

SMT SOLVING: DECIDABLE THEORIES

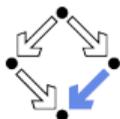
Course “Computational Logic”



Wolfgang Schreiner

Research Institute for Symbolic Computation (RISC)

Wolfgang.Schreiner@risc.jku.at



Theories

- A **theory** T is a set of first-order sentences (closed formulas) that is closed under logical consequence:

$T \models F$ if and only if $F \in T$, for every first-order formula F .

- T may be defined as the set $Th(\mathcal{M}) := \{F \mid \forall M \in \mathcal{M}. M \models F\}$ of all sentences that hold in (every element of) some class \mathcal{M} of structures.
 - Notation $Th(\mathbb{N}, 0, 1, +, \cdot, \leq)$: the theory where $0, 1, +, \cdot, \leq$ are interpreted as the usual natural number constants, functions, predicates.
- T may be also defined as the set $Cn(A) := \{F \mid A \models F\}$ of consequences of some recursively enumerable set A of first-order formulas called **axioms**.
 - A set is recursively enumerable if a machine can produce a list of its elements.
 - If $T = Cn(A)$ for some (finite) set A , then T is (finitely) **axiomatizable**.
 - Undefinability theorem (Gödel/Tarski): $Th(\mathbb{N}, 0, 1, +, \cdot, \leq)$ is not axiomatizable.

A theory describes a “domain of interest”.

Decision Problems

Theories give rise to two related decision problems.

- The problem of **Validity Modulo Theories**:
 - Given: a first-order formula F and a first-order theory T .
 - Decide: does $T \models F$ hold, i.e., is F a logical consequence of T ?
- The problem of **Satisfiability Modulo Theories (SMT)**:
 - Given: a first-order formula F and a first-order theory T .
 - Decide: is $T \cup \{F\}$ satisfiable?
- **Duality**: $T \models F$ if and only if $T \cup \{\neg F\}$ is not satisfiable.

An SMT solver is a decision procedure for the SMT problem (with respect to some theory or combination of theories); thus it also decides the dual validity problem.

Decidable Problems

For certain classes of formulas/theories, the satisfiability problem is decidable.

- Prenex normal form $\forall^n \exists^m$ (validity) or $\exists^n \forall^m$ (satisfiability) (“AE/EA fragment”).
- Formulas without functions and with only unary predicates (“monadic fragment”).
- Every theory with only finite models (e.g., the theory of fixed-size bit vectors).
- Quantifier-free theory of equality with uninterpreted functions (“equational logic”).
- Theory of arrays, theory of recursive data structures.
- Linear arithmetic over integers (“Presburger arithmetic”), natural numbers, reals.
- Theory of reals (“elementary algebra”), complex numbers, algebraically closed fields.
- Logical consequences of equalities over groups, rings, fields (“word problems”).
- ...

As we will see later, also any combination of decidable theories is decidable.

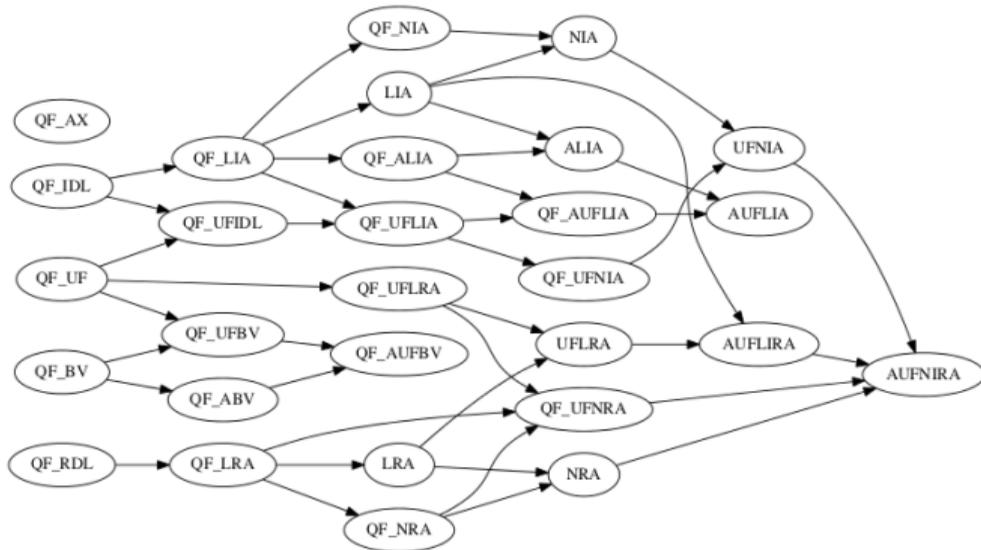
SMT-LIB: The Satisfiability Modulo Theories Library

