Automatic Software Verification **Spring Semester, 2015**

Lecture 3: March 24, 2015

Lecturer: Prof. Mooly Sagiv *Scribe: Asya Frumkin and Aviv Kuvent*

1 The SAT problem

The boolean satisfiability problem is the problem of determining if a given boolean formula ϕ is satisfiable, i.e if there exists an assignment to the formula's variables $p_1, p_2, ..., p_n$ such that the formula evaluates to true. Since every variable can be true or false, there are 2^n possible assignments. This is a NP-hard problem. However, we can use SAT solver tools to automatically decide whether a formula is satisfiable in a reasonably efficient manner.

2 The SMT problem

2.1 Problem description

In this lesson we will consider an even "harder" problem than SAT — the Satisfiability Modulo Theories (SMT) problem. Now, instead of considering boolean assignments to variables $p_1, p_2, ..., p_n$, we consider a formula ϕ where $p_1, p_2, ..., p_n$ are constraints, expressed as ground formulas (formulas with no quantifiers and no variables, only constants) in first-order logic. The given constraints can be evaluated to true/false, depending on the assignment to their variables (which are not necessarily binary). We are looking for a satisfying assignment to the formula.
Often, these constraints are expressed in a first-order theory, called a background theory. For example, when we consider piplined microprocessors the background theories that can be used are theory of equality and atoms. For timed automata we might want to use theory of integers and theory of reals. Note that, unlike the formula in question, a background theory can have variables and quantifiers.
We will describe a few applications:

Difference constraints We are given boolean combinations of inequalities of the form $a \leq b + k$ where a and b are free constants and $k \in Z$. The background theory is integer number theory.

A good example for that is the problem of parallel task scheduling, where we would want to use the following difference constraints on the start times x_k of the given tasks:

- $x_i + duration(i) \leq x_j$ for each two tasks i, j where task i should be finished before task j.

- $x_j + duration(j) - x_i \leq t$ for all tasks i, j, which expresses the fact that the total duration shouldn't exceed a maximum time t.

Uninterpreted functions The constraint formulas can use uninterpreted functions, meaning functions that we care only about their name and arity. For example, the following is a theory about memory locations:

$$read(write(X, Y, Z), Y) = Z$$
$$W \neq Y \implies read(write(X, Y, Z), W) = read(X, W)$$

The theory states that if you write a value Z to a memory location, then you'll read Z from the same location immediately afterwards. Also, if we are writing to a different memory location from the one we are reading from, the result of the read is not affected by the write operation.
Based on this theory we want to prove the following ground formula. It describes an access to an array a in a specific program, and has x and y as constants:

$$x + 2 = y \implies f(read(write(a, x, 3), y - 2) = f(y - x + 1)$$

We can easily see that both the left-hand side and the right-hand side of the implied equation are equal to $f(3)$ and therefore the formula is satisfiable.

Bounded Model Checking (BMC) Given a finite transition system M and a temporal property ϕ we want to know if M satisfies ϕ after up to k transitions. This can be applied to software by converting a program to a set of constraints which are checked against assertions about the program's reachable states. The constraints and the (negation of) assertions are given as a CNF formula to a SAT solver, and if a counterexample is found, it means that the program has a bug.

Definition 2.1 (Skolem-Lowenheim formula) *A Skolem-Lowenheim formula is a first-order formula that is in prenex normal form (PNF) and all exists quantifiers are before all forall quantifiers. For example:*

$$\phi = \exists x, y \, \forall z, w : P(x, y) \wedge \neg P(z, w)$$

Lemma 2.2 *A skolem-Lowenheim formula can be solved by a SAT solver.*

Proof sketch:
In order to use SAT solver on this formula (where the variables are from an infinite domain) we need to apply several reductions. We will use the following theorem:

Theorem 2.3 (Herbrand's theorem) *Let $\phi = \forall y_1, ..., y_n F(y_1, ..., y_n)$ be a first-order formula with $F(y_1, ..., y_n)$ quantifier-free, then ϕ is satisfiable iff there exists a finite sequence of terms t_{ij} with $1 \leq i \leq k$ and $1 \leq j \leq n$ such that $F(t_{11}, ..., t_{1n}) \wedge ... \wedge F(t_{k1}, ..., t_{kn})$ is satisfiable.*

