11.1 Nullary and Binary Sums

The abstract syntax of sums is given by the following grammar:

Typ	τ	::=	$\mathtt{void}$	$\mathtt{void}$	nullary sum
			$+(\tau_1;\tau_2)$	$\tau_1+\tau_2$	binary sum
Exp	e	::=	$\mathtt{case}\{\tau\}(e)$	$\mathtt{case}\,e\,\{\,\}$	null case
			$\mathtt{in}[\mathtt{l}]\{\tau_1;\tau_2\}(e)$	$\mathtt{l}\cdot e$	left injection
			$\mathtt{in}[\mathtt{r}]\{\tau_1;\tau_2\}(e)$	$\mathtt{r}\cdot e$	right injection
			$\mathtt{case}\{x_1.e_1;x_2.e_2\}(e)$	$\mathtt{case}\,e\,\{\mathtt{l}\cdot x_1\hookrightarrow e_1\,|\,\mathtt{r}\cdot x_2\hookrightarrow e_2\}$	case analysis

The nullary sum represents a choice of zero alternatives, and hence admits no introduction form. The elimination form, $\mathtt{case}\,e\,\{\,\}$, expresses the contradiction that e is a value of type $\mathtt{void}$. The elements of the binary sum type are labeled to show whether they are drawn from the left or the right summand, either $\mathtt{l}\cdot e$ or $\mathtt{r}\cdot e$. A value of the sum type is eliminated by case analysis.

The statics of sum types is given by the following rules.

$$\frac{\Gamma\vdash e:\mathtt{void}}{\Gamma\vdash \mathtt{case}\,e\,\{\,\}:\tau} \tag{11.1a}$$

$$\frac{\Gamma\vdash e:\tau_1}{\Gamma\vdash \mathtt{l}\cdot e:\tau_1+\tau_2} \tag{11.1b}$$

$$\frac{\Gamma\vdash e:\tau_2}{\Gamma\vdash \mathtt{r}\cdot e:\tau_1+\tau_2} \tag{11.1c}$$

$$\frac{\Gamma\vdash e:\tau_1+\tau_2 \quad \Gamma,x_1:\tau_1\vdash e_1:\tau \quad \Gamma,x_2:\tau_2\vdash e_2:\tau}{\Gamma\vdash \mathtt{case}\,e\,\{\mathtt{l}\cdot x_1\hookrightarrow e_1\,|\,\mathtt{r}\cdot x_2\hookrightarrow e_2\}:\tau} \tag{11.1d}$$

For the sake of readability, in rules (11.1b) and (11.1c) we have written $\mathtt{l}\cdot e$ and $\mathtt{r}\cdot e$ in place of the abstract syntax $\mathtt{in}[\mathtt{l}]\{\tau_1;\tau_2\}(e)$ and $\mathtt{in}[\mathtt{r}]\{\tau_1;\tau_2\}(e)$, which includes the types τ_1 and τ_2 explicitly. In rule (11.1d) both branches of the case analysis must have the same type. Because a type expresses a static "prediction" on the form of the value of an expression, and because an expression of sum type could evaluate to either form at run-time, we must insist that both branches yield the same type.

The dynamics of sums is given by the following rules:

$$\frac{e\longmapsto e'}{\mathtt{case}\,e\,\{\,\}\longmapsto \mathtt{case}\,e'\,\{\,\}} \tag{11.2a}$$

$$\frac{[e\ \mathrm{val}]}{\mathtt{l}\cdot e\ \mathrm{val}} \tag{11.2b}$$

$$\frac{[e\ \mathrm{val}]}{\mathtt{r}\cdot e\ \mathrm{val}} \tag{11.2c}$$

$$\left[\frac{e\longmapsto e'}{\mathtt{l}\cdot e\longmapsto \mathtt{l}\cdot e'}\right] \tag{11.2d}$$

$$\left[\frac{e\longmapsto e'}{\mathtt{r}\cdot e\longmapsto \mathtt{r}\cdot e'}\right] \tag{11.2e}$$

$$\frac{e\longmapsto e'}{\mathtt{case}\,e\,\{\mathtt{l}\cdot x_1\hookrightarrow e_1\,|\,\mathtt{r}\cdot x_2\hookrightarrow e_2\}\longmapsto \mathtt{case}\,e'\,\{\mathtt{l}\cdot x_1\hookrightarrow e_1\,|\,\mathtt{r}\cdot x_2\hookrightarrow e_2\}} \tag{11.2f}$$