<http://smt-lib.org>

- A **library** of theories/logics of practical relevance.
- A common input **language** for SMT solvers.
- A repository of **benchmarks**.
- The basis of the yearly **SMT-COMP** competition.
 - <https://smt-comp.github.io>

Many automated/interactive reasoners and program verifiers are equipped with SMT-LIB interfaces to external SMT solvers.

The SMT-LIB Library



- **QF_UF**: Unquantified formulas built over a signature of uninterpreted (i.e., free) sort and function symbols.
- **QF_LIA**: Unquantified linear integer arithmetic. In essence, Boolean combinations of inequations between linear polynomials over integer variables.

Not every logic is decidable, e.g., **NIA** (non-linear integer arithmetic).

Z3: An SMT solver with SMT-LIB Support

Software: <https://github.com/Z3Prover>

Tutorial: <https://microsoft.github.io/z3guide>

- An **SMT solver** developed since 2007 at Microsoft Research.
 - Nikolaj Bjørner and Leonardo de Moura.
 - Open source since 2015 under the MIT License.
- Highly **efficient** and **versatile**.
 - Frequent winner of various divisions of the SMT-COMP series.
 - Backend of various software verification systems (e.g., Microsoft Boogie).
- Uses the **SMT-LIB** language and supports various SMT-LIB logics.
 - Uninterpreted functions, linear arithmetic, fixed-size bit-vectors, algebraic datatypes, arrays, polynomial arithmetic, . . .
- Also supports **quantification**.
 - However, when using quantifiers, the solver is generally incomplete.

Z3 gradually evolves into a full-fledged automated theorem prover.

The SMT-LIB Language

```
; file example1.smt2: Integer arithmetic
(set-logic QF_LIA)
(declare-const x Int)
(declare-const y Int)
(assert (= (- x y) (+ x (- y) 1)))
(check-sat)
(exit)
```

```
debian10!1> z3 example1.smt
unsat
```

```
; file example2.smt2: Getting values or models
(set-logic QF_LIA)
(declare-const x Int)
(declare-const y Int)
(assert (= (+ x (* 2 y)) 20))
(assert (= (- x y) 2))
(check-sat)
(get-value (x y))
(get-model)
(exit)
```

```
debian10!1> z3 example2.smt2
sat
((x 8) (y 6))
(model
  (define-fun y () Int 6)
  (define-fun x () Int 8)
)
```

The SMT-LIB Language

```
; file example3.smt2:
; Modeling sequential code in SSA form
; Buggy swap: int x, y; int t = x; x = y; y = x;
(set-logic QF_UFLIA)

(declare-fun x (Int) Int)
(declare-fun y (Int) Int)
(declare-fun t (Int) Int)
(assert (= (t 0) (x 0)))
(assert (= (x 1) (y 0)))
(assert (= (y 1) (x 1)))
(assert (not
  (and (= (x 1) (y 0))
    (= (y 1) (x 0)))))

(check-sat)
(get-value ((x 0) (y 0) (x 1) (y 1)))
(get-model)
(exit)
```

```
sat
(((x 0) 2)
 (y 0) 3)
((x 1) 3)
((y 1) 3))
(model
  (define-fun y ((x!1 Int)) Int
    (ite (= x!1 0) 3
      (ite (= x!1 1) 3
        3)))
  (define-fun t ((x!1 Int)) Int
    (ite (= x!1 0) 2
      2))
  (define-fun x ((x!1 Int)) Int
    (ite (= x!1 0) 2
      (ite (= x!1 1) 3
        2)))
)
```

Example Application: Program Verification

We can reduce the verification of programs to deciding the satisfiability of formulas.

