ETH zürich

Alexander J. Summers

Program Verification

Exercise Sheet 3: SMT Solving Algorithms

Assignment 1 (Propositional Abstraction)

Let T be T_P (Presburger Arithmetic).

1. $x < 0 \wedge x > 1$ is unsatisfiable, but $p \wedge q$ (its propositional abstraction) is satisfiable.
2. Suppose A is $x > 0$ and B is $x > 1$. Then $A \models B$ but $p \not\models q$
3. The simplest option is to choose A to include only propositional literals; e.g. $p \Rightarrow q$ or $\top$. An alternative is to use theory literals whose meaning is irrelevant for the formula. For example, $x > 0 \Rightarrow x > 0$ or $\bot \Rightarrow x < 4$.

Assignment 2 (Theory Deduction)

Extra literals added to the model effectively extend the search space for the DPLL search (a new fact to backtrack on, use in clause learning, etc.). If the theory solver does this infinitely, this could prevent the search from terminating. For example, an integer theory solver could add the facts $1 > 0$, $2 > 0$, $\ldots$ indefinitely, potentially preventing termination.

Theory deductions involving new literals can still be safely allowed, provided that we have some guarantee that there will be at most finitely many of them for any given problem. For example, allowing an arithmetic solver to add facts which are consequences of transitive inequalities already in the problem (e.g. $a \leq c$ in the exercise sheet question) will be safe, since there are finitely many such literals for any given input formula. In general, if we can show that some finite set of possibly-generated literals exists for any given problem, then the theory deduction will not prevent termination (note that there is no need for the implementation to compute this set; it is enough to be sure that one exists).

In practice, of course, even adding finitely many additional facts could slow the overall search down, if these facts turn out not to be relevant. In practice, just as for the clause-learning mechanisms in CDCL, finely-tuned heuristics are employed for deciding which new facts to learn, and many tools also allow these additional facts to be *removed* during search.

Assignment 3 (Applying Nelson-Oppen)

To deal with the first literal (which includes symbols from both the uninterpreted functions and integer theories), we introduce fresh uninterpreted constants m and n and suitable equalities (as described in the lectures), obtaining:

$$\neg(m - n = 0) \land i - j = 0 \land m = f(i) \land n = f(j)$$

Note that this formula has the desired property that each literal contains only symbols understood by one of the two theories (recall, as discussed in the lecture, that we extend each theory's signature to also include the newly-introduced uninterpreted constants m and n). The set of *shared constants* is $\{m, n, i, j\}$.

There are 15 equivalence relations over 4 elements (why?). In each case, we will fail to find a model (since the formula is unsatisfiable)[1].

For each of the equivalence relations, we first conjoin the equalities and disequalities reflecting the equivalence relation, and then run DPLL (or, if preferred, CDCL) on the propositional abstraction of the formula (say $\neg p \land q \land r \land s \land \ldots$ (where $\ldots$ are all of the propositional constants abstracting the equalities and disequalities from the equivalence relation). For this particular example, there is no serious search to perform here, since all clauses are already unit clauses; by unit propagation we will arrive at the candidate propositional model containing $\neg p, q, r, s$ without needing any decision literals. For this reason, there is also no DPLL-backtracking (and using the CDCL algorithm would not make a difference, since this only affects backtracking behaviour during the search).

Having obtained this extended model, we have to ask the theory solvers to check it for consistency with the theory. That is, we will ask the arithmetic solver to check the set of literals $\{\neg(m - n = 0), i - j = 0\}$ *along with* the equalities and disequalities from our equivalence relation. Similarly, the uninterpreted functions solver will be asked to check $\{m = f(i), n = f(j)\}$ along with these (dis)equalities. Depending on the particular equivalence relation we are working under, either the arithmetic solver will report inconsistency (if we included either $i \neq j$ or $m = n$) or the uninterpreted functions solver will do so (if we included both $i = j$ and $m \neq n$). In this case, we must repeat with another such equivalence relation[2]. After trying all 15 possibilities, we will return `unsat`.

For deterministic Nelson-Oppen, we will begin the DPLL process immediately, arriving at a model $\{\neg p, q, r, s\}$ again via unit propagation. When we give the corresponding literals $\{\neg(m - n = 0), i - j = 0\}$ to the arithmetic solver, it can report consistency but also give us the *implied equality*[3] $i = j$. We add this to our current model, and give $\{m = f(i), n = f(j), i = j\}$ to the uninterpreted functions solver. This finds the model to be consistent, and produces the further

---

[1] Of course, in the satisfiable case, guessing a "right" equivalence relation can dramatically speed up the runtime.

[2] Note that this "go back and try a new equivalence relation" looks similar to the backtracking of the DPLL search instead; in practice, we will flip some of e.g. the equalities from the equivalence relation to be disequalities and try again, similarly to the way we flip decision literals. In fact, this similarity has been explored in depth: *Lazy Theory Combination* allows the DPLL/CDCL engine to manage the branching on decisions which make up the definition of the equivalence relations, rather than deciding them up-front. An attractive feature of this approach is that it works regardless of whether the theories are convex (in which case, via theory learning, one might quickly constrain the equivalence relation to the "right" one via learned clauses) or not, in which case the DPLL-search will explore the remaining possibilities.

[3] In the lecture, I claimed that we also pass disequalities between solvers; this is not what the original Nelson-Oppen procedure does.

implied interface equality $m = n$. We now invoke the arithmetic solver on $\{\neg(m-n = 0), i-j = 0, i = j, m = n\}$ and it rejects the model as inconsistent, so we can terminate and return `unsat`.

Note that Presburger arithmetic in general is not convex: e.g. $x \geq 0, x \leq 1 \models x = 0 \vee x = 1$ but $x \geq 0, x \leq 1 \not\models x = 0$ and $x \geq 0, x \leq 1 \not\models x = 1$.

Assignment 4 (Theory Combination)

Note that a theory is consistent if and only if its set of models is non-empty (since, in any model, $\perp$ is not true).

The basic idea is to pick theories which impose finiteness constraints on models. For example, let's define a theory T_1 to include an uninterpreted sort S, uninterpreted constants from this sort c, d, e and to include exactly the models validating the formula $\forall x : S.x = a \vee x = b$. In particular, these models must interpret the sort S as a set of at most two elements.

Now let's define T_2 to be a theory also including S, c, d, e in its signature, and whose models are exactly those satisfying $c \neq d \wedge d \neq e \wedge c \neq e$. These models must interpret S to be a set of at least three elements.

Each theory is consistent by itself, but the combination is not: there are no models in the combination of these two theories.