First, we will convert the formula ϕ to Skolem normal form (SNF), by replacing the variables x, y with the constants c_x, c_y. The resulting formula ϕ' is satisfiable iff the

original formula ϕ is satisfiable. Then, by Herbrand's theorem ϕ' is satisfiable iff the following formula is satisfiable:

$$\psi = P(c_x, c_y) \wedge \neg P(c_x, c_x) \wedge \neg P(c_x, c_y) \wedge \neg P(c_y, c_x) \wedge \neg P(c_y, c_y)$$

Note that we restrict the domain to contain only c_x and c_y. For the language that contains both those constants and no function symbols, these are the only terms. Now, we can assume that the domain is boolean and feed the resulting formula to the SAT solver.

2.2 Lazy approach for SMT

The lazy approach for solving SMT is based on the integration of a SAT solver and a Theory solver (T-solver), a decision procedure able to handle sets of atomic constraints in a specific theory. It is a modular and flexible method, however it's main disadvantage is that theory information is coded in a straightforward (lazy) manner to the SAT solver instead of guiding the search.

Example 2.4
Given a conjunction of a set of formulas:

$$g(a) = c \wedge f(g(a)) \neq f(c) \vee g(a) = d \wedge c \neq d$$

we pick unique numbers to all the formulas, replace them by propositions, and send the result to the SAT solver:

$$\{1, \neg 2 \vee 3, \neg 4\}$$

The solver returns a satisfying assignment: $\{1, \neg 2, \neg 4\}$.
Now, we use a T-solver to analyze the model that was found by the SAT solver. In this case, it finds out that $\{1, \neg 2\}$ *is E-unsatisfiable. We add this constraint to the set of formulas sent to the SAT solver:*

$$\{1, \neg 2 \vee 3, \neg 4, \neg 1 \vee 2\}$$

Now, the SAT solver returns the following assignment: $\{1, 2, 3, \neg 4\}$.
Again, we feed it to the T-solver, that finds that $\{1, 3, \neg 4\}$ *is E-unsatisfiable. We add this constraint:*

$$\{1, \neg 2 \vee 3, \neg 4, \neg 1 \vee 2, \neg 1 \vee \neg 3 \vee 4\}$$

The SAT solver returns UNSAT. ∎

3 Decision Procedures

3.1 Undecidability of first-order logic

Definition 3.1 (Decision procedure) *A decision procedure is an algorithm that, given a decision problem, terminates with a correct yes/no answer.*

Definition 3.2 (Decision procedure for first-order logic) *A decision procedure for first-order logic is a terminating algorithm for determining the validity (satisfiability) of a formula in a given logic.*

Definition 3.3 (Decidable problem) *A decidable problem is a problem for which there exists a terminating algorithm which solves every instance of the problem.*

Definition 3.4 (Decidable logic) *A decidable logic is a logic that has a decision procedure for every formula.*

The following theorem is one of the main results in computer science:

Theorem 3.5 (Church's theorem) *First-order logic is undecidable.*

This theorem is less known, but is also very important:

Theorem 3.6 (Trakhtenbrot's theorem) *First-order logic over finite models is undecidable.*

The second theorem might look trivial, but in fact, when you consider finite sets the problem is even harder than the general case [1]. If the formula is not satisfiable we can enumerate over all models (there is a finite number of them) and find the relevant assignment [2]. However, we cannot prove that a formula is always correct — the validity problem of finite models is not recursively enumerable and methods like resolution cannot be applied in this case.

In order to obtain a decision procedure, in spite of the undecidability of first-order logic, we can do one of the following:

- Limit the logic — for example, use formulas of the form $\exists\forall$, which are decidable.
- Limit the class of intended models - for example, consider only models that involve strings.
- Consider a specific theory, e.g the theory of real numbers.

Theorem 3.7 (Rabin's theorem) *Monadic second-order theory of infinite binary trees is decidable.*

This is an example of how limiting the model can make a problem decidable. However, decision procedure is complex, fragile and not very practical. A tool called MONA [3] is based on this observation.

Another problem that becomes decidable, if we limit the domain, was proved by Tarski:

Theorem 3.8 (Tarski's theorem) *First-order theory of reals is decidable.*

The decision procedure that he found is impractical, but it exists. However, if we limit the domain to integers, the problem is again undecidable.

In order to prove decidability we can rely on a small model theorem for a specific logic (if it exists). Such a theorem claims that every satisfiable formula over this logic has a model whose size is proportional to the size of the formula.

