From SAT to SAT-Modulo-Theories

Building Decision Procedures for Expressive Logics beyond Propositional Logics

Uwe Keller
uwe.keller@deri.at

Semantic Technology Institute (STI) Innsbruck
University of Innsbruck, Austria

June 9, 2008

Outline

1. Motivation

2. Lazy SAT-Modulo-Theories (SMT)
   - Theoretical Background
   - Algorithms: Propositional Model Enumerator & DPLL(T)
   - Basic efficiency aspects

The Power of modern SAT Solver

Propositional Logic (PL) is the most fundamental logic and the SAT problem a core problem in AI with a lot of applications (e.g. circuit verification, recap: any NP-complete problem can be represented as a SAT problem!)

SAT has been studied extensively during many decades and really impressive progress has been achieved since $\sim$15 years. Hard and fairly large real-world problems can be solved by modern SAT solver.

A range of (very) different methods has been proposed to solve SAT for propositional logics. The single methods are effective on different problem sets.

However: in applications PL is often not enough, more expressive logics are either needed (or desired) to capture domain knowledge.

Examples: Verification of pipelines micro-processors, real-time or hybrid control systems, software ...

Towards more expressive Formalisms

More expressive logics are usually derived from PL by extending (or enriching) the set A of atomic statements:

$$\phi \longrightarrow A \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi$$

Examples for such refinements of atomic statements are ...

- **Modal Logics (and Description Logics):**

$$A \longrightarrow P \mid \Box_n \phi \mid \Diamond_n \phi \quad n \in \mathbb{N}$$

- **Difference Logic (or logic of Linear Arithmetic)**

$$A \longrightarrow V - V \leq V \mid V - V = V$$

- **Logic of equality and uninterpreted function (EUF):**

$$\begin{aligned} A &\longrightarrow T \doteq T \\ T &\longrightarrow V \mid f(T, \ldots, T) \quad f \in \Sigma \end{aligned}$$

- **First-order Predicate Logic with equality (PL):**

$$\begin{aligned} A &\longrightarrow T \doteq T \mid p(T, \ldots, T) \mid \forall V.\phi \mid \exists V.\phi \quad p \in \Sigma \\ T &\longrightarrow V \mid f(T, \ldots, T) \quad f \in \Sigma \end{aligned}$$

Building SAT Procedures for expressive Logics

SAT-Modulo-Theories (SMT) = Decide the satisfiability of a formula over a specific theory T (or combination of theories) $= T-$formula

$\triangleright$ Given the success of modern SAT solver, can we use SAT solver techniques here directly?

Two main approaches:

- **Eager SMT (T-SAT $\rightarrow$ SAT):** Map the T-formula to an equi-satisfiable propositional formula (with potential (super)exponential blow-up)) and and use a state-of-the-art SAT solver
- **Lazy SMT (T-SAT = SAT + T-solver):** For a number of theories (e.g. Difference Logic, EUF) efficient (possibly non-logical) procedures to determine the satisfiability of sets of atomic T-statements already exist

  $\Rightarrow$ Combine a propositional SAT solver for boolean reasoning with such theory-specific decision procedures (theory solver) for theory-specific reasoning to solve the SMT problem

Success stories: Model Checking, AI Planning, Descrip. Logic Reasoning ...

Example Formalism: Modal Logic $\mathbf{K}$

Syntax of formulae (here Modal Logic $\mathbf{K}$):

$$\phi \longrightarrow A \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi \mid \Box\phi \mid \Diamond\phi$$

Examples: every propositional ϕ, $p \wedge \Box(q \vee s)$, $\Box\Diamond p \vee \Diamond q$

Semantics: A modal interpretation $\mathcal{I} = (W, R, o)$ with

- W a set of worlds
- $R \subseteq W \times W$ an (accessibility) relation between worlds
- $o : W \rightarrow Int(A)$ a function assigning to each world $w \in W$ a propositional interpretation $o(w)$ (observation) of all symbols A

▶ Modal Interpretation $\sim$ Graph of Propositional Interpretations

Formulae and their Meaning

Syntax of formulae (here Modal Logic K):

$$\phi \longrightarrow A \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi \mid \Box\phi \mid \Diamond\phi$$

Semantics: Define when a formula ϕ is satisfied by $\mathcal{I}$ in a world w (denoted as $\mathcal{I}, w \models \phi$):