$$\frac{[e\ \mathrm{val}]}{\mathtt{case}\,\mathtt{l}\cdot e\,\{\mathtt{l}\cdot x_1\hookrightarrow e_1\,|\,\mathtt{r}\cdot x_2\hookrightarrow e_2\}\longmapsto [e/x_1]e_1} \tag{11.2g}$$

$$\frac{[e\ \mathrm{val}]}{\mathtt{case}\,\mathtt{r}\cdot e\,\{\mathtt{l}\cdot x_1\hookrightarrow e_1\,|\,\mathtt{r}\cdot x_2\hookrightarrow e_2\}\longmapsto [e/x_2]e_2} \tag{11.2h}$$

11.3.2 Booleans

Perhaps the simplest example of a sum type is the familiar type of Booleans, whose syntax is given by the following grammar:

Typ	τ	::=	`bool`	booleans
Exp	e	::=	`true`	truth
			`false`	falsity
			$\mathtt{if}(e;e_1;e_2)$	conditional

The expression $\mathtt{if}(e;e_1;e_2)$ branches on the value of $e : \mathtt{bool}$.

The statics of Booleans is given by the following typing rules:

$$\frac{}{\Gamma \vdash \mathtt{true} : \mathtt{bool}} \tag{11.5a}$$

$$\frac{}{\Gamma \vdash \mathtt{false} : \mathtt{bool}} \tag{11.5b}$$

$$\frac{\Gamma \vdash e : \mathtt{bool} \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathtt{if}\, e\, \mathtt{then}\, e_1\, \mathtt{else}\, e_2 : \tau} \tag{11.5c}$$

The dynamics is given by the following value and transition rules:

$$\frac{}{\mathtt{true}\ \mathtt{val}} \tag{11.6a}$$

$$\frac{}{\Delta \vdash \mathtt{unit}\ \mathtt{type}} \tag{15.7b}$$

$$\frac{\Delta \vdash \tau_1\ \mathtt{type} \quad \Delta \vdash \tau_2\ \mathtt{type}}{\Delta \vdash \mathtt{prod}(\tau_1;\tau_2)\ \mathtt{type}} \tag{15.7c}$$

$$\frac{}{\Delta \vdash \mathtt{void}\ \mathtt{type}} \tag{15.7d}$$

$$\frac{\Delta \vdash \tau_1\ \mathtt{type} \quad \Delta \vdash \tau_2\ \mathtt{type}}{\Delta \vdash \mathtt{sum}(\tau_1;\tau_2)\ \mathtt{type}} \tag{15.7e}$$

$$\frac{\Delta \vdash \tau_1\ \mathtt{type} \quad \Delta \vdash \tau_2\ \mathtt{type}}{\Delta \vdash \mathtt{arr}(\tau_1;\tau_2)\ \mathtt{type}} \tag{15.7f}$$

$$\frac{\Delta, t\ \mathtt{type} \vdash \tau\ \mathtt{type} \quad \Delta \vdash t.\tau\ \mathtt{pos}}{\Delta \vdash \mathtt{ind}(t.\tau)\ \mathtt{type}} \tag{15.7g}$$

$$\frac{\Delta, t\ \mathtt{type} \vdash \tau\ \mathtt{type} \quad \Delta \vdash t.\tau\ \mathtt{pos}}{\Delta \vdash \mathtt{coi}(t.\tau)\ \mathtt{type}} \tag{15.7h}$$

The abstract syntax of **M** is given by the following grammar:

$$\begin{array}{llll} \textit{Exp}\ e ::= & \mathtt{fold}\{t.\tau\}(e) & \mathtt{fold}_{t.\tau}(e) & \text{constructor} \\ & \mathtt{rec}\{t.\tau\}(x.e_1;e_2) & \mathtt{rec}(x.e_1;e_2) & \text{recursor} \\ & \mathtt{unfold}\{t.\tau\}(e) & \mathtt{unfold}_{t.\tau}(e) & \text{destructor} \\ & \mathtt{gen}\{t.\tau\}(x.e_1;e_2) & \mathtt{gen}(x.e_1;e_2) & \text{generator} \end{array}$$

The subscripts on the concrete syntax forms are often omitted when they are clear from context.

