1 Grammar

1.1 Types

$$\begin{array}{rcl} \tau & ::= & \mid \tau_0 \to \tau_1 \mid \{\rho\} \mid [\![\rho]\!] \\ \rho & ::= & \cdot \mid \_ : \tau \mid l : \tau, \rho \\ l & ::= & \text{row field labels} \\ \alpha & ::= & \text{bound type variables} \end{array}$$

$FV(\tau)$ denotes the free type variables of τ defined in the usual way where the only type variable binder is $\forall\alpha.\tau$.

A row is *closed* if it ends in $\cdot$. We abuse , to stand for both adding a label to a row (e.g. $l : \tau, \rho$) and *appending* two rows together when the first row is closed; e.g.

$$(l_0 : \tau_0, {}_1 : \tau_1, \cdot), (l_2 : \tau_2, l_3 : \tau_3, \_ : \tau) \quad \equiv \quad l_0 : \tau_0, {}_1 : \tau_1, l_2 : \tau_2, l_3 : \tau_3, \_ : \tau$$

Furthermore, writing ρ_0, ρ_1 implies ρ_0 is closed, so $\rho, \_ : \tau$ implies ρ is closed and $\rho, \cdot \equiv \rho$.

We do not forbid multiple occurrences of the same label in a row; outer (to the left) occurrences shadow inner occurences as specified in 2.1.

1.2 Type Substitution

$$\begin{array}{rcl} \alpha[\alpha := \tau] & = & \tau \\ \alpha'[\alpha := \tau] & = & \alpha' \quad (\alpha \neq \alpha') \\ (\tau_0 \to \tau_1)[\alpha := \tau] & = & \tau_0[\alpha := \tau] \to \tau_1[\alpha := \tau] \\ \{\rho\}[\alpha := \tau] & = & \{\rho[\alpha := \tau]\} \\ [\![\rho]\!][\alpha := \tau] & = & [\![\rho[\alpha := \tau]]\!] \end{array}$$

$$\begin{array}{rcl} (\cdot)[\alpha := \tau] & = & \cdot \\ (\_ : \tau')[\alpha := \tau] & = & \_ : \tau'[\alpha := \tau] \\ (l : \tau', \rho)[\alpha := \tau] & = & l : \tau'[\alpha := \tau], \rho[\alpha := \tau] \end{array}$$

1.3 Terms

$$\begin{array}{rcl} e & ::= & x \mid \lambda x.e \mid e_0\, e_1 \mid \{r\} \mid e\#l \mid [\![l : e]\!] \mid \texttt{case}\, \texttt{e}_0\, \texttt{e}_1 \\ r & ::= & \cdot \mid \_ : e \mid l : e, r \\ x & ::= & \text{bound term variables} \end{array}$$

$FV(e)$ denotes the free term variables of e defined in the usual way where the only variable binder is $\lambda x.e$.

1.4 Term Substitution

$$
\begin{array}{rcl}
x[x := e] & = & e \\
x'[x := e] & = & x' \quad (x \neq x') \\
(\lambda x.e')[x := e] & = & \lambda x.e' \\
(\lambda x'.e')[x := e] & = & \lambda x''.e'[x' := x''][x := e] \quad (x \neq x' \wedge x \neq x'' \wedge x'' \notin FV(e)) \\
(e_0\, e_1)[x := e] & = & e_0[x := e]\, e_1[x := e] \\
\{r\}[x := e] & = & \{r[x := e]\} \\
(e'\#l)[x := e] & = & (e'[x := e])\#l \\
[\![l : e']\!][x := e] & = & [\![l : e'[x := e]]\!] \\
(\mathtt{case}\, \mathtt{e_0}\, \mathtt{e_1})[\mathtt{x} := \mathtt{e}] & = & \mathtt{case}\, \mathtt{e_0}[\mathtt{x} := \mathtt{e}]\, \mathtt{e_1}[\mathtt{x} := \mathtt{e}]
\end{array}
$$

$$
\begin{array}{rcl}
(\cdot)[x := e] & = & \cdot \\
(\_ : e')[x := e] & = & \_ : e'[x := e] \\
(l : e', r)[x := e] & = & l : e'[x := e], r[x := e]
\end{array}
$$

1.5 Contexts

$$\Gamma \quad ::= \quad \cdot \mid x : \tau, \Gamma$$

We assume that well formed contexts have a single occurrence of any variable.

