1 Grammar

\[ e ::= x \quad \text{Variables} \]
\[ | \quad n \quad \text{Integers} \]
\[ | \quad \lambda x : \tau.e \quad \text{Functions} \]
\[ | \quad e \ e \quad \text{Application} \]

\[ v ::= n \quad \text{Integers} \]
\[ | \quad \lambda x : \tau.e \quad \text{Functions} \]

\[ \tau ::= \text{Int} \quad \text{Integer Type} \]
\[ | \quad \tau \rightarrow \tau \quad \text{Function Type} \]

\[ \Gamma ::= \cdot \quad \text{Empty environment} \]
\[ | \quad \Gamma, x : \tau \quad \text{Extended environment} \]

Notice that the grammar \( e \) includes the same rules as the grammar for \( v \) (plus more rules for non-value expressions). Remember that values are irreducible expressions. In big-step semantics, they evaluate to themselves, and in small-step semantics, they do not step.

2 Static Semantics

\[ \Gamma \vdash e : \tau \]

\[ \Gamma \vdash n : \text{Int} \quad (\text{T-Num}) \]

\[ x \in \text{dom}(\Gamma) \]
\[ \Gamma \vdash x : \Gamma(x) \quad (\text{T-VAR}) \]

\[ \Gamma, x : \tau' \vdash e : \tau \]
\[ \Gamma \vdash \lambda x : \tau'.e : \tau' \rightarrow \tau \quad (\text{T-FUN}) \]

\[ \Gamma \vdash e_1 : \tau' \rightarrow \tau \quad \Gamma \vdash e_2 : \tau' \]
\[ \Gamma \vdash e_1 \ e_2 : \tau \quad (\text{T-APP}) \]

We also define variable lookup in an environment:
\[ \Gamma(x) = \tau \]
\[
(\Gamma, x : \tau)(x) = \tau \\
(\Gamma, x : \tau)(x_1) = \Gamma(x_1) \quad \text{where } x \neq x_1
\]

and environment appending:

\[ \Gamma \oplus \Gamma = \Gamma \]
\[
\Gamma_1 \oplus \cdot \quad = \Gamma_1 \\
\Gamma_1 \oplus \Gamma_2, x : \tau = (\Gamma_1 \oplus \Gamma_2), x : \tau
\]

3 Dynamic Semantics

\[ e \rightarrow e \]

\[ \begin{array}{c}
e_1 \rightarrow e'_1 \\
e_1 e_2 \rightarrow e'_1 e_2
\end{array} \quad \text{(L-STEP)} \]

\[ \begin{array}{c}
e_2 \rightarrow e'_2 \\
v_1 e_2 \rightarrow v_1 e'_2
\end{array} \quad \text{(R-STEP)} \]

\[ (\lambda x : \tau.e) v \rightarrow e[x \mapsto v] \quad (\beta) \]

We also define substitution:

\[ e[x \mapsto v] = e \]

\[
\begin{array}{ll}
x[x \mapsto v] &= v \\
x_1[x \mapsto v] &= x_1 \quad \text{where } x_1 \neq x \\
n[x \mapsto v] &= n \\
(\lambda x : \tau.e)[x \mapsto v] &= \lambda x : \tau.e \\
(\lambda x_1 : \tau.e)[x \mapsto v] &= \lambda x_1 : \tau.e[x \mapsto v] \quad \text{where } x_1 \neq x \\
(e_1 e_2)[x \mapsto v] &= (e_1[x \mapsto v])(e_2[x \mapsto v])
\end{array}
\]

This substitution will only work for closed values and the call-by-value lambda calculus. How would you change the substitution to be more general? How does that change the proof of lemma 2?
4 Progress

Lemma 1 (Progress). For all expressions e, if there exists a type \( \tau \) such that \( \cdot \vdash e : \tau \), then either:

1. \( e = v \), for some value \( v \),
2. there exists an expression \( e' \) such that \( e \rightarrow e' \).

Proof. Assume an arbitrary expression \( e \). We’ll proceed by induction on \( e \).
This induction has four cases, as there are four productions in our grammar for expressions:

Case 1. \( e = x \)
Our theorem has the hypothesis \( \cdot \vdash e : \tau \) for some particular type \( \tau \). However, \( x \) appears free in \( e \), and there is no rule for typing variables not in the environment. Therefore, we have a contradiction and \( e \) cannot be \( x \).

