CS 430/530
Formal Semantics

Zhong Shao

Yale University
Department of Computer Science

Statics and Dynamics

March 4, 2025

PL Semantics: Static & Dynamic Phases

The static phase:

- Parsing: turn concrete syntax to AST or ABT
- Type-checking to ensure the program is well-formed;
  - based on a set of typing rules (known as static semantics or statics)

The dynamic phase: execution of well-formed programs;

- based on a set of evaluation rules (dynamic / operational semantics or dynamics)

A language is **safe** when well-formed programs are well-behaved when executed

A Simple Expression Language E

Syntax of E defined as Abstract Binding Trees:

Typ	τ	$::=$	$\mathtt{num}$	$\mathtt{num}$	numbers
			$\mathtt{str}$	$\mathtt{str}$	strings
Exp	e	$::=$	x	x	variable
			$\mathtt{num}[n]$	n	numeral
			$\mathtt{str}[s]$	$\texttt{"}s\texttt{"}$	literal
			$\mathtt{plus}(e_1;e_2)$	$e_1 + e_2$	addition
			$\mathtt{times}(e_1;e_2)$	$e_1 * e_2$	multiplication
			$\mathtt{cat}(e_1;e_2)$	$e_1 \mathbin{\hat{}} e_2$	concatenation
			$\mathtt{len}(e)$	$\lvert e \rvert$	length
			$\mathtt{let}(e_1;x.e_2)$	$\mathtt{let}\, x\, \mathtt{be}\, e_1\, \mathtt{in}\, e_2$	definition

Statics (Type System) for E

$\vec{x} \mid \Gamma \vdash e : \tau,$

An inductive definition of generic hypothetical judgments

$$\overline{\Gamma, x : \tau \vdash x : \tau} \tag{4.1a}$$

$$\overline{\Gamma \vdash \mathtt{str}[s] : \mathtt{str}} \tag{4.1b}$$

$$\overline{\Gamma \vdash \mathtt{num}[n] : \mathtt{num}} \tag{4.1c}$$

$$\frac{\Gamma \vdash e_1 : \mathtt{num} \quad \Gamma \vdash e_2 : \mathtt{num}}{\Gamma \vdash \mathtt{plus}(e_1; e_2) : \mathtt{num}} \tag{4.1d}$$

$$\frac{\Gamma \vdash e_1 : \mathtt{num} \quad \Gamma \vdash e_2 : \mathtt{num}}{\Gamma \vdash \mathtt{times}(e_1; e_2) : \mathtt{num}} \tag{4.1e}$$

$$\frac{\Gamma \vdash e_1 : \mathtt{str} \quad \Gamma \vdash e_2 : \mathtt{str}}{\Gamma \vdash \mathtt{cat}(e_1; e_2) : \mathtt{str}} \tag{4.1f}$$

$$\frac{\Gamma \vdash e : \mathtt{str}}{\Gamma \vdash \mathtt{len}(e) : \mathtt{num}} \tag{4.1g}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \mathtt{let}(e_1; x.e_2) : \tau_2} \tag{4.1h}$$

Properties for Type System

Lemma 4.1 (Unicity of Typing)**.** *For every typing context* Γ *and expression* e*, there exists at most one* τ *such that* $\Gamma \vdash e : \tau$.

Proof By rule induction on rules (4.1), making use of the fact that variables have at most one type in any typing context. $\square$

Lemma 4.2 (Inversion for Typing)**.** *Suppose that* $\Gamma \vdash e : \tau$. *If* $e = \texttt{plus}(e_1; e_2)$*, then* $\tau = \texttt{num}$, $\Gamma \vdash e_1 : \texttt{num}$*, and* $\Gamma \vdash e_2 : \texttt{num}$*, and similarly for the other constructs of the language.*

Proof These may all be proved by induction on the derivation of the typing judgment $\Gamma \vdash e : \tau$. $\square$

Properties for Type System

Lemma 4.3 (Weakening)**.** *If* $\Gamma \vdash e' : \tau'$*, then* $\Gamma, x : \tau \vdash e' : \tau'$ *for any* $x \notin dom(\Gamma)$ *and any type* τ.

