Homework # 7
due April 8, 13:00

Turn in your answers to §2,3 on paper at the beginning of lecture, and send your `lambda-ref.slf` by email to `scmalte@icf.ethz.ch`.

1 Reading

Please read Chapter 13 in your textbook.

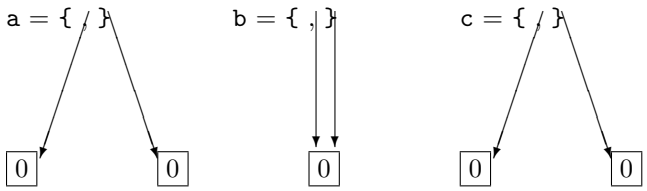
2 Problems

Please do the following problem:

- Exercise 13.1.1, and include the effects of evaluating

  ```
  c = (lambda x: Nat . {ref x, ref x}) 0
  ```

  Justify all three of your diagrams.



In the 'a' case, we evaluate two separate reference cells to store in the pair. The 'b' case is different, because by call-by-value, we evaluate the reference to a location (pointer) *before* duplicating it to two places. For the 'c' case, the duplication is of the value, there are still two separate reference cells being created, as with 'a.'

3 Discussion

1. The textbook says (page 167) that we must include a well typing in the requirements for Progress and Preservation. Give two counter-examples, one for Progress (Theorem 13.5.7) and one for Preservation (Theorem 13.5.3) if you omit the well typing requirement. In other words, give counter-examples of

   bogus progress If $\Gamma \mid \Sigma \vdash t : T$, then either t is a value or for any μ there exists t', μ' where $t \mid \mu \longrightarrow t' \mid \mu'$.

   bogus preservation If $\Gamma \mid \Sigma \vdash t : T$ and $t \mid \mu \longrightarrow t' \mid \mu'$ then there exists $\Sigma' \supseteq \Sigma$ where $\Gamma \mid \Sigma' \vdash t' : T$. (page 167)

   Progress Let $\Sigma = [l \to \texttt{Nat}]$. Then we can type `!l : Nat` using T-Loc. Now let μ be empty. Then `!l` $\mid \mu$ is stuck.

   Preservation As with the previous example but let μ be $[l \to \texttt{true}]$. Then `!l` $\mid \mu \to$ `true` $\mid \mu$, but the type of `true` does not have preserved type `Nat`.

2. Where do store typings come from; how does one get a store typing? In particular, for a programmer wanting to know whether their program will execute without getting stuck, how can they use the progress and preservation theorems? We understand that the new store typing from a preservation step can be fed back into the next use of progress, but how can one get an initial store typing?

> The three questions all are answered by that we start with the empty store typing, and thus we require that the program not have any locations in it, because otherwise they couldn't type (T-Loc won't work with an empty store typing). As execution progresses, we use the store typing provided by the preservation theorem for further progress/preservation uses.

4 Proofs

Complete the proofs of progress and preservation for STLC with references. We provide a skeleton file with all the syntax and judgments defined.

Unlike the textbook, stores are typed only in the empty context, but then we make the relation recursive over store types, so our relation is $\Sigma \vdash \mu : \Sigma$. The solution uses the following three "effectiveness" lemmas:

- Given a particular term and memory, it is always possible to allocate a cell for it.
- If we have a well-typed memory, and the memory typing has a binding for a location, then so does the memory typing. That is, if $\Sigma \vdash \mu$ and $\Sigma(l) = T$, then there exists a term t such that $\mu(l) = t$. (That it has the right type is part of preservation, not progress.)
- If we have a well-typed memory and the memory typing has a binding for a location, then we can update the memory at that location.

These three lemmas do the work of progress for allocation, dereference and assignment, respectively. Preservation requires several helper lemmas as well; see the skeleton file for more information.

Homework # 7.5
due April 8, 13:00

This is an *optional* homework assignment, due at the same time as the main homework. Another optional assignment will be assigned over the Easter break.

If you turn this assignment in and do better on it than a previous assignment, the better grade will replace it. Turn in the files `record.slf` and `fraction.slf` by email to `scmalte@inf.ethz.ch`.

5 Record Types

In the next few weeks, we will be adding subtypes and object-oriented features to record types. Prepare for these weeks by completing the proof of type soundness of record types. A skeleton file will be provided.

Repetition must be explicitly modeled in syntax. We use a new nonterminal `r` to refer to the contents of a record (and `R` for the contexts of a record type). We use `$` to refer to the empty record (type). Record fields are just natural numbers; equality and inequality are defined as done previously. Since SASyLF has no module system yet, the natural number definitions must be embedded into the proof file; starting this week, our proof files will be thousands of lines long.

6 Mechanization

Skim the paper "Checking Interference with Fractional Permissions" (Boyland 2003) and mechanize the definitions necessary to define evaluation. In other words, define the syntax and judgments for evaluation. Put the results in `fraction.slf`. Ironically, you will not need to define fractions.