$\mathcal{I}, w \models A$	iff. $o(w)(A) = \top$
$\mathcal{I}, w \models \neg\phi$	iff. not $\mathcal{I}, w \models \phi$
$\mathcal{I}, w \models \phi_1 \wedge \phi_2$	iff. $\mathcal{I}, w \models \phi_1$ and $\mathcal{I}, w \models \phi_2$
$\mathcal{I}, w \models \phi_1 \vee \phi_2$	iff. $\mathcal{I}, w \models \phi_1$ or $\mathcal{I}, w \models \phi_2$
$\mathcal{I}, w \models \Box\phi$	iff. for each w^* with $R(w, w^*)$: $\mathcal{I}, w^* \models \phi$
$\mathcal{I}, w \models \Diamond\phi$	iff. there is a $w^* \in W$ s.t. $R(w, w^*)$ and $\mathcal{I}, w^* \models \phi$

Basic notions: Atom, Truth Assignment

A T-atom is any T-formula that cannot be decomposed propositionally (i.e. whose top-level connective is not a boolean operator). A T-literal is either a T-atom or its negation.

Example: p, $\Box\neg(q \vee s)$, $\neg\Diamond(p \wedge s)$, $\Box\Diamond p \vee \Diamond q$

Denote the set of (top-level) T-atoms occurring in a T-formula ϕ by $Atoms(\phi)$.

Example: $\phi = \Box\Diamond p \vee (\neg p \wedge \Diamond\neg q) \quad \Rightarrow \quad Atoms(\phi) = \{\Box\Diamond p, \Diamond\neg q, p\}$

We call a total truth assignment μ for a $T-$formula ϕ a set

$$\mu = \{\alpha_1, \ldots, \alpha_N, \neg\beta_1, \ldots, \neg\beta_M, A_1, \ldots, A_R, \neg A_{R+1} \ldots \neg A_S\}$$

such that every atom in $Atoms(\phi)$ occurs either as a positive or negative literal in μ.

A partial truth assignment μ for ϕ is a subset of a total truth assignment for ϕ (i.e the truth value of some atoms may not be defined). If $\mu_1 \subseteq \mu_1$, then we say that μ_1 extends μ_2 and that μ_2 subsumes μ_1.

Basic notions: Propositional Satisfaction

A total truth assignment μ for ϕ propositionally satisfies ϕ (written as $\mu \models_0 \phi$) iff. μ evaluates to $\top$ under μ, i.e. for any sub-formulas ϕ_1, ϕ_2 of ϕ

$$\begin{array}{lcl}
\mu \models_0 \phi_1, \phi_1 \in \mathit{Atoms}(\phi) & \Leftrightarrow & \phi \in \mu \\
\mu \models_0 \neg\phi_1 & \Leftrightarrow & \mu \not\models_0 \phi_1 \\
\mu \models_0 \phi_1 \wedge \phi_2 & \Leftrightarrow & \mu \models_0 \phi_1 \text{ and } \mu \models_0 \phi_2 \\
\mu \models_0 \phi_1 \vee \phi_2 & \Leftrightarrow & \mu \models_0 \phi_1 \text{ or } \mu \models_0 \phi_2
\end{array}$$

We say that a partial truth assignment μ propositionally satisfies ϕ iff. all total truth assignments for ϕ that extend μ propositionally satisfy ϕ.

Example: $\phi = \Box\Diamond p \vee (\neg p \wedge \Diamond\neg q) \quad \Rightarrow \quad \mathit{Atoms}(\phi) = \{\Box\Diamond p, \Diamond\neg q, p\}$

$\mu_1 = \{\Diamond\neg q, p, \neg\Box\Diamond p\}$, $\mu_2 = \{\Box\Diamond p\} \quad \Rightarrow \quad \mu_1 \not\models_0 \phi$, $\mu_2 \models_0 \phi$

Note: Propositional satisfaction corresponds to satisfaction in PL when considering ϕ as a propositional formula over the signature $\mathit{Atoms}(\phi)$

Propositional satisfaction of a T-formula is a weaker notion than T-satisfaction:

$$I \models_T \phi \quad \Rightarrow \quad \mu^I \models_0 \phi$$

Complete sets of propositional models for a T-formula

Let $\mathcal{M} = \{\mu_1, \ldots, \mu_n\}$ be a set of partial truth assignments for ϕ that propositionally satisfy ϕ. We say that $\mathcal{M}$ is complete iff.