- Verification of program with respect to pre- and post-condition:

$$\{a[0] = x \wedge a[1] = y \wedge a[2] = z\}$$
$$i = 0; \quad m = a[i];$$
$$i = i+1; \text{ if } (a[i] < m) \text{ } m = a[i];$$
$$i = i+1; \text{ if } (a[i] < m) \text{ } m = a[i];$$
$$\{m \leq x \wedge m \leq y \wedge m \leq z \wedge (m = x \vee m = y \vee m = z)\}$$

- Satisfiability of formula:

$$a[0] = x \wedge a[1] = y \wedge a[2] = z \wedge$$
$$i_0 = 0 \wedge m_0 = a[i_0] \wedge$$
$$i_1 = i_0 + 1 \wedge (\text{if } a[i_1] < m_0 \text{ then } m_1 = a[i_1] \text{ else } m_1 = m_0) \wedge$$
$$i_2 = i_1 + 1 \wedge (\text{if } a[i_2] < m_1 \text{ then } m_2 = a[i_2] \text{ else } m_2 = m_1) \wedge$$
$$\neg(m_2 \leq x \wedge m_2 \leq y \wedge m_2 \leq z \wedge (m_2 = x \vee m_2 = y \vee m_2 = z))$$

The unsatisfiability of the formula establishes the correctness of the program with respect to its specification; a satisfying valuation determines a violating program run.

Program Verification: SMT-LIB Script

```
; file minimum.smt2:
(set-logic QF_UFLIA)

(declare-fun a (Int) Int)
(declare-const x Int) (declare-const y Int) (declare-const z Int)
(declare-const i0 Int) (declare-const i1 Int) (declare-const i2 Int)
(declare-const m0 Int) (declare-const m1 Int) (declare-const m2 Int)

(assert (= (a 0) x)) (assert (= (a 1) y)) (assert (= (a 2) z))
(assert (= i0 0)) (assert (= m0 (a i0)))
(assert (= i1 (+ i0 1))) (assert (ite (< (a i1) m0) (= m1 (a i1)) (= m1 m0)))
(assert (= i2 (+ i1 1))) (assert (ite (< (a i2) m1) (= m2 (a i2)) (= m2 m1)))
(assert (not
  (and (and (and (<= m2 x) (<= m2 y)) (<= m2 z))
    (or (or (= m2 x) (= m2 y)) (= m2 z)))))

(check-sat) (exit)
```

```
debian10!1> z3 minimum.smt2
unsat
```

Program Verification: SMT-LIB Script

```
; file minimum2.smt2:
...
; BUG: ">" rather than "<"
(assert (ite (> (a i2) m1) (= m2 (a i2)) (= m2 m1)))
...
(check-sat) (get-value (x y z i0 m0 i1 m1 i2 m2)) (get-model) (exit)

alan!89> z3 minimum2.smt2
sat
((x 1) (y 0) (z 2) (i0 0) (m0 1) (i1 1) (m1 0) (i2 2) (m2 2))
(model
  (define-fun m0 () Int 1) (define-fun i1 () Int 1) (define-fun m2 () Int 2)
  (define-fun y () Int 0) (define-fun m1 () Int 0) (define-fun i2 () Int 2)
  (define-fun i0 () Int 0) (define-fun z () Int 2) (define-fun x () Int 1)
  (define-fun a ((x!1 Int)) Int (ite (= x!1 0) 1 (ite (= x!1 1) 0 (ite (= x!1 2) 2 1)))))
```

The assignments of a buggy program with an inverted test operation.

The Theory *LRA*: Linear Real Arithmetic

Essentially the SMT-LIB logic QF_LRA.