The proof can show a direct decision procedure, or show a reduction to another decidable logic.

We will see now a direct decision procedure, that is implemented in z3.

3.2 An example of a decision procedure

The problem specification:
We consider only universal formulas, and allow a fixed scheme of first-order formulas T. We want to determine if $T \models F$, meaning if the formula F is valid in the theory T.

Definition 3.9 (Theory of uninterpreted functions (EUF)) *For each function symbol f:*

$$\forall X, Y : X = Y \implies f(X) = f(Y)$$

It was proved to be decidable by Ackerman using an impractical straitforward algorithm, that simply enumerates all options. He based his algorithm on the following theorem:

Theorem 3.10 (Small model property of uninterpreted functions) *Every satisfiable EUF formula has a model of size k, where k is the number of distinct function application terms.*

This implies that the number of models that should be checked is finite, but the algorithm is exponential, since we need to enumerate all the (finite number of) options for the elements in the formula. The next algorithm is more efficient, since it does no such enumeration.

In order to determine the validity of a formula, we will check the satisfiability of it's negation. This technique is called **proof by refutation**. When the formula is quantifier free it's enough to consider conjunction of literals. For example:

$$\forall A, B : f(A, B) = A \implies f(f(A, B, B) = A$$

We will prove that the negation is not satisfiable:

$$f(ca, cb) = ca \wedge f(f(ca, cb, cb) = ca$$

Note, that the negation is an existential function, and hence we can use specific constants without the quantifier.

Thus, when we are given a set of equation and inequation formulas over the theory of uninterpreted functions, we create a directed acyclic graph (DAG), where each node represents a term in the formula set. Edges are drawn from the function terms to their arguments. For example:

We will unify equal terms and their consequences. If we find a contradiction with the inequalities, then we report UNSAT, and hence the original formula is valid. This method is both sound and complete.

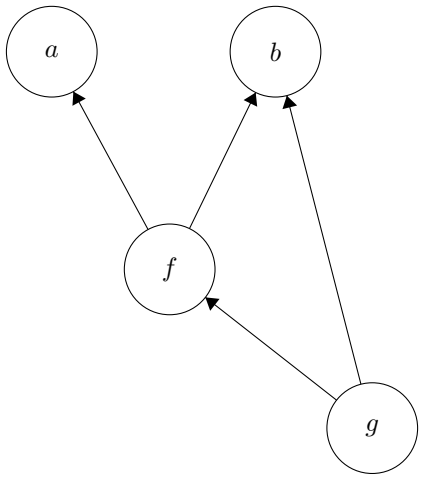


Figure 1: A DAG that corresponds to the term g(f(a,b),b)

3.3 The congruent closure problem

Definition 3.11 (Congruence relation) *A congruence relation is an equivalence relation on an algebraic structure that preserves the operations on the structure.*

For example, modulo is a congruence relation on the set of integers, as it is compatible with addition and multiplication over the integers.

Definition 3.12 (Congruence closure) *The congruence closure of an equivalence relation R is the smallest congruence relation containing R.*

Input:

- A finite labeled directed graph where nodes are labeled by function symbols and edges are also labeled.
- A binary relation R

Output: The congruence closure of R.

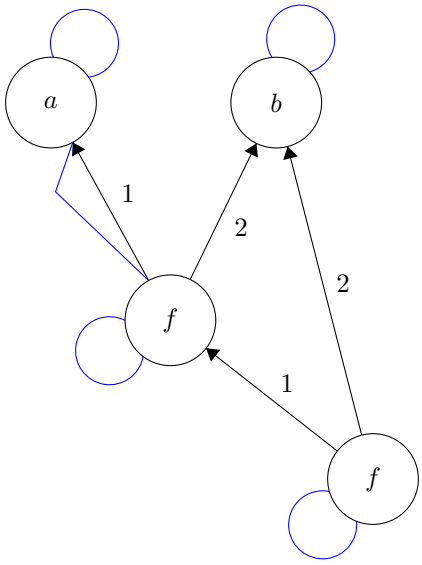
Two nodes are congruent under R if they have the same label and their arguments are in R.