The statics for **M** is given by the following typing rules:

$$\frac{\Gamma \vdash e : [\mathtt{ind}(t.\tau)/t]\tau}{\Gamma \vdash \mathtt{fold}\{t.\tau\}(e) : \mathtt{ind}(t.\tau)} \tag{15.8a}$$

$$\frac{\Gamma, x : [\tau'/t]\tau \vdash e_1 : \tau' \quad \Gamma \vdash e_2 : \mathtt{ind}(t.\tau)}{\Gamma \vdash \mathtt{rec}\{t.\tau\}(x.e_1;e_2) : \tau'} \tag{15.8b}$$

$$\frac{\Gamma \vdash e : \mathtt{coi}(t.\tau)}{\Gamma \vdash \mathtt{unfold}\{t.\tau\}(e) : [\mathtt{coi}(t.\tau)/t]\tau} \tag{15.8c}$$

$$\frac{\Gamma \vdash e_2 : \tau_2 \quad \Gamma, x : \tau_2 \vdash e_1 : [\tau_2/t]\tau}{\Gamma \vdash \mathtt{gen}\{t.\tau\}(x.e_1;e_2) : \mathtt{coi}(t.\tau)} \tag{15.8d}$$

15.3 Dynamics

The dynamics of **M** is given in terms of the positive generic extension operation described in Chapter 14. The following rules specify a lazy dynamics for **M**:

$$\frac{}{\mathtt{fold}\{t.\tau\}(e)\ \mathsf{val}} \tag{15.9a}$$

$$\frac{e_2 \longmapsto e_2'}{\mathtt{rec}\{t.\tau\}(x.e_1;e_2) \longmapsto \mathtt{rec}\{t.\tau\}(x.e_1;e_2')} \tag{15.9b}$$

$$\frac{}{\begin{array}{c}\mathtt{rec}\{t.\tau\}(x.e_1;\mathtt{fold}\{t.\tau\}(e_2)) \\ \longmapsto \\ [\mathtt{map}^+\{t.\tau\}(y.\mathtt{rec}\{t.\tau\}(x.e_1;y))(e_2)/x]e_1\end{array}} \tag{15.9c}$$

$$\frac{}{\mathtt{gen}\{t.\tau\}(x.e_1;e_2)\ \mathsf{val}} \tag{15.9d}$$

$$\frac{e \longmapsto e'}{\mathtt{unfold}\{t.\tau\}(e) \longmapsto \mathtt{unfold}\{t.\tau\}(e')} \tag{15.9e}$$

$$\frac{}{\begin{array}{c}\mathtt{unfold}\{t.\tau\}(\mathtt{gen}\{t.\tau\}(x.e_1;e_2)) \\ \longmapsto \\ \mathtt{map}^+\{t.\tau\}(y.\mathtt{gen}\{t.\tau\}(x.e_1;y))([e_2/x]e_1)\end{array}} \tag{15.9f}$$

$$\frac{\Delta, t \text{ type } \Gamma \vdash e : \tau}{\Delta\ \Gamma \vdash \Lambda(t.e) : \forall(t.\tau)} \tag{16.2d}$$

$$\frac{\Delta\ \Gamma \vdash e : \forall(t.\tau')\quad \Delta \vdash \tau \text{ type}}{\Delta\ \Gamma \vdash \mathtt{App}\{\tau\}(e) : [\tau/t]\tau'} \tag{16.2e}$$

Lemma 16.1 (Regularity). *If $\Delta\ \Gamma \vdash e : \tau$, and if $\Delta \vdash \tau_i$ type for each assumption $x_i : \tau_i$ in Γ, then $\Delta \vdash \tau$ type.*

Proof. By induction on rules (16.2). □

Lemma 16.2 (Substitution). 1. *If Δ, t type $\vdash \tau'$ type and $\Delta \vdash \tau$ type, then $\Delta \vdash [\tau/t]\tau'$ type.*

2. *If Δ, t type $\Gamma \vdash e' : \tau'$ and $\Delta \vdash \tau$ type, then $\Delta\ [\tau/t]\Gamma \vdash [\tau/t]e' : [\tau/t]\tau'$.*

3. *If $\Delta\ \Gamma, x : \tau \vdash e' : \tau'$ and $\Delta\ \Gamma \vdash e : \tau$, then $\Delta\ \Gamma \vdash [e/x]e' : \tau'$.*

The second part of the lemma requires substitution into the context Γ as well as into the term and its type, because the type variable t may occur freely in any of these positions.

