IMP: Modeling Imperative Languages

Stefan Muller, partially based on material by Jim Sasaki (for CS536)

CS 534: Types and Programming Languages, Spring 2024
Lectures 6-7

1 IMP Syntax

Today, we'll take our E language and add another layer of syntax, *statements*, to make more of a real programming language, IMP. Below, we'll use e for expressions.

$$
\begin{array}{rrcll}
\text{Expressions} & e & ::= & \ldots & \text{(everything from before)} \\
 & & | & x & \text{Variables} \\
\text{Types} & \tau & ::= & \mathsf{int} \mid \mathsf{string} & \\
\text{Statements} & s & ::= & x := e & \text{Assignment} \\
 & & | & \mathsf{if}\ e\ \mathsf{then}\ s\ \mathsf{else}\ s\ \mathsf{fi} & \text{Conditional} \\
 & & | & \mathsf{while}\ e\ \mathsf{do}\ s\ \mathsf{od} & \text{Loop} \\
 & & | & s; s & \text{Sequence} \\
 & & | & \mathsf{skip} & \text{Skip}
\end{array}
$$

A few notes on the syntax:

- We can sequence statements together using semicolons: e.g. $s_1; s_2$. We can continue this with multiple statements—while it rarely matters semantically, we'll say that sequencing is right-associative, i.e. $s_1; s_2; s_3 \equiv s_1; (s_2; s_3)$.
- This makes a sequence of statements look almost like in C/C++/Java, where a semicolon ends a statement. Technically though, we shouldn't have a semicolon at the end (e.g., $s_1; s_2;$ is not syntactically valid).
- skip is a no-op statement. Why is this useful? Maybe you don't want an "else" branch of a conditional:

$$\mathsf{if}\ x < 0\ \mathsf{then}\ x := x + \overline{1}\ \mathsf{else}\ \mathsf{skip}\ \mathsf{fi}$$

- $x := e$ evaluates e and assigns the value to variable x.
- If and while statements work like in most major programming languages.
- As with expressions, we've omitted a lot in the name of keeping the language simple, but there's a lot of room for syntactic sugar. For example, want for loops? We can represent `for (x = 0; x < n; x++) { s }` as:

$$x := \overline{0}; \mathsf{while}\ x < n\ \mathsf{do}\ s; x := x + \overline{1}\ \mathsf{od}$$

Examples.

- The following program computes the Factorial of n (assuming n is positive): at the end of the program, $n \geq 0 \rightarrow r = n!$.

```
$r := \overline{1}$;
while $(n \geq 1)$
do
    $r := r * n$;
    $n := n - 1$
od
```

- We don't have assignment expressions (like in C, where `y = x++` assigns `x + 1` to `x` and returns the old `x` which is assigned to `y`. However, we can express this as

$$y := x; x := x + \overline{1}$$

2 Small-step Operational Semantics of Programs

We have a small-step semantics for expressions (other than variables) already, so today we'll define one for statements. We'll actually define the semantics not just over statements, but over "configurations", which are a pair of a statement and a state, which we'll write as $\langle S, \sigma \rangle$. A state σ is a mapping from variables to their values. We'll write a state as a set of variable-value pairs, e.g., $\sigma = \{x = 1, y = \text{"Hello"}\}$. Like with sets, we'll write the empty state as $\emptyset$. We'll write $\sigma(x)$ to mean the value of x in state σ. For the σ above, $\sigma(x) = 1$. We'll also use the following syntax to *update* or *extend* a state with a new value (or an updated value for a variable that's already in the state:

$$\sigma[x \mapsto e]$$

Note this doesn't actually change σ in any way. If σ is the state above, then $(\sigma[x \mapsto 2])(x) = 2$, but $\sigma(x)$ is still 1.

We'll write

$$\langle s_1, \sigma_1 \rangle \mapsto \langle s_2, \sigma_2 \rangle$$

to mean that in one step, if we're in state σ_1 and we execute S_1, the program changes to S_2 (this is a bit confusing; it'll become clearer in a bit) and the state changes to σ_2.