Example 3.13

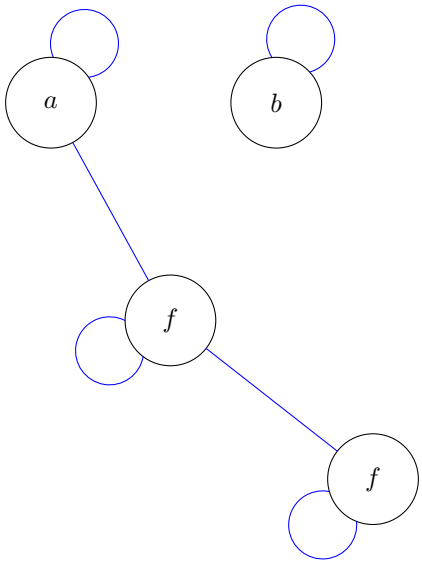
$$f(a,b) = a \wedge f(f(a,b),b) \neq a$$

This formula is not satisfiable. Here, the binary relation is the equality relation, and the labels of the edges are according to the index of the function arguments. Congruent nodes are connected by blue edges.

This is the initial state:



After applying transitivity the closure is:



∎

Example 3.14

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

This formula is not satisfiable. Here, the binary relation is the equality relation. We start by looking at the DAG corresponding to this formula:

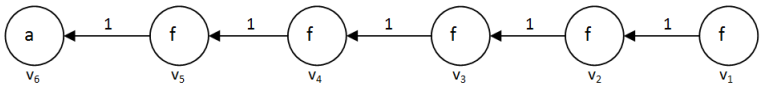


Figure 2: A DAG that corresponds to the term f(f(f(f(f(a)))))

Let R be the relation described by this formula: $(v_1, v_6), (v_3, v_6)$. *We then construct R', the congruent closure of R:*
$(v_1, v_6) \in R$ *and* $(v_3, v_6) \in R \implies (v_2, v_5)$ *are congruent under R, i.e.* $(v_2, v_5) \in R'$.
$(v_2, v_5) \in R' \implies (v_1, v_4) \in R'$, *since R' is closed under congruence.*
$(v_1, v_4) \in R'$ *and* $(v_1, v_6) \in R' \implies (v_4, v_6) \in R'$, *from transitivity of R'.*
$(v_4, v_6) \in R' \implies (v_3, v_5) \in R'$, *since R' is closed under congurence.*
$(v_3, v_5) \in R'$ *and* $(v_3, v_6) \in R' \implies (v_5, v_6) \in R'$, *from transitivity of R'.*
$(v_5, v_6) \in R'$ *and* $(v_2, v_5) \in R' \implies (v_2, v_6) \in R'$, *from transitvity of R'.*
Finally, from transitivity, this means that all vertices are equivalent in the congruence closure R'.
We found that $f(a) = a$, *therefore the formula is unsatisfiable.* ∎

3.4 Computing Congurence Closure

We now consider the problem of computing the congruence closure. We represent an equivalence relation by the set of its equivalence classes. In [9], Nelson and Oppen showed the following method for computing congruence closure:
Let u and v be vertices of a graph G, and let:

- UNION(u,v) — the procedure which combines the equivalence classes of vertices u and v.
- FIND(u) — the procedure which returns the unique name of the equivalence class of vertex u.
- label(v) — the label of vertex v.
- $\delta(v)$ — the outdegree of vertex v.
- v[i] — the i-th successor of vertex v.

Let R be a congruence relation on the vertices of the graph G.
The following procedure computes the congruence closure of the relation $R \cup \{u, v\}$:
MERGE(u,v):

1. If FIND(u) = FIND(v) — return.
2. Let P_u be the set of all predecessors of all vertices equivalent to u and let P_v be the set of all predecessors of all vertices equivalent to v.
3. Call UNION(u,v).

4. For each (x,y) such that $x \in P_u$ and $y \in P_v$, if CONGURENT(x,y) = True and FIND(x) $\neq$ FIND(Y) then call MERGE(x,y).

CONGRUENT(u,v):

1. If $label(u) \neq label(v)$ — return False.
2. If $\delta(u) \neq \delta(v)$ — return False.
3. $\forall 1 \leq i \leq \delta(u)$, if FIND(u[i]) $\neq$ FIND(v[i]) — return False.
4. Return True.

Complexity:
The worst case run time of the MERGE(u,v) procedure is $O(m^2)$, where m is the number of edges in G.
In the algorithm introduced in [4], the vertices are stored in a hash table keyed by the list of equivalence classes of their successors. Step (4) in the MERGE procedure (finding the congruent pairs) can then be implemented so that only the vertices in the shorter predecessor list will need to be rehashed. The average run time using this algorithm is $O(m \log^2 m)$.