- *LRA* is a **quantifier-free** first-order theory.
 - Interpretation over the domain \mathbb{R} of real numbers.
 - Only atomic formulas are inequalities $a \leq b$ with polynomials a, b .
 - Integer and rational constants, functions $+$ and \cdot , predicate \leq .
 - Also $-$, $<$, $>$, \geq , $=$ are allowed: $a - b$ can be reduced to $a + (-1) \cdot b$; $\{<, >\}$ can be reduced to $\{=, \leq, \geq\}$; $=$ can be reduced to $\{\leq, \geq\}$; \geq can be reduced to \leq .
 - **Linear**: in every multiplication $a \cdot b$, a must be a constant.
- ***LRA*-Satisfiability** of formula F :
 - Convert F into its disjunctive normal form $C_1 \vee \dots \vee C_n$.
 - F is *LRA*-satisfiable if and only if some C_i is *LRA*-satisfiable.

To decide the *LRA*-Satisfiability of F , it suffices to decide the satisfiability of a conjunction of (possibly negated) inequalities $a \leq b$ with linear polynomials a, b (in the following, we only consider conjunctions of unnegated inequalities).

Deciding *LRA*-Satisfiability by Fourier-Motzkin Elimination

Joseph Fourier (1826), Theodore Motzkin (1936).

```
function FOURIERMOTZKIN( $F$ )      ▶  $F$  is a conjunction of inequalities  $a \leq b$  with linear polynomials  $a, b$   
  while  $F$  contains a variable do  
    Choose some variable  $x$  in  $F$   
    Arithmetically transform every inequality in which  $x$  occurs into the form  $a \leq x$  or  $x \leq b$   
    Let  $A$  be the set of all  $a$  where  $a \leq x$  is an inequality in  $F$ .  
    Let  $B$  be the set of all  $b$  where  $x \leq b$  is an inequality in  $F$ .  
    Remove from  $F$  all inequalities of form  $a \leq x$  and  $x \leq b$ .  
    Add to  $F$  a (possibly simplified version of the) inequality  $a \leq b$  for every pair  $(a, b) \in A \times B$   
  end while  
  if  $F$  contains a constraint  $c_1 \leq c_2$  with constant  $c_1$  greater than constant  $c_2$  then  
    return false                                ▶ unsatisfiable  
  else  
    return true                                  ▶ satisfiable  
  end if  
end function
```

Example

LRA-Satisfiability of formula $F : \Leftrightarrow (z \leq x - y) \wedge (x + 2 \cdot y \leq 5) \wedge (y \leq 4 \cdot z - 2 \cdot x)$

- **Eliminate x :**
 - Transform: $(z + y \leq x) \wedge (x \leq 5 - 2 \cdot y) \wedge (x \leq 2 \cdot z - \frac{1}{2} \cdot y)$
 - Eliminate: $(z + y \leq 5 - 2 \cdot y) \wedge (z + y \leq 2 \cdot z - \frac{1}{2} \cdot y)$
 - Simplify: $(z \leq 5 - 3 \cdot y) \wedge (\frac{3}{2} \cdot y \leq z)$
- **Eliminate z :**
 - Transform: $(\frac{3}{2} \cdot y \leq z) \wedge (z \leq 5 - 3 \cdot y)$
 - Eliminate: $(\frac{3}{2} \cdot y \leq 5 - 3 \cdot y)$
 - Simplify: $(\frac{9}{2} \cdot y \leq 5)$
- **Eliminate y :**
 - Transform: $(y \leq \frac{10}{9})$
 - Eliminate: \top

F is *LRA*-satisfiable (by, e.g., $y := 0 \in [-\infty, \frac{10}{9}]$, $z := 0 \in [0, 5]$, $x := 0 \in [0, 0]$).