$$\models_0 \phi \leftrightarrow \bigvee_j \mu_j$$

Informally: $\mathcal{M}$ is complete in the sense that for every propositional model ν of ϕ there exists a $\mu_j \in \mathcal{M}$ that subsumes (and thus represents) ν. Hence, $\mathcal{M}$ is a compact representation of all propositional models of ϕ

Example: $\phi = \Box\Diamond p \vee (\neg p \wedge \Diamond\neg q) \quad \Rightarrow \quad Atoms(\phi) = \{\Box\Diamond p, \Diamond\neg q, p\}$

$\mu_1 = \{\neg p, \Diamond\neg q\}$, $\mu_2 = \{\Box\Diamond p\} \Rightarrow \mathcal{M} = \{\mu_1, \mu_2\}$ is a complete set of prop. models for ϕ

Key Property

Proposition

Let ϕ be a $T-$formula and let $\mathcal{M} = \{\mu_1, \ldots, \mu_n\}$ be a complete set of partial truth assignments for ϕ that propositionally satisfy ϕ. Then, ϕ is T-satisfiable iff. there exists a $\mu_j \in \mathcal{M}$ such that μ_j is $T-$satisfiable.

The proposition allows us to split checking for T-satisfiability in two orthogonal components:

- a purely propositional one, consisting of the search for propositional models μ_j of the T-formula ϕ
- a purely theory-specific one, determining the $T-$satisfiability of a set μ_j of T-literals (no boolean reasoning required here!)

Conceptual Components for Lazy SMT

We call a **truth assignment enumerator** a total function which takes as input a T-formula ϕ and returns a complete set $\mathcal{M}$ of propositional models for ϕ or NULL if there is none.

We call a **theory solver** a total function which takes as input a collection μ of T-literals and returns a (finite representation of a) T-model satisfying μ or NULL if there is none.

Analytic Tableau Procedure: Underlying Principles

Underlying principles are very intuitive ...

1. Assume a truth value for ϕ as $\top$
2. Derive truth values for direct subformula of a formula ϕ from the truth value of ϕ according to the semantics of the operators (**Analytical** / **Decomposition**)
3. Represent the different possibilities as an and-or-tree. A branch in the tree represents a specific case, and the tree is called tableau
4. A branch is closed (or contradictory), if it contains a literal and its complement. Closed branches do not have to be extended further.
5. If all branches are closed (contradictory), then the input formula is unsatisfiable. Otherwise, a non-closed but maximally expanded branch represents a model of the input formula.

The procedure does not require a specific normal form for the input problem!

Analytic Tableau Procedure: Decomposition Rules

$$\frac{}{\top : \phi}$$

$$\frac{\top : A \wedge B}{\begin{array}{c}\top : A \\ \top : B\end{array}} \qquad \frac{\bot : A \vee B}{\begin{array}{c}\bot : A \\ \bot : B\end{array}}$$

$$\frac{\top : A \vee B}{\top : A \mid \top : B} \qquad \frac{\bot : A \wedge B}{\bot : A \mid \bot : B}$$

$$\frac{\top : A \leftrightarrow B}{\begin{array}{ll}\top : A & \mid \bot : A \\ \top : B & \mid \bot : B\end{array}} \qquad \frac{\bot : A \leftrightarrow B}{\begin{array}{ll}\top : A & \mid \bot : A \\ \bot : B & \mid \top : B\end{array}}$$

$$\frac{\top : \neg A}{\bot : A} \qquad \frac{\bot : \neg A}{\top : A}$$

$$\frac{\begin{array}{c}\top : A \\ \bot : A\end{array}}{clash}$$

Analytic Tableau Procedure: Example

Formula to check for satisfiability:
$\phi(p,q,r,s) = (p \leftrightarrow q) \wedge (\neg p \leftrightarrow q) \wedge (r \vee s)$ (UNSAT)

Note: the search space consists of $2^4 = 16$ possible interpretations for $\{p,q,r,s\}$

(2) Perform Tableau Construction on ϕ

Observation: ...

The DPLL Procedure: Underlying Principles

Underlying principles are as well very intuitive ...

