\begin{array}{lll}
\forall x. & x = x & \text{(reflexivity)} \\
\forall x, y. & x = y \;\rightarrow\; y = x & \text{(symmetry)} \\
\forall x, y, z. & x = y \;\wedge\; y = z \;\rightarrow\; x = z & \text{(transitivity)} \\
\forall \overline{x}, \overline{y}. & \left(\bigwedge_{i=1}^{n} x_i = y_i\right) \;\rightarrow\; f(\overline{x}) = f(\overline{y}) & \text{(function congruence)} \\
\forall \overline{x}, \overline{y}. & \left(\bigwedge_{i=1}^{n} x_i = y_i\right) \;\rightarrow\; (p(\overline{x}) \leftrightarrow p(\overline{y})) & \text{(predicate congruence)}
\end{array}
$$

(the last two are schemes, hold for every function/predicate symbol)

## Satisfiability Modulo Theories

An interpretation $I$ which satisfies all axioms of $\mathcal{T}$, i.e. $I \models A_{\mathcal{T}}$, is called a **$\mathcal{T}$-interpretation**.

A $\Sigma_{\mathcal{T}}$-formula $F$ is **valid in theory $\mathcal{T}$** (or $\mathcal{T}$-valid) if $A_{\mathcal{T}} \models F$, i.e. every $\mathcal{T}$-interpretation satisfies $F$, written $\mathcal{T} \models F$.

A $\Sigma_{\mathcal{T}}$-formula $F$ is **satisfiable in $\mathcal{T}$** (or $\mathcal{T}$-satisfiable) if there exists a $\mathcal{T}$-interpretation which satisfies $F$.

## Decidability of Theories and Fragments

A theory $\mathcal{T}$ is **decidable** if $\mathcal{T} \models F$ is decidable for every $\Sigma_{\mathcal{T}}$-formula $F$.

A **fragment** of a theory $\mathcal{T}$ is a syntactically restricted subset of $\Sigma_{\mathcal{T}}$-formulas. E.g., the **quantifier-free fragment** of $\mathcal{T}$ is the set of quantifier-free $\Sigma_{\mathcal{T}}$-formulas.

A fragment of $\mathcal{T}$ is decidable if $\mathcal{T} \models F$ is decidable for every formula in the fragment.

# First-order Theories

**Examples**: Theories of

- Equality $\mathcal{T}_E$
- Arrays $\mathcal{T}_{Arrays}$
- Presburger Arithmetic $PA$
- Sets with cardinality constraints BAPA
- Rational numbers $\mathcal{T}_{\mathbb{Q}}$
- Real numbers $\mathcal{T}_{\mathbb{R}}$

## Theory of Equality

The **theory of equality** $\mathcal{T}_E$ is given by

$\Sigma_E = (\{a/0, b/0, \ldots, f/1, \ldots g/2, \ldots\}, \{=/2, p/0, \ldots q/1, \ldots\})$,

and axioms

$$
\begin{array}{lll}
\forall x. & x = x & \text{(reflexivity)} \\
\forall x, y. & x = y \;\rightarrow\; y = x & \text{(symmetry)} \\
\forall x, y, z. & x = y \;\land\; y = z \;\rightarrow\; x = z & \text{(transitivity)} \\
\forall \overline{x}, \overline{y}. & \left(\bigwedge_{i=1}^{n} x_i = y_i\right) \;\rightarrow\; f(\overline{x}) = f(\overline{y}) & \text{(function congruence)} \\
\forall \overline{x}, \overline{y}. & \left(\bigwedge_{i=1}^{n} x_i = y_i\right) \;\rightarrow\; (p(\overline{x}) \leftrightarrow p(\overline{y})) & \text{(predicate congruence)}
\end{array}
$$

(the last two are schemes, hold for every function/predicate symbol)

## Problems in $\mathcal{T}_E$

**Question**: Is $a = b \land b = c \land f(a) \neq f(c)$ satisfiable in $\mathcal{T}_E$?

From (transitivity) and $a = b \land b = c$ conclude $a = c$

From (congruence) and $a = c$ conclude $f(a) = f(c)$

$f(a) = f(c)$ contradicts $f(a) \neq f(c)$

## Problems in $\mathcal{T}_E$

**Application**: Validation of compiler translations

```
1:  y := 1                       1:  y := 1
2:  if x = y + y                 2:  R := y + y
3:  then y := x * x              3:  jmpNE(x,R,5)
4:  endif                        4:  y := R * R
```

