CMSC 245: Principles of Programming Languages
Type System Proofs

This document was prepared using \LaTeX{}, a programming language designed to typeset mathematical writing. If you do not yet know how to write \LaTeX{}, I strongly encourage you to learn. Many resources can be found online.

Here is the definition of the simply-typed \( \lambda \)-calculus (STLC):

\[
e ::= \ x \mid v \mid e_1 e_2 \mid e_1 + e_2 \quad x ::= \ \text{(named variables)}
\]

\[
v ::= \ \lambda x : \tau. e \mid n \quad n ::= \ \text{(integers)}
\]

\[
\tau ::= \ \text{Int} \mid \tau_1 \to \tau_2 \quad \Gamma ::= \ \emptyset \mid \Gamma, x : \tau
\]

\[
\begin{array}{c}
\Gamma \vdash e : \tau \\
\hline
x : \tau \in \Gamma \\
\hline
\Gamma \vdash x : \tau \quad \text{VAR} \\
\hline
\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1 \\
\hline
\Gamma \vdash e_1 e_2 : \tau_2 \quad \text{APP} \\
\hline
\Gamma \vdash \lambda x : \tau_1.e : \tau_1 \to \tau_2 \quad \text{ABS}
\end{array}
\]

\[
\begin{array}{c}
\Gamma \vdash n : \text{Int} \\
\hline
\Gamma \vdash e_1 : \text{Int} \quad \Gamma \vdash e_2 : \text{Int} \\
\hline
\Gamma \vdash e_1 + e_2 : \text{Int} \quad \text{PLUS}
\end{array}
\]

\[
\begin{array}{c}
e_1 \to e_1' \\
\hline
e_1 e_2 \to e_1' e_2 \quad \text{APP1}
\end{array}
\]

\[
\begin{array}{c}
e_2 \to e_2' \\
\hline
v_1 e_2 \to v_1 e_2' \quad \text{APP2}
\end{array}
\]

\[
\begin{array}{c}
(\lambda x : \tau. e) v \to e[x \mapsto v] \\
\hline
\beta
\end{array}
\]

\[
\begin{array}{c}
e_1 \to e_1' \\
\hline
e_1 + e_2 \to e_1' + e_2 \quad \text{PLUS1}
\end{array}
\]

\[
\begin{array}{c}
e_2 \to e_2' \\
\hline
v_1 + e_2 \to v_1 + e_2' \quad \text{PLUS2}
\end{array}
\]

\[
\begin{array}{c}
n_1 + n_2 \to \text{plus}(n_1, n_2) \\
\hline
\text{ADD}
\end{array}
\]

- The notation \( e[x \mapsto v] \) means to replace every occurrence of \( x \) in \( e \) with \( v \).
- The notation \( \text{plus}(n_1, n_2) \) indicates the sum of the numbers \( n_1 \) and \( n_2 \).
- We assume all variables in \( \Gamma \) to be pairwise distinct. Note that this can always be upheld by the use of \( \alpha \)-conversion.
- Expressions may be written with parentheses to aid in parsing; these parentheses are then ignored in this formalism.

**Theorem 1.** For all types \( \tau \), \( \tau \) mentions \( \text{Int} \).

**Proof.** By structural induction on \( \tau \).

**Case** \( \tau = \text{Int} \): We observe that \( \tau \) mentions \( \text{Int} \) and are done with this case.

**Case** \( \tau = \tau_1 \to \tau_2 \): Our induction hypothesis is that \( \tau_1 \) mentions \( \text{Int} \) and that \( \tau_2 \) mentions \( \text{Int} \). Thus \( \tau \) mentions \( \text{Int} \).

\[\square\]

**Theorem 2.** For all expressions \( e \), if \( \emptyset \vdash e : \tau \) for some type \( \tau \), then \( e \) contains either a \( \lambda \) or an integer \( n \).