Proof By induction on the derivation of $\Gamma \vdash e' : \tau'$. We will give one case here, for rule (4.1h). We have that $e' = \mathtt{let}(e_1; z.e_2)$, where by the conventions on variables we may assume z is chosen such that $z \notin dom(\Gamma)$ and $z \neq x$. By induction, we have

1. $\Gamma, x : \tau \vdash e_1 : \tau_1$,
2. $\Gamma, x : \tau, z : \tau_1 \vdash e_2 : \tau'$,

from which the result follows by rule (4.1h). $\square$

Properties for Type System

Lemma 4.4 (Substitution). *If* $\Gamma, x : \tau \vdash e' : \tau'$ *and* $\Gamma \vdash e : \tau$, *then* $\Gamma \vdash [e/x]e' : \tau'$.

Proof By induction on the derivation of $\Gamma, x : \tau \vdash e' : \tau'$. We again consider only rule (4.1h). As in the preceding case, $e' = \mathtt{let}(e_1; z.e_2)$, where z is chosen so that $z \neq x$ and $z \notin dom(\Gamma)$. We have by induction and Lemma 4.3 that

1. $\Gamma \vdash [e/x]e_1 : \tau_1$,
2. $\Gamma, z : \tau_1 \vdash [e/x]e_2 : \tau'$.

By the choice of z, we have

$$[e/x]\mathtt{let}(e_1; z.e_2) = \mathtt{let}([e/x]e_1; z.[e/x]e_2).$$

It follows by rule (4.1h) that $\Gamma \vdash [e/x]\mathtt{let}(e_1; z.e_2) : \tau'$, as desired. □

Properties for Type System

Lemma 4.5 (Decomposition). *If* $\Gamma \vdash [e/x]e' : \tau'$, *then for every type* τ *such that* $\Gamma \vdash e : \tau$, *we have* $\Gamma, x : \tau \vdash e' : \tau'$.

Proof The typing of $[e/x]e'$ depends only on the type of e wherever it occurs, if at all. $\square$

Dynamics (aka Operational Semantics)

How are programs executed?

- Structural dynamics (transition semantics)
  - Step-by-step transition system (or small-step semantics)
- Contextual dynamics
  - Structural dynamics defined under a changing evaluation context
- Equational dynamics
  - A set of rules for definitional equality
- Evaluation dynamics (big-step semantics)

Transition Systems

A *transition system* is specified by the following four forms of judgment:

1. s state, asserting that s is a *state* of the transition system.
2. s final, where s state, asserting that s is a *final* state.
3. s $\mathsf{initial}$, where s state, asserting that s is an *initial* state.
4. $s \longmapsto s'$, where s state and s' state, asserting that state s may transition to state s'.

The *iteration* of transition judgment $s \longmapsto^* s'$ is inductively defined by the following rules:

$$\frac{}{s \longmapsto^* s} \tag{5.1a}$$

$$\frac{s \longmapsto s' \quad s' \longmapsto^* s''}{s \longmapsto^* s''} \tag{5.1b}$$

Structural Dynamics for E

A *structural dynamics* for the language **E** is given by a transition system whose states are closed expressions. All states are initial. The final states are the *(closed) values*, which represent the completed computations. The judgment e val, which states that e is a value, is inductively defined by the following rules:

$$\overline{\texttt{num}[n]\ \mathsf{val}} \tag{5.3a}$$

$$\overline{\texttt{str}[s]\ \mathsf{val}} \tag{5.3b}$$

The transition judgment $e \longmapsto e'$ between states is inductively defined by the following rules:

$$\frac{n_1 + n_2 = n}{\texttt{plus}(\texttt{num}[n_1];\texttt{num}[n_2]) \longmapsto \texttt{num}[n]} \tag{5.4a}$$

$$\frac{e_1 \longmapsto e_1'}{\texttt{plus}(e_1;e_2) \longmapsto \texttt{plus}(e_1';e_2)} \tag{5.4b}$$

$$\frac{e_1\ \mathsf{val} \quad e_2 \longmapsto e_2'}{\texttt{plus}(e_1;e_2) \longmapsto \texttt{plus}(e_1;e_2')} \tag{5.4c}$$