As an example,

$$\langle x := \overline{1}; y := \overline{2}, \{x = 0, y = 0\} \rangle \mapsto \langle y := \overline{2}, \{x = 1, y = 0\} \rangle$$

(technically, this will actually be two steps with the rules we'll give, but this is just to give an idea).

We can keep going:

$$\langle x := \overline{1}; y := \overline{2}, \{x = 0, y = 0\} \rangle \mapsto \langle y := \overline{2}, \{x = 1, y = 0\} \rangle \mapsto \langle \mathsf{skip}, \{x = 1, y = 2\} \rangle$$

We also have a judgment corresponding to $e\ \mathsf{val}$ for configurations: $\langle s, \sigma \rangle\ \mathsf{final}$ means that the configuration has finished evaluating. Finally, we need to add the state σ to the semantics for expressions, since we'll need it to evaluate variables:

$$e \mapsto_\sigma e$$

We don't need configurations on both sides because evaluating expressions doesn't change the state. Nothing's stopping us from putting them, this is just less writing. Let's start off with the added rule for variables:

$$\frac{}{x \mapsto_\sigma \sigma(x)}\ \text{(S-9)}$$

Note this gives us a new way to have a "stuck state" (and a violation of Progress): an unbound variable. All of the other small step rules for expressions are the same as before, they just thread the state through. Now for the rules for statements.

$$\frac{}{\langle \mathsf{skip}, \sigma\rangle\ \mathsf{final}}\ \text{(F-1)} \qquad \frac{e \mapsto_\sigma e'}{\langle x := e, \sigma\rangle \mapsto \langle x := e', \sigma\rangle}\ \text{(IS-1)} \qquad \frac{e\ \mathsf{val}}{\langle x := e, \sigma\rangle \mapsto \langle \mathsf{skip}, \sigma[x \mapsto e]\rangle}\ \text{(IS-2)}$$

$$\frac{\langle s_1, \sigma\rangle \mapsto \langle s_1', \sigma'\rangle}{\langle s_1; s_2, \sigma\rangle \mapsto \langle s_1'; s_2, \sigma'\rangle}\ \text{(IS-3)} \qquad \frac{}{\langle \mathsf{skip}; s, \sigma\rangle \mapsto \langle s, \sigma\rangle}\ \text{(IS-4)}$$

$$\frac{e \mapsto_\sigma e'}{\langle \mathsf{if}\ e\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2\ \mathsf{fi}, \sigma\rangle \mapsto \langle \mathsf{if}\ e'\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2\ \mathsf{fi}, \sigma\rangle}\ \text{(IS-5)} \qquad \frac{n > 0}{\langle \mathsf{if}\ \overline{n}\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2\ \mathsf{fi}, \sigma\rangle \mapsto \langle s_1, \sigma\rangle}\ \text{(IS-6)}$$

$$\frac{n \le 0}{\langle \mathsf{if}\ \overline{n}\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2\ \mathsf{fi}, \sigma\rangle \mapsto \langle s_2, \sigma\rangle}\ \text{(IS-7)}$$

$$\frac{}{\langle \mathsf{while}\ e\ \mathsf{do}\ s\ \mathsf{od}, \sigma\rangle \mapsto \langle \mathsf{if}\ e\ \mathsf{then}\ s; \mathsf{while}\ e\ \mathsf{do}\ s\ \mathsf{od}\ \mathsf{else}\ \mathsf{skip}\ \mathsf{fi}, \sigma\rangle}\ \text{(IS-8)}$$

Example 1 Let's formally apply the rules to the example above.

$$\begin{array}{rl} & \langle x := \overline{1}; y := \overline{2}, \{x = 0, y = 0\}\rangle \\ \mapsto & \langle \mathsf{skip}; y := \overline{2}, \{x = 0, y = 0\}[x \mapsto 1]\rangle \\ \mapsto & \langle y := \overline{2}, \{x = 1, y = 0\}\rangle \\ \mapsto & \langle \mathsf{skip}, \{x = 1, y = 2\}\rangle \end{array}$$

Note that we write $\{x = 0, y = 0\}[x \mapsto 1]$ for clarity. This *doesn't* step to $\{x = 1, y = 0\}$, it is *equal to* $\{x = 1, y = 0\}$. We could just as easily have written it this way instead.