Returning to the motivating examples from the introduction, the polymorphic identity function, I, is written

$$\Lambda(t)\,\lambda(x:t)\,x;$$

it has the polymorphic type

$$\forall(t.t \to t).$$

Instances of the polymorphic identity are written $I[\tau]$, where τ is some type, and have the type $\tau \to \tau$.

Similarly, the polymorphic composition function, C, is written

$$\Lambda(t_1)\,\Lambda(t_2)\,\Lambda(t_3)\,\lambda(f:t_2 \to t_3)\,\lambda(g:t_1 \to t_2)\,\lambda(x:t_1)\,f(g(x)).$$

The function C has the polymorphic type

$$\forall(t_1.\forall(t_2.\forall(t_3.(t_2 \to t_3) \to (t_1 \to t_2) \to (t_1 \to t_3)))).$$

Instances of C are obtained by applying it to a triple of types, written $C[\tau_1][\tau_2][\tau_3]$. Each such instance has the type

$$(\tau_2 \to \tau_3) \to (\tau_1 \to \tau_2) \to (\tau_1 \to \tau_3).$$

16.1 Polymorphic Abstraction

The language **F** is a variant of **T** in which we eliminate the type of natural numbers, but add, in compensation, polymorphic types:[1]

Typ	τ	::=	t	t	variable
			$\rightarrow(\tau_1;\tau_2)$	$\tau_1 \rightarrow \tau_2$	function
			$\forall(t.\tau)$	$\forall(t.\tau)$	polymorphic
Exp	e	::=	x	x	
			$\lambda\{\tau\}(x.e)$	$\lambda(x:\tau)\,e$	abstraction
			$\mathsf{ap}(e_1;e_2)$	$e_1(e_2)$	application
			$\Lambda(t.e)$	$\Lambda(t)\,e$	type abstraction
			$\mathtt{App}\{\tau\}(e)$	$e[\tau]$	type application

A *type abstraction* $\Lambda(t.e)$ defines a *generic*, or *polymorphic*, function with *type variable* t standing for an unspecified type within e. A *type application*, or *instantiation* $\mathtt{App}\{\tau\}(e)$ applies a polymorphic function to a specified type, which is plugged in for the type variable to obtain the result. The *universal type*, $\forall(t.\tau)$, classifies polymorphic functions.

The statics of **F** consists of two judgment forms, the *type formation* judgment,

$$\Delta \vdash \tau \text{ type},$$

and the *typing judgment*,

$$\Delta\,\Gamma \vdash e : \tau.$$

The hypotheses Δ have the form t type, where t is a variable of sort Typ, and the hypotheses Γ have the form $x : \tau$, where x is a variable of sort Exp.

The rules defining the type formation judgment are as follows:

$$\frac{}{\Delta, t \text{ type} \vdash t \text{ type}} \tag{16.1a}$$

$$\frac{\Delta \vdash \tau_1 \text{ type} \quad \Delta \vdash \tau_2 \text{ type}}{\Delta \vdash \rightarrow(\tau_1;\tau_2) \text{ type}} \tag{16.1b}$$

$$\frac{\Delta, t \text{ type} \vdash \tau \text{ type}}{\Delta \vdash \forall(t.\tau) \text{ type}} \tag{16.1c}$$

The rules defining the typing judgment are as follows:

$$\frac{}{\Delta\,\Gamma, x : \tau \vdash x : \tau}$$

$$\frac{\Delta \vdash \tau_1 \text{ type} \quad \Delta\,\Gamma, x : \tau_1 \vdash e : \tau_2}{\Delta\,\Gamma \vdash \lambda\{\tau_1\}(x.e) : \rightarrow(\tau_1;\tau_2)}$$

$$\frac{\Delta\,\Gamma \vdash e_1 : \rightarrow(\tau_2;\tau)\quad \Delta\,\Gamma \vdash e_2 : \tau_2}{\Delta\,\Gamma \vdash \mathsf{ap}(e_1;e_2) : \tau}$$

[1] Girard's original version of System F included the natural numbers as a basic type.

