Store: Dynamics

Alex Kavvos

Reading: PFPL §34.1.2, 34.1.3

Recall that a **store** is a finite map $\mu : \mathsf{Loc} \rightharpoonup \mathsf{StoreVal}$, and that we write $\mu = \mu' \otimes \{a \mapsto v\}$ to mean that μ maps the location $a \in \mathsf{Loc}$ to the value v (i.e. $\mu(a) \simeq v$), and that μ' is the rest of the store.

The dynamics of stores consist of the following judgements.

$$e \ \mathsf{val} \qquad m \parallel \mu \ \mathsf{final} \qquad e \longmapsto e' \qquad m \parallel \mu \longmapsto_{\Sigma} m' \parallel \mu'$$

1 Values and Final States

e val is the same as for PCF expressions (but indexed by Σ) with one additional rule:

$$\textsc{Val-Cmd}\quad \frac{}{\mathsf{cmd}(m)\ \mathsf{val}}$$

The new judgement $m \parallel \mu$ final states that the command m has finished running, leaving the store in state μ. Only one command is final, viz. the one that returns a *value*:

$$\textsc{Final-Ret}\quad \frac{e\ \mathsf{val}}{\mathsf{ret}(e) \parallel \mu\ \mathsf{final}}$$

2 Transitions

The expression transitions $e \longmapsto e'$ are much the same as in PCF. The new command transitions are these:

$$\textsc{D-Get}\quad \frac{}{\mathsf{get}[a] \parallel \mu \otimes \{a \mapsto e\} \longmapsto_{\Sigma,a} \mathsf{ret}(e) \parallel \mu \otimes \{a \mapsto e\}}$$

$$\textsc{D-Set-1}\quad \frac{e \longmapsto e'}{\mathsf{set}[a](e) \parallel \mu \longmapsto_{\Sigma,a} \mathsf{set}[a](e') \parallel \mu}$$

$$\textsc{D-Set}\quad \frac{e\ \mathsf{val}}{\mathsf{set}[a](e) \parallel \mu \otimes \{a \mapsto \_\} \longmapsto_{\Sigma,a} \mathsf{ret}(e) \parallel \mu \otimes \{a \mapsto e\}}$$

$$\textsc{D-Ret-1}\quad \frac{e \longmapsto e'}{\mathsf{ret}(e) \parallel \mu \longmapsto_{\Sigma} \mathsf{ret}(e') \parallel \mu}$$

$$\textsc{D-Bnd-1}\quad \frac{e \longmapsto e'}{\mathsf{bnd}(e; x.\, m) \parallel \mu \longmapsto_{\Sigma} \mathsf{bnd}(e'; x.\, m) \parallel \mu}$$

$$\textsc{D-Bnd-Cmd}\quad \frac{m_1 \parallel \mu \longmapsto_{\Sigma} m_1' \parallel \mu'}{\mathsf{bnd}(\mathsf{cmd}(m_1); x.\, m_2) \parallel \mu \longmapsto_{\Sigma} \mathsf{bnd}(\mathsf{cmd}(m_1'); x.\, m_2) \parallel \mu'}$$

$$\textsc{D-Bnd-Ret}\quad \frac{e\ \mathsf{val}}{\mathsf{bnd}(\mathsf{cmd}(\mathsf{ret}(e)); x.\, m) \parallel \mu \longmapsto_{\Sigma} m[e/x] \parallel \mu}$$

$$\textsc{D-Dcl-1}\quad \frac{e \longmapsto e'}{\mathsf{dcl}(e; a.\, m) \parallel \mu \longmapsto_{\Sigma} \mathsf{dcl}(e'; a.\, m) \parallel \mu}$$

$$\textsc{D-Dcl-2}\quad \frac{e\ \mathsf{val} \qquad m \parallel \mu \otimes \{a \mapsto e\} \longmapsto_{\Sigma,a} m' \parallel \mu' \otimes \{a \mapsto e'\}}{\mathsf{dcl}(e; a.\, m) \parallel \mu \longmapsto_{\Sigma} \mathsf{dcl}(e; a.\, m') \parallel \mu'}$$

$$\textsc{D-Dcl-Ret}\quad \frac{e\ \mathsf{val} \qquad e'\ \mathsf{val}}{\mathsf{dcl}(e; a.\, \mathsf{ret}(e')) \parallel \mu \longmapsto_{\Sigma} \mathsf{ret}(e') \parallel \mu}$$

The rules D-Dcl-1, D-Dcl-2, and D-Dcl-Ret implicitly define the concept of **block structure**. As a result, this language can be implemented using just a stack: there is no heap allocation! (Hence everything is deterministic.)

