Homework Assignment 3

Due: November 18, 2014 at 11:00pm

Total points:	100
Deliverables:	`hw3.pdf` containing typeset solutions to Problems 1-4 and 9.
	`tree.als` containing your Alloy encoding for Problems 5-8.
	`sort.dfy` containing your Dafny implementation for Problem 10.

1 Combining Theories with Nelson-Oppen (25 points)

1. (5 points) Recall that the theory of arrays $T_A = \{\mathit{read}, \mathit{write}, =\}$ is defined by the following axioms.

$$\begin{array}{l} \forall a, i, j.\ i = j \rightarrow \mathit{read}(a, i) = \mathit{read}(a, j) \\ \forall a, v, i, j.\ i = j \rightarrow \mathit{read}(\mathit{write}(a, i, v), j) = v \\ \forall a, v, i, j.\ i \neq j \rightarrow \mathit{read}(\mathit{write}(a, i, v), j) = \mathit{read}(a, j) \end{array}$$

   Prove that T_A is not convex by constructing $n \geq 3$ formulas in T_A such that $F_1 \Rightarrow (F_2 \vee \ldots \vee F_n)$ but $F_1 \not\Rightarrow F_i$ for any $i \in [2 \ldots n]$.

2. (5 points) Purify the following $T_= \cup T_R$ formula and show the resulting $T_=$ and T_R formulas.

$$g(x + y, z) = f(g(x, y)) \wedge x + z = y \wedge z \geq 0 \wedge x \geq y \wedge g(x, x) = z \wedge f(z) \neq g(2x, 0)$$

$T_=$	T_R
...	...

   Apply purification to the (current) innermost term first. If there are several innermost terms, prefer the leftmost one. Use a_i to refer to the i^{th} auxiliary literal, starting with a_1. All occurrences of the same term should be mapped to the same auxiliary literal. You do not need to show the individual steps of the purification process, just the final result.

3. (5 points) Use the Nelson-Oppen procedure to decide the satisfiability of the purified formula from Problem 2. In one sentence, state which version of the procedure you are using (general or specialized) and justify your choice. Show the equality propagation by filling out the table below. If T_i infers the j^{th} equality (or disjunction of equalities), enter it into the j^{th} row and i^{th} column only—leave the remaining column in that row empty.

$T_=$	T_R
...	...

4. (10 points) Let F be a conjunctive formula in a non-convex theory T. Let G be a finite disjunction of equalities $\bigvee_{i=1}^{n} u_i = v_i$, also in T, such that $F \Rightarrow G$. Describe an algorithm for computing a minimal disjunction G' of the equalities in G such that $F \Rightarrow G'$. If your algorithm returns a minimal disjunction with m equalities, then it should have invoked the decision procedure for T at most $O(m \log n)$ times.

2 Finite Model Finding with Alloy (25 points)

In this part of the assignment, you will write four short Alloy specifications and check their correctness with the help of Alloy's finite model finder (Lecture 9). To start, download alloy.jar and double click on it to launch the tool. You may also want to skim Parts 1 and 2 of the Alloy tutorial.

The following questions ask you to formally define different kinds of trees. We will only consider trees that have directed edges and no unconnected nodes. Such a tree is fully described by its set of edges. In Alloy, we model the edges of a tree (or, more generally, a graph) as a binary relation from nodes to nodes.

A skeleton solution can be found in tree.als. Complete the missing definitions and submit your copy of tree.als. Solutions will be automatically checked against a reference specification, so they need to be fully contained in the submitted file.

5. (5 points) A *tree* is a graph that satisfies additional properties. What are those properties? Formalize them by completing the definition of the `tree` predicate in tree.als. Use the Alloy tool to check that your definition is correct (i.e., it rejects relations that are not trees) and non-vacuous (i.e., it admits some relations) in a universe with a small number of nodes.
6. (5 points) Formalize the properties of a *spanning tree* (of a directed graph) by completing the definition of the `spanningTree` predicate in tree.als. Check your definition for correctness and vacuity errors.
7. (5 points) Define *binary trees* in terms of their *left* and *right* relations, which map tree nodes to their left and right children (if any), respectively. Use your definition to complete the `binaryTree` predicate in tree.als. Check your definition for correctness and vacuity errors.
8. (10 points) Define *binary search trees* in terms of their *left*, *right*, and *key* relations. As above, the *left* and *right* relations map tree nodes to their left and right children (if any). The *key* relation maps tree nodes to integer keys. Use your definition to complete the `binarySearchTree` predicate in tree.als. Check your definition for correctness and vacuity errors.

3 Reasoning about Programs with Hoare Logic (25 points)

9. (25 points) Prove the validity of the following Hoare triple:

```
{n ≥ 0 ∧ d > 0}
q := 0;
r := n;
while (r ≥ d) {
  q := q + 1;
  r := r - d;
}
{n = q * d + r ∧ 0 ≤ r < d}
```

Your answer should take the form of a *proof outline*, which annotates the program S with FOL predicates inferred by applying the rules of Hoare logic. For example, if a proof outline includes n consecutive predicates $F_1, \dots, F_n$, then it must be that case that $F_1 \Rightarrow \dots \Rightarrow F_n$, corresponding to the Rule of Consequnce. Similarly, each statement $s \in S$ must be surrounded by formulas P and Q such that $\{P\}S\{Q\}$ is a valid Hoare triple, according to the inference rule for S.

4 Verifying Programs with Dafny (25 points)

In this part of the assignment, you will use Dafny (Lecture 11) to verify a modified implementation of the insertion sort. You can either download and install Dafny or use the web interface at rise4fun. To get started, read the Dafny Guide, which describes all features of Dafny that are needed to complete the assignment.

10. (25 points) sort.dfy contains an implementation of the insertion sort and a partial correctness predicate: applying the `sort` method to an array a ensures that $a[i] \leq a[j]$ for all valid indices $i < j$. This predicate is not quite right as written, however, and the implementation is missing all annotations except for the desired post-condition on `sort`.

    Get Dafny to verify sort.dfy by annotating it with sufficient pre/post conditions, assertions, loop invariants, and frame conditions. When the verification succeeds, Dafny will print the following message: "Dafny program verifier finished with n verified, 0 errors" (where n is a small number). Submit your annotated copy of sort.dfy.