2 Typing

2.1 Type Label Lookup $(\rho \mid l \,:\, \tau)$

$$\frac{}{\_ : \tau \mid l \,:\, \tau} \qquad \frac{}{l : \tau, \rho \mid l \,:\, \tau} \qquad \frac{\rho \mid l \,:\, \tau}{l' : \tau', \rho \mid l \,:\, \tau}\ (l \neq l')$$

Note that we allow instantiation of polymorphic fields during type label lookup to facilitate the formulation of the constraint, in 5.2, on the rows involved in a $\mathtt{case}\, \mathtt{e_0}\, \mathtt{e_1}$ expression.

2.2 Case Constraint $(\rho_0 \sqsubseteq \rho_1 \,:\, \tau)$

$$\frac{\rho_1 \mid l_0 \,:\, \tau_0 \to \tau \quad \rho_0 \sqsubseteq \rho_1 \,:\, \tau}{l_0 : \tau_0, \rho_0 \sqsubseteq \rho_1 \,:\, \tau} \qquad \frac{}{\cdot \sqsubseteq \rho \,:\, \tau} \qquad \frac{\rho \mid l \,:\, \tau_0 \to \tau}{\_ : \tau_0 \sqsubseteq \rho \,:\, \tau}\ (l \notin \rho)$$

The third rule above is meant to capture the idea that a variant type with a default label can only be "eliminated" by a record type with a default label.

2.3 Typing Rules $(\Gamma \vdash e \,:\, \tau)$

Functions and bound variables

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x \,:\, \tau} \qquad \frac{\Gamma, x : \tau_x \vdash e \,:\, \tau}{\Gamma \vdash \lambda x.e \,:\, \tau_x \to \tau} \qquad \frac{\Gamma \vdash e_0 \,:\, \tau_1 \to \tau \quad \Gamma \vdash e_1 \,:\, \tau_1}{\Gamma \vdash e_0\, e_1 \,:\, \tau}$$

Records

$$\frac{}{\Gamma \vdash \{\cdot\} : \{\cdot\}} \qquad \frac{\Gamma \vdash e : \tau}{\Gamma \vdash \{\_ : e\} : \{\_ : \tau\}}$$

$$\frac{\Gamma \vdash e : \tau \quad \Gamma \vdash \{r\} : \{\rho\}}{\Gamma \vdash \{l : e, r\} : \{l : \tau, \rho\}} \qquad \frac{\Gamma \vdash e : \{\rho\} \qquad \rho \mid l : \tau}{\Gamma \vdash e\#l : \tau}$$

Variants

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash [\![\_ : e]\!] : [\![\rho, \_ : \tau]\!]} \qquad \frac{\Gamma \vdash e : [\![\rho_0, l : \tau, \rho_1, \_ : \tau]\!]}{\Gamma \vdash e : [\![\rho_0, \rho_1, \_ : \tau]\!]}\ (l \notin \rho_0, \rho_1)$$

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash [\![l : e]\!] : [\![\rho_0, l : \tau, \rho_1]\!]}\ (l \notin \rho_0) \qquad \frac{\Gamma \vdash e_0 : [\![\rho_0]\!] \quad \Gamma \vdash e_1 : \{\rho_1\} \quad \rho_0 \sqsubseteq \rho_1 : \tau}{\Gamma \vdash \mathtt{case}\ \mathtt{e_0}\ \mathtt{e_1} : \tau}$$

2.4 Examples

(assume some base types and corresponding constants)