**Proof.** By structural induction on \( e \).

**Case** \( e = x \): This case cannot happen. We know that \( \emptyset \vdash e : \tau \) and that \( e = x \). Thus \( \emptyset \vdash x : \tau \). This can be proved only by \( \text{VAR} \). However, that rule’s premise requires that \( x:\tau \in \emptyset \) (in our case), which is impossible.
Case $e = e_1 e_2$: Our induction hypothesis states that, if $\emptyset \vdash e_1 : \tau_3$ (for some $\tau_3$), then $e_1$ contains a $\lambda$ or an integer $n$. The induction hypothesis holds also for $e_2$. To use the induction hypothesis, we must show that $\emptyset \vdash e_1 : \tau_3$ for some $\tau_3$. We know $\emptyset \vdash e_1 e_2 : \tau$. This can happen only by APP. Thus, the premises for APP must also hold. Thus, we know $\emptyset \vdash e_1 : \tau_1 \to \tau_2$ for some $\tau_1$ and $\tau_2$. This satisfies the premise to the induction hypothesis, so we can conclude that $e_1$ contains either a $\lambda$ or an integer $n$. Accordingly, $e_1 e_2$ contains a $\lambda$ or an integer $n$, and so we are done with this case.

Case $e = e_1 + e_2$: Similar to the previous case. Note that we use PLUS, which contains as a premise that $e_1$ and $e_2$ are well typed.

Case $e = \lambda x : \tau_1 . e_1$: Clearly, $e$ contains a $\lambda$, so we are done.

Case $e = n$: Clearly, $e$ contains an integer $n$, so we are done.

\[\square\]

Lemma 3 (Weakening). For all contexts $\Gamma$ and $\Gamma'$, expressions $e$, and types $\tau$: if $\Gamma \vdash e : \tau$, then $\Gamma, \Gamma' \vdash e : \tau$.

A simplified version of this proof is assigned for homework.

Lemma 4 (Substitution). For all expressions $e_1$: For all contexts $\Gamma$ and $\Gamma'$, all expressions $e_2$, all types $\tau_1$ and $\tau_2$, and all variables $x$, if $\Gamma, x : \tau_2, \Gamma' \vdash e_1 : \tau_1$ and $\Gamma \vdash e_2 : \tau_2$, then $\Gamma, \Gamma' \vdash e_1[x \mapsto e_2] : \tau_1$.

Proof. By structural induction on $e_1$.

Case $e_1 = x$: We must prove $\Gamma, \Gamma' \vdash x : \tau_1$; that is, we must prove $\Gamma, \Gamma' \vdash e_2 : \tau_1$. We know $\Gamma, x : \tau_2, \Gamma' \vdash x : \tau_1$. This can be proved only by VAR. Thus, we must have $x : \tau_1 \in \Gamma, x : \tau_2, \Gamma'$. But by the assumption that all elements in a context are pairwise distinct, we know that $x$ cannot occur in $\Gamma$ or $\Gamma'$. Thus, the $x : \tau_2$ binding is the only binding for $x$ in $\Gamma, x : \tau_2, \Gamma'$. Since we know $x : \tau_1 \in \Gamma, x : \tau_2, \Gamma'$, it must be that $\tau_1 = \tau_2$. Thus, we can conclude that $\Gamma \vdash e_2 : \tau_1$. By the Weakening Lemma (Lemma 3), we can further conclude that $\Gamma, \Gamma' \vdash e_2 : \tau_1$, just as we wanted to show.

Case $e_1 = x'$ (where $x \neq x'$): We must prove $\Gamma, \Gamma' \vdash x'[x \mapsto e_2] : \tau_1$; that is, we must prove $\Gamma, \Gamma' \vdash e_2 : \tau_1$. We have assumed $\Gamma, x : \tau_2, \Gamma' \vdash x' : \tau_1$. Since $x \neq x'$, then the binding for $x'$ must be in either $\Gamma$ or $\Gamma'$; that is, either $x' : \tau_1 \in \Gamma$ or $x' : \tau_1 \in \Gamma'$. Accordingly, $\Gamma, \Gamma' \vdash x' : \tau_1$ as desired.

