Urban Sorrow

April 4, 2018

1 Introduction

We want to define a translation from STlang to a pure untyped language, $\mathbf{\Lambda}$. It is the untyped lambda calculus with pairs, sum. For practicality, we assume it has primitive lists. In short, we aim to encode the state of the ST monad into a version of the *local state monad*. The (standard) local state monad was introduced, I think, by Gordon Plotkin and John Power in **??**. It provides operations to allocate, deallocate, read, and write references. Here, of course, we do not need the deallocation part. The idea why it should work is that the typing discipline of `runST` corresponds to some kind of *state passing*, like its name, State Threads, suggests.

2 The Translation

In the sequel, the letter $\mathfrak{s}$ corresponds to the state that is being passed arround. It is simply a list. The interesting cases of the translation are the following (the rest is just type erasure):

- $\underline{\texttt{readSTRef}\ r} = (\lambda x.\lambda\mathfrak{s}.\,(\mathfrak{s}[x],\mathfrak{s}))\underline{r}$
- $\underline{\texttt{writeSTRef}\ r\ t} = (\lambda xy.\lambda\mathfrak{s}.\,(\mathfrak{s}[x \mapsto y],\mathfrak{s}))\underline{r}\,\underline{t}$
- $\underline{\texttt{newSTRef}\ t} = (\lambda x.\lambda\mathfrak{s}.\,(\texttt{size}(\mathfrak{s}),\mathfrak{s}{+}{+}[\underline{x}]))\underline{t}$
- $\underline{\texttt{bind}\ t\ f} = (\lambda xg.\lambda\mathfrak{s}.\,(\lambda.(x',\mathfrak{s}').\,g\,x'\,\mathfrak{s}')(x\,\mathfrak{s}))\underline{t}\,\underline{f}$
- $\underline{\texttt{return}\ t} = (\lambda x.\lambda\mathfrak{s}.\,(x,\mathfrak{s}))\underline{t}$
- $\underline{\texttt{runST}\ t} = \pi_1(\underline{t}[])$

3 Correctness

We would like to show that this compiler is correct, in that:

Theorem 3.1 *Given a ground type τ, and $\Gamma \vdash t : \tau$, if $t \Downarrow v$, then $\underline{t} \Downarrow \underline{v}$.*

One way to achieve this is by building a logical relation $\approx$ between the two languages. The properties we with are the following.

Lemma 3.2 (Adequacy) *If $t \approx u$, then*

$$(t, \emptyset) \text{ terminates} \quad \Leftrightarrow \quad u \text{ terminates}$$

Lemma 3.3 (Fundamental lemma) *If $\Gamma \vdash t : \tau$, then $t \approx \underline{t}$.*

Corollary 3.4 (Equitermination of the translation) *A well typed term t equiterminates with its translation $\underline{t}$.*

Proposition 3.5 *$C[t]$ terminates if and only if $\underline{C}[\underline{t}]$ terminates.*

Corollary 3.6 (Reflection of contextual refinement) *Given two well typed terms t_1 and t_2,*

$$t_1 \leq^{ctx} t_2 \quad \Leftarrow \quad \underline{t_1} \leq^{ctx} \underline{t_2}$$

Hopefully, we can deduce from this (some of) the contextual inequations from the POPL paper, for we know that many contextual equivalences hold in $\mathbf{\Lambda}$. For example, for the commutativity of evaluation, this should work. The reason for this, I think, is that we translate the non-effectful fragment of the language by, essentially, the identity.

In more details, let us see how we would prove the "commutativity of effects". Consider t_1 and t_2 two well-typed programs, and write LR for the STLang program $(\lambda xy.(x,y))\, t_1\, t_2$, and RL for the program $(\lambda yx.(x,y))\, t_2\, t_1$. Now, according to Proposition 3.5, LR and RL are contextually equivalent as soon as no context of the form $\underline{C}$ can distinguish $\underline{LR}$ and $\underline{RL}$. As Λ has no effect, and as $\underline{(\lambda x.t)\, u} = (\lambda x.\underline{t})\, \underline{u}$, $\underline{LR}$ and $\underline{RL}$ *should* be equivalent.

4 The Logical Relation

The type of the logical relation would be:

$$\Gamma \vdash \cdot \approx \cdot : \tau \ : \mathbf{STLang} \times \mathbf{\Lambda} \to \mathit{iProp}$$

build, as usual, from a relation on closed values and terms

$$[\![\tau]\!]_\Delta : \mathit{Val}(\mathbf{STLang}) \times \mathit{Val}(\mathbf{\Lambda}) \to \mathit{iProp} \quad \text{and} \quad \mathcal{E}[\![\tau]\!]_\Delta : \mathbf{STLang} \times \mathbf{\Lambda} \to \mathit{iProp}.$$

To define this logical relation, let us posit the existence of a predicate $\mathsf{listelt}(\rho, \ell, pos, P)$, whose purpose is to *record* that the (heap) location ℓ corresponds to the element of $\mathfrak{s}$ at position pos, that they are part of the region ρ, and that the values they contain must be related at relation P (this is basically rel in POPL). Then, we can gather all the constraints associated to a region ρ using the $\mathsf{list}(\rho, \mathfrak{s})$ predicate to express that, for all related (ℓ, pos):

- $\mathfrak{s}$ is a list with at least $pos - 1$ element;

- if P is the predicate assocated to them, then $P(\ell, pos)$ holds.

We can now sketch the logical relation:

- $[\![\texttt{STRef}\ \rho\ \tau]\!](\ell, pos) = \mathsf{listelt}(\rho, \ell, pos, P)$
- $[\![\texttt{ST}\ \rho\ \tau]\!](v, v') = \forall \mathfrak{s}.\ \{\mathsf{list}(\rho, \mathfrak{s})\}\texttt{runST}\ v\{w.v'\ \mathfrak{s} \Downarrow w' * (w, w') \in [\![\tau]\!] * \mathsf{list}(\rho, \mathfrak{s})\}$
- ...

For now, this seems to be simpler than list time...

Question: What kind of Hoare triple would we need?