Example

LRA-Satisfiability of formula $F : \Leftrightarrow (x \leq y) \wedge (x \leq z) \wedge (y + 2 \cdot z \leq x) \wedge (1 \leq x)$

- **Eliminate x :**
 - Transform: $(y + 2 \cdot z \leq x) \wedge (1 \leq x) \wedge (x \leq y) \wedge (x \leq z)$
 - Eliminate: $(y + 2 \cdot z \leq y) \wedge (y + 2 \cdot z \leq z) \wedge (1 \leq y) \wedge (1 \leq z)$
 - Simplify: $(z \leq 0) \wedge (y + z \leq 0) \wedge (1 \leq y) \wedge (1 \leq z)$
- **Eliminate z :**
 - Transform: $(1 \leq z) \wedge (z \leq 0) \wedge (z \leq -y) \wedge (1 \leq y)$
 - Eliminate: $(1 \leq 0) \wedge (1 \leq -y) \wedge (1 \leq y)$
 - Simplify: $(1 \leq 0) \wedge (y \leq -1) \wedge (1 \leq y)$
- **Eliminate y :**
 - Transform: $(1 \leq y) \wedge (y \leq -1) \wedge (1 \leq 0)$
 - Eliminate: $(1 \leq -1) \wedge (1 \leq 0)$

F is *LRA*-unsatisfiable.

The Theory *EUF*: Equality with Uninterpreted Functions

Essentially the SMT-LIB logic QF_UF.

- *EUF* is a **quantifier-free** first-order theory with **only predicate “=”**.
 - Syntax: an arbitrary propositional combination of equalities.
 - Semantics: the fixed interpretation of “=” as “equality”.
- *EUF* is sufficient to also deal with **arbitrary other predicates** in a formula F :
 - Introduce a fresh constant T and a fresh function f_p for every other predicate p .
 - Transform every atomic formula $p(\dots)$ into an equality $f_p(\dots) = T$.
 - Formula F is satisfiable if and only if its transformed version is *EUF*-satisfiable.
- ***EUF*-satisfiability** of formula F :
 - Convert F into its disjunctive normal form $C_1 \vee \dots \vee C_n$.
 - F is *EUF*-satisfiable if and only if some C_i is *EUF*-satisfiable.

It suffices to decide the satisfiability of a conjunction of (negated) equalities.

Deciding *EUF*-Satisfiability by Congruence Closure

Greg Nelson and Derek C. Oppen (1980).

- $R \subseteq S \times S$ is a **congruence relation** if it is an equivalence relation
 - R is reflexive, symmetric, and transitivethat satisfies for every n -ary function f the congruence condition of f :
 - $\forall t, u \in S^n. (\forall 1 \leq i \leq n. R(t_i, u_i)) \Rightarrow R(f(t), f(u))$
- The **congruence closure** R^c is the smallest congruence relation covering R :
 - R^c is a congruence relation with $R \subseteq R^c$
 - $\forall R'. (R' \text{ is a congruence relation with } R \subseteq R') \Rightarrow (R^c \subseteq R')$
- ***EUF*-satisfiability** of formula $F : \Leftrightarrow (\bigwedge_{i=1}^n t_i = u_i) \wedge (\bigwedge_{j=n+1}^{n+m} t_j \neq u_j)$:
 - Let R be the relation $\{(t_i, u_i) \mid 1 \leq i \leq n\}$ on the set S of subterms of F .
 - F is *EUF*-satisfiable if and only if $\forall n+1 \leq j \leq n+m. \neg R^c(t_j, u_j)$.

To decide the *EUF*-satisfiability of F , it suffices to compute the congruence closure of the term equalities in F and check that it is compatible with the term inequalities.

Congruence Closure: Basic Idea

We compute the congruence closure by partitioning S into classes of congruent terms.

- **Partition** $S/R^c := \{[t]_{R^c} \mid t \in S\}$.
 - Congruence class $[t]_{R^c}$: $R^c(t, u)$ if and only if $[t]_{R^c} = [u]_{R^c}$.
 - Given F with equations $t_1 = u_1, \dots, t_n = u_n$, compute partitions $P_0, P_1, \dots, P_n = S/R^c$.
 - P_0 : every element of S represents a separate congruence class.
 - P_{i+1} : determined from P_i by merging $[t_{i+1}]$ and $[u_{i+1}]$, i.e., by forming their union and propagating new congruences that arise within this union.
- **Example**: satisfiability of $F : \Leftrightarrow f(a, b) = a \wedge f(f(a, b), b) \neq a$
 - Set $S := \{a, b, f(a, b), f(f(a, b), b)\}$, single equation $f(a, b) = a$.
 - $P_0 := \{\{a\}, \{b\}, \{f(a, b)\}, \{f(f(a, b), b)\}\}$
 - $P_1 := \{\{b\}, \{a, f(a, b), f(f(a, b), b)\}\}$
 - Union of $[f(a, b)]$ and $[a]$: $\{\{b\}, \{a, f(a, b)\}, \{f(f(a, b), b)\}\}$
 - Propagation: $[f(a, b)] = [a]$ implies $[f(f(a, b), b)] = [f(a, b)]$
 - F is **EUF-unsatisfiable**: $[f(f(a, b), b)] = [a]$.

