Arrays and strings

Dmitry Boulytchev

November 13, 2019

An array can be represented as a pair: the length of the array and a mapping from indices to elements. If we denote $\mathscr{E}$ the set of elements then the set of all arrays $\mathscr{A}(\mathscr{E})$ can be defined as follows:

$$\mathscr{A}(\mathscr{E}) = \mathbb{N} \times (\mathbb{N} \to \mathscr{E})$$

For an array (n, f) we assume $\texttt{dom}\, f = [0\,..\,n-1]$. An element selection function:

$$\bullet[\bullet] : \mathscr{A}(\mathscr{E}) \to \mathbb{N} \to \mathscr{E}$$

$$(n, f)[i] = \left\{ \begin{array}{rcl} f\ i & , & i < n \\ \bot & , & \mbox{otherwise} \end{array} \right.$$

We represent arrays by references. Thus, we introduce a (linearly) ordered set of locations

$$\mathscr{L} = \{l_0, l_1, \dots\}$$

Now, the set of all values the programs operate on can be described as follows:

$$\mathscr{V} = \mathbb{Z} \uplus \mathscr{L}$$

Here, every value is either an integer, or a reference (some location). The disjoint union "$\uplus$" makes it possible to unambiguously discriminate between the shapes of each value. To access arrays, we introduce an abstraction of memory:

$$\mathscr{M} = \mathscr{L} \to \mathscr{A}(\mathscr{V})$$

We now add two more components to the configurations: a memory function μ and the first free memory location l_m, and define the following primitive:

$$\mathbf{mem}\,\langle s, \mu, l_m, i, o, v\rangle = \mu$$

which gives a memory function from a configuration.

$$\frac{\begin{array}{cc} \Phi \vdash c \xrightarrow{e}_{\mathscr{E}} c' & \Phi \vdash c' \xrightarrow{j}_{\mathscr{E}} c'' \\ l = \mathbf{val}\ c' & j = \mathbf{val}\ c'' \\ l \in \mathscr{L} & j \in \mathbb{N} \\ (n, f) = \mathbf{mem}\ l & j < n \end{array}}{\Phi \vdash c \xrightarrow{e\,[j]}_{\mathscr{E}} \mathbf{ret}\ c''(f\ j)} \quad [\text{ArrayElement}]$$

$$\frac{\begin{array}{c} \Phi \vdash c_j \xrightarrow{e_j}_{\mathscr{E}} c_{j+1},\ j \in [0..k] \\ \langle s, \mu, l_m, i, o, \_\_ \rangle = c_{k+1} \end{array}}{\Phi \vdash c_0 \xrightarrow{[e_0, e_1, ..., e_k]}_{\mathscr{E}} \langle s, \mu[l_m \leftarrow (k+1, \lambda n.\mathbf{val}\ c_n)], l_{m+1}, i, o, l_m \rangle} \quad [\text{Array}]$$

$$\frac{\begin{array}{c} \Phi \vdash c \xrightarrow{e}_{\mathscr{E}} c' \\ l = \mathbf{val}\ c' \\ l \in \mathscr{L} \\ (n, f) = (\mathbf{mem}\ c')\ l \end{array}}{\Phi \vdash c \xrightarrow{e.\mathtt{length}}_{\mathscr{E}} \mathbf{return}\ c'\ n} \quad [\text{ArrayLength}]$$

Figure 1: Big-step Operational Semantics for Array Expressions

0.1 Adding arrays on expression level

On expression level, abstractly/concretely:

$$\begin{array}{rlll} \mathscr{E} += & \mathscr{E}[\mathscr{E}] & (a\,[e]) & \text{taking an element} \\ | & [\mathscr{E}^*] & ([e_1, e_2, .., e_k]) & \text{creating an array} \\ | & \mathscr{E}.\mathtt{length} & (\text{e}.\mathtt{length}) & \text{taking the length} \end{array}$$

The semantics of enriched expressions is modified as follows. First, we add two additional premises to the rule for binary operators:

$$\frac{\begin{array}{cc} \Phi \vdash c \xrightarrow{A}_{\mathscr{E}} c' & \Phi \vdash c' \xrightarrow{B}_{\mathscr{E}} c'' \\ \mathbf{val}\ c' \in \mathbb{Z} & \mathbf{val}\ c'' \in \mathbb{Z} \end{array}}{\Phi \vdash c \xrightarrow{A \otimes B}_{\mathscr{E}} \mathbf{ret}\ c''\ (\mathbf{val}\ c' \oplus \mathbf{val}\ c'')} \quad [\text{Binop}]$$

These two premises ensure that both operand expressions are evaluated into integer values. Second, we have to add the rules for new kinds of expressions (see Figure 1).

0.2 Adding arrays on statement level

On statement level, we add the single construct:

$$\mathscr{S} += \mathscr{E}[\mathscr{E}] := \mathscr{E}$$

This construct is interpreted as an assignment to an element of an array. The semantics of this construct is described by the following rule:

$$\dfrac{\begin{array}{ccc} \Phi \vdash c \xrightarrow{e}_{\mathscr{E}} c' & \Phi \vdash c' \xrightarrow{j}_{\mathscr{E}} c'' & \Phi \vdash c'' \xrightarrow{g}_{\mathscr{E}} \langle s, \mu, l_m, i, o, v \rangle \\ l = \mathbf{val}\, c' & i = \mathbf{val}\, c'' & \\ l \in \mathscr{L} & i \in \mathbb{N} & \\ & (n, f) = \mu\, l & \\ & i < n & \\ & \text{skip}, \Phi \vdash \langle s, \mu[l \leftarrow (n, f[i \leftarrow x])], l_m, i, o, -\rangle \xrightarrow{K} \widetilde{c} & \end{array}}{K, \Phi \vdash c \xrightarrow{e[j]:=g} \widetilde{c}} \quad [\text{ArrayAssign}]$$

0.3 Strings

With arrays in our hands, we can easily add strings as arrays of characters. In fact, on the source language the strings can be introduced as a syntactic extension:

1. we add a character constants 'c' as a shortcut for their integer codes;
2. we add a string literals "abcd ..." as a shortcut for arrays ['a', 'b', 'c', 'd', ...] .

Nothing else has to be done — now we have mutable reference-representable strings.