ETH zürich

Alexander J. Summers

Program Verification

Exercise Solutions 4: Quantifiers

Assignment 1 (Rewriting and Skolemization)

Technically, we can simply pull the negation out from under the existential $\exists z$, rewriting $\exists z.\neg(\ldots)$ as $\neg\forall z.(\ldots)$, and we'll have a formula in extended CNF. However, the intention is to simplify the formula, and pushing the negations inwards will help, here (especially for Assignment 2).

If we push the outermost negation inwards (using the equivalence $\neg(A \Rightarrow B) \equiv (A \wedge \neg B)$), we obtain instead:

$$\exists z.((\forall n.g(n,z) \wedge \exists m.(\neg n = z \Rightarrow s(m) = n)) \wedge c \neq z) \wedge \forall w.\neg s(s(s(w))) = s(s(c))$$

Applying Skolemization to the outer existential, we replace z with some fresh constant symbol z' in the body, obtaining:

$$(\forall n.g(n,z') \wedge \exists m.(\neg n = z' \Rightarrow s(m) = n)) \wedge c \neq z' \wedge \forall w.\neg s(s(s(w))) = s(s(c))$$

We can similarly apply Skolemization to the $\exists m.$, but since it occurs under the $\forall n.$ we have to introduce a *function* f, replacing m with $f(n)$ to obtain:

$$(\forall n.g(n,z') \wedge (\neg n = z' \Rightarrow s(f(n)) = n)) \wedge c \neq z' \wedge \forall w.\neg s(s(s(w))) = s(s(c))$$

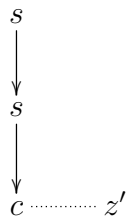
This leaves us with three (generalised) unit clauses, conjoined together.

Assignment 2 (E-graphs and E-matching)

We start from the formula that we computed in the solution for Assignment 1:

$$(\forall n.g(n,z') \wedge (\neg n = z' \Rightarrow s(f(n)) = n)) \wedge c \neq z' \wedge \forall w.\neg s(s(s(w))) = s(s(c))$$

The only ground terms are z', c, $s(c)$ and $s(s(c))$, and the only known (in)equality facts (after initial DPLL search) will be the inequality between c and z'. Thus, we should get an E-graph:



A simple choice of triggers would be $\{s(n)\}$ for the first quantifier, and $\{s(w)\}$ for the second. In both cases, we would get matching loops (can you see why?). Unfortunately, avoiding matching loops is difficult for the second quantifier (for the first, choosing e.g. $\{s(f(n))\}$ might be acceptable): choosing $\{s(s(s(w)))\}$ as a trigger would avoid matching loops but wouldn't allow us to make any instantiations of the quantifier for this example.

Sticking with the simplest choice of triggers, then, we can instantiate the first quantifier with e.g. c replacing n, since we have the term $s(c)$ in our E-graph. This yields the assertion $g(c, z') \wedge (\neg c = z' \Rightarrow s(f(c)) = c))$, which, combined with $c \neq z'$ allows us to deduce $s(f(c)) = c$. Now, we can instantiate the second quantifier, replacing w with $f(c)$ (since $s(f(c))$ will now be in our E-graph). This gives us $\neg s(s(s(f(c)))) = s(s(c))$, which contradicts $s(f(c)) = c$, giving us `unsat`.

Assignment 3 (Axiomatising Duplicate-Freeness)

The only reasonable choice of triggers is the following:

$$\forall i : \mathrm{Int}, j : \mathrm{Int}.\{\textit{lookup}(a, i), \textit{lookup}(a, j)\}\ \neg\ i{=}j \Rightarrow \neg\ \textit{lookup}(a, i){=}\textit{lookup}(a, j)$$

This will cause quadratically many instantiations of the axiom in the number of ground $\textit{lookup}(a, k)$ terms encountered in the problem; one instantiation for each pair of terms (including instantiations cause by the same term twice).

An alternative is to introduce an "inverse" function for *lookup*. Since there are no duplicates, there must exist an inverse mapping back from the array elements to the indices. We can make this assumed inverse explicit by introducing a function *lookup_inv* from Int to Int, and using the following quantifiers instead of the one from the question:

$$\forall i : \mathrm{Int}.\{\textit{lookup}(a, i)\}\ \textit{lookup\_inv}(\textit{lookup}(a, i)){=}i$$

This quantifier is sufficient to imply the previous one, but only gets instantiated once per ground *lookup* term.

It might be tempting to also add the dual axiom:

$$\forall j : \mathrm{Int}.\{\textit{lookup\_inv}(j)\}\ \textit{lookup}(a, \textit{lookup\_inv}(j)){=}j$$

but this would have the effect of guaranteeing that *every* integer occurs somewhere in the array. Even for infinite arrays, this is not necessarily true; for example, consider the array which stores twice the value of a location's index at each location (no odd integers occur in the array). This second axiom would introduce inconsistency in such an example (and is not necessary to express duplicate-freeness, in any case).