Case $e_1 = e_3 e_4$: Our induction hypothesis tells us that, if $\Gamma, x : \tau_2, \Gamma' \vdash e_3 : \tau_3$ (for some $\tau_3$), then $\Gamma, \Gamma' \vdash e_3[x \mapsto e_2] : \tau_3$ (and similarly for $e_4$). We have assumed $\Gamma, x : \tau_2, \Gamma' \vdash e_3 e_4 : \tau_1$. This can be true only by APP. Thus, the premises of APP must be true; we can conclude that $\Gamma, x : \tau_2, \Gamma' \vdash e_3 : \tau_4 \to \tau_1$ and $\Gamma, x : \tau_2, \Gamma' \vdash e_4 : \tau_4$ (for some $\tau_4$). We thus use the induction hypothesis on both $e_3$ and $e_4$ to conclude that $\Gamma, \Gamma' \vdash e_3[x \mapsto e_2] : \tau_4 \to \tau_1$ and $\Gamma, \Gamma' \vdash e_4[x \mapsto e_2] : \tau_4$. Thus, we can use APP to conclude $\Gamma, \Gamma' \vdash (e_3 e_4)[x \mapsto e_2] : \tau_1$ as desired.

Case $e_1 = e_3 + e_4$: This is similar to the previous case, using PLUS instead of APP.

Case $e_1 = \lambda x' : \tau_3 . e_3$: The induction hypothesis tells us that if $\Gamma, x : \tau_2, \Gamma' \vdash e_3 : \tau_4$ (for some $\tau_2$ and $\tau_4$), then $\Gamma, \Gamma' \vdash e_3[x \mapsto e_2] : \tau_4$.[1] We have assumed $\Gamma, x : \tau_2, \Gamma' \vdash \lambda x' : \tau_3 . e_3 : \tau_1$. This can be only by ABS. Thus, we can assume the premise of ABS: $\Gamma, x : \tau_2, \Gamma', x' : \tau_3 \vdash e_3 : \tau_4$ (and it must be that $\tau_1 = \tau_3 \to \tau_4$). That satisfies the premise of the induction hypothesis (choosing $\Gamma_2 = \Gamma'$, $x' : \tau_3$; remember that the induction hypothesis is universally quantified over this choice of $\Gamma_2$, so this is legal), and so we conclude $\Gamma, \Gamma' \vdash e_3[x \mapsto e_2] : \tau_4$. By ABS, we can thus conclude $\Gamma, \Gamma' \vdash (\lambda x' : \tau_3 . e_3)[x \mapsto e_2] : \tau_4$ as desired (remembering that $\tau_1 = \tau_3 \to \tau_4$).

[1] Why so I suddenly change $\Gamma'$ to $\Gamma_2$ here? Note the way the lemma is stated. It first quantifies universally over $e_1$, and only afterward over all the other variables. This means that the induction hypothesis is actually universally quantified over all the other variables involved. In the APP case, we don’t need the extra flexibility, except for the type. In the $\lambda$ case, though, we do need to be flexible around the context, and so I’ve written it that way.
Case $e_1 = n$: We must prove that $\Gamma, \Gamma' \vdash n[x \mapsto e_2] : \textbf{Int}$; that is, that $\Gamma, \Gamma' \vdash n : \textbf{Int}$. We have this by INT, and we are done.

\[ \square \]

**Theorem 5** (Preservation). For all contexts $\Gamma$, expressions $e$ and $e'$, and types $\tau$, if $\Gamma \vdash e : \tau$ and $e \rightarrow e'$, then $\Gamma \vdash e' : \tau$.

*Proof.* By structural induction on $\Gamma$.

Case $e = x$: This case is impossible, because there is no $e'$ such that $x \rightarrow e'$.

Case $e = e_1 e_2$: We have assumed that $\Gamma \vdash e_1 e_2 : \tau$. This can be true only by App. Thus, we know the premises of that rule must be true: $\Gamma \vdash e_1 : \tau' \rightarrow \tau$ (for some $\tau'$) and $\Gamma \vdash e_2 : \tau'$. The induction hypothesis tells us that, if $e_1 \rightarrow e_1'$ (for some $e_1'$), then $\Gamma \vdash e_1' : \tau' \rightarrow \tau$; it also says that if $e_2 \rightarrow e_2'$ (for some $e_2'$), then $\Gamma \vdash e_2' : \tau'$. We have also assumed that $e_1 e_2 \rightarrow e'$. This fact can be true in one of three ways; thus we have three cases to consider:

**Case App1:** In this case, we know $e' = e_1' e_2$ and $e_1 \rightarrow e_1'$. By the induction hypothesis, $\Gamma \vdash e_1' : \tau' \rightarrow \tau$. We can thus use App to prove that $\Gamma \vdash e_1' e_2 : \tau$ as desired.

