Chapter 2

Untyped Lambda Calculus

We assume the existence of a denumerable set \( \text{VAR} \) of (object) variables \( x^0, x^1, x^2, \ldots \), and use \( x, y, z \) to range over these variables. Given two variables \( x_1 \) and \( x_2 \), we write \( x_1 = x_2 \) if both \( x_1 \) and \( x_2 \) denote the same \( x^n \) for some natural number \( n \); similarly, we write \( x_1 < x_2 \) \( (x_1 \leq x_2) \) if \( x_1 \) and \( x_2 \) denote \( x^{n_1} \) and \( x^{n_2} \), respectively, for some natural numbers \( n_1 \) and \( n_2 \) satisfying \( n_1 < n_2 \) \( (n_1 \leq n_2) \); we write \( x_1 > x_2 \) \( (x_1 \geq x_2) \) to mean \( x_2 < x_1 \) \( (x_2 \leq x_1) \).

**Definition 2.0.1 (\( \lambda \)-terms)** The (pure) \( \lambda \)-terms are formally defined below:

\[
\begin{align*}
t &::= x | \lambda x.t | (t_1)t_2
\end{align*}
\]

We use \( \text{TERM} \) for the set of all \( \lambda \)-terms. Given a \( \lambda \)-term \( t \), \( t \) is either a variable, or a \( \lambda \)-abstraction of the form \( \lambda x.t_1 \), or an application of the form \((t_1)t_2\). We write \( t_1 \equiv t_2 \) to mean that \( t_1 \) and \( t_2 \) are syntactically the same.

**Definition 2.0.2 (Size of \( \lambda \)-terms)** We define a unary function \( \text{size}(\cdot) \) to compute the size of a given \( \lambda \)-term:

\[
\begin{align*}
\text{size}(x) &= 0 \\
\text{size}(\lambda x.t) &= 1 + \text{size}(t) \\
\text{size}((t_1)t_2) &= 1 + \text{size}(t_1) + \text{size}(t_2)
\end{align*}
\]

There is often a need to refer to a subterm in a given \( \lambda \)-term. For this purpose, we introduce \( \text{paths} \) defined as finite sequences of natural numbers:

\[
\begin{align*}
\text{paths} \ p &::= \emptyset | n.p
\end{align*}
\]

We use \( \emptyset \) for the empty sequence and \( n.p \) for the sequence whose head and tail are \( n \) and \( p \), respectively, where \( n \) ranges over natural numbers. We use \( \text{PATH} \) for the set of all paths and \( \overline{p} \) to range over finite sets of paths. Given two paths \( p_1 \) and \( p_2 \), we write \( p_1 \oplus p_2 \) for the concatenation of \( p_1 \) and \( p_2 \). We say that \( p_1 \) is a prefix of \( p_2 \) if \( p_2 = p_1 \oplus p_3 \) for some path \( p_3 \); this prefix is proper if \( p_3 \) is not empty.

**Definition 2.0.3** We define as follows a binary partial function \( \text{subterm}(\cdot, \cdot) \) from \((\text{TERM}, \text{PATH})\) to \( \text{TERM} \):

\[
\begin{align*}
\text{subterm}(t, \emptyset) &= t \\
\text{subterm}((t_1)t_2, 0.p) &= \text{subterm}(t_1, p) \\
\text{subterm}((t_1)t_2, 1.p) &= \text{subterm}(t_2, p) \\
\text{subterm}(\lambda x.t, 0.p) &= \text{subterm}(t, p)
\end{align*}
\]
Given two \( \lambda \)-terms \( t_1, t_2 \) and a path \( p \), we say that \( t_1 \) is a subterm of \( t_2 \) at \( p \) if \( \text{subterm}(t_2, p) = t_1 \); this subterm is proper if \( p \) is not empty. We may simply say that \( t_1 \) is a subterm of \( t_2 \) if \( \text{subterm}(t_2, p) = t_1 \) for some path \( p \). Also, we may say that \( t_1 \) has an occurrence in \( t_2 \) (at \( p \)) if \( t_1 \) is a subterm of \( t_2 \) (at \( p \)). Note that for a \( \lambda \)-term of the form \( \lambda x.t \), the variable \( x \) following the binder \( \lambda \) does not count as an occurrence (in the formal sense).

