CS120: Intro. to Algorithms and their Limitations Hesterberg & Vadhan

Lecture 24: Satisfiability Modulo Theories and Cook–Levin

Harvard SEAS - Fall 2022 *2022-11-22*

1 Announcements

Recommended Reading:

- De Moura & Bjørner: https://cacm.acm.org/magazines/2011/9/122785-satisfiability-modulo-theories/abstract

Announcements:

- PS9 due this Fri 12/2
- Modified office hours and section this week
- Final exam Thu 12/8
- Final review sessions starting this Thurs
- PP3, revisions on PS8 & 9 due 12/8 but will be accepted on 12/9 without penalty
- Lowest Participation Portfolio will be dropped
- Please fill out Q survey
- Vote on T-shirts!
- The material below is mostly for fun/culture; you do not need to be able to work with it at a technical level.

2 Satisfiability Modulo Theories

Definition 2.1. A *theory* $\mathcal{T}$ consists of a domain $\mathcal{D}$ and a collection $\mathcal{P}$ of predicates $p : \mathcal{D}^k \to \{0,1\}$, $k \geq 0$.

Examples:

- Theory of Naturals: $\mathcal{D} = \mathbb{N}$, with constraints
  - $p_+(x,y,z) = [x + y \stackrel{?}{=} z]$,
  - $p_\times(x,y,z) = [x \times y \stackrel{?}{=} z]$, and
  - $p_c(x) = [x \stackrel{?}{=} c]$ for each $c \in \mathbb{N}$
- Theory of Bitvectors of length w: $\mathcal{D} = \{0,1,\ldots,2^w - 1\} \equiv \{0,1\}^w$, with constraints

- $p_{\texttt{op}}(x, y, z) = [x \ \texttt{op} \ y \stackrel{?}{=} z]$ for whatever word operations `op` we want to allow (e.g. saturation arithmetic or modular arithmetic, possible bitwise XOR/AND/OR, etc.)
- $p_c(x) = [x \stackrel{?}{=} c]$ for each $c \in \mathcal{D}$.

- Theory of Difference Arithmetic: $\mathcal{D} = \mathbb{Q}$, with constraints
  - $p_c(x, y) = [x - y \stackrel{?}{\leq} c]$ for each $c \in \mathbb{Q}$.

- Theory of Disjunctions: $\mathcal{D} = \{0, 1\}$, with constraints
  - $p_{k,\ell}(x_0, \ldots, x_{k+\ell-1}) = [x_0 \vee \cdots \vee x_{k-1} \vee \neg x_k \vee \cdots \vee \neg x_{k+\ell-1}]$ for every $k, \ell \in \mathbb{N}$.

Input	**:** A CNF formula $\varphi(x_0, \ldots, x_{n_p-1}, y_0, \ldots, y_{n_q-1})$ on n_p *propositional variables* and n_q *auxiliary variables*, a sequence $z = (z_0, \ldots, z_{n_t-1})$ of n_t *theory variables*, and a sequence of n_q *theory predicates* $P_0(z), \ldots, P_{n_q-1}(z)$, each of which is obtained by applying a predicate $p \in \mathcal{P}$ to a sequence $(z_{i_0}, \ldots, z_{i_{k-1}})$ of theory variables.
Output	**:** Assignments $\alpha \in \{0,1\}^{n_p}$, $\beta \in \mathcal{D}^{n_q}$ such that $$\varphi(\alpha, P_0(\beta), \ldots, P_{n_q-1}(\beta)) = 1,$$ if such assignments exist.