**Case App2:** In this case, we know $e_1 = v_1$, $e' = v_1 e_2'$, and $e_2 \rightarrow e_2'$. By the induction hypothesis, $\Gamma \vdash e_2' : \tau'$. We can thus use App to prove that $\Gamma \vdash v_1 e_2' : \tau$ as desired.

**Case Beta:** In this case, we know $e_1 = \lambda x : \tau'. e_3$ and $e_2 = v$ (for some $e_3$ and $v$). We must prove that $\Gamma \vdash e_3[x \mapsto v] : \tau$. We showed above that $\Gamma \vdash \lambda x : \tau'. e_3 : \tau' \rightarrow \tau$. Thus must be by Abs. Thus, we may conclude the premise to Abs: $\Gamma, x : \tau' \vdash e_3 : \tau$. Now, we use the Substitution Lemma (Lemma 4), instantiating $\Gamma'$ in that lemma statement to be the empty context. The Substitution Lemma concludes that $\Gamma \vdash e_3[x \mapsto v] : \tau$, as desired.

Case $e = e_1 + e_2$: We have assumed that $\Gamma \vdash e_1 + e_2 : \tau$. This can be true only by PLUS. We can thus conclude that $\tau = \textbf{Int}$. We can also conclude the premises of PLUS: $\Gamma \vdash e_1 : \textbf{Int}$ and $\Gamma \vdash e_2 : \textbf{Int}$. The induction hypothesis tells us that, if $e_1 \rightarrow e_1'$ (for some $e_1'$) then $\Gamma \vdash e_1' : \textbf{Int}$, and similarly for $e_2$.

We have assumed that $e_1 + e_2 \rightarrow e'$. This fact can be true in one of three ways; thus we have three cases to consider:

**Case Plus1:** In this case, we know $e' = e_1' + e_2$ and that $e_1 \rightarrow e_1'$. We thus use the induction hypothesis to get $\Gamma \vdash e_1' : \textbf{Int}$. We can now use PLUS to conclude $\Gamma \vdash e_1' + e_2 : \textbf{Int}$ as desired.

**Case Plus2:** In this case, we know $e' = e_1 + e_2'$, $e_1 = v_1$, and $e_2 \rightarrow e_2'$. We thus use the induction hypothesis to get $\Gamma \vdash e_2' : \textbf{Int}$. We can now use PLUS to conclude $\Gamma \vdash v_1 + e_2' : \textbf{Int}$ as desired.

**Case Add:** In this case, we know $e_1 = n_1$, $e_2 = n_2$ and $e' = \text{plus}(n_1, n_2)$. Because $\text{plus}(n_1, n_2)$ is a number, we use INT to conclude $\Gamma \vdash \text{plus}(n_1, n_2) : \textbf{Int}$ as desired.

Case $e = \lambda x : \tau'. e_1$: Impossible, because there exists no $e'$ such that $e \rightarrow e'$.

Case $e = n$: Impossible, because there exists no $e'$ such that $e \rightarrow e'$.

\[ \square \]

**Theorem 6** (Progress). For all expressions $e$ and types $\tau$, if $\emptyset \vdash e : \tau$, then either $e$ is a value $v$ or there exists $e'$ such that $e \rightarrow e'$.

*Proof.* By structural induction on $e$.