Given a \( \lambda \)-term, we use \( \text{paths}(t) \) for the set of paths such that \( p \in \text{paths}(t) \) if and only if \( \text{subterm}(t, p) \) is defined. Clearly, for every \( \lambda \)-term \( t \), we have

- \( \emptyset \in \text{paths}(t) \), and
- \( p_0 \in \text{paths}(t) \) implies that \( p \in \text{paths}(t) \) holds for every prefix \( p \) of \( p_0 \).

It is also clear for every path \( p, p \in \text{paths}(t) \) implies \( p \) being a sequence of 0's and 1's.

**Definition 2.0.4** We define a function \( \text{vars} \) as follows that maps \( \lambda \)-terms to finite sets of variables:

\[
\begin{align*}
\text{vars}(x) & = \{x\} \\
\text{vars}(\lambda x.t) & = \text{vars}(t) \cup \{x\} \\
\text{vars}((t_1)t_2) & = \text{vars}(t_1) \cup \text{vars}(t_2)
\end{align*}
\]

Clearly, for every \( \lambda \)-term \( t_0, x \in \text{vars}(t_0) \) if and only if \( t_0 \) has a subterm of the form \( x \) or \( \lambda x.t \).

**Definition 2.0.5 (Free Variables)** We define a function \( \text{FV} \) as follows that maps \( \lambda \)-terms to finite sets of variables:

\[
\begin{align*}
\text{FV}(x) & = \{x\} \\
\text{FV}(\lambda x.t) & = \text{FV}(t) \setminus \{x\} \\
\text{FV}((t_1)t_2) & = \text{FV}(t_1) \cup \text{FV}(t_2)
\end{align*}
\]

Given a \( \lambda \)-term \( t \), we refer to \( \text{FV}(t) \) as the set of free variables in \( t \). We say that a variable \( x \) is free in \( t \) if and only if \( x \in \text{FV}(t) \) holds.

Given a \( \lambda \)-term \( t \) and a variable \( x \), an occurrence of \( x \) in \( t \) at \( p_0 \) is a free occurrence if \( \text{subterm}(t_0, p) \) is not of the form \( \lambda x.t \) for any prefix \( p \) of \( p_0 \). It is clear from the definition of \( \text{FV} \) that \( x \in \text{FV}(t) \) if and only if \( x \) has at least one free occurrence in \( t \).

**Definition 2.0.6 (Variable Replacement)** Given a \( \lambda \)-term \( t \) and two variables \( x \) and \( y \), we define \( t[y/x] \) as follows by structural induction on \( t \):

\[
\begin{align*}
x[y/x] & ::= y \\
x'[y/x] & ::= x' \text{ if } x' \text{ is not } x \\
(t_1)t_2[y/x] & ::= (t_1[y/x])t_2[y/x] \\
(\lambda x.t)[y/x] & ::= \lambda x.t \\
(\lambda x'.t)[y/x] & ::= \lambda x'.t[y/x] \text{ if } x \neq x'
\end{align*}
\]

We refer to \( t[y/x] \) as the \( \lambda \)-term obtained from replacing \( x \) with \( y \) in \( t \).

Clearly, \( \text{size}(t[y/x]) = \text{size}(t) \) for all \( \lambda \)-terms \( t \) and variables \( x \) and \( y \).
Proposition 2.0.7 Assume \( y \not\in \text{vars}(t) \). We have \( \text{FV}(t[y/x]) = \text{FV}(t) \) if \( x \not\in \text{FV}(t) \), or \( \text{FV}(t[y/x]) = (\text{FV}(t) \setminus \{x\}) \cup \{y\} \) if \( x \in \text{FV}(t) \).

