Chapter udf

Syntax

syn.1 Terms

lam:syn:trm:sec

The terms of the lambda calculus are built up inductively from an infinite supply of variables v_0, v_1, ..., the symbol "λ", and parentheses. We will use x, y, z, ... to designate variables, and M, N, P, ... to desginate terms.

lam:syn:trm:defn:term

Definition syn.1 (Terms). The set of *terms* of the lambda calculus is defined inductively by:

lam:syn:trm:defn:term-var

1. If x is a variable, then x is a term.

lam:syn:trm:defn:term-abs

2. If x is a variable and M is a term, then $(\lambda x.\, M)$ is a term.

lam:syn:trm:defn:term-app

3. If both M and N are terms, then (MN) is a term.

If a term $(\lambda x.\, M)$ is formed according to (2) we say it is the result of an *abstraction*, and the x in λx is called a *parameter*. A term (MN) formed according to (3) is the result of an *application*.

The terms defined above are fully parenthesized. This can get rather cumbersome, as the term $(\lambda x.\, ((\lambda x.\, x)(\lambda x.\, (xx))))$ demnostrates. We will introduce conventions for avoiding parentheses. However, the official definition makes it easy to determine how a term is constructed according to Definition syn.1. For example, the last step of forming the term $(\lambda x.\, ((\lambda x.\, x)(\lambda x.\, (xx))))$ must be abstraction where the parameter is x. It results by abstraction from the term $((\lambda x.\, x)(\lambda x.\, (xx)))$, which is an application of two terms. Each of these two terms is the result of an abstraction, and so on.

Problem syn.1. Describe the formation of $(\lambda g.\, (\lambda x.\, (g(xx)))(\lambda x.\, (g(xx))))$.

syn.2 Unique Readability

lam:syn:unq:sec

We may wonder if for each term there is a unique way of forming it, and there is. For each lambda term there is only one way to construct and interpret it.

In the following discussion, a *formation* is the procedure of constructing a term using the formation rules (one or several times) of Definition syn.1.

Lemma syn.2. *A term starts with either a variable or a parenthesis.* *lam:syn:unq: lem:term-start*

Proof. Something counts as a term only if it is constructed according to Definition syn.1. If it is the result of (1), it must be a variable. If it is the result of (2) or (3), it starts with a parenthesis. □

Lemma syn.3. *The result of an application starts with either two parentheses or a parenthesis and a variable.* *lam:syn:unq: lem:app-start*

Proof. If M is the result of an application, it is of the form (PQ), so it begins with a parenthesis. Since P is a term, by Lemma syn.2, it begins either with a parenthesis or a variable. □

Lemma syn.4. *No proper initial part of a term is itself a term.* *lam:syn:unq: lem:initial*

Problem syn.2. Prove Lemma syn.4 by induction on the length of terms.

Proposition syn.5 (Unique Readability). *There is a unique formation for each term. In other words, if a term M is formed by a formation, then it is the only formation that can form this term.* *lam:syn:unq: prop:unq*

Proof. We prove this by induction on the formation of terms.

1. M is of the form x, where x is some variable. Since the results of abstractions and applications always start with parentheses, they cannot have been used to construct M; Thus, the formation of M must be a single step of Definition syn.1(1).

2. M is of the form $(\lambda x.\, N)$, where x is some variable and N is a term. It could not have been constructed according to Definition syn.1(1), because it is not a single variable. It is not the result of an application, by Lemma syn.3. Thus M can only be the result of an abstraction on N. By inductive hypothesis we know that formation of N is itself unique.