Congruence Closure: Algorithm

```
function CONGRUENCECLOSURE( $S, R$ )  
   $P := \{\{t\} \mid t \in S\}$   $\triangleright$  compute partition  $P := S/(R^c)$   
  for  $(t, u) \in R$  do  
     $P := \text{MERGE}(S, P, t, u)$   
  end for  $\triangleright$  return relation determined by  $P$   
  return  $\{(t, u) \in S \times S \mid \text{FIND}(P, t) = \text{FIND}(P, u)\}$   
end function
```

```
function CONGRUENT( $P, t, u$ )  
  if  $t$  and  $u$  are  $f(t_1, \dots, t_n)$  and  $f(u_1, \dots, u_n)$  then  
    return  $\forall 1 \leq i \leq n. \text{FIND}(P, t_i) = \text{FIND}(P, u_i)$   
  else  
    return false  
  end if  
end function
```

P can be represented by a “disjoint-set” data structure with efficient merge/find algorithms.

```
function MERGE( $S, P, t, u$ )  $\triangleright$  merge  $[t]$  and  $[u]$   
   $p_t, p_u := \text{FIND}(P, t), \text{FIND}(P, u)$   
  if  $p_t = p_u$  return  $P$   
   $P := (P \setminus \{p_t, p_u\}) \cup \{p_t \cup p_u\}$   
  for  $(t_1, t_2) \in S \times S$  do  
     $p_1, p_2 := \text{FIND}(P, t_1), \text{FIND}(P, t_2)$   
    if  $p_1 \neq p_2 \wedge \text{CONGRUENT}(P, t_1, t_2)$  then  
       $P := \text{MERGE}(P, t_1, t_2)$   
    end if  
  end for  
  return  $P$   
end function
```

```
function FIND( $P, t$ )  $\triangleright$  find congruence class  $[t] \in P$   
  choose  $p \in P$  with  $t \in p$   
  return  $p$   
end function
```

Congruence Closure: More Examples

- **Example:** satisfiability of $F : \Leftrightarrow f(f(f(a))) = a \wedge f(f(f(f(f(a)))) = a \wedge f(a) \neq a$.
 - $P_0 := \{\{a\}, \{f(a)\}, \{f^2(a)\}, \{f^3(a)\}, \{f^4(a)\}, \{f^5(a)\}\}$
 - $P_1 := \{\{a, f^3(a)\}, \{f(a), f^4(a)\}, \{f^2(a), f^5(a)\}\}$
 - Union of $[f^3(a)]$ and $[a]$: $\{\{a, f^3(a)\}, \{f(a)\}, \{f^2(a)\}, \{f^4(a)\}, \{f^5(a)\}\}$
 - Propagation: $[f^3(a)] = [a]$ implies $[f^4(a)] = [f(a)]$ and $[f^5(a)] = [f^2(a)]$.
 - $P_2 := \{\{a, f(a), f^2(a), f^3(a), f^4(a), f^5(a)\}\}$
 - Union of $[f^5(a)]$ and $[a]$: $\{\{a, f^2(a), f^3(a), f^5(a)\}, \{f(a), f^4(a)\}\}$
 - Propagation: $[f^2(a)] = [a]$ implies $[f^3(a)] = [f(a)]$.
 - F is **EUF-unsatisfiable**: $[f(a)] = [a]$.
- **Example:** satisfiability of $F : \Leftrightarrow f(x) = y \wedge x \neq f(y)$.
 - $P_0 := \{\{x\}, \{y\}, \{f(x)\}, \{f(y)\}\}$
 - $P_1 := \{\{x\}, \{y, f(x)\}, \{f(y)\}\}$
 - Union of $[f(x)]$ and $[y]$: $\{\{x\}, \{y, f(x)\}, \{f(y)\}\}$
 - No more propagation.
 - F is **EUF-satisfiable**: $[x] \neq [f(y)]$.