Computational Problem SatisfiabilityModuloT

Example: Recall the constraints 1, 3, and 4 from the Binary Search SMT solver in Lec. 23. The below formula is $1 \wedge 3 \wedge 4$:

$$((0 \leq \ell) \wedge (\ell \leq u)) \wedge (x_2) \wedge (\neg(x_2 \wedge (\ell < u)) \vee x_3)$$

We can express this as an SMT instance over the theory of naturals by introducing some additional theory variables z_0, z_1, z_2, z_3 as follows:

$$(z_0 = 0) \wedge (z_0 + z_1 = \ell) \wedge (\ell + z_2 = u) \wedge (x_2) \wedge (\neg x_2 \vee (u + z_3 = \ell) \vee x_3),$$

which corresponds to the following inputs to SMT:

- CNF formula: $\varphi(x_2, x_3, y_0, y_1, y_2, y_3) = (y_0) \wedge (y_1) \wedge (y_2) \wedge (x_2) \wedge (\neg x_2 \vee y_3 \vee x_3)$
- Theory variables: $\ell, u, z_0, z_1, z_2, z_3$
- Theory predicates:

$$\begin{aligned} P_0 &= [z_0 \stackrel{?}{=} 0] \\ P_1 &= [z_0 + z_1 \stackrel{?}{=} \ell] \\ P_2 &= [\ell + z_2 \stackrel{?}{=} u] \\ P_3 &= [u + z_3 \stackrel{?}{=} \ell] \end{aligned}$$

The solvability of Satisfiability Modulo $\mathcal{T}$ depends on the choice of the theory $\mathcal{T}$. It can be simplified using the following formulation.

Input : A sequence $z = (z_0, \ldots, z_{n-1})$ of n *theory variables*, and a sequence of *theory predicates* $Q_0(z), \ldots, Q_{m-1}(z)$, each of which is obtained by applying a predicate $p \in \mathcal{P}$ *or its negation* to a sequence $(z_{i_0}, \ldots, z_{i_{k-1}})$ of theory variables.

Output : An assignment $\beta \in \mathcal{D}^n$ such that

$$Q_0(\beta) = Q_1(\beta) = \cdots = Q_{m-1}(\beta) = 1.$$

if such an assignment exists.

Computational Problem ConstraintSatisfactionInT

Example: Constraint Satisfaction in the Theory of Disjunctions is equivalent to SAT

Theorem 2.2. *For every theory $\mathcal{T}$, Satisfiability Modulo $\mathcal{T}$ is solvable iff Constraint Satisfaction in $\mathcal{T}$ is solvable.*

Proof sketch.
$\Rightarrow$ Immediate because conjunctions of literals are a special case of CNF formulas.
$\Leftarrow$ We give a (exponential-time) reduction from Satisfiability Modulo $\mathcal{T}$ to Constraint Satisfaction in $\mathcal{T}$:

Input : An Satisfiability Modulo $\mathcal{T} = (\mathcal{D}, \mathcal{P})$ instance $\varphi(x_0, \ldots, x_{n_p-1}, y_0, \ldots, y_{n_q-1})$, $z = (z_0, \ldots, z_{n_t-1})$, $P_0(z), \ldots, P_{n_q-1}(z)$
Output : $\alpha \in \{0,1\}^{n_p}$, $\beta \in \mathcal{D}^{n_q}$ s.t. $\varphi(\alpha, P_0(\beta), \ldots, P_{n_q-1}(\beta)) = 1$, or $\perp$ if none exist.

```
foreach α ∈ {0,1}^{n_p}, γ ∈ {0,1}^{n_q} do
    if φ(α, γ) = 1 then
        foreach i = 0 to n_q-1 do
            if γ_i = 1 then Q_i = P_i;
            else Q_i = ¬P_i;
        β = ConstraintSatisfaction(z, Q_0, ..., Q_{n_q-1});
        if β ≠ ⊥ then return (α, β);
return ⊥
```

Algorithm 1: Exhaustive-Search SMT Solver

□

In practice, it's better to use a SAT solver rather than exhaustive search to find a satisfying assignment (α, γ) to φ, and then apply a constraint satisfaction solver. If the constraint satisfaction solver determines there is no corresponding assignment to the theory variables, use its output to construct a new clause to add to φ. For example, we can add a clause of length n_q over the auxiliary variables y ruling out the specific assignment γ, but sometimes the constraint solver can provide more information that allows for constructing a shorter clause that rules out more assignments.

It turns that the Theory of Naturals is unsolvable (proven similarly to the optional problem 3a on PS9), but the Theory of Difference Arithmetic, the Theory of Bitvectors are solvable.