**To prove:** (equivalence of left- and right-hand side)

$$y_1 = 1 \land ((x_2 \neq y_1 + y_1 \land y_3 = y_1) \lor (x_0 = y_1 + y_1 \land y_3 = x_0 * x_0))$$
$$\land\ y_1' = 1 \land R_2 = y_1' + y_1' \land ((x_0' \neq R_2 \land y_4' = y_1') \lor (x_0' = R_2 \land y_4' = R_2 * R_2))$$
$$\land\ x_0 = x_0' \land y_0 = y_0' \quad \to \quad x_0 = x_0' \land y_3 = y_4'$$

$\Rightarrow$ this can be proved for uninterpreted $*$ and $+$

## Theory of Arrays

The **theory of arrays** $\mathcal{T}_{Array}$ is given by[a]

$\Sigma_{Array} = (\{read/2, write/3\}, \{=/2\})$,

and axioms

$$
\begin{array}{llr}
\forall x. & x = x & \text{(reflexivity)} \\
\forall x, y. & x = y \;\rightarrow\; y = x & \text{(symmetry)} \\
\forall x, y, z. & x = y \;\wedge\; y = z \;\rightarrow\; x = z & \text{(transitivity)} \\
\forall a, x, y. & x = y \;\rightarrow\; read(a, x) = read(a, y) & \text{(array congruence)} \\
\forall a, v, x, y. & x = y \;\rightarrow\; read(write(a, x, v), y) = v & \text{(read-over-write 1)} \\
\forall a, v, x, y. & x \neq y \;\rightarrow\; read(write(a, x, v), y) = read(a, y) & \text{(read-over-write 2)}
\end{array}
$$

[a]Note that we omit sorts, which is less restrictive and less efficient.

## Problems in $\mathcal{T}_{Array}$

**Question**: Is $i = j \land i \neq k \land read(a, j) = e_1 \land e_1 \neq e_2 \land read(write(a, k, e_2), i) = e_2$ satisfiable in $\mathcal{T}_{Array}$?

From (array congruence) and $i = j$ conclude $read(a, i) = read(a, j)$

From (transitivity), $read(a, i) = read(a, j)$, and $read(a, j) = e_1$ conclude $read(a, i) = e_1$

From (read-over-write 2) and $i \neq k$ conclude $read(write(a, k, e_2), i) = read(a, i)$

From (symmetry), (transitivity), $read(write(a, k, e_2), i) = read(a, i)$, and $read(write(a, k, e_2), i) = e_2$, conclude $read(a, i) = e_2$

From (symmetry), (transitivity), $read(a, i) = e_1$, and $read(a, i) = e_2$ conclude $e_1 = e_2$

## Problems in $\mathcal{T}_{Array}$

**Applications**:
Arrays are used to model problems in hard- and software verification, e.g. for memory accesses, the stack of programs or for arrays in programs.

## Presburger Arithmetic

The **theory of Presburger Arithmetic** *PA* is given by

$\Sigma_{\mathbb{N}} = (\{0/0, 1/0, +/2\}, \{=/2\})$,

and axioms

$$
\begin{array}{lll}
\forall x. & x + 1 \neq 0 & \text{(zero)} \\
\forall x, y. & x + 1 = y + 1 \ \rightarrow \ x = y & \text{(successor)} \\
\multicolumn{2}{l}{F[0] \ \wedge \ (\forall x. \ \ F[x] \ \rightarrow \ F[x+1]) \ \rightarrow \ \forall x. \ \ F[x]} & \text{(induction)} \\
\forall x. & x + 0 = x & \text{(plus zero)} \\
\forall x, y. & x + (y + 1) = (x + y) + 1 & \text{(plus successor)}
\end{array}
$$

((induction) is a scheme, holds for every $\Sigma_{\mathbb{N}}$-formula $F$)

## Boolean Algebra

The **theory of Boolean Algebra** *BA* is given by

$\Sigma_{PA} = (\{\emptyset/0, \mathcal{U}/0, \ ^c/1, \cup/2, \cap/2\}, \{=/2, \subseteq/2\})$,

and axioms

| | | |
|---|---|---|
| | Axioms of $\mathcal{T}_E$ | |
| $\forall x, y, z.$ | $x \cup (y \cup z) = (x \cup y) \cup z$ | (associativity of $\cup$) |
| $\forall x, y.$ | $x \cup y = y \cup x$ | (commutativity of $\cup$) |
| $\forall x, y.$ | $x \cup (x \cap y) = x$ | (absorption 1) |
| $\forall x, y, z.$ | $x \cup (y \cap z) = (x \cup y) \cap (x \cup z)$ | (distributivity 1) |
| $\forall x.$ | $x \cup x^c = \mathcal{U}$ | (complement 1) |
| | (continued on next slide) | |