Proof As an exercise.

Proposition 2.0.8 We have the following.

1. \( t[x/x] \equiv t \).
2. \( t[y/x] \equiv t \) if \( x \not\in \text{FV}(t) \).
3. \( t[y/x][z/y] \equiv t[z/x] \) if \( y \not\in \text{vars}(t) \).

Proof Both (1) and (2) are straightforward. We prove (3) by structural induction on \( t \).

- \( t \) is \( x \). Then both \( t[y/x][z/y] = z \) and \( t[z/x] = z \) hold, and we are done.
- \( t \) is \( x' \) for some variable \( x' \neq x \). Then \( x' \neq y \) also holds as \( y \not\in \text{vars}(t) \). So \( t[y/x][z/y] = x' \) and \( t[z/x] = x' \), and we are done.
- \( t \) is \( (t_1)_{t_2} \). For \( i = 1, 2 \), we have \( t_i[y/x][z/y] = t_i[z/x] \) by induction hypotheses on \( t_i \). Therefore, \( t[y/x][z/y] = t[z/x] \) holds as well.
- \( t \) is \( \lambda x.t_0 \). Then \( t[y/x][z/y] = t[z/y] \), and \( t[z/x] = t \). By (2), \( t[z/y] = t \) holds, and we are done.
- \( t \) is \( \lambda x'.t_0 \) for some \( x' \neq x \). We have \( t_0[y/x][z/y] = t_0[z/x] \) by induction hypothesis on \( t_0 \). Note that \( x' \neq y \) since \( y \not\in \text{vars}(t) \). So we have \( t[y/x][z/y] = t[z/x] \).

We conclude the proof as all the cases are covered.

Definition 2.0.9 We use \( \Gamma \) for a sequence of variables defined as follows:

\[
\Gamma ::= \cdot \mid \Gamma, x
\]

We write \( x \in \Gamma \) to indicate that \( x \) occurs in \( \Gamma \). If \( x \in \Gamma \) holds, we define \( \Gamma(x) \) as follows: \( \Gamma(x) = 1 \) if \( \Gamma = \Gamma_1, x \) and \( \Gamma(x) = 1 + \Gamma_1(x) \) if \( \Gamma = \Gamma_1, x_1 \) for some \( x_1 \neq x \).

Definition 2.0.10 (\( \alpha \)-normal forms) We use \( \mathcal{L} \) for \( \alpha \)-normal forms defined as follows:

\[
\alpha\text{-normal forms } t ::= x \mid n \mid \lambda t \mid (t_1)t_2
\]

where \( n \) ranges over positive natural numbers.

Definition 2.0.11 (\( \alpha \)-equivalence) Given a sequence \( \Gamma \) of variables and a term \( t \), \( \text{NF}_\alpha(\Gamma; t) \) is defined inductively as follows:

\[
\text{NF}_\alpha(\Gamma; t) = \begin{cases} 
  x & \text{if } t = x \text{ for some } x \not\in \Gamma; \\
  \Gamma(x) & \text{if } t = x \text{ for some } x \in \Gamma; \\
  \lambda \langle t_0 \rangle & \text{if } t = \lambda x.t_0 \text{ and } \langle t_0 \rangle = \text{NF}_\alpha(\Gamma, x; t_0); \\
  (t_1)t_2 & \text{if } t = (t_1)t_2 \text{ and } t_1 = \text{NF}_\alpha(\Gamma; t_1) \text{ and } t_2 = \text{NF}_\alpha(\Gamma; t_2).
\end{cases}
\]

We use \( \text{NF}_\alpha(t) \) as a shorthand for \( \text{NF}_\alpha(\cdot; t) \). Given two terms \( t_1 \) and \( t_2 \), we say that \( t_1 \) and \( t_2 \) are \( \alpha \)-equivalent if \( \text{NF}_\alpha(t_1) = \text{NF}_\alpha(t_2) \) holds, and we use \( t_1 \equiv_\alpha t_2 \) to indicate that \( t_1 \) and \( t_2 \) are \( \alpha \)-equivalent. Clearly, \( \equiv_\alpha \) is an equivalence relation.
Proposition 2.0.12 Assume \( t \equiv_\alpha t' \). Then we have the following.