Structural Dynamics for E

$$\frac{s_1 \mathbin{\hat{}} s_2 = s \ \mathsf{str}}{\mathtt{cat}(\mathtt{str}[s_1];\mathtt{str}[s_2]) \longmapsto \mathtt{str}[s]} \tag{5.4d}$$

$$\frac{e_1 \longmapsto e_1'}{\mathtt{cat}(e_1;e_2) \longmapsto \mathtt{cat}(e_1';e_2)} \tag{5.4e}$$

$$\frac{e_1 \ \mathsf{val} \quad e_2 \longmapsto e_2'}{\mathtt{cat}(e_1;e_2) \longmapsto \mathtt{cat}(e_1;e_2')} \tag{5.4f}$$

$$\left[\frac{e_1 \longmapsto e_1'}{\mathtt{let}(e_1;x.e_2) \longmapsto \mathtt{let}(e_1';x.e_2)}\right] \tag{5.4g}$$

$$\frac{[e_1 \ \mathsf{val}]}{\mathtt{let}(e_1;x.e_2) \longmapsto [e_1/x]e_2} \tag{5.4h}$$

Structural Dynamics for E

A derivation sequence in a structural dynamics has a two-dimensional structure, with the number of steps in the sequence being its "width" and the derivation tree for each step being its "height." For example, consider the following evaluation sequence:

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathtt{let}(\mathtt{plus}(\mathtt{num}[1];\mathtt{num}[2]);x.\mathtt{plus}(\mathtt{plus}(x;\mathtt{num}[3]);\mathtt{num}[4]))} \\
\longmapsto & \mathtt{let}(\mathtt{num}[3];x.\mathtt{plus}(\mathtt{plus}(x;\mathtt{num}[3]);\mathtt{num}[4])) \\
\longmapsto & \mathtt{plus}(\mathtt{plus}(\mathtt{num}[3];\mathtt{num}[3]);\mathtt{num}[4]) \\
\longmapsto & \mathtt{plus}(\mathtt{num}[6];\mathtt{num}[4]) \\
\longmapsto & \mathtt{num}[10]
\end{array}
$$

Each step in this sequence of transitions is justified by a derivation according to rules (5.4). For example, the third transition in the preceding example is justified by the following derivation:

$$
\dfrac{\dfrac{}{\mathtt{plus}(\mathtt{num}[3];\mathtt{num}[3]) \longmapsto \mathtt{num}[6]}\ (5.4\mathrm{a})}{\mathtt{plus}(\mathtt{plus}(\mathtt{num}[3];\mathtt{num}[3]);\mathtt{num}[4]) \longmapsto \mathtt{plus}(\mathtt{num}[6];\mathtt{num}[4])}\ (5.4\mathrm{b})
$$

Structural Dynamics for E

Lemma 5.2 (Finality of Values)**.** *For no expression e do we have both e val, and $e \longmapsto e'$ for some e'.*

Proof By rule induction on rules (5.3) and (5.4). $\square$

Lemma 5.3 (Determinacy)**.** *If $e \longmapsto e'$ and $e \longmapsto e''$, then e' and e'' are α-equivalent.*

Proof By rule induction on the premises $e \longmapsto e'$ and $e \longmapsto e''$, carried out either simultaneously or in either order. The primitive operators, such as addition, are assumed to have a unique value when applied to values. $\square$

Contextual Dynamics for E

The instruction transition judgment $e_1 \to e_2$ for **E** is defined by the following rules, together with similar rules for multiplication of numbers and the length of a string.

$$\frac{m + n \text{ is } p \text{ nat}}{\texttt{plus}(\texttt{num}[m];\texttt{num}[n]) \to \texttt{num}[p]} \tag{5.5a}$$

$$\frac{s \mathbin{\hat{}} t = u \text{ str}}{\texttt{cat}(\texttt{str}[s];\texttt{str}[t]) \to \texttt{str}[u]} \tag{5.5b}$$

$$\frac{}{\texttt{let}(e_1;x.e_2) \to [e_1/x]e_2} \tag{5.5c}$$

Contextual Dynamics for E

The judgment $\mathcal{E}$ ectxt determines the location of the next instruction to execute in a larger expression. The position of the next instruction step is specified by a "hole," written $\circ$, into which the next instruction is placed, as we shall detail shortly. (The rules for multiplication and length are omitted for concision, as they are handled similarly.)