Case 2. \( e = n \)
Conclusion 1 holds trivially.

Case 3. \( e = \lambda x : \tau.e \)
Conclusion 1 holds trivially.

Case 4. \( e = e_1 e_2 \)
We will need our induction hypotheses. We have two, one for each expression subterm:

Induction Hypothesis (IH1). If there exists a \( \tau \) such that \( \cdot \vdash e_1 : \tau \), then either:

1. \( e_1 = v_1 \) for some \( v_1 \), or
2. there exists an \( e'_1 \) such that \( e_1 \rightarrow e'_1 \).

Induction Hypothesis (IH2). If there exists a \( \tau \) such that \( \cdot \vdash e_2 : \tau \), then either:

1. \( e_2 = v_2 \) for some \( v_2 \), or
2. there exists an \( e'_2 \) such that \( e_2 \rightarrow e'_2 \).
From our hypothesis $\vdash e : \tau$, we can invert the typing rule $T\text{-App}$, since there’s only one typing rule for application, to deduce that there exists some type $\tau'$ such that $\vdash e_1 : \tau' \to \tau$ and $\vdash e_2 : \tau'$. Thus, $e_1$ and $e_2$ each have a valid type, so we can discharge the antecedents of IH1 and IH2, which means we can deduce that $e_1$ and $e_2$ are either values or take a step. Let’s examine each case in turn.

**Case 4a.** $e_1$ takes a step (whether or not $e_2$ takes a step).

That means there exists an $e'_1$ such that $e_1 \rightarrow e'_1$. Using rule $L\text{-Step}$ from our dynamic semantics, we form the judgment $e_1 e_2 \rightarrow e'_1 e_2$. Thus we have discovered an $e'$ such that $e \rightarrow e'$, namely, $e' = e'_1 e_2$.

**Case 4b.** $e_1$ is a value $v_1$ and $e_2$ takes a step.

That means there exists an $e'_2$ such that $e_2 \rightarrow e'_2$. Using rule $R\text{-Step}$ from our dynamic semantics, we form the judgment $v_1 e_2 \rightarrow v_1 e'_2$. Thus we have discovered an $e'$ such that $e \rightarrow e'$, namely, $e' = v_1 e'_2$.

**Case 4c.** $e_1$ is a value $v_1$ and $e_2$ is a value $v_2$.

From $\vdash e_1 : \tau' \to \tau$, we can invert the typing rule $T\text{-Fun}$, since it is the only rule that forms judgments on function types, to deduce that $v_1$ must be a function $\lambda x : \tau'. e_b$. Since $e_2$ is a value $v_2$, we can apply rule $\beta$ to take a step and get $e' = e_b[x \mapsto v_2]$.

---

### 5 Substitution

**Lemma 2** (Substitution). For all expressions $e$, environments $\Gamma$, variables $x$, values $v$, and types $\tau$ and $\tau'$, if $\vdash v : \tau'$ and $\Gamma, x : \tau' \vdash e : \tau$, then $\Gamma \vdash e[x \mapsto v] : \tau$.

Here, notice that the value $v$ must be closed.

**Proof.** We’ll assume an arbitrary $e$ and induct on $e$, which again has four cases due to its syntactic structure. In each case, we’ll assume an arbitrary $\Gamma$, $x$, $v$, $\tau$, and $\tau'$.

**Case 1.** $e = x_1$

From our hypotheses, we have that $\vdash v : \tau'$ and $\Gamma, x : \tau' \vdash x_1 : \tau$. There are two possibilities:
Case 1a. \( x_1 = x \)

From our definition of substitution, \( x[x \mapsto v] = v \). From our hypothesis \( \Gamma, x : \tau' \vdash x_1 : \tau \) and typing rule T-VAR, we get that \( \tau' = \tau \), and so we can use that equality with our other hypothesis \( \cdot \vdash \tau' \) to get \( \cdot \vdash x[x \mapsto v] : \tau \).