3 Type safety

We define

$$\mu : \Sigma \quad \stackrel{\text{def}}{=} \quad \forall a \in \Sigma.\ \exists e.\ \mu(a) \simeq e \ \wedge\ e \text{ val} \wedge\ \vdash_\emptyset e : \text{Nat}$$

Type safety is given by the following two theorems. Both are shown by **simultaneous induction**.

Theorem 1 (Preservation)**.**

1. If $\vdash_\Sigma e : \tau$ and $e \longmapsto e'$ then $\vdash_\Sigma e' : \tau$.
2. If $\vdash_\Sigma m$ ok, $\mu : \Sigma$, and $m \parallel \mu \ \longmapsto_\Sigma m' \parallel \mu'$ then $\vdash_\Sigma m'$ ok and $\mu' : \Sigma$.

Theorem 2 (Progress)**.**

1. If $\vdash_\Sigma e : \tau$ then either e val or $e \longmapsto e'$ for some e'.
2. If $\vdash_\Sigma m$ ok and $\mu : \Sigma$ then either $m \parallel \mu$ final or $m \parallel \mu \ \longmapsto_\Sigma m' \parallel \mu'$ for some m', μ'.

4 First-order, mobility, and CBV

Notice that the dynamics of MA have a strong call-by-value flavour. For example, when executing a command $\text{ret}(e)$ the rule D-Ret-1 forces e to be fully evaluated to a numeral before returning it. Similarly, when executing $\text{dcl}(e; a.\, m)$ the rules D-Dcl-1 and D-Dcl-2 force e to be fully evaluated before assigning it to the store location a.

Furthermore, the judgement $\mu : \Sigma$ in the type safety proof requires that everything in the store be (a) numerical, (b) a value, and (c) typable without referring to anything in the store (the subscript Σ is required to be empty).

It is interesting to contemplate what would happen if these requires were not in place. In fact, the problem becomes evident even without far-reaching changes. Suppose we considered $\text{succ}(e)$ to be a value. Consider

$$m \stackrel{\text{def}}{=} \text{dcl}(\text{succ}(\text{let}(\text{cmd}(\text{get}[a]); x.\, \text{zero})); a.\, \{y \leftarrow \text{get}[a]; \text{ret}(y)\})$$

Executing this command in the empty store would allocate a with an initial value $\text{succ}(\text{let}(\text{cmd}(\text{get}[a]); x.\, \text{zero}))$. It would then fetch this value from a, and return it as-is. The final state would then be

$$\text{ret}(\text{succ}(\text{let}(\text{cmd}(\text{get}[a]); x.\, \text{zero}))) \parallel \emptyset$$

Notice that this value now refers to the location a; that is, the the location a has escaped its **scope**. In this case it is immaterial as the $\text{get}[a]$ command never runs. However, the preservation theorem fails: we have $\vdash_\emptyset m$ ok, yet $\not\vdash_\emptyset \text{ret}(\text{succ}(\text{let}(\text{cmd}(\text{get}[a]); x.\, \text{zero})))$ ok.

Thus, for MA to be type-safe, we must include the rule $\dfrac{e \text{ val}}{\text{succ}(e) \text{ val}}\ \textsc{Val-Succ}$ in the dynamics.

A similar problem occurs if the dynamics allow CBN-like behaviour in $\text{ret}(-)$ or $\text{dcl}(-; -.\, -)$.

Furthermore, a similar problems happens if we extend the store to allow non-numerical values. For example, if we allowed expressions of the form $\lambda x.\, e$, then the body e—which is almost never a value—could contain locations a which would then escape their scope of declaration.

How does the type safety proof work then? The key is the property of **mobility** which natural numbers satisfy.

Lemma 3 (Mobility)**.** If $\vdash_\Sigma e : \text{Nat}$ and e val then $\vdash_\emptyset e : \text{Nat}$.

Mobile data is data that do not refer to locations; they can safely be placed in the store.

We have left the dynamics of the PCF part of MA unspecified. The only restriction is that natural numbers need to be strict, i.e. the rule Val-Succ must be included. Otherwisewe are free to do anything we like: our language is **stratified** in a mostly-pure part (PCF) and the command language, so that the first is independent of the second.