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. Command transitions:

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

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

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

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

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

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

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

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

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

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

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

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

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 $\mathsf{ret}(e)$ the rule D-Ret-1 forces e to be fully evaluated to a numeral before returning it. Similarly, when executing $\mathsf{dcl}(e; a.\, m)$ 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 $\mathsf{succ}(e)$ to be a value, irrespective of the status of e. (This would give natural numbers a **lazy** semantics.) Then, consider the command

$$m \stackrel{\text{def}}{=} \mathsf{dcl}(a; \mathsf{zero}.\ \mathsf{ret}(\mathsf{proc}\,(x : \mathsf{Nat})\,\{\mathsf{set}[a](x)\}))$$

Executing this in the empty store $\emptyset$ allocates a with zero, and then returns a procedure. The final state is

$$\mathsf{ret}(\mathsf{proc}\,(x : \mathsf{Nat})\,\{\mathsf{set}[a](x)\}) \parallel \emptyset$$

When this procedure is called, it attempts to store the value of its argument x in a. But this is nonsensical, as a is no longer allocated! Thus, the location a has **escaped the scope** of the declaration $\mathsf{dcl}(a; \mathsf{zero}.\, -)$. Not knowing what a is, this procedure will generate a **stuck state**: the progress theorem will fail.

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

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

Furthermore, other issues occur 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** of natural numbers satisfy.

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

Mobile data are data that do not refer to locations, and hence can be placed in the store. Mobility does not hold if we omit Val-Succ.

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.