Case $e = x$: We have assumed $\emptyset \vdash x : \tau$, but this is impossible, because the premise of VAR cannot be satisfied in an empty context.
Case $e = e_1 e_2$: We have assumed that $\emptyset \vdash e_1 e_2 : \tau$. This can be only by APP. Thus, we can conclude the premises of APP, that $\emptyset \vdash e_1 : \tau' \rightarrow \tau$ and $\emptyset \vdash e_2 : \tau'$ for some $\tau'$. The induction hypothesis says that if $e_1$ is well-typed in an empty context, then it is either a value or it steps to some $e'_1$. We use this induction hypothesis, giving us two possibilities:

Case $e_1$ is a value $v_1$: We now must use the induction hypothesis on $e_2$ (which we know is well-typed by the premise of APP), giving us two further possibilities:

Case $e_2$ is a value $v_2$: We know $e_1$ is a value, and furthermore that $\emptyset \vdash e_1 : \tau' \rightarrow \tau$. Values have two possibilities: they are either $\lambda$-expressions or integers. If $v_1$ were an integer, then it could be well-typed only by INT; this would be a contradiction with the fact that $v_1$'s type is $\tau' \rightarrow \tau$. Thus, $v_1$ must be a $\lambda$-expression. We can conclude then that $v_1 = \lambda x : \tau'. e_3$. Putting this all together, we see that our original expression $e = (\lambda x : \tau'. e_3) v_2$. Thus, BETA applies, and we say that $e' = e_3[x \mapsto v_2]$ and $e \rightarrow e'$.

Case $e_2 \rightarrow e'_2$: In this case, APP2 applies. We choose $e' = v_1 e'_2$ and we are done.

Case $e_1 \rightarrow e'_1$: In this case, APP1 applies. We choose $e' = e'_1 e_2$ and we are done.

Case $e = e_1 + e_2$: We have assumed that $\emptyset \vdash e_1 + e_2 : \tau$. This can be only by PLUS, and thus $\tau = \Int$. We can also conclude that $\emptyset \vdash e_1 : \Int$ and $\emptyset \vdash e_2 : \Int$. The induction hypothesis says that either $e_1$ is a value or $e_1 \rightarrow e'_1$; also, either $e_2$ is a value or $e_2 \rightarrow e'_2$. We invoke the induction hypothesis on $e_1$, giving us two cases:

Case $e_1$ is a value $v_1$: We now invoke the induction hypothesis on $e_2$, giving us two further cases:

Case $e_2$ is a value $v_2$: We have assumed that both $v_1$ and $v_2$ have type $\Int$. There are two forms for values: either $\lambda$-expressions or integers. All $\lambda$-expressions have a type mentioning $\rightarrow$; thus these values must be integers $n_1$ and $n_2$. Accordingly, ADD applies, and we choose $e' = \text{plus}(n_1, n_2)$ and we are done.

Case $e_2 \rightarrow e'_2$: In this case, PLUS2 applies. We choose $e' = v_1 + e'_2$ and we are done.

Case $e_1 \rightarrow e'_1$: In this case, PLUS1 applies. We choose $e' = e'_1 + e_2$ and we are done.

Case $e = v$: If $e$ is a value, we are trivially done.

\[ \square \]

Theorem 7 (Type safety). For any natural number $k$: For all expressions $e$ and types $\tau$, if $\emptyset \vdash e : \tau$, then there exists $e'$ such that $e \rightarrow^k e'$ (that is, $e$ steps to $e'$ in $k$ steps) and $\emptyset \vdash e' : \tau$, or there exists $v$ and $k'$ such that $e \rightarrow^{k'} v$ and $k' < k$.

More intuitively, this means that any well-typed term either steps forever or eventually reduces to a value.

Proof. By induction on $k$.

Case $k = 0$: Trivially true, as $e \rightarrow^0 e$.

Case $k = j + 1$: The induction hypothesis gives us either $e'$ such that $e \rightarrow^j e'$ with $\emptyset \vdash e' : \tau$ or a value $v$ and $e \rightarrow^j v$ where $j' < j$. We thus have two cases:

Case $e \rightarrow^j e'$: We invoke the Progress Theorem (Theorem 6), giving us two cases:

Case $e'$ is a value $v$: We thus have $e \rightarrow^j v$, and $j < k$ (because $k = j + 1$); we are done.

Case $e' \rightarrow v''$: In this case, we invoke the Preservation Theorem (Theorem 5) to get $\emptyset \vdash e'' : \tau$. We conclude that $e \rightarrow^k e''$ and are done.

Case $e \rightarrow^j v$ where $j' < j$: We can observe that $j' < k$ as well, and we are done.

\[ \square \]