3. M is of the form (PQ), where P and Q are terms. Since it starts with a parentheses, it cannot also be constructed by Definition syn.1(1). By Lemma syn.2, P cannot begin with λ, so (PQ) cannot be the result of an abstraction. Now suppose there were another way of constructing M by application, e.g., it is also of the form $(P'Q')$. Then P is a proper initial segment of P' (or vice versa), and this is impossible by Lemma syn.4. So P and Q are uniquely determined, and by inductive hypothesis we know that formations of P and Q is unique. □

A more readable paraphrase of the above proposition is as follows:

Proposition syn.6. *A term M can only be one of the following forms:*

1. *x, where x is a variable uniquely determined by M.*

2. *$(\lambda x.\, N)$, where x is a variable and N is another term, both of which is uniquely determined by M.*

3. *(PQ), where P and Q are two terms uniquely determined by M.*

syn.3 Abbreviated Syntax

lam:syn:abb:sec

Terms as defined in Definition syn.1 are sometimes cumbersome to write, so it is useful to introduce a more concise syntax. We must of course be careful to make sure that the terms in the concise notation also are uniquely readable. One widely used version called *abbreviated terms* is as follows.

1. When parentheses are left out, application takes place from left to right. For example, if M, N, P, and Q are terms, then $MNPQ$ abbreviates $(((MN)P)Q)$.

2. Again, when parentheses are left out, lambda abstraction is given the widest scope possible. From example, $\lambda x.\, MNP$ is read as $(\lambda x.\, MNP)$.

3. A lambda can be used to abstract multiple variables. For example, $\lambda xyz.\, M$ is short for $\lambda x.\, \lambda y.\, \lambda z.\, M$.

For example,

$$\lambda xy.\, xxyx\lambda z.\, xz$$

abbreviates

$$(\lambda x.\, (\lambda y.\, ((((xx)y)x)(\lambda z.\, (xz))))).$$

Problem syn.3. Expand the abbreviated term $\lambda g.\, (\lambda x.\, g(xx))\lambda x.\, g(xx)$.

syn.4 Free Variables

lam:syn:fv:sec

Lambda calculus is about functions, and lambda abstraction is how functions arise. Intuitively, $\lambda x.\, M$ is the function with values given by M when the argument to the function is assigned to x. But not every occurrence of x in M is relevant: if M contains another abstract $\lambda x.\, N$ then the occurrences of x in N are relevant to $\lambda x.\, N$ but not to $\lambda x.\, M$. So, a lambda abstract λx inside $\lambda x.\, M$ *binds* those occurrences of x in M that are not already bound by another lambda abstract—the *free* occurrences of x in M.

Definition syn.7 (Scope). If $\lambda x.\, M$ occurs inside a term N, then the corresponding occurrence of N is the *scope* of the λx.

Definition syn.8 (Free and bound occurrence). An occurrence of variable x in a term M is *free* if it is not in the scope of a λx, and *bound* otherwise. An occurrence of a variable x in $\lambda x.\, M$ is bound by the initial λx iff the occurrence of x in M is free.

Example syn.9. In $\lambda x.\, xy$, both x and y are in the scope of λx, so x is bound by λx. Since y is not in the scope of any λy, it is free. In $\lambda x.\, xx$, both occurrences of x are bound by λx, since both are free in xx. In $((\lambda x.\, xx)x)$, the last occurrence of x is free, since it is not in the scope of a λx. In $\lambda x.\, (\lambda x.\, x)x$, the scope of the first λx is $(\lambda x.\, x)x$ and the scope of the second λx is the second-to-last occurrence of x. In $(\lambda x.\, x)x$, the last occurrence of x is free, and the second-to-last is bound. Thus, the second-to-last occurrence of x in $\lambda x.\, (\lambda x.\, x)x$ is bound by the second λx, and the last occurrence by the first λx.

For a term P, we can check all variable occurrences in it and get a set of free variables. This set is denoted by $\mathrm{FV}(P)$ with a natural definition as follows:

Definition syn.10 (Free variables of a term). The set of *free variables* of a term is defined inductively by: lam:syn:fv: def:fv

1. $\mathrm{FV}(x) = \{x\}$ lam:syn:fv: def:fv1
2. $\mathrm{FV}(\lambda x.\, N) = \mathrm{FV}(N) \setminus \{x\}$ lam:syn:fv: def:fv2
3. $\mathrm{FV}(PQ) = \mathrm{FV}(P) \cup \mathrm{FV}(Q)$ lam:syn:fv: def:fv3

Problem syn.4.