The statics of T is given by the following typing rules:

$$\frac{}{\Gamma, x : \tau \vdash x : \tau} \tag{9.1a}$$

$$\frac{}{\Gamma \vdash \mathtt{z} : \mathtt{nat}} \tag{9.1b}$$

$$\frac{\Gamma \vdash e : \mathtt{nat}}{\Gamma \vdash \mathtt{s}(e) : \mathtt{nat}} \tag{9.1c}$$

$$\frac{\Gamma \vdash e : \mathtt{nat} \quad \Gamma \vdash e_0 : \tau \quad \Gamma, x : \mathtt{nat}, y : \tau \vdash e_1 : \tau}{\Gamma \vdash \mathtt{rec}\{e_0; x.y.e_1\}(e) : \tau} \tag{9.1d}$$

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda\{\tau_1\}(x.e) : \to(\tau_1; \tau_2)} \tag{9.1e}$$

$$\frac{\Gamma \vdash e_1 : \to(\tau_2; \tau) \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \mathtt{ap}(e_1; e_2) : \tau} \tag{9.1f}$$

As usual, admissibility of the structural rule of substitution is crucially important.

Lemma 9.1. *If* $\Gamma \vdash e : \tau$ *and* $\Gamma, x : \tau \vdash e' : \tau'$, *then* $\Gamma \vdash [e/x]e' : \tau'$.

9.2

9.2 Dynamics

Th
laz

The closed values of T are defined by the following rules:

$$\frac{}{\mathtt{z}\ \mathtt{val}} \tag{9.2a}$$

$$\frac{[e\ \mathtt{val}]}{\mathtt{s}(e)\ \mathtt{val}} \tag{9.2b}$$

$$\frac{}{\lambda\{\tau\}(x.e)\ \mathtt{val}} \tag{9.2c}$$

$$\left[\frac{e_1\ \mathtt{val} \quad e_2 \longmapsto e_2'}{\mathtt{ap}(e_1; e_2) \longmapsto \mathtt{ap}(e_1; e_2')}\right] \tag{9.3c}$$

$$\frac{[e_2\ \mathtt{val}]}{\mathtt{ap}(\lambda\{\tau\}(x.e); e_2) \longmapsto [e_2/x]e} \tag{9.3d}$$

$$\frac{e \longmapsto e'}{\mathtt{rec}\{e_0; x.y.e_1\}(e) \longmapsto \mathtt{rec}\{e_0; x.y.e_1\}(e')} \tag{9.3e}$$

$$\frac{}{\mathtt{rec}\{e_0; x.y.e_1\}(\mathtt{z}) \longmapsto e_0} \tag{9.3f}$$

$$\frac{\mathtt{s}(e)\ \mathtt{val}}{\mathtt{rec}\{e_0; x.y.e_1\}(\mathtt{s}(e)) \longmapsto [e, \mathtt{rec}\{e_0; x.y.e_1\}(e)/x, y]e_1} \tag{9.3g}$$

Lemma 9.2 (Canonical Forms). *If $e : \tau$ and e val, then*

1. *If $\tau = \mathtt{nat}$, then either $e = \mathtt{z}$ or $e = \mathtt{s}(e')$ for some e'.*
2. *If $\tau = \tau_1 \to \tau_2$, then $e = \lambda(x : \tau_1)\, e_2$ for some e_2.*

The abstract syntax of products is given by the following grammar:

Typ	τ	::=	$\mathtt{unit}$	$\mathtt{unit}$	nullary product
			$\times(\tau_1;\tau_2)$	$\tau_1 \times \tau_2$	binary product
Exp	e	::=	$\mathtt{triv}$	$\langle\rangle$	null tuple
			$\mathtt{pair}(e_1;e_2)$	$\langle e_1, e_2\rangle$	ordered pair
			$\mathtt{pr}[\mathtt{l}](e)$	$e \cdot \mathtt{l}$	left projection
			$\mathtt{pr}[\mathtt{r}](e)$	$e \cdot \mathtt{r}$	right projection