## Boolean Algebra

The **theory of Boolean Algebra** $BA$ is given by

$\Sigma_{PA} = (\{\emptyset/0, \mathcal{U}/0, {}^{c}/1, \cup/2, \cap/2\}, \{=/2, \subseteq/2\})$,

and axioms

(continued from previous slide)

| | | |
|---|---|---|
| $\forall x, y, z.$ | $x \cap (y \cap z) = (x \cap y) \cap z$ | (associativity of $\cap$) |
| $\forall x, y.$ | $x \cap y = y \cap x$ | (commutativity of $\cap$) |
| $\forall x, y.$ | $x \cap (x \cup y) = x$ | (absorption 2) |
| $\forall x, y, z.$ | $x \cap (y \cup z) = (x \cap y) \cup (x \cap z)$ | (distributivity 2) |
| $\forall x.$ | $x \cap x^c = \emptyset$ | (complement 2) |
| $\forall x, y.$ | $x \cup y = y \leftrightarrow x \subseteq y$ | (partial order $\subseteq$) |

## Boolean Algebra with Presburger Arithmetic

The **theory of Boolean Algebra with Presburger Arithmetic** *BAPA* is given by

$$\Sigma_{BAPA} = \Sigma_{PA} \cup \Sigma_{BA} \cup (\{|\cdot|/1\}, \{\}),$$

and axioms

$$\begin{array}{lll} & \text{(Axioms of PA)} & \\ & \text{(Axioms of BA)} & \\ \forall x, y. & x \cap y = \emptyset \ \rightarrow \ |x \cup y| = |x| + |y| & \text{(disjoint sets)} \\ \forall x. & |x| = 0 \leftrightarrow x = \emptyset & \text{(empty set)} \end{array}$$

## Problems in *BAPA*

**Applications**:

- static analysis of data structure consistency properties: for a data structure that keeps track of its own size internally, prove that the internal size value is equal to the actual size of the set of elements in the data structure
- proving conditions for refinement and equivalence of program fragments with such data structures
- proving the termination of programs that manipulate such data structures

## Rational Numbers

The **theory of rational numbers** $\mathcal{T}_{\mathbb{Q}}$ is given by

$\Sigma_{\mathbb{Q}} = (\{0/0, 1/0, -/1, +/2\}, \{=/2, \leq/2\})$,

and axioms

$$\begin{array}{lll}
\forall x, y. & x \leq y \ \wedge \ y \leq x \ \rightarrow \ x = y & \text{(antisymmetry)} \\
\forall x, y, z. & x \leq y \ \wedge \ y \leq z \ \rightarrow \ x \leq z & \text{(transitivity)} \\
\forall x, y. & x \leq y \vee y \leq x & \text{(totality)} \\
\forall x, y, z. & x + (y + z) = (x + y) + z & \text{(associativity of } +) \\
\forall x. & x + 0 = x & \text{(identity wrt. } +) \\
\forall x. & x + (-x) = 0 & \text{(inverse wrt. } +) \\
\forall x, y. & x + y = y + x & \text{(commutativity of } +) \\
\forall x, y, z. & x \leq y \ \rightarrow \ x + z \leq y + z & \text{(ordered wrt. } +)
\end{array}$$

(continued on next slide)

## Rational Numbers

The **theory of rational numbers** $\mathcal{T}_{\mathbb{Q}}$ is given by

$\Sigma_{\mathbb{Q}} = (\{0/0, 1/0, -/1, +/2\}, \{=/2, \leq/2\})$,

and axioms

(continued from last slide)

| | | |
|---|---|---|
| $\forall x.$ | $n \cdot x = 0 \;\rightarrow\; x = 0$ | (torsion-free) |
| $\forall x. \exists y.$ | $x = n \cdot y$ | (divisible) |

((torsion-free) and (divisible) are schemes, hold for every $n \in \mathbb{N}^+$, where $n \cdot x$ stands for $\underbrace{x + \ldots + x}_{n \text{ times}}$)

