1 Simple Types

We have the simple type assignment rules from previous classes:

\[
\begin{align*}
\Gamma(x) &= T \\
\Gamma \vdash x : T \\
\Gamma \vdash t_1 : T_2 \rightarrow T_1 \\
\Gamma \vdash t_2 : T_2 \\
\Gamma, x : T_1 \vdash t : T_2 \\
\Gamma \vdash \lambda x.t : T_1 \rightarrow T_2
\end{align*}
\]

We wish to prove that full \(\beta\)-reduction is terminating on every simply typable term. There are many, many different proofs of this in the literature. We will use a proof from “Proofs and Types” (PaT) by Jean-Yves Girard, which you can find linked from the “Other Materials” section of the course web page [1]. The core idea of this proof is attributed to William Tait. The proof is tricky, but very elegant. It can also be extended to much more complicated type systems for lambda calculus (though that is outside the scope of our class).

2 Some Notation

We will write \(\text{next}(t)\) for the set of all terms \(t'\) for which \(t \sim t'\). For example, \(\text{next}(((\lambda w.w) \ (\lambda x.x) \ ((\lambda y.y) \ (\lambda z.z))))\) is

\[\{(\lambda x.x) \ ((\lambda y.y) \ (\lambda z.z)), \ ((\lambda w.w) \ (\lambda x.x) \ (\lambda z.z))\}\]

Here, we will define a context to be a term \(t^*\) with a designated free variable \(\ast\), which may be instantiated in a capture-avoiding way by a term \(s\) using the notation \(t^*[s]\).

Also, if \(S\) is a set of terms, then we will allow ourselves to write a term which has \(S\) inserted for the hole of some context \(t^*\). For example, we may write \(\lambda x.\{x, (x \ x)\}\) (here \(t^* = \lambda x.\ast\)). The meaning of this notation is the set of terms \(\{t^*[s]|s \in S\}\).

So in the example: \(\{\lambda x.x, \lambda x. (x \ x)\}\). We will consider predicates the same as their extensions (the set of elements where the predicate is true).

3 Reducibility

Write \(SN\) for the set of all strongly normalizing terms. If \(SN(t)\), then we make use of a (finite) natural number \(\nu(t)\) bounding the lengths of the reduction sequences from \(t\). We write \(b\) for a base type (we assume we have at least one base type), and define reducibility (at \(T\)) of a term \(t\) of type \(T\) in context \(\Gamma\), with notation \(Red_T(t)\), by recursion on \(T\):

\[
t \in Red_b \iff t \in SN
\]

\[
t \in Red_{T,\rightarrow T'} \iff \forall t' \in Red_{T}. (t \ t') \in Red_{T'}
\]

Notice that this definition is indeed structurally recursive. We define the set \(Red_T\) recursively in terms of \(Red_{T'}\), where \(T'\) is a subexpression of \(T\). So this is a valid definition. We will sometimes omit the type subscript on \(Red\).
4 Critical Properties

A term is defined to be neutral iff it is of the form \((t t')\), or if it is a variable. We now prove several critical properties of reducibility at type \(T\). These correspond to (CR 1), (CR 2), and (CR 3), respectively, in the PaT proof.

- **R-SN.** \(\text{Red}_T(t) \Rightarrow \text{SN}(t)\).
- **R-Pres.** \(\text{Red}_T(t) \Rightarrow \text{Red}_T(\text{next}(t))\).
- **R-Prog.** If \(t\) is neutral, then \(\text{Red}_T(\text{next}(t)) \Rightarrow \text{Red}_T(t)\).

The first property says that every reducible term is strongly normalizing. The second says that if every term you can reach in one step of \(\beta\)-reduction from a reducible term is also reducible. The third says that for neutral terms, if everything you can reach in one step from \(t\) is reducible, then \(t\) is also reducible. We prove these properties simultaneously by structural induction on \(T\).

4.1 Critical properties for \(b\)

**R-SN** is immediate. **R-Pres** and **R-Prog** follow easily since we have \(\text{SN}(t) \Leftrightarrow \text{SN}(\text{next}(t))\).

4.2 Critical properties for \(\rightarrow\)

4.2.1 Proof of R-SN

Suppose \(t\) is reducible at type \(T \rightarrow T'\). Let \(x\) be a variable of type \(T\). By **R-Prog**, \(x\) is reducible, since \(x\) is neutral and \(\text{next}(x) = \emptyset\). By definition of \(\text{Red}\), \((t x)\) is reducible. So by **R-SN** at smaller type \(T'\), we also have \(\text{SN}(t x)\), which implies \(\text{SN}(t)\).

4.2.2 Proof of R-Pres

Consider an arbitrary \(t' \in \text{Red}_T\). By definition of \(\text{Red}\), we have \(\text{Red}(t t')\). We also have

\[
\text{next}(t t') \subset \text{next}(t t')
\]

By **R-Pres** at smaller type \(T'\), we obtain \(\text{Red}(\text{next}(t t'))\), which by (1) implies \(\text{Red}(\text{next}(t) t')\).

So we conclude this for all \(t' \in \text{Red}_T\). By the definition of \(\text{Red}\), this implies \(\text{Red}(\text{next}(t))\).