Here are some expressions with their (not necessarily unique) deriveable types:

$$\begin{array}{lcl}
\mathtt{case}\ [\![\mathtt{b} : 5]\!]\ \{\mathtt{a} : \lambda \mathtt{x.x} + 1, _ : \lambda \mathtt{x.x} * 0\} & : & \mathtt{Num} \\
[\![b : \text{“}\mathtt{b}\text{”}]\!] & : & [\![a : \{\}, _ : \mathtt{Text}]\!] \\
\mathtt{case}\ [\![\mathtt{b} : \text{“}\mathtt{b}\text{”}]\!]\ \{\mathtt{a} : \lambda \mathtt{x.x} + 1, _ : \lambda \mathtt{x.0}\} & : & \mathtt{Num} \\
\mathtt{case}\ [\![_ : \text{“}\mathtt{b}\text{”}]\!]\ \{\mathtt{a} : \lambda \mathtt{x.x} + 1, _ : \lambda \mathtt{x.0}\} & : & \mathtt{Num}
\end{array}$$

Here are some untypeable expressions:

$$\begin{array}{l}
\mathtt{case}\ [\![\mathtt{b} : \text{“}\mathtt{b}\text{”}]\!]\ \{\mathtt{a} : \lambda \mathtt{x.x} + 1, _ : \lambda \mathtt{x.x} * 0\} \\
\mathtt{case}\ [\![_ : \text{“}\mathtt{b}\text{”}]\!]\ \{\mathtt{a} : \lambda \mathtt{x.x} + 1, \mathtt{b} : \lambda \mathtt{x.0}\}
\end{array}$$

3 Evaluation

3.1 Term Label Lookup $(r \mid l \hookrightarrow e)$

$$\frac{}{\_ : e \mid l \hookrightarrow e} \qquad \frac{}{l : e, r \mid l \hookrightarrow e} \qquad \frac{r \mid l \hookrightarrow e}{l' : e', r \mid l \hookrightarrow e}\ (l \neq l')$$

3.2 Evaluation Rules $(e_0 \hookrightarrow e_1)$

$$\frac{}{\lambda x.e \hookrightarrow \lambda x.e} \qquad \frac{e_0 \hookrightarrow \lambda x.e' \quad e'[x := e_1] \hookrightarrow e}{e_0\, e_1 \hookrightarrow e}$$

$$\frac{}{\{r\} \hookrightarrow \{r\}} \qquad \frac{e \hookrightarrow \{r\} \quad r \mid l \hookrightarrow e_l \quad e_l \hookrightarrow e'}{e\#l \hookrightarrow e'}$$

$$\frac{}{[\![l : e]\!] \hookrightarrow [\![l : e]\!]} \qquad \frac{e_0 \hookrightarrow [\![l : e_l]\!] \quad e_1 \hookrightarrow \{r\} \quad r \mid l \hookrightarrow e_r \quad e_r\, e_l \hookrightarrow e}{\mathtt{case}\ \mathtt{e_0}\ \mathtt{e_1} \hookrightarrow \mathtt{e}}$$

4 Correctness

Theorem 1. $\forall e\,\tau.\exists e'.\ \cdot \vdash e : \tau$ *and* $e \hookrightarrow e'$ *implies* $\cdot \vdash e' : \tau$

5 A Tighter Coupling of Records and Variants

Record terms are currently used to consume variant terms, i.e. in $\texttt{case}\,\texttt{e}_0\,\texttt{e}_1$, the second argument e_1 should be a record; however there is a bit of lost symmetry since variant terms are not involved in consuming records. In this section we slightly reformulate the system to allow this symmetry. Specifically we will use projection as the single elimination form for both records and variants, and we will modify projection to take a variant as a second argument, rather than a label; this change also removes the reliance on raw labels, which are not first class values, in the core expressions.

Variants
No `case` construct.