1. \( \text{FV}(t) = \text{FV}(t') \).
2. If \( y \notin \text{vars}(t) \cup \text{vars}(t') \), then \( t[y/x] \equiv_\alpha t'[y/x] \) for any variable \( x \).

Proof As an exercise.

Definition 2.0.13 (Substitutions) We use \( \theta \) for substitutions, which are finite mappings from variables to \( \lambda \)-terms:

\[
\text{substitutions} \quad \theta ::= [] \mid \theta[x \mapsto t]
\]

We use \( \text{dom}(\theta) \) for the (finite) domain of \( \theta \) and \( \text{vars}(\theta) \) for the following (finite) set of variables:

\[
\text{dom}(\theta) \cup (\cup_{x \in \text{dom}(\theta)} \text{vars}(\theta(x)))
\]

Definition 2.0.14 Given a \( \lambda \)-term \( t \) and a substitution \( \theta \), we use \( t[\theta] \) for the result of applying the substitution \( \theta \) to \( t \), which is formally defined below by induction on the size of \( t \):

- \( x[\theta] = \theta(x) \) if \( x \in \text{dom}(\theta) \).
- \( x[\theta] = x \) if \( x \notin \text{dom}(\theta) \).
- \( (\lambda x.t)[\theta] = \lambda y.(t[y/x])[\theta] \), where \( y \) is the first variable not in \( \text{vars}(t) \cup \text{vars}(\theta) \).
- \( ((t_1)t_2)[\theta] = (t_1[\theta])t_2[\theta] \).

Lemma 2.0.15 If \( x \notin \text{vars}(\theta) \) and \( y \notin \text{vars}(t) \cup \text{vars}(\theta) \), then \( (t[\theta])[y/x] \equiv_\alpha (t[y/x])[\theta] \).

Given two substitutions \( \theta_1 \) and \( \theta_2 \), we write \( \theta_1 \equiv_\alpha \theta_2 \) to mean that \( \theta_1(x) \equiv_\alpha \theta_2(x) \) holds for every \( x \in \text{dom}(\theta_1) = \text{dom}(\theta_2) \).

Lemma 2.0.16 If \( \theta \equiv_\alpha \theta' \) and \( t \equiv_\alpha t' \), then \( t[\theta] \equiv_\alpha t'[\theta'] \)

Proof As an exercise.

Lemma 2.0.17 \( (t[\theta_1])[\theta_2] \equiv_\alpha t[\theta_2 \circ \theta_1] \).

Proof As an exercise.

Proposition 2.0.18 \( \lambda x.t \equiv_\alpha \lambda y.t[x := y] \) if \( y \notin \text{FV}(t) \).

Proof As an exercise.

Given a \( \lambda \)-abstraction \( \lambda x.t \) and a finite set of variables, we can also choose another \( \lambda \)-abstraction \( \lambda x'.t' \) that is \( \alpha \)-equivalent to \( \lambda x.t \) while guaranteeing that \( x' \) does not occur in the given finite set of variables. By Proposition 2.0.18, \( x' \) can be chosen to be any variable that is not in \( \text{FV}(\lambda x.t) \). This is often called \( \alpha \)-conversion or \( \alpha \)-renaming (of a bound variable).

