T-501-FMAL Programming languages, Assignment 2
Spring 2021
Due Friday 26 February 2021 at 23:59

The file Assignment2.fs contains partial implementations of two languages. Both languages support variables, arithmetical operations, and comparisons of the form `if positive e then et else ef` (represented by `IIfPositive(e, et, ef)` and `IfPositive(e, et, ef)`). All of these are meant to support both integers and floating point numbers; in particular, variables can hold both integers and floats.

In `iexpr`, integers are implicitly converted to floats when necessary, for example when adding an integer and a float. In `expr`, there are no implicit conversions (and so adding an integer to a float will fail), but an explicit cast `IntToFloat e` yields a float when `e` is an integer. `expr` also supports matching on values to determine if they are `int`s or `float`s: an expression `match e with I xi → ei | F xf → ef` (represented by `Match(e, xi, ei, xf, ef)`) evaluates to the value of `ei` if `e` is an integer and to the value of `ef` if `e` is a `float`; the local variable `xi` resp. `xf` obtains the value of `e`.

1. The evaluation function `ieval` fails for an expression using a variable that does not appear in the environment. Modify it so that variables default to integer `0` (for example, `ieval (IVar "x") []` should yield `I 0`).

   Hint: You need to replace the call of `lookup` with a call to a variation of it.

2. The implementation of the evaluation function for the explicit language is incomplete. Modify the evaluation function `eval` so that `Plus` works when works when both operands are `float`s (in addition to when both operands are integers). If one operand is a `float` and the other is an integer, evaluation should fail. Similarly modify `Times` and `Neg`. Also add clauses for `IfPositive` (which should support both integers and floats) and for `Match`.

3. Code a function `to_float : value -> float` that converts an element of `value` to a `float` (use F#'s builtin function `float : int -> float`).

4. Using `Match` among other things, implement functions

   ```
   to_float_expr : expr -> expr
   plus_expr : expr * expr -> expr
   times_expr : expr * expr -> expr
   ```

   such that

   ```
   eval (to_float_expr e) env = F (to_float (eval e env))
   eval (plus_expr (e1, e2)) env = plus_value (eval e1 env, eval e2 env)
   eval (times_expr (e1, e2)) env = times_value (eval e1 env, eval e2 env)
   ```

   You may assume that `e`, `e1`, and `e2` do not contain variables whose names begin with `_` (for example `_x`). Note that evaluation of `IntToFloat e` will fail when `e` is not an integer. For `plus_expr (e1, e2)` and `times_expr (e1, e2)`, you should allow one operand to be an integer and one to be a `float`, but if both are integers then the result should be an integer.

   Hint: You need to apply `Match` to the parameters of the functions.

5. Implement a function `add_matches : iexpr -> expr` to convert from the implicit language to the explicit one. You should use the functions you defined in the previous problem so that evaluation does not fail because of operands having different types. Your function should satisfy

   ```
   eval (add_matches e) env = ieval e env
   ```

   when `env` contains all of the variables that appear in `e`.

6. Write a type inference function `infer : expr -> tyenvir -> typ`. The `tyenvir` argument contains the types of the variables. Type inference should fail when the two operands of a `Plus` or `Times` have different types and also when the two branches of `IfPositive` have different types. Type inference should also fail when `e` is not an integer in `IntToFloat e`.

   Hint: `eval` is a good starting point for writing `infer`.

7. Write a function `add_casts : iexpr -> tyenvir -> expr` to convert from `iexpr` to `expr`, inserting casts `IntToFloat` where necessary, but not using `Match`. If `tyenv` contains the type of every variable that appears in `e`, then `infer (add_casts e tyenv) tyenv` should never fail. You should only insert the casts that are absolutely necessary to accomplish this; do not unnecessarily convert integers to floats.

8. Can `eval (add_matches e) []` and `eval (add_casts e []) []` have different behaviour?

   What about `infer (add_matches e) []` and `infer (add_casts e []) []`?

9. The file Assignment2.fs also contains a copy of the first stack machine from Expressions3.fs. Write a function `rlower : rcode -> rcode` that compiles away all of the `RPop`, `RDup` and `RSwap` instructions by replacing them with sequences of `RLoad`, `RStore` and `RErase` instructions. The instructions `rlower inss` should have the same behaviour as `inss`, but should not contain `RPop`, `RDup` or `RSwap`.