4 Applications of Congruence Closure

4.1 Application 1 — EUF

The decision problem for quantifier-free EUF theory can be reduced to the congurence closure problem [9].
The following algorithm will determine the satsifiability of the conjunction of equalities and inequalities of the form:

$$t_1 = u_1 \wedge ... \wedge t_p = u_p \wedge r_1 \neq s_1 \wedge ... \wedge r_q \neq s_q$$

1. Construct a graph G which corresponds to the set of all terms appearing in the conjunction. For each term i appearing in the conjunction, let $\tau(i)$ be the vertex in G representing term i.
2. Let R be the identity relation on the vertices of G.
3. $\forall 1 \leq i \leq p$, call MERGE($\tau(t_i), \tau(u_i)$).
4. If for some $1 \leq j \leq q, \tau(r_j)$ is equivalent to $\tau(s_j)$, report UNSAT.
5. Report SAT.

Extensions and Improvements:
Lahiri et al suggest in [5] some improvements to handiling the decision problem of EUF formulas. These improvements rely on identifying "positive terms" (positive equality) in a formula — terms which only appear in the context of equality in a formula, and not in the context of inequality or in the controling formula of an If-then-else (ITE) term (which is implicitly negated when choosing the "else" branch).
Positive equality analysis is used to reduce the number of interpretations required to check the validity of a formula in EUF theory.

4.2 Application 2 — Theory of Lisp List Structure

The theory of Lisp List Structure is the quantifier-free theory of equality with function symbols car, cdr and cons and predicate symbol atom (well known functions and predicates of Lisp).
This theory satisfies the following axioms:

- car(cons(x,y)) = x
- cdr(cons(x,y)) = y
- ¬atom(x) $\implies$ cons(car(x),cdr(x)) = x
- ¬atom(cons(x,y))

Note that we do not restrict the domain of the Lisp functions to non-circular lists (i.e. a formula such as car(x) = x is SAT). If we include axioms enforcing acyclity of list structure, and exclude uninterpreted function symbols, there is a faster algorithm (described in [6]) for determining the satisfiability of conjunctions.
The following algorithm (described in [9]) will determine the satisfiability of a conjunction of the form:

$$atom(u_1) \wedge ... \wedge atom(u_q) \wedge v_1 = w_1 \wedge ... \wedge v_r = w_r \wedge x_1 \neq y_1 \wedge ... \wedge x_s \neq y_s$$

where the terms in the literals may contain uninterpreted function symbols as well as the function car, cdr and cons:

1. Construct a graph G which correpsonds to the set of all terms appearing in the conjunction. For each term i appearing in the conjunction, let $\tau(i)$ be the vertex in G representing term i.
2. Let R be the identity relation on the vertices of G.
3. $\forall 1 \leq i \leq r$, call MERGE($\tau(v_i), \tau(w_i)$).
4. For each vertex in G labeled cons, add a vertex v labeled car and a vertex w labeled cdr, both with outdegree 1, s.t. v[1]=w[1]=u. Call MERGE(v,u[1]) and MERGE(w,u[2]). This means that when we have a term cons(x,y), we add two vertices representing car(cons(x,y)) and cdr(cons(x,y)) and merge them with the vertices representing x and y.
5. If for some $1 \leq j \leq s, \tau(x_j)$ is equivalent to $\tau(y_j)$, report UNSAT.
6. If for some $1 \leq j \leq q$, the equivalence class of $\tau(u_j)$ contains a vertex labeled cons, report UNSAT.
7. Report SAT.

Integrating Values:
The problem of satisfiability of conjunction of literals can become NP-Hard even by what appear as minor changes in the theory to which the literals belong.