3 Program Verification and the Cook–Levin Theorem

SMT Solvers are a powerful tool for verifying properties of time-bounded algorithms, like we saw with Binary Search. Indeed, we will see how they can be used to solve the following very general problem.

Input : A Word RAM program P, an array of natural numbers $x = (x_0, \ldots, x_{n-1})$ and parameters $w, m, t \in \mathbb{N}$.

Output : An array $y = (y_0, \ldots, y_{m-1})$ of natural numbers y such that:

1. Throughout the computation of P on input (x, y) the word length is at most w.
2. P halts on input (x, y) within t steps, and
3. $P(x, y) = 1$,

if such a y exists.

Computational Problem WordRAMSatisfiability

Many debugging problems can easily be reduced to WordRAMSatisfiability, if we fix bounds on the input length and the running time that we care about. For example, if we take $n = 0$ and modify P to output 1 only when an arithmetic overflow occurs, then WordRAMSatisfiability will tell us whether there is an input y of length m that causes an arithmetic overflow within t steps.

Furthermore, we can easily reduce every $\mathsf{NP}_{\mathsf{search}}$ problem Π to WordRAMSatisfiability, by taking P to be the verifier of solutions for Π: given an instance x of Π, we can set $m = n^{O(1)}$ to be the length of solutions for Π on input x and $t = n^{O(1)}$ to be the running time of the verifier. By using bignums, it can be shown that the word length w can be assumed to be $O(\log n)$ without loss of generality.

Theorem 3.1. *WordRAMSatisfiability can be reduced to Satisfiability Modulo the Theory of Bitvectors. Given an instance (P, x, w, m, t) of WordRAMSatisfiability, the (mapping) reduction produces an SMT instance with the same word size parameter w and runs in time polynomial in $|P|$, $|x|$, m, and t.*

Proof sketch. Given a WordRAMSatisfiability instance (P, x, w, m, t), we construct our SMT instance very similarly to the binary search example:

- Propositional variables: $u_{i,j}$ for $i = 0, \ldots, \ell$ and $j = 0, \ldots, t$, meant to capture whether the computation of P on (x, y) is at line i at time j.
- Theory variables: $\texttt{var}_{i,j}$ for $i = 0, \ldots, v-1$ and $j = 0, \ldots, t$ and $M_j[i]$ for $i = 0, \ldots, n+t-1$ and $j = 0, \ldots, t$ corresponding to the value of P's v variables and the values in the first $n+t$ memory locations of P at time j. (Note that in t time steps, P can use at most $n+t$ memory locations, since it can issue at most t MALLOC commands.)
- Constraints: initialization of the variables, memory locations, and line number of P at time 0; updates from time t to time $t+1$; and requirement that P halts in t steps with output 1

□

Theorem 3.2. *Satisfiability Modulo the Theory of Bitvectors reduces to SAT in polynomial time.*

Proof sketch. In case $w = O(\log n)$: we can just replace each w-bit theory variable with w Boolean variables, and each constraint $p(x, y, z)$ on ≤ 3 w-bit words with the brute-force CNF of size $2^{3w} = n^{O(1)}$.

A more efficient solution (to avoid the exponential dependence on w): like the optional problem on Problem Set 7, introducing auxiliary variables to simplify the verification of the predicate. For each predicate $p(x, y, z)$ (e.g. $[x + y \stackrel{?}{=} z]$), we can construct a CNF formula $\psi(x, y, z, a)$ of size $w^{O(1)}$ with auxiliary Boolean variables a such that $p(x, y, z) \Leftrightarrow \exists r\ \psi(x, y, z, a)$. For example, to verify binary addition with a small CNF, we can take a to represent the carry bits in the addition. For multiplication, we can use all the intermediate values in the computation of the grade-school multiplication algorithm.

□

Note that the $\mathsf{NP}_{\mathsf{search}}$-hardness of SAT (i.e. the hard part of the Cook–Levin Theorem, which we stated without proof in Lecture 19) follows by combining Theorems 3.1 and 3.2.