Lecture Notes for

Logic and Computability

Course Number: IND04033UF

by

Bettina Könighofer
Bernd Grabner
Belinda Uhl

Contact

Bettina Könighofer
Institute for Applied Information Processing and Communications (IAIK)
Graz University of Technology, Austria
bettina.koenighofer@iaik.tugraz.at

March, 2021

Table of Contents

9

Satisfiability Modulo Theories

In computer science, the satisfiability modulo theories (SMT) problem refers to the problem of determining whether a formula in predicate logic is satisfiable with respect to some theory. *A theory fixes the interpretation/meaning of certain predicate and function symbols.* Checking whether a formula in predicate logic is satisfiable with respect to a theory means that we are not interested in arbitrary models but in models that interpret the functions and predicates contained in the theory as defined by the axioms in the theory. Consider the following formula that uses arithmetic:

$$\neg(a \geq b) \wedge (a + 1 > b).$$

We are not interested in models that use a nonstandard interpretation of the symbols $<$, $+$, and 1. We only want to consider models that also satisfy the theory which defines the meaning of the used constants, functions, and predicates. There exists several commonly used theories in computer science. For example, Presburger arithmetic is the theory of natural numbers with addition, more complex theories include the theory of integers or real numbers with arithmetic, and the theories of data structures such as lists, arrays, or bit vectors.

One way to enforce that all considered models are consistent with a background theory is to explicitly incorporate all the axioms of the theory into the input formula (called eager encoding). But even when this is possible, the performance of such solvers is often unacceptable. To avoid the explicit encoding of axioms, specialized theory solvers in combination with SAT solvers can be used to decide whether a formula is satisfiable within a theory (called lazy encoding). This chapter provides a brief overview of SMT and focuses on the two most used approaches for implementing SMT solvers: eager and lazy encoding.

9.1 Definitions and Notations

A theory in predicate logic is typically defined by its signature Σ and its set of axioms $\mathcal{A}$.

Definition - Theory. A theory is as a pair $(\Sigma; \mathcal{A})$ where Σ is a signature which defines a set of constant, function, and predicate symbols. The set of axioms $\mathcal{A}$ is a set of closed predicate logic formulas in which only constant, function, and predicate symbols of Σ appear.

Definition - $\mathcal{T}$-satisfiablility. We say that a *formula φ is satisfiable in a theory* $\mathcal{T}$, or $\mathcal{T}$-satisfiable, if and only if there is a model $\mathcal{M}$ within $\mathcal{T}$ that satisfies φ (i.e., $\mathcal{M} \vDash A$ for every $A \in \mathcal{A}$ and $\mathcal{M} \vDash \varphi$).

Definition - $\mathcal{T}$-valid. We say that a *formula φ is valid in a theory* $\mathcal{T}$, or $\mathcal{T}$-valid, if and only if all models within $\mathcal{T}$ satisfy φ.

Definition - $\mathcal{T}$-entailment. A set of formulas $\varphi_1, \dots \varphi_n$ $\mathcal{T}$-entails a formula ψ, written as $\varphi_1, \dots \varphi_n \vDash_{\mathcal{T}} \psi$, if every model of $\mathcal{T}$ that satisfies all formulas $\varphi_1, \dots \varphi_n$ satisfies ψ as well.

Definition $\mathcal{T}$-decidable. A theory $\mathcal{T}$ is decidable if there exists an algorithm that always terminates with "yes" if φ is $\mathcal{T}$-valid or with "no" if φ is $\mathcal{T}$-invalid.

9.2 Theory of Equality and Uninterpreted Functions

In this chapter we discuss the theory of equality with uninterpreted functions $\mathcal{T}_{EUF}$, since is the simplest first-order theory.

Uninterpreted functions are often used as an abstraction technique to remove unnecessarily complex or irrelevant details of a system being modeled. For example, suppose we want to prove that the following set of literals is unsatisfiable: $\{a \cdot (f(b) + f(c)) = d, b \cdot (f(a) + f(c)) \neq d, a = b\}$. At first, it may appear that this requires reasoning in the theory of arithmetic. However, if we abstract $+$ and $\cdot$ by replacing them with uninterpreted functions g and h respectively, we get a new set of literals: $\{h(a, g(f(b), f(c))) = d, h(b, g(f(a), f(c))) \neq d, a = b\}$. This set of literals can be proved unsatisfiable without any arithmetic.