Definition 2.0.19 (\( \beta \)-redexes) A \( \lambda \)-term \( t \) is a \( \beta \)-redex if it is of the form \( (\lambda x.t_1)t_2 \), and its contractum is \( t_1[x := t_2] \). We may also refer to the contractum of a \( \beta \)-redex as the reduct of the \( \beta \)-redex.
Given a \( \lambda \)-term \( t \), \( \mathcal{R} \) is a set of \( \beta \)-redexes in \( t \) if \( \mathcal{R} \) a finite set of paths such that \( \text{subterm}(t, p) \) is a \( \beta \)-redex for each \( p \in \mathcal{R} \).

**Definition 2.0.20 (\( \lambda \)-terms and \( \beta \)-redexes)** A \( \lambda \)-term \( t_0 \) is a \( \lambda \)-term if for every subterm \( t \) of \( t_0 \), \( t \) being of the form \( \lambda x.t_1 \) implies \( x \in \text{FV}(t_1) \). Moreover, a \( \beta \)-redex \( (\lambda x.t_1)t_2 \) is a \( \beta \)-redex if \( x \in \text{FV}(t_1) \).

**Definition 2.0.21 (\( \lambda \)-term Contexts)**

\[
\text{contexts} \quad C \ ::= \ [] \mid \lambda x.C \mid (C)t \mid (t)C
\]

Given a context \( C \) and a \( \lambda \)-term \( t \), we use \( C[t] \) for the \( \lambda \)-term obtained from replacing the hole \( [] \) in \( C \), which is formally defined below:

\[
C[t] = \begin{cases} 
  t & \text{if } C \text{ is } []; \\
  \lambda x.(C_0[t]) & \text{if } C \text{ is } \lambda x.C_0; \\
  (C_1[t])t_2 & \text{if } C \text{ is } (C_1)t_2; \\
  (t_1)(C_2[t]) & \text{if } C \text{ is } (t_1)C_2.
\end{cases}
\]

Given a context \( C \) and a path \( p \), we use \( \text{subterm}(C, p) \) for either a context or a term defined below:

\[
\begin{align*}
  \text{subterm}(C, \emptyset) &= C \\
  \text{subterm}((C)t, 0, p) &= \text{subterm}(C, p) \\
  \text{subterm}((t)C, 0, p) &= \text{subterm}(t, p) \\
  \text{subterm}((C)t, 1, p) &= \text{subterm}(t, p) \\
  \text{subterm}((t)C, 1, p) &= \text{subterm}(C, p) \\
  \text{subterm}(\lambda x.C, 0, p) &= \text{subterm}(C, p)
\end{align*}
\]

**Definition 2.0.22 (\( \beta \)-reduction)** Given two \( \lambda \)-terms \( t_1, t_2 \) and a path \( p \), we write \( [p] : t_1 \rightarrow^{\beta} t_2 \) if \( t_1 = C[t] \) for some context \( C \) and \( \beta \)-redex \( t \), where \( \text{subterm}(C, p) = [], \) and \( t_2 = C[t'] \) for the reduct \( t' \) of \( t \). We may also write \( t_1 \rightarrow^{\beta} t_2 \) to mean \( [p] : t_1 \rightarrow^{\beta} t_2 \) for some \( p \).

We refer to the binary relation \( \rightarrow^{\beta} \) as (one-step) \( \beta \)-reduction, and use \( \rightarrow^{\beta+} \) and \( \rightarrow^{\beta*} \) for the transitive closure and the reflexive and transitive closure of \( \rightarrow^{\beta} \), respectively. We may also refer to \( \rightarrow^{\beta*} \) as multi-step \( \beta \)-reduction.

**Definition 2.0.23 (\( \beta \)-reduction Sequences)** We use \( \sigma \) for \( \beta \)-reduction sequences defined below:

\[
\beta \text{-reduction sequences} \quad \sigma \ ::= \ [] \mid [p] + \sigma
\]

where \( \emptyset \) stands for the empty \( \beta \)-reduction sequence.