As we see, the first assignment actually steps to skip and then we use another rule (and another step) to remove that skip. This will get a little tiresome, so we can also write

$$\begin{array}{rl} & \langle x := \overline{1}; y := \overline{2}, \{x = 0, y = 0\}\rangle \\ \mapsto^2 & \langle y := \overline{2}, \{x = 1, y = 0\}\rangle \\ \mapsto & \langle \mathsf{skip}, \{x = 1, y = 2\}\rangle \end{array}$$

where $\mapsto^2$ means that this is actually 2 steps. (More generally, we'll use $\mapsto^n$ to mean n steps and $\mapsto^*$ to mean any number of steps, including 0.) We can use this to skip over "boring" steps. How do we know if a step is "boring"? There's no hard and fast rule, but if a step doesn't update the state, check a condition or do something else of interest like that, it's probably safe to skip. When in doubt, write out the steps.

Example 2 Below, let $W = \mathsf{while}\ x\ \mathsf{do}\ x := x - \overline{1}\ \mathsf{od}$.

$$\begin{array}{rl} & \langle W, \{x = 1\}\rangle \\ \mapsto & \langle \mathsf{if}\ x\ \mathsf{then}\ x := x - \overline{1}; W\ \mathsf{else}\ \mathsf{skip}\ \mathsf{fi}, \{x = 1\}\rangle \\ \mapsto & \langle \mathsf{if}\ \overline{1}\ \mathsf{then}\ x := x - \overline{1}; W\ \mathsf{else}\ \mathsf{skip}\ \mathsf{fi}, \{x = 1\}\rangle \\ \mapsto & \langle x := x - \overline{1}; W, \{x = 1\}\rangle \\ \mapsto & \langle x := \overline{1} - \overline{1}; W, \{x = 1\}\rangle \\ \mapsto & \langle x := \overline{0}; W, \{x = 1\}\rangle \\ \mapsto^2 & \langle W, \{x = 1\}[x \mapsto 0]\rangle \\ \mapsto & \langle \mathsf{if}\ x\ \mathsf{then}\ x := x - \overline{1}; W\ \mathsf{else}\ \mathsf{skip}\ \mathsf{fi}, \{x = 0\}\rangle \\ \mapsto & \langle \mathsf{if}\ \overline{0}\ \mathsf{then}\ x := x - \overline{1}; W\ \mathsf{else}\ \mathsf{skip}\ \mathsf{fi}, \{x = 0\}\rangle \\ \mapsto & \langle \mathsf{skip}, \{x = 0\}\rangle \end{array}$$

3 Typing rules for IMP

We'll now present a type system for IMP. The most interesting part is the type system for expressions, but that now needs to change from what we've seen before for an important reason: what's the type of the expression x? Well, it depends on what the type of the variable x is. We have no way of knowing this without additional information, so we need a *context* to give us this information. The typing judgment for expressions now has an extra context Γ, which maps variables to their *types* (in much the same way a state maps variables to their values). We'll write a context without the set notation though, like this:

$$x : \mathsf{int}, y : \mathsf{string}$$

We'll write the empty context as $\cdot$ (you'll also sometimes see $\emptyset$). We'll write $\Gamma, x : \tau$ to mean "the context Γ but extended with x having type τ." Generally, we'll only write this if x isn't already in Γ (more on what to do if it is later in the course...) The order of variables in a context doesn't matter, so we can write any context that has $x : \tau$ in it in this way. The new typing judgment is $\Gamma \vdash e : \tau$. We need the context for all of the rules, even though it's only useful for the variable rule:

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}\ \text{(T-0)} \qquad \frac{}{\Gamma \vdash \overline{n} : \mathsf{int}}\ \text{(T-1)} \qquad \frac{}{\Gamma \vdash \text{“}s\text{”} : \mathsf{string}}\ \text{(T-2)} \qquad \frac{\Gamma \vdash e_1 : \mathsf{int} \qquad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1 + e_2 : \mathsf{int}}\ \text{(T-3)}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{string} \qquad \Gamma \vdash e_2 : \mathsf{string}}{\Gamma \vdash e_1 \mathbin{\hat{}} e_2 : \mathsf{string}}\ \text{(T-4)} \qquad \frac{\Gamma \vdash e : \mathsf{string}}{\Gamma \vdash |e| : \mathsf{int}}\ \text{(T-5)}$$