Definition - Theory of Equality and Uninterpreted Functions. The theory of equality and uninterpreted functions $\mathcal{T}_{EUF}$ has the signature

$$\Sigma_{EUF} \coloneqq \{=, a, b, c, d, \dots, f, g, h, \dots, P, Q, R, \dots\}$$

. The axioms $\mathcal{A}_{EUF}$ are the following:

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)
4. $\forall \overline{x}, \overline{y}.\ \ (\bigwedge_{i=1}^{n} x_i = y_i) \rightarrow f(\overline{x}) = f(\overline{y})$ (congruence)

5. $\forall \overline{x}, \overline{y}.\ \ (\bigwedge_{i=1}^{n} x_i = y_i) \rightarrow (P(\overline{x}) \leftrightarrow P(\overline{y}))$ (equivalence)

The signature Σ_{EUF} consists the binary predicate $=$ (equality) and all constant, function and predicate symbols. The axioms of the theory of equality include the axioms of reflexivity, symmetry, transitivity, function congruence, and equivalence.

9.3 Eager Encoding

The eager approach to SMT solving involves *translating the original formula to an equisatisfiable Boolean formula in a single step.* The translation is done by encoding enough relevant consequences of the theory $\mathcal{T}$ into the Boolean formula. The eager approach applies in principle to any theory with a decidable satisfiability problem, possibly however at the cost of a significant blow-up in the translation.

The main idea of eager encoding is that the input formula is translated into a propositional formula with all relevant theory-specific information encoded into the formula. The large propositional formula can then be given to any off-the-shelf SAT solver.

A algorithm based on the direct encoding of axioms (i.e., eager method) operates in three steps:

1. Replace each unique constraint in the original formula φ with a fresh Boolean variable to get a Boolean formula $\hat{\varphi}$.
2. Generate a Boolean formula φ_{cons} that constrains values of the introduced Boolean variables so as to preserve the information of the theory.
3. Invoke a SAT solver on the Boolean formula $\varphi_{prop} \coloneqq \hat{\varphi} \wedge \varphi_{cons}$ that corresponds to an equisatisfiable propositional formula to φ.

The translations used in the eager approach are of course theory specific. We discuss the translation of a formula within the the theory of uninterpreted functions and equalities to an equisatisfiable propositional formula. The translation consists of two separate steps to construct the propositional formula φ_{prop}. First, we remove all function instances using the Ackermann algorithm. In a second step, we remove all equality instances by a graph-based algorithm.

Elimination of Function Applications - via Ackermann Algorithm

The first step in the transformation to a Boolean formula is to eliminate applications of function and predicate symbols of non-zero arity. These applications are replaced by new propositional symbols and additional constraints are added on this fresh variables to maintain functional consistency (the congruence property).

The Ackermann methods involves creating sufficient instances of the congruence axiom to preserve satisfiability. The method works by replacing every function application occurring in the input formula φ_{EUF} with a fresh variable

resulting in $\hat{\varphi}_{EUF}$, and then adding to $\hat{\varphi}_{EUF}$ all the needed functional congruence constraints encoded in φ_{FC}. The formula $\hat{\varphi}_E$ obtained is equisatisfiable with $\hat{\varphi}_{EUF}$, and contains no uninterpreted function symbols.

We illustrate Ackermann's method using an example. Suppose that you have the following formula φ_{UEF} with three occurrences of the function symbol f.

$$\varphi_{UEF} \coloneqq (f(a) = f(b)) \wedge \neg(f(b) = f(c)).$$

First, we generate three fresh constant symbols f_a, f_b, and f_c, one for each of the three different terms containing f. First, we replace those terms using the fresh variables leading to the formula:

$$\hat{\varphi}_{EUF} \coloneqq (f_a = f_b) \wedge \neg(f_b = f_c).$$

Second, we encode the functional consistency constraints for f:

$$\varphi_{FC} \coloneqq ((a = b) \rightarrow f_a = f_b) \wedge ((b = c) \rightarrow f_b = f_c) \wedge ((a = c) \rightarrow f_a = f_c).$$