Example 4.1 *Nelson and Oppen show in [9] that if we change the axioms for the theory of the Lisp List Structure so that now they specify the result of a selector function (car, cdr) on an atom, the satisfiability of a conjunction of literals becomes NP-Hard. Consider the following axioms for the theory of Lisp List Structure with the single atom NIL:*

- *car(cons(x,y)) = x*
- *cdr(cons(x,y)) = y*
- *x* $\neq$ *NIL* $\implies$ *cons(car(x),cdr(x)) = x*
- *cons(x,y)* $\neq$ *NIL*
- *car(NIL) = cdr(NIL) = NIL*

Note the last axiom, which specifies the result of car and cdr on the NIL atom.
We show that the conjunction satisifiability problem for this theory is NP-Hard by reducing the 3-CNF satisfiability problem for propositional calculus to it:
Let $p_1, ..., p_n$ *be propositional variables and F a conjunction of 3-element clauses over these variables.*
We construct a conjunction G of equalities and inequalities between terms from the Lisp List Structure theroy with the single atom NIL using the 2n variables $x_1, y_1, ..., x_n, y_n$ *such that G is satisfiable iff F is satisfiable.*
The first part of G is of the form:
$car(x_1) = car(y_1) \wedge cdr(x_1) = cdr(y_1) \wedge x_1 \neq y_1 \wedge$
...
$car(x_n) = car(y_n) \wedge cdr(x_n) = cdr(y_n) \wedge x_n \neq y_n$
Note that from these conjucntions and the axioms, $\forall 1 \leq i \leq n$*, one of* x_i, y_i *equals NIL and the other equals cons(NIL,NIL).*
Now, based on the clauses in F, we can construct a series of additional conjunctions in G so that $\forall 1 \leq i \leq n, p_i = True \implies x_i = NIL$*, and F is satisfiable iff G is satisfiable.*
We demonstrate how this construction is done by an example: if one of the clauses in F is $p_1 \vee \neg p_2 \vee p_3$*, we want to add a conjunct to G which is equivalent to:* $x_1 = NIL \vee x_2 \neq NIL \vee x_3 = NIL$*. Because of the first part of G, this is equivalent to:* $\neg(y_1 = NIL \wedge x_2 = NIL \wedge y_3 = NIL)$*, which in the form of a single literal is equivalent to:* $cons(y_1, cons(x_2, y_3))) \neq cons(NIL, cons(NIL, NIL))$.
∎

4.3 Application 3 — Theory of Arrays

Stump et al use in [7] the congurence closure procedure in order to find decision procedure for quantifier-free formuulas from the extensional theory of arrays.
The theory of arrays allows performing read and write operations over cells of arrays. The extensional theory of arrays requires in addition that if two arrays store the same value at index i, for each index i, then the arrays must be the same.
This theory includes the following axioms:
For all arrays v and a, indices i and j and values e and f:

- read(write(v,i,e),j) = if i=j then e, else read(v,j).
- write(v,i,read(v,i)) = v
- write(write(v,i,e),i,f) = write(v,i,f)
- i $\neq$ j $\implies$ write(write(v,i,e),j,f) = write(write(v,j,f),i,e)
- For all arrays v and a, if for every index i, read(v,i) = read(a,i) then v = a

Since this is the extensional theory of arrays (as shown by the last axiom), we can eliminate write operations from a formula by replacing them with a conjunction in the following manner:
write(v,i,e) = a $\iff$ (v $=_i$ a $\wedge$ read(a,i) = e). The equality v $=_i$ a is called a partial equality, and means that the values in array v are equal to a in all indices except index i. After eliminating write operations, we then observe that this theory behaves similarly to the theory of EUF, and we can therefore use the congruence closure algorithm (with some modifications to support partial equalities) to solve the satisfiability problem of formulas from this theory.

5 Combining Decision Procedures

Programming languages combine different features from different theories. We would like to be able to solve formulas which combine different theories using the solvers of these theories.

5.1 Equality Propogation Procedure

Nelson and Oppen suggest in [8] the method of equality propogation to allow solving formulas that combine different theories.
We divide a theory into logical symbols and non-logical symbols. Logical symbols are: $=, \wedge, \vee, \neg, \implies, if-then-else, \forall, \exists$.
All other symbols in a theory are non-logical symbols.
The method of combining decision procedures via equality propogation applies only to formulas which are qunatifier free and does not consider combination of theories which share non-logical symbols, or theories which do not interpret equality.

The Idea:
Given a formula combining several theories:

1. Separate the conjunct into separate conjuncts A $\wedge$ B such that A and B use different theories and only constants are shared between them.
2. If A is UNSAT or B is UNSAT — report UNSAT.
3. A is SAT and B is SAT — propogate equalities between A and B and repeat from step (2).