4.2.3 Proof of R-Prog

Suppose \(t\) is neutral. By assumption, we have

\[
\text{Red}_{T \rightarrow T'}(\text{next}(t))
\]

It suffices, by the definition of \(\text{Red}\), to show that for all \(t' \in \text{Red}_T\), \(\text{Red}_{T'}(t t')\). So consider arbitrary \(t' \in \text{Red}_T\). Since \(t\) is neutral, \((t t')\) cannot be a \(\beta\)-redex. Since \(\text{SN}(t')\) by **R-SN** at smaller type \(T\), we may reason by inner induction on the number \(\nu(t')\) to prove that for all \(t' \in \text{Red}_T\), we have \(\text{Red}(t t')\). By **R-Prog** at smaller type \(T'\), it
suffices to prove \( \text{Red}(\text{next}(t \ t')) \), since the term in question is neutral. The possibilities for reduction are summarized by:

\[
\text{next}(t \ t') \subset (\text{next}(t) \ t') \cup (t \ \text{next}(t'))
\]  

(3)

We have \( \text{Red}(\text{next}(t \ t')) \) from (2), by the definition of \( \text{Red} \). For reducibility of the second set, we use our inner induction hypothesis, together with \( \text{R-Pres} \) (which ensures that reducibility of \( t' \) implies reducibility of \( \text{next}(t') \)).

5 Preservation of Reducibility

We must show that reducibility is preserved by all term-constructing operations. For applications, this comes directly from the definition of reducibility. So we must just consider \( \lambda \)-abstractions.

**Lemma 1** If for all \( t' \in \text{Red}_T \), we have \( \left[ t' / x \right] t \in \text{Red}_{T'} \), then \( \lambda x . t \in \text{Red}_{T \rightarrow T'} \).

It suffices by the definition of \( \text{Red} \) to show \( \text{Red}_{T'} \left( (\lambda x . t) \ t_1 \right) \), for all \( t_1 \in \text{Red}_T \). We prove this by induction on \( \nu(t) + \nu(t_1) \), which exists by \( \text{R-SN} \) (and \( [x/x] t \) is reducible by hypothesis). By \( \text{R-Prog} \), it suffices to prove \( \text{Red}_{T'} \left( \text{next}( (\lambda u . t) \ t_1 ) \right) \), since the term in question is neutral. The possibilities for reduction are summarized by:

\[
\text{next}( (\lambda x . t) \ t_1 ) \subset (\lambda x . \text{next}(t)) \ t_1 \cup (\lambda x . t) \ \text{next}(t_1) \cup \{[t_1 / x] t \}
\]  

(4)

The first and second sets on the right hand side are reducible by the induction hypothesis. The third set is reducible by the hypothesis of the lemma.

6 Reducibility Theorem

We are in a position now to prove the main theorem:

**Theorem 1 (Reducibility)** Suppose \( \{ x_1 : T_1, \ldots, x_n : T_n \} \vdash t : T \), and consider arbitrary \( t_i \in \text{Red}_{T_i} \), for all \( i \in \{1, \ldots, n\} \). Let \( \sigma \) be the substitution \( \{ (x_1, t_1), \ldots, (x_n, t_n) \} \). Then \( \sigma t \in \text{Red}_T \).

The proof is by rule induction on the assumed typing derivation, with the following cases.

6.1 Base Case

The typing derivation looks like:

\[
\begin{align*}
\Gamma(x) = T \\
\Gamma \vdash x : T
\end{align*}
\]

We have to prove that \( \sigma x_i \in \text{Red}_{T_i} \), for an arbitrary \( i \in \{1, \ldots, n\} \). Of course, we have \( \sigma x_i = t_i \), so what we are trying to prove is equivalent to \( t_i \in \text{Red}_{T_i} \), which we are assuming.
6.2 Application Case

The typing derivation looks like:

\[
\Gamma \vdash t_1 : T_2 \rightarrow T_1 \quad \Gamma \vdash t_2 : T_2 \\
\Gamma \vdash t_1 \ t_2 : T_1
\]

We have to prove that \( \sigma(t_1 \ t_2) \in \text{Red}_{T_1} \). Our induction hypothesis tells us that \( \sigma(t_1) \in \text{Red}_{T_2 \rightarrow T_1} \) and \( \sigma(t_2) \in \text{Red}_{T_2} \). Directly from the definition of \( \text{Red} \), this implies \( \sigma(t_1 \ t_2) \in \text{Red}_{T_1} \), as required.

6.3 Lambda Case

The typing derivation looks like:

\[
\Gamma, x : T_1 \vdash t : T_2 \\
\Gamma \vdash \lambda x. t : T_1 \rightarrow T_2
\]

We have to prove that \( \sigma(\lambda x. t) \in \text{Red}_{T_1 \rightarrow T_2} \). Our induction hypothesis tells us that for all \( t' \in \text{Red}_{T_1} \), \( \sigma(t'/x)t \in \text{Red}_{T_2} \). Lemma 1 then directly applies (indeed, we proved it just for this case), to give us the desired conclusion.

7 Concluding Strong Normalization

Theorem 2 (Strong Normalization) Term reduction is strongly normalizing for well-typed terms.

This follows since all well-typed terms are reducible, and by \textbf{R-SN}, reducible terms are strongly normalizing.

Theorem 3 (Normal Forms) Well-typed terms in normal form are of the form \( S \) described by the following grammar:

\[
N ::= x \mid \lambda x. N \mid n \\
n ::= x \mid (n \ N)
\]

We cannot have a \( \lambda \)-abstraction applied to an argument, but all other terms are allowed.

References