In a similar fashion, functional consistency constraints are generated for each function and predicate symbol in φ_{UEF} and are conected with a conjunction to form φ_{FC}. The resulting equisatisfiable formula in the theory of equaility without function instances is as follows:

$$\varphi_E \coloneqq \hat{\varphi}_{EUF} \wedge \varphi_{FC}.$$

To obtain a formula that is *equivalid*, the conjcuntion needs to be replaced by an implication, i.e.,

$$\varphi_E \coloneqq \hat{\varphi}_{EUF} \rightarrow \varphi_{FC}.$$

Example. Perform Ackermann's reduction on the following formula to compute an equsatisfiable formula without function instances.

$$(z = f(x, z) \leftrightarrow f(x, y) = x) \wedge (y \neq x \vee f(y, z) = f(x, y) \vee x = z) \rightarrow f(x, z) = z$$

Solution.

$$\begin{aligned}
\varphi_{FC} \equiv & (x = x \wedge y = z) \rightarrow (f_{xy} = f_{xz}) \wedge \\
& (x = y \wedge y = z) \rightarrow (f_{xy} = f_{yz}) \wedge \\
& (x = y \wedge z = z) \rightarrow (f_{xz} = f_{yz}) \\
\hat{\varphi}_{EUF} \equiv & (z = f_{xz} \leftrightarrow f_{xy} = x) \wedge \\
& (y \neq x \vee f_{yz} = f_{xy} \vee x = z) \rightarrow f_{xz} = z
\end{aligned}$$

$$\textit{Formula in Theory of Equality} : \varphi_E \equiv \varphi_{FC} \wedge \hat{\varphi}_{EUF}$$

Elimination of Equalities - Graph-based Reduction

Ackermann's reduction gives us a formula in the theory of equality $\mathcal{T}_E$. The last step required to reduce this to an equisatisable propositional formula is to remove equalities. Bryant and Velev have introduced such a reduction based on an equality graph. For an input formula φ_E in $\mathcal{T}_E$, the algorithm works a follows.

First, every *refexivity* instance $a = a$ in φ_E is replaced by *true*. Second, every equality atom is rewritten such that the first term precedes the second term with respect to some total order. This takes care of ensuring *symmetry*. Next, every equality atom $a = b$ is replaced by a fresh propositional variable $e_{a=b}$. This results in the formula $\hat{\varphi}_E$.

In order to take care of *transitivity*, we construct a so-called *non-polar equality graph*: This graph has a node for every term and an edge for every equality and disequality in the formula (there is no difference between equality and disequality in the graph). This graph is then made *chordal*.

Definition - Chords, Chord-free Cycles, and Chordal Graphs. In a graph G, let n_1 and n_2 be two non-adjacent nodes in a cycle. An edge between n_1 and n_2 is called a chord. A cycle is said to be *chord-free*, if in the cycle there exist no non-adjacent nodes that are connected by an edge. A graph is called *chordal*, if it contains no chord-free cycles with size greater than 3.

A graph can be made chordal by adding additional edges. By only having such triangles in the graph, we avoid an exponential blow-up in the number of transitivity constraints. Based on the chordal graph, we can compute the transitivity constraints. For every triangle (x, y, z) in the graph, we add the following constraints:

$$(e_{x=y} \wedge e_{y=z} \rightarrow e_{x=z}) \wedge (e_{x=y} \wedge e_{x=z} \rightarrow e_{y=z}) \wedge (e_{y=z} \wedge e_{x=z} \rightarrow e_{x=y})$$

We connect all transitivity constraints for all triangles via conjunction to obtain φ_{TC}. The resulting equisatisfiable propositional formula:

$$\varphi_{prop} \coloneqq \hat{\varphi}_E \wedge \varphi_{TC}.$$

To obtain a formula that is *equivalid*, the conjcuntion needs to be replaced by an implication, i.e.,

$$\varphi_{prop} \coloneqq \hat{\varphi}_E \rightarrow \varphi_{TC}.$$

Example. Perform the graph-based reduction reduction on the following formula to compute an equsatisfiable formula in propositional logic.