5.1 Project Fun $(\rho_0 \trianglelefteq \rho_1 @ \tau)$

$$\frac{\rho_0 \mid l_1 : \tau_1 \to \tau \quad \rho_0 \trianglelefteq \rho_1 @ \tau}{\rho_0 \trianglelefteq l_1 : \tau_1, \rho_1 @ \tau} \qquad \frac{}{\rho \trianglelefteq \cdot @ \tau} \qquad \frac{\rho \mid l : \tau_0 \to \tau}{\rho \trianglelefteq \_ : \tau_0 @ \tau}\ (l \notin \rho)$$

Third rule above is interpretation of term variant default label as only being able to project the term record default label; removing the condition would change this to allowing the (non-deterministic) projection of any suitable label.

5.2 Project Arg $(\rho_0 \trianglerighteq \rho_1 @ \tau)$

$$\frac{\rho_0 \mid l_1 : \tau_1 \quad \rho_0 \trianglerighteq \rho_1 @ \tau}{\rho_0 \trianglerighteq l_1 : \tau_1 \to \tau, \rho_1 @ \tau} \qquad \frac{}{\rho \trianglerighteq \cdot @ \tau} \qquad \frac{\rho \mid l : \tau_0 \to \tau}{\rho \trianglerighteq \_ : \tau_0 @ \tau}\ (l \notin \rho)$$

Row Elimination (Projection)

$$\frac{\Gamma \vdash e_0 : \{\rho_0\} \quad \Gamma \vdash e_1 : [\![\rho_1]\!] \quad \rho_0 \trianglelefteq \rho_1 @ \tau}{\Gamma \vdash e_0 \mathbin{/\!\!/\!\!/} e_1 : \tau} \qquad \frac{\Gamma \vdash e_0 : \{\rho_0\} \quad \Gamma \vdash e_1 : [\![\rho_1]\!] \quad \rho_0 \trianglerighteq \rho_1 @ \tau}{\Gamma \vdash e_0 /\!\!/\!\!/ e_1 : \tau}$$

Projection Evaluation

$$\frac{e_0 \hookrightarrow \{r\} \quad e_1 \hookrightarrow [\![l : e_l]\!] \quad r \mid l \hookrightarrow e_r \quad e_r\, e_l \hookrightarrow e}{e_0 \mathbin{/\!\!/\!\!/} e_1 \hookrightarrow e} \qquad \frac{e_0 \hookrightarrow \{r\} \quad e_1 \hookrightarrow [\![l : e_l]\!] \quad r \mid l \hookrightarrow e_r \quad e_l\, e_r \hookrightarrow e}{e_0 /\!\!/\!\!/ e_1 \hookrightarrow e}$$

6 Default Labels

There are multiple interpretations we can think for default labels, specifically with regards to projection and `case` of the default label itself. Everything else works similarly between interpretations. Since projection can be thought of as a special `case`, we'll only deal with the latter.

1. The way that is described in this system presently treats the default label as a regular label in a record. A variant with a default label matches the default label in a record it is `case`d with.

   However, the presence of a default label makes all lookups suceed. If a variant fails to match any labels direclty in a record for a `case`, it matches with the default label.

   So this assigns a special meaning to an otherwise normal label.

2. Another way we can think of the default label as is a selector for everything that is not in the variant. However, if the default label in a variatn matches with multiple functions in its corresponding record, what do we do with all of their return values? A function typically cannot return a variable number of values.

   One solution that lets us match everything missing is to impose a constraint upon `case` statements. Specifically, `case` statements would take a third argument: a combining function f. If the return type of each function in the record argument is τ, then $f : \tau \to \tau \to \tau$. By necessity, f would need to be commutative.

   This would let us, for example, collect all of the results that don't match into a list like so

$$\texttt{case}\ [\![\_ : ()]\!]\ \{\texttt{a} : \lambda\texttt{x}.[1], \texttt{b} : \lambda\texttt{x}.[2]\} = [1, 2]$$

7 Small-Step Semantics

These semantics are agnostic about whether we are using default labels. If not, we could simplify the system by writing the typing and evaluation rules without using special lookup judgments (e.g., "$r \mid l \hookrightarrow e_r$" could be replaced by "$r = r_1, l{:}e_r, r_2$ where $l \notin r_1$".)