Now we just need to apply the following lemma:

**Lemma** (Weakening). For all \( \Gamma_1, \Gamma_2, e, \tau \), if \( \text{dom}(\Gamma_1) \cap \text{dom}(\Gamma_2) = \emptyset \) and \( \Gamma_1 \vdash e : \tau \), then \( \Gamma_1 \oplus \Gamma_2 \vdash e : \tau \).

to get \( \Gamma \vdash x[x \mapsto v] : \tau \).

Case 1b. \( x_1 \neq x \)

From our definition of substitution, \( x_1[x \mapsto v] = x_1 \). Since \( \Gamma, x : \tau' \vdash x_1 : \tau \), but \( x_1 \neq x \), then \( x_1 \in \text{dom}(\Gamma) \) and \( \Gamma(x_1) = \tau \). That means \( \Gamma \vdash x_1 : \tau \), and so we can use that equality with our other hypothesis \( \cdot \vdash v : \tau' \) to get \( \cdot \vdash x_1[x \mapsto v] : \tau \).

Case 2. \( e = n \)

From our definition of substitution, \( n[x \mapsto v] = n \). From T-Num, we get \( \Gamma, x : \tau' \vdash n : \text{Int} \). (That is, Int is the only \( \tau \) that satisfies our hypothesis.) From there, we get \( \Gamma \vdash n[x \mapsto v] : \tau \).

Case 3. \( e = \lambda x_1 : \tau_1.e \)

From our hypotheses, we know there exists a \( \tau' \) such that \( \Gamma \vdash v : \tau' \) and \( \Gamma, x : \tau' \vdash \lambda x_1 : \tau_1.e : \tau \). We also know, by the form of the judgment produced by the typing rule T-Fun, that \( \tau = \tau_1 \rightarrow \tau_2 \) for some \( \tau_2 \). Like the variable case, we have two possibilities:

Case 3a. \( x_1 = x \)

From our definition of substitution, \( (\lambda x : \tau_1.e)[x \mapsto v] = \lambda x : \tau_1.e \). From our hypothesis \( \Gamma, x : \tau' \vdash \lambda x : \tau_1.e : \tau_1 \rightarrow \tau_2 \) and the typing rule T-Fun, we have \( \Gamma, x : \tau', x : \tau_1 \vdash e : \tau_2 \). We then use the following lemma:

**Lemma** (Contraction). For all \( \Gamma, e, \tau, x, \tau_1, \) and \( \tau_2 \), if \( \Gamma, x : \tau_1, x : \tau_2 \vdash e : \tau \), then \( \Gamma, x : \tau_2 \vdash e : \tau \).

to get \( \Gamma, x : \tau_1 \vdash e : \tau_2 \). From that judgment and T-Fun, we get \( \Gamma \vdash \lambda x : \tau_1.e : \tau_1 \rightarrow \tau_2 \), so \( \Gamma \vdash (\lambda x : \tau_1.e)[x \mapsto v] : \tau \).

---

1^Try proving this!
2^Try proving this!
Case 3b. \( x_1 \neq x \)

From our definition of substitution,

\[(\lambda x_1 : \tau_1.e)[x \mapsto v] = \lambda x_1 : \tau_1.(e[x \mapsto v]).\]

From our hypothesis \( \Gamma, x : \tau' \vdash \lambda x_1 : \tau_1.e : \tau_1 \rightarrow \tau_2 \) and the typing rule

\( \text{T-Fun} \), we have \( \Gamma, x : \tau', x_1 : \tau_1 \vdash e : \tau_2 \). We use the following lemma:

**Lemma** (Exchange). *For all \( \Gamma, e, \tau, x_1, x_2, \tau_1, \) and \( \tau_2 \), if \( x_1 \neq x_2 \) and \( \Gamma, x_1 : \tau_1, x_2 : \tau_2 \vdash e : \tau \), then \( \Gamma, x_2 : \tau_2, x_1 : \tau_1 \vdash e : \tau \).* \(^3\)

we reorder the environment to get \( \Gamma, x_1 : \tau_1, x : \tau' \vdash e : \tau_2 \).

We then apply our induction hypothesis for the body \( e \):

**Induction Hypothesis** (IH). *For all \( \Gamma, x, v, \) and \( \tau \), if there exists a \( \tau' \) such that \( \cdot \vdash v : \tau' \) and \( \Gamma, x : \tau' \vdash e : \tau \), then \( \Gamma \vdash e[x \mapsto v] : \tau \).