$$\frac{}{\circ \ \mathsf{ectxt}} \tag{5.6a}$$

$$\frac{\mathcal{E}_1 \ \mathsf{ectxt}}{\mathtt{plus}(\mathcal{E}_1; e_2) \ \mathsf{ectxt}} \tag{5.6b}$$

$$\frac{e_1 \ \mathsf{val} \quad \mathcal{E}_2 \ \mathsf{ectxt}}{\mathtt{plus}(e_1; \mathcal{E}_2) \ \mathsf{ectxt}} \tag{5.6c}$$

Contextual Dynamics for E

An evaluation context is a template that is instantiated by replacing the hole with an instruction to be executed. The judgment $e' = \mathcal{E}\{e\}$ states that the expression e' is the result of filling the hole in the evaluation context $\mathcal{E}$ with the expression e. It is inductively defined by the following rules:

$$\frac{}{e = \circ\{e\}} \tag{5.7a}$$

$$\frac{e_1 = \mathcal{E}_1\{e\}}{\texttt{plus}(e_1; e_2) = \texttt{plus}(\mathcal{E}_1; e_2)\{e\}} \tag{5.7b}$$

$$\frac{e_1\ \mathsf{val} \quad e_2 = \mathcal{E}_2\{e\}}{\texttt{plus}(e_1; e_2) = \texttt{plus}(e_1; \mathcal{E}_2)\{e\}} \tag{5.7c}$$

There is one rule for each form of evaluation context. Filling the hole with e results in e; otherwise, we proceed inductively over the structure of the evaluation context.

Finally, the contextual dynamics for **E** is defined by a single rule:

$$\frac{e = \mathcal{E}\{e_0\} \quad e_0 \to e_0' \quad e' = \mathcal{E}\{e_0'\}}{e \longmapsto e'} \tag{5.8}$$

Relating Structural & Contextual Dynamics

Theorem 5.4. *$e \longmapsto_{\mathsf{s}} e'$ if, and only if, $e \longmapsto_{\mathsf{c}} e'$.*

Proof From left to right, proceed by rule induction on rules (5.4). It is enough in each case to exhibit an evaluation context $\mathcal{E}$ such that $e = \mathcal{E}\{e_0\}$, $e' = \mathcal{E}\{e_0'\}$, and $e_0 \to e_0'$. For example, for rule (5.4a), take $\mathcal{E} = \circ$, and note that $e \to e'$. For rule (5.4b), we have by induction that there exists an evaluation context $\mathcal{E}_1$ such that $e_1 = \mathcal{E}_1\{e_0\}$, $e_1' = \mathcal{E}_1\{e_0'\}$, and $e_0 \to e_0'$. Take $\mathcal{E} = \mathtt{plus}(\mathcal{E}_1; e_2)$, and note that $e = \mathtt{plus}(\mathcal{E}_1; e_2)\{e_0\}$ and $e' = \mathtt{plus}(\mathcal{E}_1; e_2)\{e_0'\}$ with $e_0 \to e_0'$.

From right to left, note that if $e \longmapsto_{\mathsf{c}} e'$, then there exists an evaluation context $\mathcal{E}$ such that $e = \mathcal{E}\{e_0\}$, $e' = \mathcal{E}\{e_0'\}$, and $e_0 \to e_0'$. We prove by induction on rules (5.7) that $e \longmapsto_{\mathsf{s}} e'$. For example, for rule (5.7a), e_0 is e, e_0' is e', and $e \to e'$. Hence, $e \longmapsto_{\mathsf{s}} e'$. For rule (5.7b), we have that $\mathcal{E} = \mathtt{plus}(\mathcal{E}_1; e_2)$, $e_1 = \mathcal{E}_1\{e_0\}$, $e_1' = \mathcal{E}_1\{e_0'\}$, and $e_1 \longmapsto_{\mathsf{s}} e_1'$. Therefore, e is $\mathtt{plus}(e_1; e_2)$, e' is $\mathtt{plus}(e_1'; e_2)$, and therefore by rule (5.4b), $e \longmapsto_{\mathsf{s}} e'$. $\square$