Definition 5.1 *If ϕ and ψ are two theories with no common non-logical symbols, their combination is a theory whose set of non-logical symbols is the union of the sets of non-logical symbols of ϕ and ψ, and whose set of axioms is the union of the sets of axioms of ϕ and ψ.*

Let F be a conjunction of literals whose non-logical symbols are among those of theories ϕ and ψ. The following algorithm determines whether F is SAT:

1. Assign conjunctions to F_ϕ and F_ψ so that F_ϕ contains only literals of the theory ϕ and F_ψ contains only literals of the theory ψ, and $F_\phi \wedge F_\psi$ is SAT iff F is SAT.
2. If F_ϕ is UNSAT or F_ψ is UNSAT — return UNSAT.
3. If either F_ϕ or F_ψ entails some equality between variables not entaileed by the other, then add the equality as a new conjunct to the one that does not entail it. Go to step 2.
4. If either F_ϕ or F_ψ entails a disjunction $u_1 = v_1 \vee ... \vee u_k = v_k$ of equalities between variables, without entailing any of the equalities alone, then apply the procedure recursively to the k formulas
   $F_\phi \wedge F_\psi \wedge u_1 = v_1$
   ...
   $F_\phi \wedge F_\psi \wedge u_k = v_k$
   If any of the formulas are SAT, then return SAT. Otherwise, return UNSAT.
5. Return SAT.

Example 5.2 *Using equality propogation, we will show that the following formula F is UNSAT:*

$$x \leq y \wedge y \leq x + car(cons(0, x)) \wedge P(h(x) - h(y)) \wedge \neg P(0)$$

This formula is a combination of literals from 3 different theories: Real Numbers, List-Structure and EUF. We construct 3 conjunctions $F_{Reals}, F_{List}, F_{EUF}$ so that each conjunct contains only literals from the relevant theory, and F is SAT iff $F_{Reals} \wedge F_{List} \wedge F_{EUF}$ is SAT.
We construct these conjunctions by introducing new variables g_1, ...,g_5 to replace terms of "wrong" type, and adding equalities defining these new variables:

F_{Reals}	F_{EUF}	F_{List}
$x \leq y$	$P(g_2) = true$	$g_1 = car(cons(g_5, x))$
$y \leq x + g_1$	$P(g_5) = false$	
$g_2 = g_3 - g_4$	$g_3 = h(x)$	
$g_5 = 0$	$g_4 = h(y)$	

From the semantics of List-Structure theory, the equality in F_{List} entails that g_1 = g_5. After propogating this equality, we get in F_{Reals} that x = y (since g_1 = g_5 = 0 means $x \leq y \wedge y \leq x$).

Propogating this equality, we get in F_{EUF} that $g_3 = g_4$. After propogating this equality, we get in F_{Reals} that $g_2 = g_5 = 0$. Finally, after propogating this equality, we get the following in F_{EUF}:
$g_2 = g_5 \wedge P(g_2) = true \wedge P(g_5) = false$ — which is UNSAT, and therefore F is UNSAT.
∎

Example 5.3 *Using equality propogation, we will show that the following formula F is SAT:*

$$x \leq y \wedge y \leq car(cons(x, x)) \wedge P(x) = P(y)$$

This formula is a combination of literals from 3 different theories: Real Numbers, List-Structure and EUF. We construct 3 conjunctions $F_{Reals}, F_{List}, F_{EUF}$ so that each conjunct contains only literals from the relevant theory, and F is SAT iff $F_{Reals} \wedge F_{List} \wedge F_{EUF}$ is SAT.
We construct these conjunctions by introducing new variable g_1 to replace term of "wrong" type, and adding equality defining this new variable:

F_{Reals}	F_{EUF}	F_{List}
$x \leq y$	$P(x) = P(y)$	$g_1 = car(cons(x, x))$
$y \leq g_1$		

From the semantics of List-Structure theory, the equality in F_{List} entails that $g_1 = x$. After propogating this equality, we get in F_{Reals} that y = x (since $g_1 = x$ means $x \leq y \wedge y \leq x$).
Propogating this equality, we get in F_{EUF} that indeed $P(x) = P(y)$, and the formula F is SAT.
∎

Soundness: The algorithm of equality propogation is sound — it is clear that we only report UNSAT when we discover that a formula is indeed UNSAT.