1. Iteratively build up an interpretation I for ϕ that satisfies ϕ (**Search-based procedure**)
2. At each step: first check if the SAT status is already clear, or if it is clear that we can not continue (**Backtracking**).
3. If not, then derive (deterministically) as much as possible how I must be extended without any guessing (**Boolean Constraint Propagation**)
4. If some variables can not be assigned this way: simply guess (**Heuristic guessing**)
5. After each extension step for I: Simplify the current formula ϕ by evaluation wrt. I. Shrinks the remaining problem, less variables (**Simplification**)

Choose a representation of ϕ that simplifies (2), (3), (4), (5):
Conjunctive Normal Form (CNF)

The DPLL Procedure: Example 1

Formula to check for satisfiability: $\phi(p,q,r) = (p \vee q) \wedge r \wedge \neg(p \vee \neg r) \wedge q$

(1) Convert ϕ to CNF: $CNF(\phi) = \{p \vee q\} \wedge \{r\} \wedge \{\neg p\} \wedge \{q\}$
Note: the search space consists of $2^3 = 8$ possible interpretations for $\{p,q,r\}$

(2) Perform DPLL search on $CNF(\phi)$:

$F_0 := \{p \vee q\} \wedge \{r\} \wedge \{\neg p\} \wedge \{q\}$ and $I_0 := \{\}$

(a) Are we done with F_0? No! No empty clause, and still clauses left!

(b) Derive determ. variable assignments from F_0:
Unit clauses $\{r\}$, $\{\neg p\}$ and $\{q\}$ enforce $I_1 := I_0 \cup \{r, \neg p, q\}$

(c) Simplify F_0 according to the inferred variable assignments:
$F_1 := \{\bot \vee \top\} \wedge \{\top\} \wedge \{\top\} \wedge \{\top\} = \top$

(d) Are we done with F_1? Yes! No clauses are left which can not be be assigned a truth value under I_1. Since $F_1 = \top$, $I_1 = \{r, \neg p, q\}$ is a model of $CNF(\phi)$ and hence for ϕ.

Observation: No guessing was needed. Truth functional analysis of $CNF(\phi)$ lead deterministically to the (only) model for $CNF(\phi)$!

The DPLL Procedure: Example 2

Formula to check for satisfiability:
$\phi(p,q,r,s) = (p \leftrightarrow q) \wedge (\neg p \leftrightarrow q) \wedge (r \vee s)$ (UNSAT)

(1) Convert ϕ to CNF:
$CNF(\phi) = \{\neg p \vee q\} \wedge \{\neg q \vee p\} \wedge \{\neg q \vee \neg p\} \wedge \{p \vee q\} \wedge \{r \vee s\}$
Note: the search space consists of $2^4 = 16$ possible interpretations for $\{p,q,r,s\}$

(2) Perform DPLL search on $CNF(\phi)$:

$F_0 := \{\neg p \vee q\} \wedge \{\neg q \vee p\} \wedge \{\neg q \vee \neg p\} \wedge \{p \vee q\} \wedge \{r \vee s\}$ and $I_0 := \{\}$

Observation: Using a „suitable" variable ordering (e.g. MOMS: $p, \neg p, q, \neg q \prec r, s$), we only construct (therefore consider) 2 cases (i.e. partial truth assignments) instead of 16 possible candidate interpretations.

Truth Assignment Enumeration via DPLL

```
function enumerate-models(φ);
DPLL(φ, ∅, 0);

function DPLL(φ, μ, current-level);
if φ = ⊤ then
    output μ;
    return current-level - 1;
end
if φ = ⊥ then
    blevel := analyze-conflict(φ, μ);
    return blevel;
end
if unit clause {l} occurs in φ then
    return DPLL(assign(l,φ), μ ∪ {l}, current-level);
end
l := choose-literal(φ,μ);
blevel := DPLL(assign(l,φ), μ ∪ {l}, current-level + 1);
if blevel > 0 and blevel = current-level then
    return DPLL(assign(¬l,φ), μ ∪ {¬l}, current-level + 1);
else
    return blevel;
end
```

DPLL vs. Analytic Tableau

DPLL is significantly more efficient than Analytic Tableau!

Summary: The power of DPLL stems from a few major sources:

- Guess only when there is no other choice, i.e. maximize determ. inference
- Simplify the inference problem as soon as new „knowledge” has been derived (to avoid redundancy)
- Branching on truth values (aka. semantic branching) instead of branching on subformulas (aka. syntactic branching)
- It is easy to incorporate conflict-driven search (learn from your failures for the future!)