Equational Dynamics

$$\frac{}{\Gamma \vdash e \equiv e : \tau} \quad (5.10a)$$

$$\frac{\Gamma \vdash e' \equiv e : \tau}{\Gamma \vdash e \equiv e' : \tau} \quad (5.10b)$$

$$\frac{\Gamma \vdash e \equiv e' : \tau \quad \Gamma \vdash e' \equiv e'' : \tau}{\Gamma \vdash e \equiv e'' : \tau} \quad (5.10c)$$

$$\frac{\Gamma \vdash e_1 \equiv e_1' : \mathtt{num} \quad \Gamma \vdash e_2 \equiv e_2' : \mathtt{num}}{\Gamma \vdash \mathtt{plus}(e_1; e_2) \equiv \mathtt{plus}(e_1'; e_2') : \mathtt{num}} \quad (5.10d)$$

$$\frac{\Gamma \vdash e_1 \equiv e_1' : \mathtt{str} \quad \Gamma \vdash e_2 \equiv e_2' : \mathtt{str}}{\Gamma \vdash \mathtt{cat}(e_1; e_2) \equiv \mathtt{cat}(e_1'; e_2') : \mathtt{str}} \quad (5.10e)$$

$$\frac{\Gamma \vdash e_1 \equiv e_1' : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 \equiv e_2' : \tau_2}{\Gamma \vdash \mathtt{let}(e_1; x.e_2) \equiv \mathtt{let}(e_1'; x.e_2') : \tau_2} \quad (5.10f)$$

$$\frac{n_1 + n_2 \text{ is } n \text{ nat}}{\Gamma \vdash \mathtt{plus}(\mathtt{num}[n_1]; \mathtt{num}[n_2]) \equiv \mathtt{num}[n] : \mathtt{num}} \quad (5.10g)$$

$$\frac{s_1 \hat{\ } s_2 = s \text{ str}}{\Gamma \vdash \mathtt{cat}(\mathtt{str}[s_1]; \mathtt{str}[s_2]) \equiv \mathtt{str}[s] : \mathtt{str}} \quad (5.10h)$$

$$\frac{}{\Gamma \vdash \mathtt{let}(e_1; x.e_2) \equiv [e_1/x]e_2 : \tau} \quad (5.10i)$$

Equational Dynamics

Theorem 5.5. *For the expression language* E*, the relation* $e \equiv e' : \tau$ *holds iff there exists* e_0 val *such that* $e \longmapsto^* e_0$ *and* $e' \longmapsto^* e_0$.

Proof The proof from right to left is direct, because every transition step is a valid equation. The converse follows from the following, more general, proposition, which is proved by induction on rules (5.10): if $x_1 : \tau_1, \ldots, x_n : \tau_n \vdash e \equiv e' : \tau$, then when $e_1 : \tau_1, e_1' : \tau_1, \ldots, e_n : \tau_n, e_n' : \tau_n$, if for each $1 \leq i \leq n$ the expressions e_i and e_i' evaluate to a common value v_i, then there exists e_0 val such that

$$[e_1, \ldots, e_n/x_1, \ldots, x_n]e \longmapsto^* e_0$$

and

$$[e_1', \ldots, e_n'/x_1, \ldots, x_n]e' \longmapsto^* e_0.$$

□

Type Safety for E

Theorem 6.1 (Type Safety)**.**

1. *If* $e : \tau$ *and* $e \longmapsto e'$*, then* $e' : \tau$.
2. *If* $e : \tau$*, then either* e val*, or there exists* e' *such that* $e \longmapsto e'$.

The first part, called *preservation*, says that the steps of evaluation preserve typing; the second, called *progress*, ensures that well-typed expressions are either values or can be further evaluated. Safety is the conjunction of preservation and progress.

