Concepts of Programmig Language Design

MinHs Exercises

Gabriele Keller

November 6, 2024

1. **Extending MinHs:** Add let-bindings (with a single binding, like in our arithmetic expression language) to MinHs and extend the static semantics accordingly.

   How would the static semantics rule change if we extend the scope of the bound variable to the right hand side of the binding (like in Haskell)?

   Does the following expression type check according to your rules?

   ```
   let
      f = recfun g (Int -> Int) x = if x == 0 then 0 else 1 + f (x-1)
   in f 1
   ```

   If not, is this fixable?

2. **Type safety in MinHs :** Give a (partial) proof for type safety in MinHs. For a full proof, we need to show that

   - Progress: If we can derive $\vdash e : \tau$, then either e is a final value, or there is e' such that $e \mapsto e'$.
   - Preservation: If $e \mapsto e'$, and $\vdash e : \tau$ then $\vdash e' : \tau$

   Give for both progress and preservation all the base cases for the proof, and all inductive cases for `if`-expressions.

3. **Equational reasoning in (purely) functional languages.** In Haskell, computations which depend on an explicit state (such as computations of type `IO`) can be composed in a user-friendly way using the `do`-notation. However, this notation can be used for any type constructor `m` for which we can define the function `return :: a -> m a` and the infix operator `(>>=) :: m a -> (a -> m b) -> m b` such that the following equalities hold:

   - `return a >>= f` is equivalent to `f a`
   - `m >>= return` is equivalent to `m`
   - `(m >>= f) >>= g` is equivalent to `m >>= (\x -> f x >>= g)`

   For the type constructor `Maybe` we can define these functions as follows:

   ```
   data Maybe a = Nothing | Just a

   return :: a -> Maybe a
   return a = Just a

   (>>=) :: Maybe a -> (a -> Maybe b) -> Maybe b
   (>>=) Nothing  f = Nothing
   (>>=) (Just a) f = f a
   ```

   Show that the equalities listed above (also called Monad laws) indeed hold for any type correct application.