Types and Expressions

Stefan Muller, based on material by Jim Sasaki

CS 536: Science of Programming, Fall 2023
Lecture 4

1 A Simple Programming Language

- The next few classes will build up the simple programming language, "IMP", we'll be working with in the class.
- When studying programming languages, we use these simple *models* because they're easier to reason about than full programming languages. We include just a few features which capture the main ideas. Adding more features usually turns out to not be that interesting, and these features can often be built in terms of the features we already have as "syntactic sugar."
- As we saw in the first couple lectures, a language consists of *syntax* and *semantics*.
- A language also has several layers. IMP has three: types, expressions and statements. Today we'll see the syntax of types and expressions and the semantics of expressions.

The data types we'll have in our language are:

- Integers
- Booleans
- Arrays of integers and booleans (with integer indices).
- Other types like floats, chars, strings, etc., are pretty easy to add and don't add much interesting complexity. We may pretend we have these to make some examples more interesting.

Now we specify the syntax of expressions using a form called Backus-Naur form.

Expressions	e	::=	$\overline{n}$	Numbers
		|	true	True value
		|	false	False value
		|	x	Variables
		|	e iop e	Integer operations (e.g., +, -)
		|	$\mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e$	Conditional (like e ? e : e in C, Java)
		|	e bop e	Boolean operations (e.g., $\vee, \wedge$)
		|	$a[e]$	Array access (we'll use a, b, etc., for arrays)
		|	$size(a)$	Size of array

All of the things on the right let us build up expressions; in the definitions, e stands for another expression (which is also built using these rules, and so on).

This doesn't have:

- Assignment expressions
- Pointers
- Functions

- Arrays as values (e.g., we can't do something like (if x then a else b)$[\overline{0}]$, but we can do (if x then $a[\overline{0}]$ else $b[\overline{0}]$)

Other notes about arrays:

- Arrays are fixed-size and zero-indexed
- We assume out-of-bounds accesses are a runtime error.
- We'll assume we have multidimensional arrays; these aren't in the syntax above but they can pretty easily be defined in terms of single-dimensional arrays: we can define $a[i][j]$ as $a[i \cdot \mathsf{sizex}(a) + j]$ where $\mathsf{sizex}(a)$ is the x-dimension of a. We call this "syntactic sugar" because it's just more convenient (sweeter?) syntax for something that in theory already exists in the language.

Some syntactically well-formed expressions:

- if $x < \overline{0}$ then $x + y$ else $x * y + z$
- $a[$if $x < size(a)$ then x else $size(a) - \overline{1}]$
- if $x < \overline{0}$ then $\overline{0}$ else $\mathsf{sqrt}(x)$

Some non-well-formed expressions:

- $A[x] + A[b][c]$ (A is either a single-dimensional array or a two-dimensional array; it can't be both)
- (if b then min else max)(x, y) (functions must be applied)

2 Syntactic Values and Semantic Values

- Note that we have numbers, e.g. $\overline{2}$ and math symbols like + in the language. We also have them in math, which we're going to use to talk about programs.
- The former (things that appear in the program) we call *syntactic*. Things that we're using to talk about the program we call *semantic*. So, e.g., semantic + and semantic 2 are just the operators and numbers from math that you know and love.
- How do we tell the difference?
- For operators, it's usually clear from context.
- For numbers, this is why we put the bar over the numbers for *syntactic* (program) numbers, e.g., $\overline{2}$ instead of 2. We'll say $\overline{2}$ is syntactic and 2 is semantic.
- When writing on the board, I'll forget the lines. In your homeworks, you'll forget the lines. But let's at least try and keep these straight :)
- The syntactic values for Booleans are true and false, and the semantic values for Booleans are (still) T and F.

3 Values of Expressions (Semantics)

- What's the value of $x + y$?
- It depends what x and y are! Like with propositions, we need a state σ to tell us the values.
- We'll extend the notation to have $\sigma(e)$ mean the value of e in state σ.
- Note that we still need σ to be *proper*, i.e., assign a value to all (free) variables in e (expressions don't have a way for variables to not be free).
- $\sigma(e)$ is always a semantic value.

$$
\begin{array}{rcll}
\sigma(\overline{n}) & = & n & \\
\sigma(\mathsf{true}) & = & T & \\
\sigma(\mathsf{false}) & = & F & \\
\sigma(x) & = & \sigma(x) & \text{(seems obvious, but just so we cover all the cases...)} \\
\sigma(e_1 + e_2) & = & \sigma(e_1) + \sigma(e_2) & \text{(note the plus on the left is syntactic, the right is semantic)} \\
\sigma(e_1 \wedge e_2) & = & \sigma(e_1) \wedge \sigma(e_2) & \\
\sigma(\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3) & = & \begin{cases} \sigma(e_2) & \sigma(e_1) = T \\ \sigma(e_3) & \sigma(e_1) = F \end{cases} & \\
\sigma(a[e]) & = & \sigma(a[\sigma(e)]) & \\
\sigma(\mathit{size}(a)) & = & |\sigma(a)| &
\end{array}
$$

Example. Let $\sigma = \{x = 1, y = 2, x = 3\}$. (Note that states still contain *semantic* values.)

$$
\begin{array}{rcll}
\sigma(\mathsf{if}\ x < \overline{0}\ \mathsf{then}\ x + y\ \mathsf{else}\ x * y + z) & = & \sigma(x * y) + \sigma(z) & \text{(because } \sigma(x < \overline{0}) = \sigma(x) < \sigma(\overline{0}) = 1 < 0 = F\text{)} \\
& = & \sigma(x) * \sigma(y) + \sigma(z) & \\
& = & 1 * 2 + 3 & \\
& = & 5 &
\end{array}
$$

3.1 Representing Array Values in States

There are a few options for how to do this:

- A finite sequence (we'll use this one)
- A set of ordered pairs, i.e., a relation
- A function from integers to values

We'll write a state containing an array as, e.g., $\sigma = \{a = [8; 2; 5], x = 3, y = 0\}$.
Here, $\sigma(a) = [8; 2; 5]$ and $\sigma(a[0]) = 8$ and $|\sigma(a)| = 3$.

Examples

- Using the σ above,
$$\sigma(a[x * y]) = \sigma(a[\sigma(x * y)]) = \sigma(a[3 * 0]) = \sigma(a[0]) = 8$$
- If $\sigma = \{x = 1, a = [2; 0; 4]\}$,

$\sigma(a[x{+}\overline{1}]{-}\overline{2}) = \sigma(a[x{+}\overline{1}]){-}\sigma(\overline{2}) = \sigma(a[\sigma(x{+}\overline{1})]){-}2 = \sigma(a[\sigma(x){+}\sigma(\overline{1})]){-}2 = \sigma(a[1{+}1]){-}2 = \sigma(a[2]){-}2 = 4{-}2 = 2$