We say that an expression e is *stuck* iff it is not a value, yet there is no e' such that $e \longmapsto e'$. It follows from the safety theorem that a stuck state is necessarily ill-typed. Or, putting it the other way around, that well-typed states do not get stuck.

Preservation for E

Theorem 6.2 (Preservation). *If $e : \tau$ and $e \longmapsto e'$, then $e' : \tau$.*

Proof We will give the proof in two cases, leaving the rest to the reader. Consider rule (5.4b),

$$\frac{e_1 \longmapsto e_1'}{\texttt{plus}(e_1; e_2) \longmapsto \texttt{plus}(e_1'; e_2)} .$$

Assume that $\texttt{plus}(e_1; e_2) : \tau$. By inversion for typing, we have that $\tau = \texttt{num}$, $e_1 : \texttt{num}$, and $e_2 : \texttt{num}$. By induction, we have that $e_1' : \texttt{num}$, and hence $\texttt{plus}(e_1'; e_2) : \texttt{num}$. The case for concatenation is handled similarly.

Now consider rule (5.4h),

$$\frac{}{\texttt{let}(e_1; x.e_2) \longmapsto [e_1/x]e_2} .$$

Assume that $\texttt{let}(e_1; x.e_2) : \tau_2$. By the inversion Lemma 4.2, $e_1 : \tau_1$ for some τ_1 such that $x : \tau_1 \vdash e_2 : \tau_2$. By the substitution Lemma 4.4 $[e_1/x]e_2 : \tau_2$, as desired.

It is easy to check that the primitive operations are all type-preserving; for example, if a nat and b nat and $a + b$ is c nat, then c nat. $\square$

Progress for E

Lemma 6.3 (Canonical Forms)**.** *If e val and $e : \tau$, then*

1. *If $\tau = \mathtt{num}$, then $e = \mathtt{num}[n]$ for some number n.*
2. *If $\tau = \mathtt{str}$, then $e = \mathtt{str}[s]$ for some string s.*

Proof By induction on rules (4.1) and (5.3). $\square$

Progress is proved by rule induction on rules (4.1) defining the statics of the language.

Theorem 6.4 (Progress)**.** *If $e : \tau$, then either e val, or there exists e' such that $e \longmapsto e'$.*

Progress for E

Theorem 6.4 (Progress)**.** *If $e : \tau$, then either e val, or there exists e' such that $e \longmapsto e'$.*

Proof The proof proceeds by induction on the typing derivation. We will consider only one case, for rule (4.1d),

$$\frac{e_1 : \mathtt{num} \quad e_2 : \mathtt{num}}{\mathtt{plus}(e_1; e_2) : \mathtt{num}} ,$$

where the context is empty because we are considering only closed terms.

By induction, we have that either e_1 val, or there exists e_1' such that $e_1 \longmapsto e_1'$. In the latter case, it follows that $\mathtt{plus}(e_1; e_2) \longmapsto \mathtt{plus}(e_1'; e_2)$, as required. In the former, we also have by induction that either e_2 val, or there exists e_2' such that $e_2 \longmapsto e_2'$. In the latter case, we have that $\mathtt{plus}(e_1; e_2) \longmapsto \mathtt{plus}(e_1; e_2')$, as required. In the former, we have, by the Canonical Forms Lemma 6.3, $e_1 = \mathtt{num}[n_1]$ and $e_2 = \mathtt{num}[n_2]$, and hence

$$\mathtt{plus}(\mathtt{num}[n_1]; \mathtt{num}[n_2]) \longmapsto \mathtt{num}[n_1 + n_2].$$

$\square$

E + Runtime Errors

Suppose that we wish to extend **E** with, say, a quotient operation that is undefined for a zero divisor. The natural typing rule for quotients is given by the following rule:

$$\frac{e_1 : \mathtt{num} \quad e_2 : \mathtt{num}}{\mathtt{div}(e_1; e_2) : \mathtt{num}} \, .$$

But the expression $\mathtt{div}(\mathtt{num}[3]; \mathtt{num}[0])$ is well-typed, yet stuck! We have two options to correct this situation:

1. Enhance the type system, so that no well-typed program may divide by zero.
2. Add dynamic checks, so that division by zero signals an error as the outcome of evaluation.