We then apply IH1 to the judgments \( \cdot \vdash v : \tau' \) and \( \Gamma, x : \tau' \vdash e_1 : \tau_1 \rightarrow \tau \) to deduce \( \Gamma \vdash e_1[x \mapsto v] : \tau_1 \rightarrow \tau \). From IH2, \( \cdot \vdash v : \tau' \), and \( \Gamma, x : \tau' \vdash e_2 : \tau_1 \), we can deduce \( \Gamma \vdash e_2[x \mapsto v] : \tau_1 \). Combining these two judgments using

\( \text{T-App} \), we produce the judgment \( \Gamma \vdash (e_1[x \mapsto v]) \ (e_2[x \mapsto v]) : \tau \), which means \( \Gamma \vdash (e_1 \ e_2)[x \mapsto v] : \tau \) due to our definition of substitution; namely, that

\[(e_1 \ e_2)[x \mapsto v] = (e_1[x \mapsto v]) \ (e_2[x \mapsto v]).\]

\(^3\)Try proving this!
6 Preservation

Lemma 6 (Preservation). For all $e, e'$, and $\tau$, if $\cdot \vdash e : \tau$ and $e \rightarrow e'$, then $\cdot \vdash e' : \tau$.

Alternative formulation:

Lemma 6 (Preservation). For all $e$ and $e'$, if $e \rightarrow e'$, then for all $\tau$, if $\cdot \vdash e : \tau$, then $\cdot \vdash e' : \tau$.

Notice that reordering the lemma definition this way allows us to talk about the step that $e$ takes to $e'$ without tying ourselves to a particular type $\tau$.

Proof. Assuming arbitrary $e$ and $e'$, we have the hypothesis $e \rightarrow e'$. We’ll proceed by induction on the derivation of $e \rightarrow e'$.

There are three cases according to our dynamic semantics:

Case 1. Rule L-Step

We have that $e = e_1 \ e_2$ and $e' = e'_1 \ e_2$. We assume an arbitrary $\tau$ and get the hypothesis $\cdot \vdash e : \tau$. From this hypothesis and the rule T-App, we have that for some type $\tau_1$, $\cdot \vdash e_1 : \tau_1 \rightarrow \tau$ and $\cdot \vdash e_2 : \tau_1$. We need the induction hypothesis for $e_1 \rightarrow e'_1$, namely:

Induction Hypothesis (IHe1). For all $\tau$, if $\cdot \vdash e_1 : \tau$, then $\cdot \vdash e'_1 : \tau$.

We have $\cdot \vdash e_1 : \tau_1 \rightarrow \tau$, so we can deduce $\cdot \vdash e'_1 : \tau_1 \rightarrow \tau$ by IHe1. Combining this with $\cdot \vdash e_2 : \tau_1$ and rule T-App, we get $\cdot \vdash e'_1 \ e_2 : \tau$, i.e., $\cdot \vdash e' : \tau$.

Case 2. Rule R-Step

We have that $e = v_1 \ e_2$ and $e' = v_1 \ e'_2$. We assume an arbitrary $\tau$ and get the hypothesis $\cdot \vdash e : \tau$. From this hypothesis and the rule T-App we have that for some type $\tau_1$, $\cdot \vdash v_1 : \tau_1 \rightarrow \tau$ and $\cdot \vdash e_2 : \tau_1$. We need the induction hypothesis for $e_2 \rightarrow e'_2$, namely:

Induction Hypothesis (IHe2). For all $\tau$, if $\cdot \vdash e_2 : \tau$, then $\cdot \vdash e'_2 : \tau$.

We have $\cdot \vdash e_2 : \tau_1$, so we can deduce $\cdot \vdash e'_2 : \tau_1$ by IHe2. Combining this with $\cdot \vdash v_1 : \tau_1 \rightarrow \tau$ and rule T-App, we get $\cdot \vdash v_1 \ e'_2 : \tau$, i.e., $\cdot \vdash e' : \tau$. 