Statements don't really have a type, but we still want to make sure they "make sense", e.g., if x has type int, we don't want to do $x :=$ "Hello". We'll use the judgment $\Gamma \vdash s\ \mathsf{ok}$ to say that s is well-typed under the context Γ.

$$\frac{}{\Gamma \vdash \mathsf{skip}\ \mathsf{ok}}\ \text{(IT-1)} \qquad \frac{\Gamma, x : \tau \vdash e : \tau}{\Gamma, x : \tau \vdash x := e\ \mathsf{ok}}\ \text{(IT-2)} \qquad \frac{\Gamma \vdash e : \mathsf{int} \qquad \Gamma \vdash s_1\ \mathsf{ok} \qquad \Gamma \vdash s_2\ \mathsf{ok}}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2\ \mathsf{fi}\ \mathsf{ok}}\ \text{(IT-3)}$$

$$\frac{\Gamma \vdash e : \mathsf{int} \qquad \Gamma \vdash s\ \mathsf{ok}}{\Gamma \vdash \mathsf{while}\ e\ \mathsf{do}\ s\ \mathsf{od}\ \mathsf{ok}}\ \text{(IT-4)} \qquad \frac{\Gamma \vdash s_1\ \mathsf{ok} \qquad \Gamma \vdash s_2\ \mathsf{ok}}{\Gamma \vdash s_1; s_2\ \mathsf{ok}}\ \text{(IT-5)}$$

We also need to know that a state is well-typed, i.e., that all the values have the right types. We'll say that $\Gamma \vdash \sigma$ if for all $x : \tau \in \Gamma$, we have $x = e \in \sigma$ for some e and $\cdot \vdash e : \tau$. Note that we don't need a context to type these values because they shouldn't have any variables in them (we say they are *closed*). So, for example,

$$x : \mathsf{int}, y : \mathsf{string} \vdash \{x = 5, y = \text{“Hello”}\}$$

but the following *are not* true:

$$x : \mathsf{int}, y : \mathsf{string} \vdash \{x = 5\}$$

$$x : \mathsf{int} \vdash \{x = 5, y = \text{“Hello”}\}$$

4 Type Safety

We have an extra typing rule and extra small-step rule for expressions, so that means we have another case each for Progress and Preservation for expressions (and, technically, every other case has to change slightly, but it just means writing σs and Γs in the right places). We also need an extra assumption on preservation and progress, and an additional lemma:

Lemma 1 (Weakening)**.** *If* $\Gamma \vdash e : \tau$ *and* $x \notin Dom(\Gamma)$*, then* $\Gamma, x : \tau' \vdash e : \tau$*.*

Proof. By induction on the derivation of $\Gamma \vdash e : \tau$.

- T-0. Then $e = y$ and $\Gamma = \Gamma', y : \tau$ and so $\Gamma, x : \tau' = \Gamma', x : \tau', y : \tau$ (because order doesn't matter). Apply T-0.
- T-1. Then $e = \overline{n}$ and $\tau = \mathsf{int}$. Apply T-1.
- T-2. Similar.
- T-3. Then $e = e_1 + e_2$ and $\tau = \mathsf{int}$ and $\Gamma \vdash e_1 : \mathsf{int}$ and $\Gamma \vdash e_2 : \mathsf{int}$. By induction, $\Gamma, x : \tau' \vdash e_1 : \mathsf{int}$ and $\Gamma, x : \tau' \vdash e_2 : \mathsf{int}$. Apply rule T-3.
- T-4, T-5. Similar.

□

Note, as a corollary, we can apply weakening as many times as we want to add more variables to the context as long as they're all different and none of them were there before. In particular, if $\cdot \vdash e : \tau$, then $\Gamma \vdash e : \tau$ for any Γ (that doesn't repeat variables).

The reverse (that e is well-typed under the empty context if it's well-typed under a non-empty context) is true for values (which in IMP are just integer and string literals).