7.1 Core Rules

$$\frac{e_0 \rightsquigarrow e_0'}{e_0\, e_1 \rightsquigarrow e_0'\, e_1} \qquad \frac{}{(\lambda x.e')\, e_1 \rightsquigarrow e'[x := e_1]}$$

7.2 If we are using projection

$$\frac{e_0 \rightsquigarrow e_0'}{e_0 \# l \rightsquigarrow e_0' \# l} \qquad \frac{r \,|\, l \hookrightarrow e_l}{\{r\} \# l \rightsquigarrow e_l}$$

$$\frac{e_0 \rightsquigarrow e_0'}{\texttt{case}\ \texttt{e}_0\ \texttt{e}_1 \rightsquigarrow \texttt{case}\ \texttt{e}_0'\ \texttt{e}_1} \qquad \frac{e_1 \rightsquigarrow e_1'}{\texttt{case}\ \texttt{e}_0\ \texttt{e}_1 \rightsquigarrow \texttt{case}\ \texttt{e}_0\ \texttt{e}_1'} \qquad \frac{r \,|\, l \hookrightarrow e_r}{\texttt{case}\ [\![\texttt{l} : \texttt{e}_1]\!]\ \{\texttt{r}\} \rightsquigarrow \texttt{e}_\texttt{r}\ \texttt{e}_1}$$

We could enforce left-to-right evaluation by having the second "case" rule start working on e_1 only if e_0 is already a value, but in the absence of side-effects it doesn't matter.

7.3 If we are using the symmetric version:

$$\frac{e_0 \rightsquigarrow e_0'}{e_0 \mathbin{\#\!\!\#} e_1 \rightsquigarrow e_0' \mathbin{\#\!\!\#} e_1} \qquad \frac{e_1 \rightsquigarrow e_1'}{e_0 \mathbin{\#\!\!\#} e_1 \rightsquigarrow e_0 \mathbin{\#\!\!\#} e_1'} \qquad \frac{r \,|\, l \hookrightarrow e_r}{\{r\} \mathbin{\#\!\!\#} [\![l : e_l]\!] \rightsquigarrow e_r\ e_l}$$

$$\frac{e_0 \rightsquigarrow e_0'}{e_0 \mathbin{\#\!\!\#} e_1 \rightsquigarrow e_0' \mathbin{\#\!\!\#} e_1} \qquad \frac{e_1 \rightsquigarrow e_1'}{e_0 \mathbin{\#\!\!\#} e_1 \rightsquigarrow e_0 \mathbin{\#\!\!\#} e_1'} \qquad \frac{r \,|\, l \hookrightarrow e_r}{\{r\} \mathbin{\#\!\!\#} [\![l : e_l]\!] \rightsquigarrow e_l\ e_r}$$

8 Small-Step Correctness

check that proofs work with defaults (will mostly require extending lemmas and some logic involving case constraints).

We use v to refer to a value. We assume call-by-name, so that record values and variant values can contain "unevaluated" expressions.

$$v \quad ::= \quad \lambda x.e \mid \{r\} \mid [\![l : e]\!]$$

Lemma 1. *Canonical Forms*

1. *If* $\cdot \vdash v : \tau_1 \to \tau_2$ *then* v *is* $\lambda x.e$.
2. *If* $\cdot \vdash v : \{\rho\}$ *then* v *is* $\{r\}$.
3. *If* $\cdot \vdash v : [\![\rho]\!]$ *then* v *is* $[\![l : e]\!]$.

Proof. Without polymorphism, it follows directly by inspection of the typing rules. (E.g., the only typing rule that lets us conclude a value has an arrow type is the rule for typing lambdas, so any value with an arrow type must be a lambda.) □

Theorem 2 (Progress)**.** *if* $\cdot \vdash e : \tau$ *then either* e *is a value or else* $e \rightsquigarrow e'$ *for some* e'.