---

The statics of product types is given by the following rules.

$$\frac{}{\Gamma \vdash \langle\rangle : \mathtt{unit}} \tag{10.1a}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \langle e_1, e_2\rangle : \tau_1 \times \tau_2} \tag{10.1b}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash e \cdot \mathtt{l} : \tau_1} \tag{10.1c}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash e \cdot \mathtt{r} : \tau_2} \tag{10.1d}$$

The dynamics of product types is defined by the following rules:

$$\frac{}{\langle\rangle \text{ val}} \tag{10.2a}$$

$$\frac{[e_1 \text{ val}] \quad [e_2 \text{ val}]}{\langle e_1, e_2\rangle \text{ val}} \tag{10.2b}$$

$$\left[\frac{e_1 \longmapsto e_1'}{\langle e_1, e_2\rangle \longmapsto \langle e_1', e_2\rangle}\right] \tag{10.2c}$$

$$\left[\frac{e_1 \text{ val} \quad e_2 \longmapsto e_2'}{\langle e_1, e_2\rangle \longmapsto \langle e_1, e_2'\rangle}\right] \tag{10.2d}$$

$$\frac{e \longmapsto e'}{e \cdot \mathtt{l} \longmapsto e' \cdot \mathtt{l}} \tag{10.2e}$$

$$\frac{e \longmapsto e'}{e \cdot \mathtt{r} \longmapsto e' \cdot \mathtt{r}} \tag{10.2f}$$

$$\frac{[e_1 \text{ val}] \quad [e_2 \text{ val}]}{\langle e_1, e_2\rangle \cdot \mathtt{l} \longmapsto e_1} \tag{10.2g}$$

$$\frac{[e_1 \text{ val}] \quad [e_2 \text{ val}]}{\langle e_1, e_2\rangle \cdot \mathtt{r} \longmapsto e_2} \tag{10.2h}$$

The bracketed rules and premises are omitted for a lazy dynamics, and included for an eager

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

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

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

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

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

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

6.2 Progress

The progress theorem captures the idea that well-typed programs cannot "get stuck". The proof depends crucially on the following lemma, which characterizes the values of each type.

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

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

Proof. By induction on rules (4.1) and (5.3). □

7.1 Evaluation Dynamics

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:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\frac{}{\mathtt{num}[n]\ \mathsf{val}} \tag{5.3a}$$

$$\frac{}{\mathtt{str}[s]\ \mathsf{val}} \tag{5.3b}$$

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

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

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

The dynamics of **F** is given as follows:

$$\frac{}{\lambda\{\tau\}(x.e)\ \mathsf{val}} \tag{16.3a}$$

$$\frac{}{\Lambda(t.e)\ \mathsf{val}} \tag{16.3b}$$

$$\frac{[e_2\ \mathsf{val}]}{\mathsf{ap}(\lambda\{\tau_1\}(x.e);e_2) \longmapsto [e_2/x]e} \tag{16.3c}$$

$$\frac{e_1 \longmapsto e_1'}{\mathsf{ap}(e_1;e_2) \longmapsto \mathsf{ap}(e_1';e_2)} \tag{16.3d}$$

$$\left[\frac{e_1\ \mathsf{val} \quad e_2 \longmapsto e_2'}{\mathsf{ap}(e_1;e_2) \longmapsto \mathsf{ap}(e_1;e_2')}\right] \tag{16.3e}$$

$$\frac{}{\mathtt{App}\{\tau\}(\Lambda(t.e)) \longmapsto [\tau/t]e} \tag{16.3f}$$

$$\frac{e \longmapsto e'}{\mathtt{App}\{\tau\}(e) \longmapsto \mathtt{App}\{\tau\}(e')} \tag{16.3g}$$

The bracketed premises and rule are included for a call-by-value interpretation, and omitted for a call-by-name interpretation of **F**.

It is a simple matter to prove safety for **F**, using familiar methods.

