# IIT CS534: Types and Programming Languages

E Syntax and Small-step Semantics

Stefan Muller

Tuesday, Sept. 2

## 1 Syntax

We will be working with a small language called E consisting of integer and string expressions. The grammar below is in BNF (Backus-Naur Form). We use $e ::= A \mid B \mid ...$ to mean that an expression (with metavariable $e$) can look like form $A$ or $B$, and so on.

| | | | |
|---|---|---|---|
| $e$ | ::= | $\overline{n}$ | Numbers |
| | $\mid$ | "$s$" | Strings |
| | $\mid$ | $e + e$ | Addition |
| | $\mid$ | $e \hat{} e$ | Concatenation |
| | $\mid$ | $\lvert e \rvert$ | String Length |

**Important note:** You might notice one thing missing from the above that's important for specifying expressions: parentheses. (Of course, it's not that important for this language because the operators commute, but in general it will be.) This is a difference between *abstract syntax*, which is how we generally specify programming languages when writing the theory and *concrete syntax*, which gets into these lower-level details such as operator precedence and parenthesization. Concrete syntax is very important if we're, for example, writing a parser, but generally the details are very complicated and obscure the points we're trying to make about the language. So, we'll write abstract syntax, use standard conventions for operator precedence, add parentheses as necessary (even if they're not actually in the syntax) and specify in words if it's unclear.

## 2 Small-step semantics

Next, we look at the small-step semantics of E. There are two judgments, $e$ val meaning that $e$ is a value and can't step anymore, and $e \mapsto e'$ meaning that $e$ steps to $e'$. The rules for these judgments are below. Note that in rules StepAdd and StepLen, when we do $n_1 + n_2$ or $|s|$ (as opposed to $\overline{n_1} + \overline{n_2}$ and $|"s"|$), these are actually taking the mathematical addition of two integers and the actual number of characters in a string literal. We just use the same symbols for the actual underlying operation and for the syntax of the programming language. Rules other than the first three are "search" rules that allow us to step subexpressions.

$$\frac{}{\overline{n}\ \mathsf{val}}\ (\textsc{ValNum}) \qquad \frac{}{\text{“}s\text{”}\ \mathsf{val}}\ (\textsc{ValString}) \qquad \frac{}{\overline{n_1} + \overline{n_2} \mapsto \overline{n_1 + n_2}}\ (\textsc{StepAdd})$$

$$\frac{}{\text{“}s_1\text{”} \mathbin{\hat{}} \text{“}s_2\text{”} \mapsto \text{“}s_1 s_2\text{”}}\ (\textsc{StepCat}) \qquad \frac{}{|\text{“}s\text{”}| \mapsto \overline{|s|}}\ (\textsc{StepLen}) \qquad \frac{e_1 \mapsto e_1'}{e_1 + e_2 \mapsto e_1' + e_2}\ (\textsc{StepSearchAddLeft})$$

$$\frac{e_2 \mapsto e_2'}{\overline{n_1} + e_2 \mapsto \overline{n_1} + e_2'}\ (\textsc{StepSearchAddRight}) \qquad \frac{e_1 \mapsto e_1'}{e_1 \mathbin{\hat{}} e_2 \mapsto e_1' \mathbin{\hat{}} e_2}\ (\textsc{StepSearchCatLeft})$$

$$\frac{e_2 \mapsto e_2'}{\text{“}s_1\text{”} \mathbin{\hat{}} e_2 \mapsto \text{“}s_1\text{”} \mathbin{\hat{}} e_2'}\ (\textsc{StepSearchCatRight}) \qquad \frac{e \mapsto e'}{|e| \mapsto |e'|}\ (\textsc{StepSearchLen})$$

This is a “left-to-right” semantics because we enforce that the left side of an addition or concatenation is stepped fully before we step the right. We could have done the opposite and made a “right-to-left” semantics (or not specified an ordering and left the semantics nondeterministic).

Note that this means that we build a whole derivation using these rules to show that the program takes one step. For example:

$$\dfrac{\dfrac{\dfrac{}{\overline{1} + \overline{2} \mapsto \overline{3}}\ (\textsc{StepAdd})}{(\overline{1} + \overline{2}) + \overline{3} \mapsto \overline{3} + \overline{3}}\ (\textsc{StepSearchAddLeft})}{\overline{0} + (\overline{1} + \overline{2}) + \overline{3} \mapsto \overline{0} + (\overline{3} + \overline{3})}\ (\textsc{StepSearchAddRight})$$

And that’s just one step! The derivation generally consists of a bunch of the “search rules” at the bottom to find the one part of the expression that’s able to step now, and then one of the “actual” step rules to take the step.

We don’t usually write this full derivation, but instead just the steps:

$$\begin{array}{ll} & \overline{0} + (\overline{1} + \overline{2}) + \overline{3} \\ \mapsto & \overline{0} + (\overline{3} + \overline{3}) \\ \mapsto & \overline{0} + \overline{6} \\ \mapsto & \overline{6} \end{array}$$

Another example:

$$\begin{array}{ll} & |\text{“}ab\text{”} \mathbin{\hat{}} \text{“}c\text{”}| \\ \mapsto & |\text{“}abc\text{”}| \\ \mapsto & \overline{3} \end{array}$$

Again, each of these steps is its own derivation but they’re pretty straightforward and mechanical to build. In fact, we often don’t even write all of the steps since a lot of them are kind of uninteresting. The judgment $e \mapsto^n e'$ means “$e$ steps to $e'$ in $n$ steps.”

$$\frac{}{e \mapsto^0 e}\ (\textsc{StepZero}) \qquad \frac{e \mapsto e' \qquad e' \mapsto^n e''}{e \mapsto^{n+1} e''}\ (\textsc{StepOne})$$

There’s also a judgment $e \mapsto^* e'$ if we don’t even care about the number of steps.

$$\frac{}{e \mapsto^* e}\ (\textsc{StepStarZero}) \qquad \frac{e \mapsto e' \qquad e' \mapsto^* e''}{e \mapsto^* e''}\ (\textsc{StepStarOne})$$

The following theorem is obvious but technically necessary since these are different judgments. It’s also a good warmup example of the types of things we’ll prove about these judgments by rule induction.

**Theorem 1.** *$e \mapsto^* e'$ if and only if $e \mapsto^n e'$ for some $n \geq 0$.*

*Proof.* Forward direction ($e \mapsto^* e' \Rightarrow e \mapsto^n e'$):
By induction on the derivation of $e \mapsto^* e'$.

- Case StepStarZero. Then $e = e'$. By StepZero, $e \mapsto^0 e$.
- Case StepStarOne. Then $e \mapsto e''$ and $e'' \mapsto^* e'$. By induction, $e'' \mapsto^n e'$ for some $n \geq 0$. By StepOne, $e \mapsto^{n+1} e'$. We also have $n + 1 \geq 0$.

Reverse direction ($e \mapsto^n e' \Rightarrow e \mapsto^* e'$):
By induction on the derivation of $e \mapsto^n e'$.

- Case StepZero. Then $n = 0$ and $e = e'$. By StepStarZero, $e \mapsto^* e$.
- Case StepStarOne. Then $n = m + 1$ and $e \mapsto e''$ and $e'' \mapsto^m e'$. By induction, $e'' \mapsto^* e'$. By StepStarOne, $e \mapsto^* e'$.

□