Lemma 2. *If $\Gamma \vdash e : \tau$ and e val then $\cdot \vdash e : \tau$.*

Proof. By induction on the derivation of e val.

- V-1. Then $e = \overline{n}$. By inversion, $\tau = \mathsf{int}$. Apply T-1.
- V-2. Then $e =$ "s". By inversion, $\tau = \mathsf{string}$. Apply T-2.

□

Lemma 3 (Preservation)**.** *If $\Gamma \vdash e : \tau$ and $\Gamma \vdash \sigma$ and $e \mapsto_\sigma e'$ then $\Gamma \vdash e' : \tau$.*

- S-9. Then $e = x$ and $e' = \sigma(x)$. By inversion on T-0, $\Gamma(x) = \tau$. By the definition of $\Gamma \vdash \sigma$, we know $\cdot \vdash \sigma(x) : \tau$. By weakening, $\Gamma \vdash \sigma(x) : \tau$.

Lemma 4 (Progress)**.** *If $\Gamma \vdash e : \tau$ and $\Gamma \vdash \sigma$ then either e val or there exists e' such that $e \mapsto_\sigma e'$.*

- T-0. Then $e = x$ and $\Gamma(x) = \tau$. By the definition of $\Gamma \vdash \sigma$, we know that there exists some e' such that $\sigma(x) = e'$. By S-9, $e \mapsto_\sigma e'$.

We've set things up so that type safety now also implies there are no unbound variables at runtime (assuming we start with a well-typed state)!

We need preservation and progress results for statements too. They'll use the progress and preservation results for expressions from above.

Lemma 5 (Preservation for Statements)**.** *If $\Gamma \vdash s$ ok and $\Gamma \vdash \sigma$ and $\langle s, \sigma \rangle \mapsto \langle s', \sigma' \rangle$ then $\Gamma \vdash s'$ ok and $\Gamma \vdash \sigma'$*

Proof. By induction on the derivation of $\langle s, \sigma \rangle \mapsto \langle s', \sigma' \rangle$.

- IS-1. Then $s = x := e$ and $s' = x := e'$ and $e \mapsto_\sigma e'$ and $\sigma' = \sigma$. By inversion, $x : \tau \in \Gamma$ and $\Gamma \vdash e : \tau$. By Lemma 3, $\Gamma \vdash e' : \tau$. Apply IT-2. We have $\Gamma \vdash \sigma'$ because $\sigma' = \sigma$.
- IS-2. Then $s = x := e$ and e val and $s' = \mathsf{skip}$ and $\sigma' = \sigma[x \mapsto e]$. By inversion, $x : \tau \in \Gamma$ and $\Gamma \vdash e : \tau$. By Lemma 2, we have $\cdot \vdash e : \tau$. We therefore have $\Gamma \vdash \sigma'$. We have $\Gamma \vdash \mathsf{skip}$ ok by IT-1.
- IS-3. Then $s = s_1; s_2$ and $s = s_1'; s_2$ and $\langle s_1, \sigma \rangle \mapsto \langle s_1', \sigma' \rangle$. By inversion on IT-5, $\Gamma \vdash s_1$ ok and $\Gamma \vdash s_2$ ok. By induction, $\Gamma \vdash s_1'$ ok and $\Gamma \vdash \sigma'$. Apply rule IT-5.
- IS-4. Then $s = \mathsf{skip}; s'$ and $\sigma' = \sigma$, so $\Gamma \vdash \sigma'$. By inversion on IT-5, $\Gamma \vdash s'$ ok.

- IS-5. Then $s = \mathsf{if}\ e\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2\ \mathsf{fi}$ and $s' = \mathsf{if}\ e'\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2\ \mathsf{fi}$ and $e \mapsto_\sigma e'$ and $\sigma' = \sigma$. By inversion, $\Gamma \vdash e : \mathsf{int}$ and $\Gamma \vdash s_1\ \mathsf{ok}$ and $\Gamma \vdash s_2\ \mathsf{ok}$. By Lemma 3, $\Gamma \vdash e' : \mathsf{int}$. Apply IT-3.