$$f_x = f_a \wedge f_y \neq y \vee f_x = f_y \wedge f_y = y \vee f_y = f_x \wedge y \neq f_a \vee f_y = g_x \wedge f_y = y$$

Solution.

$$
\begin{aligned}
\varphi_{TC} \equiv & (e_{f_x=f_a} \wedge e_{y=f_a} \rightarrow e_{f_x=y}) \wedge \\
& (e_{f_x=f_a} \wedge e_{f_x=y} \rightarrow e_{y=f_a}) \wedge \\
& (e_{y=f_a} \wedge e_{f_x=y} \rightarrow e_{f_x=f_a}) \wedge \\
& (e_{f_x=y} \wedge e_{y=f_y} \rightarrow e_{f_x=f_y}) \wedge \\
& (e_{f_x=y} \wedge e_{f_x=f_y} \rightarrow e_{y=f_y}) \wedge \\
& (e_{y=f_y} \wedge e_{f_x=f_y} \rightarrow e_{f_x=f_y}) \\
\hat{\varphi}_E \equiv & e_{f_x=f_a} \wedge \neg e_{f_y=y} \vee \\
& e_{f_x=f_y} \wedge e_{f_y=y} \vee \\
& e_{f_y=f_x} \wedge \neg e_{y=f_a} \vee \\
& e_{f_y=g_x} \wedge e_{f_y=y}
\end{aligned}
$$

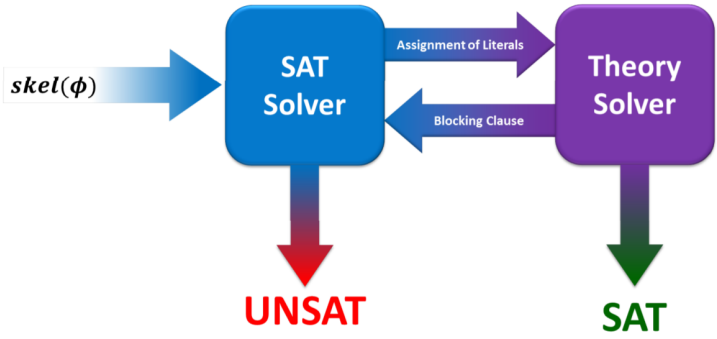
$$\textit{Boolean Formula} : \varphi_{prop} \equiv \varphi_{TC} \wedge \hat{\varphi}_E$$

9.4 Lazy Encoding

Lazy encoding is based on the interaction between a SAT solver and a so-called theory solver. A theory solver is an algorithm that can decide satisfiability of the conjunctive fragment of a theory. In contrast to eager encoding, where a sufficient set of constraints is computed at the beginning, lazy encoding starts with no constraints at all, and lazily adds constraints only when required.

The principle of lazy encoding is shown in Figure 9.1. To decide whether or not a $\mathcal{T}$-formula φ is $\mathcal{T}$-satisfiable, the propositional skeleton skel(φ) is given to a SAT solver. If the SAT solver returns *unsatisfiable*, the procedure is done and we know that φ is not $\mathcal{T}$-satisfiable. If, however, the SAT solver returns *satisfiable*, we obtain a satisfying assignment for the truth values of the theory atoms in φ. This assignment is a formula in the conjunctive fragment of $\mathcal{T}$, which we pass to the theory solver. If the theory solver returns *satisfiable*, we have found an assignment of truth values to the theory atoms that is *consistent.* with $\mathcal{T}$. Thus we know that is $\mathcal{T}$-satisfiable. If, however, the theory solver returns *unsatisfiable*, we have to look for another assignment, as the present one is not consistent with $\mathcal{T}$. To obtain a different assignment, we negate the inconsistent assignment - which conveniently turns it into a clause - and add it as a so-called blocking clause to the CNF of skel(φ). The blocking clause ensures that the next satisfying assignment obtained from the SAT solver (if one exists) is different from the current, $\mathcal{T}$-inconsistent assignment. This loop is repeated until we encounter one of the following two terminal cases. Either the SAT solver suggests an assignment that the theory solver finds to be consistent with $\mathcal{T}$, in which case φ is $\mathcal{T}$-satisfiable. Or we have added so many blocking clauses that the SAT solver cannot find any more assignments, in which case

Figure 9.1: The propositional skeleton of φ is given to a SAT solver. If a satisfying assignment is found, it is checked by a theory solver. If the assignment is consistent with the theory, φ is $\mathcal{T}$-satisfiable. Otherwise, a blocking clause is generated and the SAT solver searches for a new assignment. This is repeated until either a $\mathcal{T}$-consistent assignment is found, or the SAT solver cannot find any more assignments.