E + Runtime Errors

One approach to modeling checked errors is to give an inductive definition of the judgment e err stating that the expression e incurs a checked run-time error, such as division by zero. Here are some representative rules that would be present in a full inductive definition of this judgment:

$$\frac{e_1 \ \mathsf{val}}{\mathtt{div}(e_1; \mathtt{num}[0]) \ \mathsf{err}} \tag{6.1a}$$

$$\frac{e_1 \ \mathsf{err}}{\mathtt{div}(e_1; e_2) \ \mathsf{err}} \tag{6.1b}$$

$$\frac{e_1 \ \mathsf{val} \quad e_2 \ \mathsf{err}}{\mathtt{div}(e_1; e_2) \ \mathsf{err}} \tag{6.1c}$$

E + Runtime Errors

Once the error judgment is available, we may also consider an expression, `error`, which forcibly induces an error, with the following static and dynamic semantics:

$$\frac{}{\Gamma \vdash \mathtt{error} : \tau} \tag{6.2a}$$

$$\frac{}{\mathtt{error}\ \mathsf{err}} \tag{6.2b}$$

The preservation theorem is not affected by checked errors. However, the statement (and proof) of progress is modified to account for checked errors.

Theorem 6.5 (Progress With Error)**.** *If $e : \tau$, then either e err, or e val, or there exists e' such that $e \longmapsto e'$.*

Evaluation Dynamics for E (big-step operational semantics)

An *evaluation dynamics* consists of an inductive definition of the evaluation judgment $e \Downarrow v$ stating that the closed expression e evaluates to the value v. The evaluation dynamics of **E** is defined by the following rules:

$$\frac{}{\texttt{num}[n] \Downarrow \texttt{num}[n]} \tag{7.1a}$$

$$\frac{}{\texttt{str}[s] \Downarrow \texttt{str}[s]} \tag{7.1b}$$

$$\frac{e_1 \Downarrow \texttt{num}[n_1] \quad e_2 \Downarrow \texttt{num}[n_2] \quad n_1 + n_2 \text{ is } n \text{ nat}}{\texttt{plus}(e_1; e_2) \Downarrow \texttt{num}[n]} \tag{7.1c}$$

$$\frac{e_1 \Downarrow \texttt{str}[s_1] \quad e_2 \Downarrow \texttt{str}[s_2] \quad s_1 \,\hat{}\, s_2 = s \text{ str}}{\texttt{cat}(e_1; e_2) \Downarrow \texttt{str}[s]} \tag{7.1d}$$

$$\frac{e \Downarrow \texttt{str}[s] \quad |s| = n \text{ nat}}{\texttt{len}(e) \Downarrow \texttt{num}[n]} \tag{7.1e}$$

eval-by-name

$$\frac{[e_1/x]e_2 \Downarrow v_2}{\texttt{let}(e_1; x.e_2) \Downarrow v_2} \tag{7.1f}$$

Evaluation Dynamics for E

An *evaluation dynamics* consists of an inductive definition of the evaluation judgment $e \Downarrow v$ stating that the closed expression e evaluates to the value v. The evaluation dynamics of **E** is defined by the following rules:

$$\frac{}{\texttt{num}[n] \Downarrow \texttt{num}[n]} \tag{7.1a}$$

$$\frac{}{\texttt{str}[s] \Downarrow \texttt{str}[s]} \tag{7.1b}$$

$$\frac{e_1 \Downarrow \texttt{num}[n_1] \quad e_2 \Downarrow \texttt{num}[n_2] \quad n_1 + n_2 \textsf{ is } n \textsf{ nat}}{\texttt{plus}(e_1; e_2) \Downarrow \texttt{num}[n]} \tag{7.1c}$$

$$\frac{e_1 \Downarrow \texttt{str}[s_1] \quad e_2 \Downarrow \texttt{str}[s_2] \quad s_1 \hat{\ } s_2 = s \textsf{ str}}{\texttt{cat}(e_1; e_2) \Downarrow \texttt{str}[s]} \tag{7.1d}$$

$$\frac{e \Downarrow \texttt{str}[s] \quad |s| = n \textsf{ nat}}{\texttt{len}(e) \Downarrow \texttt{num}[n]} \tag{7.1e}$$

eval-by-value

$$\frac{e_1 \Downarrow v_1 \quad [v_1/x]e_2 \Downarrow v_2}{\texttt{let}(e_1; x.e_2) \Downarrow v_2} \tag{7.2}$$