Congruence Closure in OCaml

```
let congruent eqv (s,t) = (* Test whether subterms are congruent under an equivalence. *)
  match (s,t) with
  | Fn(f,a1),Fn(g,a2) -> f = g & forall2 (equivalent eqv) a1 a2
  | _ -> false;;

let rec emerge (s,t) (eqv,pfn) = (* Merging of terms, with congruence closure. *)
  let s' = canonize eqv s and t' = canonize eqv t in
  if s' = t' then (eqv,pfn) else
  let sp = tryapply1 pfn s' and tp = tryapply1 pfn t' in
  let eqv' = equate (s,t) eqv in
  let st' = canonize eqv' s' in
  let pfn' = (st' |-> union sp tp) pfn in
  itlist (fun (u,v) (eqv,pfn) ->
    if congruent eqv (u,v) then emerge (u,v) (eqv,pfn)
    else eqv,pfn)
  (allpairs (fun u v -> (u,v)) sp tp) (eqv',pfn');;
```

EUF-Satisfiability/Validity in OCaml

```
let predecessors t pfn =
  match t with
  | Fn(f,a) -> itlist (fun s f -> (s |-> insert t (tryapply1 f s)) f) (setify a) pfn
  | _ -> pfn;;
let ccsatisfiable fms = (* Satisfiability of conjunction of ground equations and inequations. *)
  let pos,neg = partition positive fms in
  let eqps = map dest_eq pos and eqns = map (dest_eq ** negate) neg in
  let lrs = map fst eqps @ map snd eqps @ map fst eqns @ map snd eqns in
  let pfn = itlist predecessors (unions(map subterms lrs)) undefined in
  let eqv,_ = itlist emerge eqps (unequal,pfn) in
  forall (fun (l,r) -> not(equivalent eqv l r)) eqns;;
let ccvalid fm = (* Validity checking a universal formula. *)
  let fms = simpdnf(askolemize(Not(generalize fm))) in
  not (exists ccsatisfiable fms);;

# ccvalid <<f(f(f(f(f(c)))) = c /\ f(f(f(c))) = c ==> f(c) = c \/ f(g(c)) = g(f(c))>>;
- : bool = true
# ccvalid <<f(f(f(f(c)))) = c /\ f(f(c)) = c ==> f(c) = c>>;
- : bool = true
```

The Theory E : Equality Logic

EUF without uninterpreted functions (i.e., only with constants).

- **Decision of E -satisfiability:**

- Computation of congruence closure without the need to propagate congruences:

```
function MERGE( $S, P, t, u$ )  
   $p_t, p_u := \text{FIND}(P, t), \text{FIND}(P, u)$   
  return  $(P \setminus \{p_t, p_u\}) \cup \{p_t \cup p_u\}$   
end function
```

▷ equals P , if $p_t = p_u$

- **Ackermann's Reduction:** transformation of an EUF -formula into an E -formula.

- Replace every function application $f(t_1, \dots, t_n)$ by a fresh constant f_{t_1, \dots, t_n} .
- For every pair of applications $f(t_1, \dots, t_n)$ and $f(u_1, \dots, u_n)$, add the constraint

$$(t_1 = u_1 \wedge \dots \wedge t_n = u_n) \Rightarrow f_{t_1, \dots, t_n} = f_{u_1, \dots, u_n}$$

- The result is E -satisfiable if and only if the original formula is EUF -satisfiable.

The theory E needs larger formulas but has a simpler decision algorithm than EUF .