Lemma 16.3 (Canonical Forms). *Suppose that $e : \tau$ and e val, then*

1. *If $\tau = \rightarrow(\tau_1;\tau_2)$, then $e = \lambda\{\tau_1\}(x.e_2)$ with $x : \tau_1 \vdash e_2 : \tau_2$.*
2. *If $\tau = \forall(t.\tau')$, then $e = \Lambda(t.e')$ with t type $\vdash e' : \tau'$.*

Proof. By rule induction on the statics. □

17.1.1 Statics

The statics of **FE** is given by these rules:

$$\frac{\Delta, t\ \mathsf{type} \vdash \tau\ \mathsf{type}}{\Delta \vdash \exists(t.\tau)\ \mathsf{type}} \tag{17.1a}$$

$$\frac{\Delta \vdash \rho\ \mathsf{type} \quad \Delta, t\ \mathsf{type} \vdash \tau\ \mathsf{type} \quad \Delta\ \Gamma \vdash e : [\rho/t]\tau}{\Delta\ \Gamma \vdash \mathsf{pack}\{t.\tau\}\{\rho\}(e) : \exists(t.\tau)} \tag{17.1b}$$

$$\frac{\Delta\ \Gamma \vdash e_1 : \exists(t.\tau) \quad \Delta, t\ \mathsf{type}\ \Gamma, x : \tau \vdash e_2 : \tau_2 \quad \Delta \vdash \tau_2\ \mathsf{type}}{\Delta\ \Gamma \vdash \mathsf{open}\{t.\tau\}\{\tau_2\}(e_1;t,x.e_2) : \tau_2} \tag{17.1c}$$

Rule (17.1c) is complex, so study it carefully! There are two important things to notice:

1. The type of the client, τ_2, must not involve the abstract type t. This restriction prevents the client from attempting to export a value of the abstract type outside of the scope of its definition.
2. The body of the client, e_2, is type checked without knowledge of the representation type, t. The client is, in effect, polymorphic in the type variable t.

Lemma 17.1 (Regularity). *Suppose that $\Delta\ \Gamma \vdash e : \tau$. If $\Delta \vdash \tau_i$ type for each $x_i : \tau_i$ in Γ, then $\Delta \vdash \tau$ type.*

Proof. By induction on rules (17.1), using substitution for expressions and types. □

17.1.2 Dynamics

The dynamics of **FE** is defined by the following rules (including the bracketed material for an eager interpretation, and omitting it for a lazy interpretation):

$$\frac{[e\ \mathsf{val}]}{\mathtt{pack}\{t.\tau\}\{\rho\}(e)\ \mathsf{val}} \tag{17.2a}$$

$$\left[\frac{e \longmapsto e'}{\mathtt{pack}\{t.\tau\}\{\rho\}(e) \longmapsto \mathtt{pack}\{t.\tau\}\{\rho\}(e')}\right] \tag{17.2b}$$

$$\frac{e_1 \longmapsto e_1'}{\mathtt{open}\{t.\tau\}\{\tau_2\}(e_1; t, x.e_2) \longmapsto \mathtt{open}\{t.\tau\}\{\tau_2\}(e_1'; t, x.e_2)} \tag{17.2c}$$

$$\frac{[e\ \mathsf{val}]}{\mathtt{open}\{t.\tau\}\{\tau_2\}(\mathtt{pack}\{t.\tau\}\{\rho\}(e); t, x.e_2) \longmapsto [\rho, e/t, x]e_2} \tag{17.2d}$$

It is important to see that, according to these rules, *there are no abstract types at run time*! The representation type is propagated to the client by substitution when the package is opened, thereby eliminating the abstraction boundary between the client and the implementor. Thus, data abstraction is a *compile-time discipline* that leaves no traces of its presence at execution time.

17.1.3 Safety

Safety of **FE** is stated and proved by decomposing it into progress and preservation.

Theorem 17.2 (Preservation). *If $e : \tau$ and $e \longmapsto e'$, then $e' : \tau$.*

Proof. By rule induction on $e \longmapsto e'$, using substitution for both expression- and type variables. □

Lemma 17.3 (Canonical Forms). *If $e : \exists(t.\tau)$ and e val, then $e = \mathtt{pack}\{t.\tau\}\{\rho\}(e')$ for some type ρ and some e' such that $e' : [\rho/t]\tau$.*

Proof. By rule induction on the statics, using the definition of closed values. □

Theorem 17.4 (Progress). *If $e : \tau$ then either e val or there exists e' such that $e \longmapsto e'$.*

Proof. By rule induction on $e : \tau$, using the canonical forms lemma. □