- IS-6. Then $s = \mathsf{if}\ \overline{n}\ \mathsf{then}\ s'\ \mathsf{else}\ s_2\ \mathsf{fi}$ and $\sigma' = \sigma$. By inversion, $\Gamma \vdash s'\ \mathsf{ok}$.

- IS-7. Similar to above.

- IS-8. Then $s = \mathsf{while}\ e\ \mathsf{do}\ s\ \mathsf{od}$ and $\sigma' = \sigma$. and $s' = \mathsf{if}\ e\ \mathsf{then}\ (s; \mathsf{while}\ e\ \mathsf{do}\ s\ \mathsf{od})\ \mathsf{else\ skip\ fi}$. By inversion, $\Gamma \vdash e : \mathsf{int}$ and $\Gamma \vdash s\ \mathsf{ok}$. By assumption, $\Gamma \vdash \mathsf{while}\ e\ \mathsf{do}\ s\ \mathsf{od}\ \mathsf{ok}$. By IT-5, $\Gamma \vdash s; \mathsf{while}\ e\ \mathsf{do}\ s\ \mathsf{od}\ \mathsf{ok}$. By IT-1, $\Gamma \vdash \mathsf{skip\ ok}$. By IT-3, $\Gamma \vdash s'\ \mathsf{ok}$.

□

Lemma 6 (Progress for statements)**.** *If $\Gamma \vdash s$ ok and $\Gamma \vdash \sigma$, then either $\langle s, \sigma\rangle$ final or there exist s' and σ' such that $\langle s, \sigma\rangle \mapsto \langle s', \sigma'\rangle$.*

Proof. By induction on the derivation of $\Gamma \vdash s\ \mathsf{ok}$.

- IT-1. Then $s = \mathsf{skip}$. By F-1, $\langle s, \sigma\rangle\ \mathsf{final}$.
- IT-2. Then $s = x := e$ and $\Gamma = \Gamma', x : \tau$ and $\Gamma \vdash e : \tau$. By Lemma 4, either $e\ \mathsf{val}$ or $e \mapsto_\sigma e'$.
  - If $e\ \mathsf{val}$, then $\langle s, \sigma\rangle \mapsto \langle \mathsf{skip}, \sigma[x \mapsto e]\rangle$ by IS-2.
  - If $e \mapsto_\sigma e'$, then $\langle s, \sigma\rangle \mapsto \langle x := e', \sigma\rangle$ by IS-1.
- IT-3. Then $s = \mathsf{if}\ e\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2\ \mathsf{fi}$ and $\Gamma \vdash e : \mathsf{int}$ and $\Gamma \vdash s_1\ \mathsf{ok}$ and $\Gamma \vdash s_2\ \mathsf{ok}$. By Lemma 4, either $e\ \mathsf{val}$ or $e \mapsto_\sigma e'$.
  - If $e\ \mathsf{val}$, then $\langle s, \sigma\rangle$ steps by IS-6 or IS-7 depending on the value of n.
  - If $e \mapsto_\sigma e'$, then $\langle s, \sigma\rangle \mapsto \langle \mathsf{if}\ e'\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2\ \mathsf{fi}, \sigma\rangle$ by IS-5.
- IT-4. Then $s = \mathsf{while}\ e\ \mathsf{do}\ s\ \mathsf{od}$, and $\langle s, \sigma\rangle$ steps by IS-8.
- IT-5. Then $s = s_1; s_2$ and $\Gamma \vdash s_1\ \mathsf{ok}$ and $\Gamma \vdash s_2\ \mathsf{ok}$. By induction, either $\langle s_1, \sigma\rangle\ \mathsf{final}$ or $\langle s_1, \sigma\rangle \mapsto \langle s_1', \sigma\rangle$.
  - If $\langle s_1, \sigma\rangle\ \mathsf{final}$, then by inversion on F-1, $s_1 = \mathsf{skip}$. By IS-4, $\langle s, \sigma\rangle \mapsto \langle s_2, \sigma\rangle$.
  - If $\langle s_1, \sigma\rangle \mapsto \langle s_1', \sigma'\rangle$, then $\langle s, \sigma\rangle \mapsto \langle s_1'; s_2, \sigma'\rangle$ by IS-3.

□