Let \( \sigma \) be a \( \beta \)-reduction sequence of length \( n \), that is, \( \sigma \) is of the form \( [p_1] + \ldots + [p_n] + \emptyset \). We say that \( \sigma \) is a \( \beta \)-reduction sequence from a \( \lambda \)-term \( t \) if \( [p_i] : t_i \rightarrow^{\beta} t_{i+1} \) holds for each \( 1 \leq i \leq n \) and \( t = t_1 \), and we use \( t/\sigma \) for \( t_{n+1} \).

**Proposition 2.0.24** Assume that \( \sigma \) is a \( \beta \)-reduction sequence from \( t_1 \). Then for every variable \( x \) and \( \lambda \)-term \( t_2 \), \( \sigma \) is also a \( \beta \)-reduction sequence from \( t_1[x := t_2] \), and \( \sigma(t_1[x := t_2]) = (\sigma(t_1))[x := t_2] \) if \( \sigma \) is finite.
Proof It is straightforward to verify that \([p] : t_1 \rightarrow_\beta t'_1\) implies \([p] : t_1[x := t_2] \rightarrow_\beta t'_1[x := t_2]\). Then the proposition follows from an induction on the length of \(\sigma\).

It is clear from Proposition \(2.0.24\) that for every \(\lambda\)-term \(t\), a \(\beta\)-reduction sequence from \(t\) can also be viewed as a \(\beta\)-reduction sequence from \(t[x := t']\) for any \(x\) and \(t'\).

Definition 2.0.25 (*Residual of a \(\beta\)-reduction*)

Clearly, the residuals of a \(\beta\)-redex are \(\beta\)-redexes themselves.

2.1 Developments

Definition 2.1.1 (Developments) Assume that \(t\) is a \(\lambda\)-term and \(\mathcal{R}\) is a set of \(\beta\)-redexes in \(t\). A \(\beta\)-reduction sequence \(\sigma\) is from \(\langle t, \mathcal{R} \rangle\) is called a development if \(\sigma\) is empty, or \(\sigma = [p] + \sigma_1\) for some \(p \in \mathcal{R}\) and \([p]\) reduces \(\langle t, \mathcal{R} \rangle\) to \(\langle t_1, \mathcal{R}_1 \rangle\) and \(\sigma_1\) is a development of \(\langle t_1, \mathcal{R}_1 \rangle\).

A finite development of \(\langle t, \mathcal{R} \rangle\) is complete if \(\sigma(\langle t, \mathcal{R} \rangle) = \langle t', \emptyset \rangle\) for some \(t'\).

Lemma 2.1.2 Let \(P\) be a unary predicate on marked \(\lambda\)-terms. Assume that \(P\) is modulo \(\alpha\)-equivalence, that is, \(P(\langle t_1, \mathcal{R} \rangle)\) implies \(P(\langle t_2, \mathcal{R} \rangle)\) whenever \(t_1 \equiv_\alpha t_2\) holds, and

1. \(P(\langle x, \emptyset \rangle)\) holds for every variable \(x\).
2. For every marked \(\lambda\)-term \(\langle t, \mathcal{R} \rangle\), \(P(\langle t, \mathcal{R} \rangle)\) implies \(P(\langle \lambda x.t, 0.\mathcal{R} \rangle)\).
3. For every pair of marked \(\lambda\)-terms \(\langle t_1, \mathcal{R}_1 \rangle, \langle t_2, \mathcal{R}_2 \rangle\) and \(\mathcal{R} = 0.\mathcal{R}_1 \cup 1.\mathcal{R}_2\), \(P(\langle t_1, \mathcal{R}_1 \rangle)\) and \(P(\langle t_2, \mathcal{R}_2 \rangle)\) implies \(P(\langle (t_1)t_2, \mathcal{R} \rangle)\).
4. For every pair of \(\lambda\)-terms \(t_1, t_2\) and a set \(\mathcal{R}\) of redexes in \(t = (\lambda x.t_1)t_2\), \(P(\langle t_1[x := t_2], \mathcal{R}' \rangle)\) implies \(P(\langle t, \mathcal{R} \cup \{0\} \rangle)\), where \(\langle t, \mathcal{R} \rangle\) reduces to \(\langle t', \mathcal{R}' \rangle\) by a head \(\beta\)-reduction.