# Real Numbers

The **theory of real numbers** $\mathcal{T}_{\mathbb{R}}$ is given by

$\Sigma_{\mathbb{R}} = \Sigma_{\mathbb{Q}} \cup (\{\cdot/2\}, \{\})$,

and axioms

| | | |
|---|---|---|
| $\forall x, y.$ | $x \leq y \ \wedge \ y \leq x \ \rightarrow \ x = y$ | (antisymmetry) |
| $\forall x, y, z.$ | $x \leq y \ \wedge \ y \leq z \ \rightarrow \ x \leq z$ | (transitivity) |
| $\forall x, y.$ | $x \leq y \vee y \leq x$ | (totality) |
| $\forall x, y, z.$ | $x + (y + z) = (x + y) + z$ | (associativity of $+$) |
| $\forall x.$ | $x + 0 = x$ | (identity wrt. $+$) |
| $\forall x.$ | $x + (-x) = 0$ | (inverse wrt. $+$) |
| $\forall x, y.$ | $x + y = y + x$ | (commutativity of $+$) |
| $\forall x, y, z.$ | $x \leq y \ \rightarrow \ x + z \leq y + z$ | (ordered wrt. $+$) |

(continued on next slide)

# Real Numbers

| | | |
|---|---|---|
| | (continued from last slide) | |
| $\forall x, y, z.$ | $x \cdot (y \cdot z) = (x \cdot y) \cdot z$ | (associativity of $\cdot$) |
| $\forall x.$ | $x \cdot 1 = x$ | (identity wrt. $\cdot$) |
| $\forall x.$ | $x \neq 0 \;\rightarrow\; \exists y.\, x \cdot y = 1$ | (inverse wrt. $\cdot$) |
| $\forall x, y.$ | $x \cdot y = y \cdot x$ | (commutativity of $\cdot$) |
| $\forall x, y.$ | $0 \leq x \;\wedge\; 0 \leq y \;\rightarrow\; 0 \leq x \cdot y$ | (ordered wrt. $\cdot$) |
| $\forall x, y, z.$ | $x \cdot (y + z) = x \cdot y + x \cdot z$ | (distributivity) |
| | $0 \neq 1$ | (separate identities) |
| $\forall x. \exists y.$ | $x = y^2 \vee -x = y^2$ | (square-root) |
| $\forall \overline{x}. \exists y.$ | $y^n + x_1 \cdot y^{n-1} + \ldots + x_{n-1} \cdot y + x_n = 0$ | (at least one root) |

((at least one root) is a scheme that holds for every odd integer $n$)

# Rational and Real Numbers

**Application**: (one among many)
Analysis and verification of hybrid systems, i.e. systems whose state space is spanned by both discrete and continuous variables.

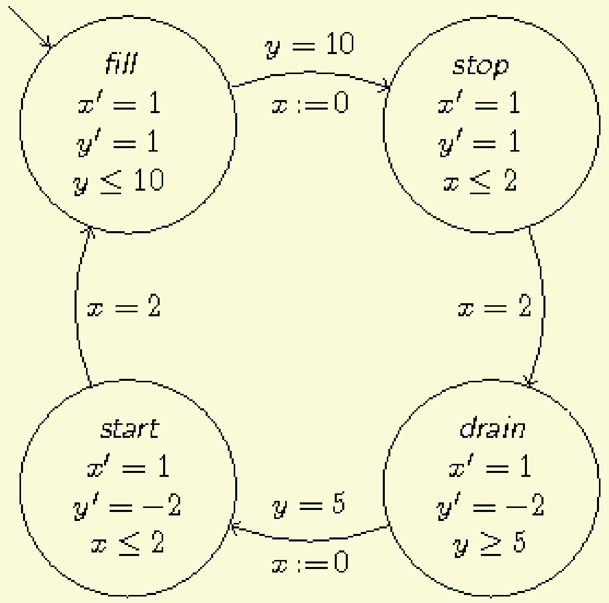
Hybrid automaton of a watertank:

4 discrete states, 2 continuous variables

$x$ represents time, $y$ amount of water

Want to prove properties like:

- never more than 12 water in tank
- never less than 1 water in tank



## Decidablity of Theories

Overview of decidablity of the conjunctive fragments (i.e. only conjunctions of literals are allowed) of our example theories:

| Theory | Full | QF |
|---|---|---|
| $\mathcal{T}_E$ | undecidable | O(n log n) |
| $\mathcal{T}_{Arrays}$ | undecidable | NP-complete |
| $PA$ | $\sim$3EXPTIME | NP-complete |
| $BAPA$ | $\sim$3EXPTIME | NP-complete |
| $\mathcal{T}_{\mathbb{Q}}$ | 2EXPTIME | PTIME |
| $\mathcal{T}_{\mathbb{R}}$ | 2EXPTIME | 2EXPTIME |

## Decidablity of Theories

For problems in these theories, general-purpose methods like resolution

- are in general no decision procedures
- are in general not as efficient as specialized methods
- often have an infinite search space even for the simplest problems, because of infinite sets of axioms

Therefore, we will investigate **specialized decision procedures** for important theories.