E-Satisfiability: Example

EUF-satisfiability of formula $F : \Leftrightarrow x_2 = x_3 \wedge f(x_1) = f(x_3) \wedge f(x_1) \neq f(x_2)$

- Ackermann's reduction to *E*-formula F' :

$$x_2 = x_3 \wedge f_1 = f_3 \wedge f_1 \neq f_2 \wedge$$

$$(x_1 = x_2 \Rightarrow f_1 = f_2) \wedge (x_1 = x_3 \Rightarrow f_1 = f_3) \wedge (x_2 = x_3 \Rightarrow f_2 = f_3)$$

- Disjunctive normal form of F' :

$$(\underline{x_2 = x_3} \wedge f_1 = f_3 \wedge f_1 \neq f_2 \wedge x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge \underline{x_2 \neq x_3}) \vee$$

$$(x_2 = x_3 \wedge f_1 = f_3 \wedge f_1 \neq f_2 \wedge x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge f_2 = f_3) \vee$$

$$(\underline{x_2 = x_3} \wedge f_1 = f_3 \wedge f_1 \neq f_2 \wedge x_1 \neq x_2 \wedge f_1 = f_3 \wedge \underline{x_2 \neq x_3}) \vee$$

$$(x_2 = x_3 \wedge f_1 = f_3 \wedge f_1 \neq f_2 \wedge x_1 \neq x_2 \wedge f_1 = f_3 \wedge f_2 = f_3) \vee$$

$$(x_2 = x_3 \wedge f_1 = f_3 \wedge \underline{f_1 \neq f_2} \wedge \underline{f_1 = f_2} \wedge x_1 \neq x_3 \wedge x_2 \neq x_3) \vee$$

$$(x_2 = x_3 \wedge f_1 = f_3 \wedge \underline{f_1 \neq f_2} \wedge \underline{f_1 = f_2} \wedge x_1 \neq x_3 \wedge f_2 = f_3) \vee$$

$$(x_2 = x_3 \wedge f_1 = f_3 \wedge \underline{f_1 \neq f_2} \wedge \underline{f_1 = f_2} \wedge f_1 = f_3 \wedge x_2 \neq x_3) \vee$$

$$(x_2 = x_3 \wedge f_1 = f_3 \wedge \underline{f_1 \neq f_2} \wedge \underline{f_1 = f_2} \wedge f_1 = f_3 \wedge f_2 = f_3)$$

E-Satisfiability: Example

E-satisfiability of DNF of F' : only two clauses do not have conflicting literals.

- Satisfiability of $(x_2 = x_3 \wedge f_1 = f_3 \wedge f_1 \neq f_2 \wedge x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge f_2 = f_3)$:
 - $P_0 := \{\{x_1\}, \{x_2\}, \{x_3\}, \{f_1\}, \{f_2\}, \{f_3\}\}$
 - $P_1 := \{\{x_1\}, \{x_2, x_3\}, \{f_1\}, \{f_2\}, \{f_3\}\}$
 - $P_2 := \{\{x_1\}, \{x_2, x_3\}, \{f_1, f_3\}, \{f_2\}\}$
 - $P_3 := \{\{x_1\}, \{x_2, x_3\}, \{f_1, f_2, f_3\}\}$
 - $[f_1] = [f_2]$: clause is *E*-unsatisfiable.
- Satisfiability of $(x_2 = x_3 \wedge f_1 = f_3 \wedge f_1 \neq f_2 \wedge x_1 \neq x_2 \wedge f_1 = f_3 \wedge f_2 = f_3)$:
 - $P_0 := \{\{x_1\}, \{x_2\}, \{x_3\}, \{f_1\}, \{f_2\}, \{f_3\}\}$
 - $P_1 := \{\{x_1\}, \{x_2, x_3\}, \{f_1\}, \{f_2\}, \{f_3\}\}$
 - $P_2 := \{\{x_1\}, \{x_2, x_3\}, \{f_1, f_3\}, \{f_2\}\}$
 - $P_3 := \{\{x_1\}, \{x_2, x_3\}, \{f_1, f_2, f_3\}\}$
 - $[f_1] = [f_2]$: clause is *E*-unsatisfiable.

DNF of F' is *E*-unsatisfiable, thus F is *EU*F-unsatisfiable.