Relating Structural & Evaluation Dynamics

Lemma 7.3. *If $e \Downarrow v$, then $e \longmapsto^* v$.*

Proof By induction on the definition of the evaluation judgment. For example, suppose that $\texttt{plus}(e_1; e_2) \Downarrow \texttt{num}[n]$ by the rule for evaluating additions. By induction, we know that $e_1 \longmapsto^* \texttt{num}[n_1]$ and $e_2 \longmapsto^* \texttt{num}[n_2]$. We reason as follows:

$$\begin{array}{rcl}
\texttt{plus}(e_1; e_2) & \longmapsto^* & \texttt{plus}(\texttt{num}[n_1]; e_2) \\
& \longmapsto^* & \texttt{plus}(\texttt{num}[n_1]; \texttt{num}[n_2]) \\
& \longmapsto & \texttt{num}[n_1 + n_2]
\end{array}$$

Therefore, $\texttt{plus}(e_1; e_2) \longmapsto^* \texttt{num}[n_1 + n_2]$, as required. The other cases are handled similarly. $\square$

Relating Structural & Evaluation Dynamics

Lemma 7.4. *If $e \longmapsto e'$ and $e' \Downarrow v$, then $e \Downarrow v$.*

Proof By induction on the definition of the transition judgment. For example, suppose that $\texttt{plus}(e_1; e_2) \longmapsto \texttt{plus}(e_1'; e_2)$, where $e_1 \longmapsto e_1'$. Suppose further that $\texttt{plus}(e_1'; e_2) \Downarrow v$, so that $e_1' \Downarrow \texttt{num}[n_1]$, and $e_2 \Downarrow \texttt{num}[n_2]$, and $n_1 + n_2$ is n $\texttt{nat}$, and v is $\texttt{num}[n]$. By induction $e_1 \Downarrow \texttt{num}[n_1]$, and hence $\texttt{plus}(e_1; e_2) \Downarrow \texttt{num}[n]$, as required. $\square$

Cost Dynamics

Evaluation judgments have the form $e \Downarrow^k v$, with the meaning that e evaluates to v in k steps.

$$\overline{\mathtt{num}[n] \Downarrow^0 \mathtt{num}[n]} \tag{7.4a}$$

$$\frac{e_1 \Downarrow^{k_1} \mathtt{num}[n_1] \quad e_2 \Downarrow^{k_2} \mathtt{num}[n_2]}{\mathtt{plus}(e_1; e_2) \Downarrow^{k_1+k_2+1} \mathtt{num}[n_1 + n_2]} \tag{7.4b}$$

$$\overline{\mathtt{str}[s] \Downarrow^0 \mathtt{str}[s]} \tag{7.4c}$$

$$\frac{e_1 \Downarrow^{k_1} s_1 \quad e_2 \Downarrow^{k_2} s_2}{\mathtt{cat}(e_1; e_2) \Downarrow^{k_1+k_2+1} \mathtt{str}[s_1 \,\hat{}\, s_2]} \tag{7.4d}$$

$$\frac{[e_1/x]e_2 \Downarrow^{k_2} v_2}{\mathtt{let}(e_1; x.e_2) \Downarrow^{k_2+1} v_2} \tag{7.4e}$$

For a by-value interpretation of `let`, rule (7.4e) is replaced by the following rule:

$$\frac{e_1 \Downarrow^{k_1} v_1 \quad [v_1/x]e_2 \Downarrow^{k_2} v_2}{\mathtt{let}(e_1; x.e_2) \Downarrow^{k_1+k_2+1} v_2} \tag{7.5}$$

Theorem 7.7. *For any closed expression e and closed value v of the same type, $e \Downarrow^k v$ iff $e \longmapsto^k v$.*