## Congruence Closure

**Congruence closure** is a method to decide quantifier-free $\mathcal{T}_E$-formulas

**Applications** are in verification of programs:

- directly, by abstraction of function properties
- extensions to prove properties of data structures like arrays, lists
- is the basis for combining theories

*"Almost all proofs require reasoning about equalities."*

— Greg Nelson and Derek C. Oppen

*Fast Decision Procedures Based on Congruence Closure*, 1980

## Congruence Closure

**Definition**: a binary relation $\sim$ is a **congruence relation** if

$$\begin{array}{ll}
\forall x. & x \sim x \\
\forall x, y. & x \sim y \;\rightarrow\; y \sim x \\
\forall x, y, z. & x \sim y \;\wedge\; y \sim z \;\rightarrow\; x \sim z \\
\forall \overline{x}, \overline{y}. & \left(\bigwedge_{i=1}^{n} x_i \sim y_i\right) \;\rightarrow\; f(\overline{x}) \sim f(\overline{y}) \quad \text{(for every function } f\text{)}
\end{array}$$

**Definition**: given a set $S$ and a congruence relation $\sim$ on $S$, the **congruence class** of $s \in S$ under $\sim$ is

$$[s]_\sim = \{s' \in S \mid s \sim s'\}$$

## Congruence Closure

A congruence relation $\sim$ on a set $S$ defines a **partition** on this set: if $P$ is the set of all congruence classes of $S$ under $\sim$, then

$$\left(\bigcup_{S' \in P} S'\right) = S,$$

$$\textit{and} \quad \forall\ S_1, S_2 \in P.\ S_1 \neq S_2\ \rightarrow\ S_1 \cap S_2 = \emptyset.$$

**Definition**: A relation $\sim'$ is a **refinement** of $\sim$, written $\sim' \prec \sim$, if

$$\forall\ s_1, s_2 \in S.\ s_1 \sim' s_2\ \rightarrow\ s_1 \sim s_2.$$

## Congruence Closure

**Definition**: If $\sim$ is a binary relation on $S$, then the **congruence closure** $\sim^c$ of $\sim$ is the congruence relation such that

- $\sim \prec \sim^c$, and
- for all other congruence relations $\sim'$ such that $\sim \prec \sim'$, either $\sim' = \sim^c$ or $\sim^c \prec \sim'$.

(i.e. it is the smallest congruence relation that includes $\sim$)

## Congruence Closure

**Abstract Algorithm**: Given a $\mathcal{T}_E$-formula

$$F: \quad s_1 = t_1 \land \ldots \land s_n = t_n \land s_{n+1} \neq t_{n+1} \land \ldots \land s_m \neq t_m,$$

1. start with the relation $\sim$ in which each **subterm** of $F$ is its own congruence class
2. then, for every equality $s_i = t_i$ in $F$, merge $[s_i]_\sim$ and $[t_i]_\sim$ by
   (a) forming the union $[s_i]_\sim \cup [t_i]_\sim$, and
   (b) propagating the new congruences that arise in this union.
3. $F$ is unsatisfiable if and only if there is a disequality $s_j \neq t_j$ in $F$ such that $s_j \sim t_j$ in the resulting relation.

## Congruence Closure

**Question**: Is $g(a, b) = a \wedge g(g(a, b), b) \neq a$ $\mathcal{T}_E$-satisfiable?

Initial congruence classes: $\{a\}, \{b\}, \{g(a, b)\}, \{g(g(a, b), b)\}$

Using $g(a, b) = a$ to merge: $\{a, g(a, b)\}, \{b\}, \{g(g(a, b), b)\}$

By congruence, $g(a, b) \sim a$ and $b \sim b$ imply $g(g(a, b), b) \sim g(a, b)$

Thus, merge: $\{a, g(a, b), g(g(a, b), b)\}, \{b\}$

This is the congruence closure.
The formula is unsatisfiable, since it contains $g(g(a, b), b) \neq a$, but we have $g(g(a, b), b) \sim a$ in the congruence closure.

## Congruence Closure

**Efficient implementation**: using a directed acyclic graph (DAG)

**Idea**: All subterms are represented in one DAG with different connections for subterm relationship, equalities in formula and deduced equalities.
Once the DAG for subterm relationship is constructed, only connections for equalities need to be updated.

**Details**: The Calculus of Computation

**Original Results**:
[Downey, Sethi, Tarjan 1980], [Nelson, Open 1980]