7
Case 3. Rule \( \beta \).

We have that \( e = (\lambda x_1 : \tau_1.e_b) v_2 \) and \( e' = e_b[x_1 \mapsto v_2] \). Assuming an arbitrary \( \tau \) and given the hypothesis \( \cdot \vdash e : \tau \), from rule T-App we get that for some type \( \tau_1 \), \( \cdot \vdash \lambda x_1 : \tau_1.e_b : \tau_1 \rightarrow \tau \) and \( \cdot \vdash v_2 : \tau_1 \). From \( \cdot \vdash \lambda x_1 : \tau_1.e_b : \tau_1 \rightarrow \tau \) and rule T-Fun, we get \( \cdot, x_1 : \tau_1 \vdash e_b : \tau \). We can now apply Lemma 2 to \( \cdot, x_1 : \tau_1 \vdash e_b : \tau \) and \( \cdot \vdash v_2 : \tau_1 \) to get \( \cdot \vdash e_b \[ x_1 \mapsto v_2 \] : \tau \), that is, \( \cdot \vdash e' : \tau \). \( \blacksquare \)

7 Soundness

Theorem 1 (Soundness). For all \( e \) and \( \tau \), if \( \cdot \vdash e : \tau \) then either:

1. \( e \uparrow \) (i.e., there does not exist a value \( v \) such that \( e \rightarrow^* v \)), or
2. there exists a value \( v \) such that \( e \rightarrow^* v \) and \( \cdot \vdash v : \tau \).

Proof. Our proof will be by induction on the length of the reduction sequence for \( e \).

Case 1. The sequence is of length 0.

By lemma 1, since we have a well-typed term \( e \), the only possibility where the term does not step is when \( e \) is a value. Thus, \( e \) is the value \( v \) such that \( e \rightarrow^* v \), and it trivially has the correct type needed for conclusion 2.

Case 2. The sequence is of at least length 1.

Then the sequence is of some form: \( e \rightarrow e_1 \rightarrow \ldots \). By lemma 6, since \( \cdot \vdash e : \tau \) and \( e \rightarrow e_1 \), we know that \( \cdot \vdash e_1 : \tau \). By our inductive hypothesis, we know that \( e_1 \) either diverges, which means that \( e \) also diverges, or that \( e_1 \rightarrow^* v \), where \( \cdot \vdash v : \tau \). That means that \( e \rightarrow e_1 \rightarrow^* v \), or \( e \rightarrow^* v \). Thus, \( e \rightarrow^* v \) which has the same type \( \tau \), as needed for conclusion 2. \( \blacksquare \)

A Some Notes about Proofs

Proofs in first-order logic, which is what we’ll be using here, are all about syntax. We have terms, that is, objects in our logic (here, terms and types in our extended lambda calculus), and we have propositions, that is, statements about our terms that we deduce using inference rules. For example, the
rules given in our static and dynamic semantics above use inference rules to define judgments, \( e \rightarrow e' \) and \( \Gamma \vdash e : \tau \), which are propositions about our terms. I’ll use lowercase letters to refer to terms, and uppercase to refer to propositions.

There are additional inference rules available to us in first-order logic than those we’ve defined for our judgments in the previous sections. (For a full list, see sites like the Wikipedia page on natural deduction.) Here, we’ll focus on introduction and elimination rules: that is, how we introduce and eliminate logical connectives when needed. Logical connectives are how we construct propositions from propositions, as opposed to judgments which construct propositions from terms.

A simple example are the rules for conjunction:

\[
\begin{array}{c}
A \\
B \\
\hline
A \land B
\end{array} \\
(I-And)
\]

\[
\begin{array}{c}
A \land B \\
\hline
A
\end{array} \\
(E-AndL)
\]

\[
\begin{array}{c}
A \land B \\
\hline
B
\end{array} \\
(E-AndR)
\]

These describe the usual expectations about conjunction. That is, if we have proven two propositions, then we can prove their conjunction by I-And, and if we have proven the conjunction of two propositions, then by the elimination forms we can use either proposition on its own.

Here are the rules for disjunction (\( \lor \)), where the elimination rule is slightly more complicated:

\[
\begin{array}{c}
A \\
\hline
A \lor B
\end{array} \\
(I-OrL)
\]

\[
\begin{array}{c}
B \\
\hline
A \lor B
\end{array} \\
(I-OrR)
\]

\[
\begin{array}{c}
A \lor B \\
A \Rightarrow C \\
B \Rightarrow C \\
\hline
C
\end{array} \\
(E-Or)
\]
As we might expect, if we have proven a proposition, then we can also prove the disjunction of that proposition with another (I-OrL and I-OrR).

