COMSM0067: Advanced Topics in Programming Languages

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.

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

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*:

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

Monday 16th September, 2024

2 Transitions

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

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

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

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

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

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

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

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

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

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

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

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

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 $\vdash_\Sigma m \ \mathsf{ok}$, $\mu : \Sigma$, and $m \parallel \mu \ \longmapsto_\Sigma m' \parallel \mu'$ then $\vdash_\Sigma m' \ \mathsf{ok}$ and $\mu' : \Sigma$.

Theorem 2 (Progress)**.**

1. If $\vdash_\Sigma e : \tau$ then either $e \ \mathsf{val}$ or $e \longmapsto e'$ for some e'.
2. If $\vdash_\Sigma m \ \mathsf{ok}$ and $\mu : \Sigma$ then either $m \parallel \mu \ \mathsf{final}$ 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 requirements were not in place. Suppose we allowed commands to return other types. Then, consider the command

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

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! In other words, the location a has **escaped the scope** of the declaration $\mathsf{dcl}(\mathsf{zero}; a. -)$. Not knowing what a is, this procedure will generate a **stuck state**: the progress theorem will fail.

Similar issues occur if we allow the store to contain non-strict numbers, i.e. if we admit that $\mathsf{succ}(e)$ val no matter what e is. The reason is that e could contain an occurrence to a location a which may have been deallocated.

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

How does the type safety proof work then? The key is the property of **mobility** that 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, so that the mobility lemma holds. Otherwise, the language is **stratified** in a mostly-pure part (PCF) and the command language, so that the first is independent of the second.