Class exercise: Equational reasoning about filesystems

COMP 150 - Applied Functional Programming

September 17, 2012

Equational reasoning about mutable data

We can do equational reasoning most easily about purely functional data structures (like `git`'s filestore), but it is also possible to use algebraic laws to reason about mutable data structures, such as the real filesystem. When reasoning about mutable data, we normally reason about a sequence of *commands*. Here are some examples of commands:

- Push 7 on the stack
- Pop the stack
- Put 83 in the queue
- Get the first element from the queue
- Remove file `/h/nr/cs/40/server/www/solutions/intro/fgroups.c`

When reasoning about a sequence of commands, a particularly easy and effective form of specification is to abstract away from (i.e., ignore) some of the crucial information that is *observed* about the mutable data structure. This form specifies only effects on underlying mutable structures.

Work all three problems below:

1. Specify a set of commands for a stack, and give algebraic laws that make it possible to prove when sequences of commands are equivalent.
2. Specify a set of commands for a queue, and give algebraic laws that relate sequences of commands.

   A particularly useful relation is the *approximation* or *refinement* relation, written $\sqsubseteq$ and pronounced "at least as defined as." If you took COMP 105 in 2011 or have studied denotational semantics, you may recognize this relation. We write

   $$C_1 \sqsubseteq C_2$$

   when C_2 is defined on as many data structures as C_1 is, and when both are defined, they have the same effect.
3. Write an equational specification of a filesystem. This includes a *signature* and *algebraic laws*.

   - Abstract away from irrelevant or difficult detail.
   - Try specifying both a purely functional data structure and a mutable data structure. See what you can get out of the two different forms.

Be prepared to argue *why your specifications are good* (or why they are not good).