Solution for Homework №1

Homework

PSV 2020/21

Author: *Jakub Bednarz*

To recap, we are given a “base” $\texttt{State}_0$, with an evaluator $[\![\cdot]\!]_0 : \mathit{Instr}_0 \to (\texttt{State}_0 \nrightarrow \texttt{State}_0)$, which contains every part of the described syntax and big-step semantics except for the transaction mechanism. Our goal is to construct $\texttt{State}$ and corresponding $[\![\cdot]\!] : \mathit{Instr} \to (\texttt{State} \nrightarrow \texttt{State})$.

Modifying the state First of all, we can assume that $\texttt{State}_0 = \texttt{Memory}_0 =: \texttt{Memory}$, since in the base language, only “stateful” feature is variable assigment (`while` loop isn't breakable), and nothing about it is modified in our extension of the language; we shall assume the isomorphisms to be implicitly there in the definitions. The new features are the following:

1. We need an indicator for whether `fail` has been successfully invoked (and, if so, about which particular `try`-block);
2. We need to create the snapshot of `Memory` to revert to if a transaction needs to be rescinded (or save to if `commit` is invoked); there is, however, some subtlety involved, as (from what I understand), saving the state at the beginning of a `try`-block is different from `commit`ing.
3. We also need to maintain some memory of the enveloping `try`-blocks (so as to ignore `fail` if the referenced transaction is not enveloping)

The following should therefore suffice:

$$\begin{aligned} \texttt{State} &:= (\texttt{Memory}, \mathit{TrId}?, \mathit{TrId} \nrightarrow \texttt{Memory}, [\mathit{TrId}]) \\ \ell(m) &= (m, \bot, \varnothing, []) : \texttt{Memory}_0 \to \texttt{Memory} \end{aligned}$$

These represent, respectively, base state, (optional) failed transaction indicator, memory snapshots' *map* and a list/stack of enveloping `try`-blocks; we also attach a state lifting function.

Semantics “Mathematically”:

$$[\texttt{OnFail}] \quad \frac{f \neq \bot, I \in \{\texttt{try}\ t : I, \texttt{fail}\ t, \texttt{commit}\}}{[\![I]\!]\ (m, f, \mu_s, T) = (m, f, \mu_s, T)}$$

$$[\texttt{Try}] \quad \frac{[\![I]\!]\ (m, \bot, \mu_s[t \leftarrow m], [t, *T]) = (m', t', \mu_s', [t, *T])}{[\![\texttt{try}\ t : I]\!]\ (m, \bot, \mu_s, T) = \begin{cases} (\mu_s'(t), \bot, \mu_s', T), & t = t' \\ (m', t', \mu_s', T), & t \neq t' \end{cases}}$$

$$[\texttt{Fail}] \quad [\![\texttt{fail } t]\!]\ (m, \bot, \mu_s, T) = \begin{cases} (m, t, \mu_s, T), & t \in T \\ (m, \bot, \mu_s, T), & t \notin T \end{cases}$$

$$[\texttt{Commit}] \quad [\![\texttt{commit}]\!]\ (m, \bot, \mu_s, T) = (m, \bot, \lambda\, t \in T.m, T)$$

$$[\texttt{Seq}] \quad [\![I_1;\ I_2]\!] = [\![I_2]\!] \circ [\![I_1]\!]$$

$$[\texttt{Lift}] \quad \frac{[\![I]\!]_0\ m = m'}{[\![I]\!]\ (m, t, \mu_s, T) = \begin{cases} (m, t, \mu_s, T), & t = \bot \\ (m', t, \mu_s, T), & t \neq \bot \end{cases}}$$

In the natural language:

[`OnFail`] If we are in "failed" state, it can only be reverted "higher-up" (in [`Try`]), and all the other instructions should be skipped. Here we handle the new instructions; the original ones are handled in `Lift`;

[`Try`] For `try` $t : I$, we evaluate I with t added to the stack (and memory snapshot for t saved), and if `fail` t has not been handled up to this point, we revert the memory to the snapshot associated with the flag and clear the flag; in any case, we pop t from the stack;

[`Fail`] For `fail` t, if a transaction named t indeed envelops the operation, we raise the failure flag; otherwise, the operation is supposed to do nothing;

[`Commit`] For `commit`, we assign to every current transaction a snapshot of current memory.

[`Seq`] No modification should be required in the implementation of the semicolon;

[`Lift`] We lift all the operations in the original language, according to the `fail` semantics.

Implementation I have created a simple implementation of an interpreter of the language. The relevant code and README.md is in `https://github.com/vitreusx/sem1/tree/master/hs`.