5.2 Convexity

Definition 5.4 *A formula F is non-convex if it entails a disjunction $u_1 = v_1 \vee ... \vee u_k = v_k$ of equalities between variables, but for no $i = 1..n$ does F entail $x_i = y_i$. Otherwise, F is convex.*
A theory ϕ is convex if every conjunction of ϕ-literals is convex.

Some convex theories: EUF, List-Structure, relation linear algebra.
Some non-convex theories: Arrays, reals under multiplications.

Example 5.5 *Real numbers under multiplication is not convex: xy = 0 $\implies$ x = 0 $\vee$ y = 0.*

Non-convex theories require performing step (4) in the algorithm of equality propogation, i.e. enumarating over all possibilties.

5.3 Completeness of Equality Propogation

Definition 5.6 *Residue is the storngest boolean combination of equalities between variables entailed by a formula*

Example 5.7 *Simple formulas and their residues:*

Formula	*Residue*
$x = f(a) \wedge y = f(b)$	$a = b \implies x = y$
$x + y - a - b > 0$	$\neg(x = a \wedge y = b) \wedge \neg(x = b \wedge y = a)$

Example 5.8 *Residues in the equality-propogation example 5.2 (in blue):*

F_{Reals}	F_{EUF}	F_{List}
$x \leq y$	$P(g_2) = true$	$g_1 = car(cons(g_5, x))$
$y \leq x + g_1$	$P(g_5) = false$	$g_1 = g_5$
$g_2 = g_3 - g_4$	$g_3 = h(x)$	
$g_5 = 0$	$g_4 = h(y)$	
$(g_1 = g_5 \implies x = y) \wedge$	$(g_2 \neq g_5) \wedge$	
$(g_5 = g_2 \iff g_3 = g_4)$	$(x = y \implies g_3 = g_4)$	

Residues are used in proving the completness of the eqality-propogation algorithm. We mark the residue of formula F as Res(F).
In [10], Manna and Zarba defined the following:

Definition 5.9 *A theory T is stably-infinite if for every T-satisfiable quantifier-free formula there exists a T-model which satisfies it, whose domain is infinite.*

Most theories are stably-infinite, and therefore the equality-propogation procedure can be used on their combinations to correctly determine satisfiability.

Example 5.10 *The following theory T is not stably-infinite:*
$T = \forall x \ (x = a \vee x = b)$*, where a and b are constants.*
T is not stably-infinite. For every quantifier-free formula F in T, there does not exist a model in T with infinite domain which satisfies it. In fact, every model with domain cardinality > 2 *will not satisfy F.*

Lemma 5.11 *If A and B are formulas whose only common parameters are variables, and are from theories which are disjoint (do not have any non-logical symbols in common) and stably-infinite, then* $Res(A \wedge B) = Res(A) \wedge Res(B)$.

This lemma allows us to show that the equality-propogation procedure is complete if the combined theories are:

1. disjoint
2. stably-inifinte

5.4 Handling Quantifiers

Once we introduce quantifiers to the formulas, the problem becomes undecidable.
Some refutantionally resolution-based complete procedures exist (see [11]) and implemented (e.g. SPASS, Vampiere). However, they do not handle theories, and are not guaranteed to terminate.
Z3 employes incomplete heuristics to handle quantifiers, and allows tuning by the user.

References

[1] Fagin, Ronald. Finite-model theory - a personal perspective. Theoretical Computer Science 116: 331, 1993.

[2] Boolos, Burgess, Jeffrey. Computability and Logic, Cambridge University Press, 2002.

[3] MONA - http://www.brics.dk/mona/index.html

[4] Downey, Sethi and Trajan. Variations on the Common Subexpression Problem, JACM, 1980.

[5] Lahiri, Bryant, Goel, Talpur. Revisiting Positive Equality, TACAX, 2004.

[6] Oppen. Reasoning about Recursively Defined Data Structures, ACM 1978.

[7] Stump, Dill, Barrett, Levitt. A Decision Procedure for Extensional Theory of Arrays, LICS, 2001.

[8] Nelson, Oppen. Simplification by Cooperating Decision Procedures, ACM, 1980.

[9] Nelson, Oppen. Fast Decision Procedures Based on Congruence Closure, JACM, 1979.

[10] Manna, Zarba. Combining Decision Procedures, Stanford University, 2003.

[11] Flanagan, Joshi, Saxe. Verifun: An Explicating Theorem Prover for Quantified Formulas, HPL, 2004.