Unlike conjunction, we cannot eliminate a disjunction directly; instead, we must find a proposition that depends on each branch (that is, each branch of the disjunction implies the proposition), and then from the disjunction we can conclude the implied proposition holds. This rule, E-Or, is extremely useful when proving a proposition by cases. That is, if we have an arbitrary term $a$, and we can show that each possibility for $a$ implies the proposition $P$ we want to prove, then from the disjunction of those cases we have the proposition. (Proof by induction is similar, but where $a$ is defined inductively, and for a case that includes an inductive subterm $a'$, we can use $P[a \mapsto a']$ in our proof of that case.)

Speaking of implication, let’s look at the rules for implication:

\[
\begin{array}{c}
  A \\
  \vdash \\
  B \\
  \hline
  A \implies B
\end{array}
\] (I-Impl)

\[
\begin{array}{c}
  A \implies B \\
  A \\
  \hline
  B
\end{array}
\] (E-Impl)

That is, if we can deduce $B$ from $A$, then $A$ implies $B$. (We’ll also often say “if $A$, then $B$” instead of writing $A \implies B$.) Similarly, if we know that $A \implies B$ and we also know $A$, then we can deduce $B$. This latter rule is often referred to as modus ponens.

In first-order logic, we also have two quantifiers that are often used: $\exists x.A$ and $\forall x.A$. Like $\lambda x : \tau.e$, these bind a variable that may appear free in the body. Here, the “type” of the variable they bind are terms: that is, they talk about objects in our logic, not about propositions. In this section, I’ll use the letter $a$ to refer to an abstract, or arbitrary, term and $t$ to refer to a concrete, or specific, term. Here are the introduction and elimination rules for $\forall$:

\[
\begin{array}{c}
a \\
\vdash \\
A \\
\hline
\forall x.A[a \mapsto x]
\end{array}
\] (I-All)
\[
\begin{array}{c}
\forall x. A \quad t \\
\overline{A[x \mapsto t]} \\
\end{array}
\quad \text{(E-All)}
\]

That is, if we assume an abstract term \(a\) and from that, can show that a proposition \(A\) holds (where \(a\) can be free in \(A\)), then we know that for any concrete term \(t\), that \(A\) holds where \(a\) is replaced with \(t\).

Here are the introduction and elimination rules for \(\exists\):

\[
\begin{array}{c}
A[x \mapsto t] \\
\overline{\exists x. A} \\
\end{array}
\quad \text{(I-Ex)}
\]

\[
\begin{array}{c}
\exists x. A \quad a \land A[x \mapsto a] \implies C \\
\overline{C} \\
\end{array}
\quad \text{(E-Ex)}
\]

The introduction rule is straightforward: if we have a proposition that contains a concrete term \(t\), then we know that the proposition holds if we replace \(t\) with \(x\). The elimination rule, like that for E-Or, is a bit more complicated: if we know that there exists a term \(x\) such that \(A\) holds, and we have a proposition \(C\) that depends on \(A\) holding for some abstract term \(a\), then we can deduce \(C\). That is, since \(C\) doesn’t care what term \(A\) holds for (remember, \(a\) is abstract), we can arbitrarily choose the term that proves \(\exists x. A\).

Basically, these rules give us ways of working with “for all ...” and “there exists ...” in our theorems. If we want to prove a proposition \(A\) holds for all terms \(x\), then we just need to show that \(A[x \mapsto a]\) holds for an arbitrary term \(a\). If we already know or have shown that \(\forall x. A\) is true, i.e., \(A\) holds for all terms \(x\), then we know it holds for a concrete term \(t\), so we can deduce \(A[x \mapsto t]\). If we want to deduce \(\exists x. A\), then we just need that to prove \(A[x \mapsto t]\) for some concrete term \(t\). Similarly, if we know \(\exists x. A\), then we can assume the existence of some term \(a\) such that \(A\) holds when \(x\) is replaced with that particular term (that is, \(A[x \mapsto a]\) is true).