DPLL(T): Extending DPLL by a theory solver

```
function DPLL-T(φ, μ, current-level);
if φ = ⊤ then
    | I := T-solver(μ);
    | if I ≠ NULL then
    |     | output I;
    |     | return 0;
    | else
    |     | return current-level - 1;
    | end
end
if φ = ⊥ then
    | blevel := analyze-conflict(φ, μ, blevel);
    | return blevel;
end
if unit clause {l} occurs in φ then
    | return DPLL(assign(l,φ), μ ∪ {l}, current-level);
end
l := choose-literal(φ,μ);
blevel := DPLL(assign(l,φ), μ ∪ {l}, current-level + 1);
if blevel > 0 and blevel = current-level then
    | return DPLL(assign(¬l,φ), μ ∪ {¬l}, current-level + 1);
else
    | return blevel;
end
```

Example: KSAT Procedure for $\mathbf{K}$

```
function KSAT(φ);
DPLL-T(φ, ∅, 0);
```

```
function T-solver(μ);
// μ = {□α_1, ..., □α_N, ¬□β_1, ..., ¬□β_M, A_1, ..., A_R, ¬A_{R+1} ... ¬A_S};
for each literal ¬□β_j ∈ do
    if KSAT(⋀_k α_k ∧ ¬β_j) outputs false then
        return false;
    end
end
return true;
```

KSAT corresponds to a depth-first tableau construction procedure using DPLL for propositional model enumeration

KSAT had a decisive influence on modern Description Logic systems (outperformed previous tableau procedures impressively!)

Improving the efficiency of Lazy SMT

A loose interaction between the propositional solver and the T-solver does not give the optimal performance

A tighter integration (or richer interaction) between T-solver and propositional solver is often desirable: exchange of more theory-specific information to the propositional solver

Important property for propositional solver: non-redundant enumeration of models (e.g. provided by DPLL, OBDDs)

Desired Features of Theory Solvers

Model Generation: when invoked on a T-consistent set μ the T-solver can produce a T-model $\mathcal{I}$ witnessing the satisfiability of μ

Conflict Set Generation: when invoked on a T-inconsistent set μ the T-solver can produce a (possibly minimal) subset $\nu \subseteq \mu$ which caused the inconsistency (ν = theory conflict set)

Incrementality: T-solver „remembers" its computation status from one call to another and thus can avoid the recomputation of already known partial results

Backtrackability: it is possible for the T-solver to undo steps and return to a previous state in a efficient manner

Deduction of unassigned literals: when invoked on a T-consistent set μ the T-solver can produce a set of T-literals l occuring in ϕ such that $\mu \models_T l$

DPLL(T): tighter integration of the theory solver

```
function DPLL-T(φ, μ, current-level);
if φ = ⊤ then
    I := T-solver(μ);
    if I ≠ NULL then
        output I;
        return 0;
    else
        blevel := analyze-T-conflict(μ,blevel);
        return blevel;
    end
end
if φ = ⊥ then
    blevel := analyze-conflict(φ, μ, blevel);
    return blevel;
end
if unit clause {l} occurs in φ  or l ∈ T-deduce(φ, μ) then
    return DPLL(assign(l,φ), μ ∪ {l}, current-level);
end
if likely-unsatisfiable(μ) then
    I := T-solver(μ);
    if I = NULL then
        blevel := analyze-T-conflict(μ,blevel);
        return blevel;
    end
end
l := choose-T-literal(φ,μ);
blevel := DPLL(assign(l,φ), μ ∪ {l}, current-level + 1);
if blevel > 0 and blevel = current-level then
    return DPLL(assign(¬l,φ), μ ∪ {¬l}, current-level + 1);
else
    return blevel;
end
```

Further Techniques to Improve Efficiency

Preprocessing theory literals / input formula: normalize theory literals occurring in ϕ such that previously syntactically different, but semantically equivalent theory atoms are made syntactically the same (e.g. associativity, sorting, removing dual operators, exploit theory specific properties, ...)

Memoizing / SAT status Caching: When a T-solver is invoked on a propositional model μ, use previous T-satisfiability / unsatisfiability results for supersets / subsets of μ

Static Learning: Analyze input before the propositional solver is invoked and identify sets of literals μ occurring in the input formula which are T-inconsistent. Add then an additional constraint $\neg\mu$ to the input formula ϕ

Theory-driven Learning: If the theory solver is invoked on a T-inconsistent set of literals μ, then add (or learn) an additional constraint $\neg\mu$ to the input formula ϕ considered for T-satisfiability