Proof. By (structural) induction on the proof that $\cdot \vdash e : \tau$.

If e is a value, then the proof is complete.

If e is not a value, we examine the proof tree for $\cdot \vdash e : \tau$.

We consider the possible cases for e, ignoring ones that are values.

- Case
$$\frac{\Gamma \vdash e_0 : \tau_1 \to \tau \quad \Gamma \vdash e_1 : \tau_1}{\Gamma \vdash e_0\, e_1 : \tau}$$

  We know that $\cdot \vdash e_0 : \tau_1 \to \tau$ and $\cdot \vdash e_1 : \tau_1$.

  There are two cases to consider for e_0.

  If e_0 is a value, then by lemma 1 we conclude that $e_0 = \lambda x.e_0'$ for some e_0'. We may then apply the $(\lambda x.e_0')e_1 \rightsquigarrow e_0'[x := e_1]$ rule. So for this case, we conclude $e \rightsquigarrow e'$ for an $e' = e_0'[x := e_1]$.

  If e_0 is not a value, then by the induction hypothesis we know that there exists some e_0' such that $e \rightsquigarrow_0 e_0'$. We may then apply the $e_0 e_1 \rightsquigarrow e_0' e_1$ rule. For this case, we conclude $e \rightsquigarrow e'$ for an $e' = e_0' e_1$.

- Case
$$\frac{\Gamma \vdash e : \{\rho\} \qquad \rho \mid l : \tau}{\Gamma \vdash e\#l : \tau}$$

  We know that $\cdot \vdash e_0 : \{\rho\}$ and that $\rho \mid \ell : \tau$.

  If e_0 is a value, then by lemma 1, e_0 is some record $\{r\}$. By lemma 3, $r \mid \ell \hookrightarrow e_0'$. We may then apply the $\{r\}\#\ell \rightsquigarrow e_0'$ rule. For this case, we conclude $e \rightsquigarrow e_0'$.

  If e_0 is not a value, then by the induction hypothesis we know that there exists some e_0' such that $e \rightsquigarrow_0 e_0'$. We may then apply the $e_0\#\ell \rightsquigarrow e_0'\#\ell$ rule. For this case, we conclude $e \rightsquigarrow e'$ for an $e' = e_0'\#\ell$.

- Case
$$\frac{\Gamma \vdash e_0 : [\![\rho_0]\!] \quad \Gamma \vdash e_1 : \{\rho_1\} \quad \rho_0 \sqsubseteq \rho_1 : \tau}{\Gamma \vdash \mathtt{case}\, \mathtt{e_0}\, \mathtt{e_1} : \tau}$$

  We know $\cdot \vdash e_0 : [\![\rho_0]\!]$, $\cdot \vdash e_1 : \{\rho_1\}$, and $\rho_0 \sqsubseteq \rho_1 : \tau$.

  If either e_0 or e_1 is not a value, we may proceed as in the non-value cases of the application or projection parts of the proof by using the induction hypothesis and the corresponding step rule.

  If both e_0 and e_1 are values, we conclude from lemma 1 that $e_0 = [\![\ell : e_0']\!]$ and $e_1 = \{r\}$. Examining the rules for case constraint, it must be the case that $\rho_1 \mid \ell : \tau_0 \to \tau$ because ρ_0 is nonempty (this is trivially true because e_0 is a value with type $[\![\rho_0]\!]$). The only way for ρ_1 to not have label ℓ would be for ρ_0 to not have label ℓ. By lemma 3, it must be the case that $r \mid \ell \hookrightarrow e_1'$. Therefore, we may apply rule $\mathtt{case}\, [\![\ell : \mathtt{e_0'}]\!]\, \{\mathtt{r}\} \rightsquigarrow \mathtt{e_1' e_0'}$ and this case is complete.

These cases cover all of the non-values. Since the values constitute the base cases of the structural induction, and the inductive hypothesis holds, the proof is complete.