Then \(P(\langle t, \mathcal{R} \rangle)\) holds for every marked \(\lambda\)-term \(\langle t, \mathcal{R} \rangle\) satisfying \(\mathcal{R} \subseteq \beta\)-redexes(\(t\)).

Proof We first prove that for every marked \(\lambda\)-term \(\langle t, \mathcal{R} \rangle\), \(P(\langle t, \mathcal{R} \rangle [\theta])\) holds for every substitution \(\theta\) that maps variables to marked \(\lambda\)-terms satisfying \(P\), that is, \(P(\theta(x))\) for each \(x \in \text{dom}(\theta)\). We proceed by structural induction on \(t\).

- \(t\) is some variable \(x\). Then \(t[\theta]\) is either \(x\) or \(\theta(x)\). So \(P(t[\theta])\) holds.

- \(t\) is \(\lambda x_0.t_0\) for some \(\lambda\)-term \(t_0\). Then \(\mathcal{R} = 0.\mathcal{R}_0\) for some set \(\mathcal{R}_0\) of redexes in \(t_0\). Given that \(P\) is modulo \(\alpha\)-equivalence, we may assume \(t[\theta] \equiv \lambda x_0.t_0[\theta]\) without loss of generality. By induction hypothesis on \(t_0\), we have \(P(t_0[\theta])\). By (2), we have \(P(\langle t[\theta], \mathcal{R} \rangle)\).

- \(t\) is \((t_1)t_2\) and \(\emptyset \notin \mathcal{R}\). Then \(\mathcal{R} = 0.\mathcal{R}_1 \cup 1.\mathcal{R}_2\) for some sets \(\mathcal{R}_0\) and \(\mathcal{R}_1\) of redexes in \(t_1\) and \(t_2\), respectively. Clearly, \(t[\theta] = (t_1[\theta])(t_2[\theta])\). By induction hypothesis, both \(P(\langle t_1, \mathcal{R}_0 \rangle[\theta])\) and \(P(\langle t_2, \mathcal{R}_1 \rangle[\theta])\) hold. By (3), we have \(P(\langle t, \mathcal{R} \rangle[\theta])\).

- \(t\) is \((t_1)t_2\).
2.2. FUNDAMENTAL THEOREMS OF λ-CALCULUS

Theorem 2.1.3 Assume that $\sigma_1$ and $\sigma_2$ are two complete developments from $t$. Then $\sigma_1(t) = \sigma_2(t)$.

Lemma 2.1.4 (Confluence of Developments) Assume that $\sigma_1$ and $\sigma_2$ are developments from $\langle t, R \rangle$. Then there exist reduction sequences $\sigma'_1$ and $\sigma'_2$ such that $(\sigma_1 + \sigma'_2)(t) = (\sigma_2 + \sigma'_1)(t)$ and both $\sigma_1 + \sigma'_2$ and $\sigma_2 + \sigma'_1$ are developments from $\langle t, R \rangle$.

Theorem 2.1.5 (Finite Developments) All developments are finite.

Proof We are to prove a stronger result stating that for each λ-term $t$, length($\sigma$) < $2^\text{size}(t)$ holds whenever $\sigma$ is a development of $t$.

Given $\langle t, R \rangle$, let $\mu_0(\langle t, R \rangle)$ be the maximum of length($\sigma$), where $\sigma$ ranges over all the developments of $\langle t, R \rangle$, and $\mu_0(t)$ for $\mu_0(\langle t, \beta\text{-redexes}(t) \rangle)$. By Theorem 2.1.5, $\mu_0(\langle t, R \rangle) < \infty$ for every λ-term $t$ and set of β-redexes $R$ in $t$.