φ is not $\mathcal{T}$-satisfiable. As every blocking clause excludes (at least) one of only finitely many assignments, the loop is guaranteed to terminate.

9.5 Theory Solvers and Congruence Closure

Theory Solvers. Theory solvers are specialized on deciding a specific background theory, or a fragment of a background theory $\mathcal{T}_{EUF}$. The common practice is to write theory solvers just for deciding *conjunctions of literals*; i.e., atomic formulas and their negations. The main advantage of theory-specific solvers is that one can use whatever specialized algorithms and data structures are best for the theory in question, which typically leads to better performance.

The role of the theory solver is to accept a set of literals and report whether the set is $\mathcal{T}$-satisfiable or not. The congruence closure algorithm is the most common theory solver for $\mathcal{T}_{EUF}$.

Congruence Closure. Given a conjunction of $\mathcal{T}_{EUF}$-literals, it computes a set of *congruence classes*, such that the conjunction of literals implies that all terms in the same congruence class are equal.This is done in the following way. First, all terms for which there is a (positive) equality in the conjunction of literals are put into the same congruence class. All remaining terms are put in singleton classes. Next, classes are merged, if they contain common terms. This accounts for the transitivity of the equality predicate. Next, classes are merged based on function congruence. That is, if two classes both contain an instance of the same uninterpreted function, and corresponding parameters are already in the same congruence class (which means that they are equal), the classes of the function instances are merged. These steps are repeated until

no more merging can be done. In the last step, all the disequalities from the conjunction of literals are checked against the merged congruence classes. If there is a disequality that contradicts the congruence classes, that is, both its terms are in the same congruence class, the conjunction of literals is unsatisfiable. If no such disequality exists, the conjunction of literals is satisfiable.

Example. Use the congruence closure algorithm to check if the following assignment for the equalities is satisfiable.

$f(a) = e \wedge f(c) \neq f(e) \wedge a = f(b) \wedge f(b) \neq c \wedge b \neq a \wedge f(a) = d \wedge$
$d \neq f(c) \wedge b = d \wedge a \neq e \wedge c = d$

Solution.

$$\{\underline{f(a)}, e\}, \{a, f(b)\}, \{\underline{f(a)}, d\}, \{b, d\}, \{c, d\}, \{f(c)\}, \{f(d)\}, \{f(e)\}$$
$$\{f(a), e, \underline{d}\}, \{a, f(b)\}, \{b, \underline{d}\}, \{c, \underline{d}\}, \{f(c)\}, \{f(d)\}, \{f(e)\}$$
$$\{f(a), \underline{e}, \underline{d}, b, \underline{c}\}, \{a, f(b)\}, \{f(c)\}, \{f(d)\}, \{f(e)\}$$
$$\{f(a), e, d, b, c\}, \{a, f(b)\}, \{f(c), f(d), f(e)\}$$

Checking the disequality $f(c) \neq f(e)$ leads to the result that the assignment is UNSAT, since $f(c)$ and $f(e)$ are in the same congurence class.

9.6 Exercises

9.6.1 Examples

Example 1. Perform Ackermann's reduction to translate a formula in $\mathcal{T}_{EUF}$ into an equisatifiable formula in $\mathcal{T}_E$.
$f(x) = f(g(y)) \wedge f(y) \neq y \vee$
$f(x) = f(y) \wedge f(y) = y \vee$
$f(y) = f(x) \wedge y \neq f(g(y)) \vee$
$f(y) = g(x) \wedge f(y) = y$

Example 2. Perform graph-based reduction to translate a formula in $\mathcal{T}_E$ into an equisatifiable formula in propositional logic.

$$(z = f_{xz} \leftrightarrow f_{xy} = x) \wedge (\neg y = x \vee f_{yz} = f_{xy} \vee x = z) \rightarrow z = f_{xz}$$