□

Lemma 2. *Inversion*
The following hold:

1. *If* $\Gamma \vdash e_0\, e_1 : \tau$ *then* $\Gamma \vdash e_0 : \tau_1 \to \tau$ *and* $\Gamma \vdash e_1 : \tau_1$ *for some type* τ_1.
2. *If* $\Gamma \vdash \lambda x.e : \tau$ *then* $\tau = (\tau_1 \to \tau_2)$ *for some types* τ_1 *and* τ_2, *and* $\Gamma, x : \tau_1 \vdash e : \tau_2$.
3. *If* $\Gamma \vdash e\#\ell : \tau$, *then* $\Gamma \vdash e : \{\rho\}$ *and* $\rho \mid \ell : \tau$.
4. *If* $\Gamma \vdash \mathtt{case\ e\ }_0\mathtt{e}_1 : \tau$, *then* $\Gamma \vdash e_0 : [\![\rho_0]\!]$, $\Gamma \vdash e_1 : \{\rho_1\}$, *and* $\rho_0 \sqsubseteq \rho_1 : \tau$.
5. *If* $\Gamma \vdash [\![\ell : e]\!] : [\![\rho]\!]$, *then* $\rho \mid \ell : t$ *and* $\Gamma \vdash e : \tau$.
6. *If* $\Gamma \vdash \lambda x.e : \tau_0 \to \tau$, *then* $\Gamma, x : \tau_0 \vdash e : \tau$.
7. *Other facts as needed...*

Proof. By inspection of the inference rules. (I.e., if we have a complete typing proof, and there is only one possible typing rule that can give us the corresponding conclusion, then the proofs of the premises of that rule must be contained within our assumed typing proof.)

One caveat: the variant case can either come from a deault variant derivation or a regular variant derivation. The latter follows immediately from the rules. The former follows from the fact that if a label is in the variant term but not in its type, its expression must have the default type.

□

Lemma 3. *Row Type-Term Correspondance*

1. *If* $\Gamma \vdash \{r\} : \{\rho\}$, *then* $\rho \mid \ell : \tau$ *if and only if* $r \mid \ell \hookrightarrow e$ *and* $\Gamma \vdash e : \tau$.
2. *If* $\Gamma \vdash [\![\ell : e]\!] : [\![\rho]\!]$, *then* $\rho \mid \ell : \tau$ *and* $\Gamma \vdash e : \tau$.

Proof. The proofs follow from the respective rules.

1. ($\Rightarrow$) Suppose that $\rho \mid \ell : \tau$. There are two cases.
   - ℓ is in ρ. Then for $\{r\}$ to have type $\{\rho\}$, it must contain ℓ and furthermore e must have type τ. This follows from the fact that only one rule could result in $\{r\}$ having type $\{\rho\}$.
   - ℓ is not in ρ. Then for $\{r\}$ to have type $\{\rho\}$, it must contain a default and furthermore the e corresponding to this default must have type τ. This again follows from the fact that there is one rule that could result in $\{r\}$ having type $\{\rho\}$.

($\Leftarrow$) This follows in a similar manner as the forward direction, except with the logic reversed (observe that the term and type level lookups are almost identical; they just deal with different objects).

2. This follows from lemma 2.

□

Theorem 3 (Preservation)**.** *If* $\cdot \vdash e : \tau$ *and* $e \rightsquigarrow e'$ *then* $\cdot \vdash e' : \tau$*.*

Proof. By (structural) induction on the term e and typing derivation $\cdot \vdash e : \tau$. We consider all of the possible forms that e can take.

- Case $e_0 e_1 \rightsquigarrow e_0' e_1$. Since $\cdot \vdash e_0 e_1 : \tau$, lemma 2 means that $\cdot \vdash e_0 : \tau_0 \to \tau$ and $\cdot \vdash e_1 : \tau_0$.