Definition 2.1.6 (Standard Developments) A standard development $\sigma$ from $\langle t, R \rangle$ is standard if

1. $\sigma$ is empty, or
2. $\sigma = [p] + \sigma_1$ for the leftmost $p$ in $R$ and $\sigma_1$ is a standard development of $[p](\langle t, R \rangle)$.

Exercise 1 Assume that $\sigma$ is a development of $t$. Please show size($\sigma(t)$) < $2^\text{size}(t)$.

Exercise 2 Assume that $\sigma$ is a standard development of $\langle t, R \rangle$ that is also complete. Please show that length($\sigma$) = $\mu_0(\langle t, R \rangle)$ if $R$ contains only $\beta_1$-redexes.

2.2 Fundamental Theorems of λ-calculus

Lemma 2.2.1 Assume that $\sigma_1$ is a development from $t$. For every finite $\beta$-reduction sequence $\sigma_2$ from $t$, there exists a development $\sigma'_1$ and a $\beta$-reduction sequence $\sigma'_2$ such that $(\sigma_1 + \sigma'_2)(t) = (\sigma_2 + \sigma'_1)(t)$.

Proof We proceed by induction on the length of $\sigma_2$.

- $\sigma_2 = \emptyset$. Let $\sigma'_1 = \sigma_1$ and $\sigma'_2 = \emptyset$, and we are done.

- $\sigma_2 = \sigma_{20} + \sigma_{21}$, where $\sigma_{20}$ is a nonempty development. Let $\sigma_{10}$ be $\sigma_1$. By Lemma 2.1.4, there exists two developments $\sigma'_{10}$ and $\sigma'_{20}$ such that $(\sigma_{10} + \sigma'_2)(t) = (\sigma_{20} + \sigma'_{10})(t)$. By induction hypothesis on, there exist a development $\sigma''_{10}$ and a $\beta$-reduction sequence $\sigma'_{21}$ such that $(\sigma'_{10} + \sigma'_{20})(\sigma_{20}(t)) = (\sigma_{21} + \sigma''_{10})(\sigma_{20}(t))$. Let $\sigma'_1 = \sigma''_{10}$ and $\sigma'_2 = \sigma'_{20} + \sigma'_{21}$, and we are done.

Theorem 2.2.2 (Church-Rosser) Assume that $\sigma_1$ and $\sigma_2$ are finite $\beta$-reduction sequences from $t$. Then there exists finite $\beta$-reduction sequences $\sigma'_1$ and $\sigma'_2$ such that $(\sigma_1 + \sigma'_2)(t) = (\sigma_2 + \sigma'_1)(t)$.

Proof
Lemma 2.2.3 Assume $\sigma = \sigma_1 + \sigma_2$ is finite $\beta$-reduction sequence for a $\lambda$-term $t$, where $\sigma_1$ is a standard development and $\sigma_2$ is a standard $\beta$-reduction sequence. Then we can construct a standard (finite) $\beta$-reduction sequence $\sigma'$ from $t$ such that $\sigma(t) = \sigma'(t)$.

Proof We are to define a binary function $std_2$ that takes the arguments $\sigma_1$ and $\sigma_2$ and returns $\sigma'$.

Theorem 2.2.4 (Standardization)

Given a $\lambda$-term $t$, we use $\text{norm}_\beta(t)$ for the (possibly infinite) reduction sequence $\sigma$ from $t$ such that each $\beta$-reduction step in $\sigma$ is leftmost and $\sigma(t)$ is in normal form $\sigma$ is finite.

Theorem 2.2.5 (Normalization) Assume $\nu(t) < \infty$. Then $\text{norm}_\beta(t)$ is finite.

Theorem 2.2.6 (Conservation) Assume $p : t \rightarrow t'$ and $\mu(t') < \infty$. If $\text{subterm}(t, p)$ is $\beta_1$-redex, then $\mu(t) < \infty$. 