Example 3. Use the congruence closure algorithm to check if the following assignment for the equalities is satisfiable.
$f(b) = a \wedge c \neq d \wedge f(e) = b \wedge$
$d \neq f(b) \wedge f(a) = f(e) \wedge b \neq f(b) \wedge$
$a \neq e \wedge f(a) = e \wedge a = c \wedge$
$f(b) \neq e \wedge d = f(c)$

Example 4. Use the congruence closure algorithm to check if the following assignment for the equalities is satisfiable.
$x = y \wedge v = w \wedge z = f(w) \wedge z \neq x \wedge w \neq f(y) \wedge f(x) = w \wedge$
$f(z) = f(x) \wedge f(z) = f(v)$

9.6.2 Solutions

Solution 1.

$$\begin{aligned}\varphi_{FC} \equiv (x = y) \rightarrow (f_x = f_y) \wedge \\ (x = g_y) \rightarrow (f_x = f_{g_y}) \wedge \\ (y = g_y) \rightarrow (f_y = f_{g_y}) \wedge \\ (x = y) \rightarrow (g_x = g_y)\end{aligned}$$

$$\begin{aligned}\hat{\varphi}_{EUF} \equiv f_x = f_{g_y} \wedge f_y \neq y \vee \\ f_x = f_y \wedge f_y = y \vee \\ f_y = f_x \wedge y \neq f_{g_y} \vee \\ f_y = g_x \wedge f_y = y\end{aligned}$$

$$\varphi_E \equiv \varphi_{FC} \wedge \hat{\varphi}_{EUF}$$

Solution 2. $\hat{\varphi}_E \equiv (e_{z=f_{xz}} \leftrightarrow e_{f_{xy}=x}) \wedge (\neg e_{y=x} \vee e_{f_{yz}=f_{xy}} \vee e_{x=z}) \rightarrow e_{z=f_{xz}}$
$\varphi_{TC} \coloneqq \mathit{true}$
$\varphi_{prop} \equiv \varphi_{TC} \wedge \hat{\varphi}_E$

Solution 3.

$\{f(b), a\}, \{f(e), b\}, \{\underline{f(a)}, f(e)\}, \{\underline{f(a)}, e\}, \{a, c\}, \{d, f(c)\}, \{f(d)\}$
$\{f(b), a\}, \{\underline{f(e)}, b\}, \{f(a), \underline{f(e)}, e\}, \{a, c\}, \{d, f(c)\}, \{f(d)\}$
$\{f(b), \underline{a}\}, \{f(a), f(e), e, b\}, \{\underline{a}, c\}, \{d, f(c)\}, \{f(d)\}$
$\{f(b), a, c\}, \{f(a), f(e), \underline{e}, \underline{b}\}, \{d, f(c)\}, \{f(d)\}$
$\{f(b), a, c, f(a), f(e), e, b\}, \{d, f(c)\}, \{f(d)\}$

Checking the disequality $f(b) \neq e$ leads to the result that the assignment is UNSAT, since $f(b)$ and e are in the same congurence class.

Solution 4. $\{x, y\}, \{v, \underline{w}\}, \{z, f(w)\}, \{f(x), \underline{w}\}, \{f(z), f(x)\}, \{f(z), f(v)\}$
$\{x, y\}, \{v, w, \underline{f(x)}\}, \{z, f(w)\}, \{f(z), \underline{f(x)}\}, \{f(z), f(v)\}$
$\{x, y\}, \{v, w, \underline{f(x)}, \underline{f(z)}\}, \{z, f(w)\}, \{\underline{f(z)}, f(v)\}$
$\{x, y\}, \{v, w, f(x), \underline{f(z)}, f(v)\}, \{z, f(w)\}$
$z \neq x$✓
$w \neq f(y)$✓
φ is SAT

Declaration of Sources

Chapter 9 was based on the following books.

- A. Biere, M. Heule, H. van Maaren, and T. Walsh: Handbook of Satisfiability. Volume 185 of Frontiers in Artificial Intelligence and Applications, IOS Press, (2009)
- Georg Hofferek: Controller Synthesis with Uninterpreted Functions. PhD Thesis. 2014. Graz University of Technology.