  By the induction hypothesis, $\cdot \vdash e_0' : \tau_0 \to \tau$. This allows us to use the application typing rule to derive that $\cdot \vdash e_0' e_1 : \tau$, as desired.

- Case $(\lambda x.e')e_1 \rightsquigarrow e'[x := e_1]$. Like the above, since $\cdot \vdash (\lambda x.e')e_1 : \tau$, lemma 2 means that $\cdot \vdash (\lambda x.e') : \tau_0 \to \tau$ and $\cdot \vdash e_1 : \tau_0$. We may apply lemma 2 again to conclude that $x : \tau_0 \vdash e' : \tau$. Lemma 4 gives that $\cdot \vdash e'[x := e_1] : \tau$, as desired.

- All other cases involving steps in subparts of the equation have proofs following the same direction (appeal to lemma 2 followed by application of the induction hypothesis, then an amendment to the derivation of $\cdot \vdash e : \tau$ that results in $\cdot \vdash e' : \tau$).

- Case $\{r\}\#\ell \rightsquigarrow e_\ell$. By lemma 2, we conclude that $\cdot \vdash e : \{\rho\}$ and $\rho \mid \ell : \tau$. From this, we may use lemma 3 to conclude that $\cdot \vdash e_\ell : \tau$, as desired.

- Case $\texttt{case}\,[\![\ell : \texttt{e}_\ell]\!]\,\{\texttt{r}\} \rightsquigarrow \texttt{e}_\texttt{r}\texttt{e}_\ell$. By lemma 2, we conclude that $\cdot \vdash [\![\ell : e_\ell]\!] : [\![\rho_0]\!]$, $\cdot \vdash \{r\} : \{\rho_1\}$, and $\rho_0 \sqsubseteq \rho_1 : \tau$. By lemma 3, we know that $\cdot \vdash e_\ell : \tau_\ell$ and $\rho_0 \mid \ell : \tau_\ell$. Because $\rho_0 \sqsubseteq \rho_1 : \tau$, it must be that $\rho_1 \mid \ell : \tau_\ell \to \tau$. By lemma 3, we know that $r \mid \ell \hookrightarrow e_r$ and $\cdot \vdash e_r : \tau_\ell \to \tau$. The types of e_ℓ and e_r allow us to derive $\cdot \vdash e_\ell e_r : \tau$, as desired.

□

Lemma 4 (Substitution)**.** *If* $\Gamma_0, x : \tau_x, \Gamma_1 \vdash e : \tau$ *and* $\Gamma_0, \Gamma_1 \vdash e' : \tau_x$*, then* $\Gamma_0, \Gamma_1 \vdash e[x := e'] : \tau$*.*

Proof. We will modify the typing derivation $\Gamma_0, x : \tau_x, \Gamma_1 \vdash e : \tau$ so that it becomes $\Gamma_0, \Gamma_1 \vdash e[x := e'] : \tau$. This modification can happen recursively, and in most cases all it requires is recursive traversals or empty base cases. The only missing case is that for variables.

$$\frac{x : \tau_x \in \Gamma}{\Gamma \vdash x : \tau_x}$$

Since the substitution will always replace x with e', we may modify this case to simply be the proof $\Gamma \vdash e' : \tau_x$, which is a premise. Since this is the only place where $x : \tau_x$ is used in the context, it's safe to be removed from the updated context.

$$\frac{y : \tau \in \Gamma}{\Gamma \vdash y : \tau} \ (y \neq x)$$

In this case, nothing needs to happen since no substitution will happen in y. And since $y : \tau$ is not $x : \tau_x$, it will not be removed from the context of the resulting derivation. □

Theorem 4 (Soundness)**.** *If $\cdot \vdash e : \tau$ then either $e \rightsquigarrow^* v$ for some result value v (with $\cdot \vdash v : \tau$), or there is an infinite sequence $e \rightsquigarrow e_1 \rightsquigarrow e_2 \rightsquigarrow \cdots$.*

Proof. Direct consequence of Preservation and Progress. □