1. Identify the scopes of λg and the two λx in this term: $\lambda g.\, (\lambda x.\, g(xx))\lambda x.\, g(xx)$.
2. In $\lambda g.\, (\lambda x.\, g(xx))\lambda x.\, g(xx)$, are all occurrences of variables bound? By which abstractions are they bound respectively?
3. Give $\mathrm{FV}(\lambda x.\, (\lambda y.\, (\lambda z.\, xy)z)y)$

explanation

A free variable is like a reference to the outside world (the *environment*), and a term containing free variables can be seen as a partially specified term, since its behaviour depends on how we set up the environment. For example, in the term $\lambda x.\, fx$, which accepts an argument x and returns f of that argument, the variable f is free. This value of the term is dependent on the environment it is in, in particular the value of f in that environment.

If we apply abstraction to this term, we get $\lambda f.\, \lambda x.\, fx$. This term is no longer dependent on the environment variable f, because it now designates a function that accepts two arguments and returns the result of applying the first to the second. Changing f in the environment won't have any effect on the behavior of this term, as the term will only use whatever is passed as an argument, and not the value of f in the environment.

Definition syn.11 (Closed term, combinator). A term with no free variables is called a *closed term*, or a *combinator*.

Lemma syn.12. *lam:syn:fv: lem:fv*

1. *If $y \neq x$, then $y \in \mathrm{FV}(\lambda x.\, N)$ iff $y \in \mathrm{FV}(N)$.* *lam:syn:fv: lem:fv-abs*
2. *$y \in \mathrm{FV}(PQ)$ iff $y \in \mathrm{FV}(P)$ or $y \in \mathrm{FV}(Q)$.* *lam:syn:fv: lem:fv-app*

Proof. Exercise. □

Problem syn.5. Prove Lemma syn.12.

syn.5 Substitution

lam:syn:sub:sec

Free variables are references to environment variables, thus it makes sense to actually use a specific value in the place of a free variable. For example, we may want to replace f in $\lambda x.\, fx$ with a specific term, like the identity function $\lambda y.\, y$. This results in $\lambda x.\, (\lambda y.\, y)x$. The process of replacing free variables with lambda terms is called substitution. explanation

lam:syn:sub:defn:substitution

Definition syn.13 (Substitution). The *substitution* of a term N for a variable x in a term M, $M[N/x]$, is defined inductively by:

lam:syn:sub:defn:substitution-1
1. $x[N/x] = N$.

lam:syn:sub:defn:substitution-2
2. $y[N/x] = y$ if $x \neq y$.

lam:syn:sub:def:substitution-3
3. $PQ[N/x] = (P[N/x])(Q[N/x])$.

lam:syn:sub:defn:substitution-4
4. $(\lambda y.\, P)[N/x] = \lambda y.\, P[N/x]$, if $x \neq y$ and $y \notin \mathrm{FV}(N)$, otherwise undefined.

In Definition syn.13(4), we require $x \neq y$ because we don't want to replace *bound* occurrences of the variable x in M by N. For example, if we compute the substitution $\lambda x.\, x[y/x]$, the result should not be $\lambda x.\, y$ but simply $\lambda x.\, x$. explanation

When substituting N for x in $\lambda y.\, P$, we also require that $y \notin \mathrm{FV}(N)$. For example, we cannot substitute y for x in $\lambda y.\, x$, i.e., $\lambda y.\, x[y/x]$, because it would result in $\lambda y.\, y$, a term that stands for the function that accepts an argument and returns it directly. But the term $\lambda y.\, x$ stands for a function that always returns the term x (or whatever x refers to). So the result we actually want is a function that accepts an argument, drop it, and returns the environment variable y. To do this properly, we would first have to "rename" the bound variable y.

Problem syn.6. What is the result of the following substitutions?

1. $\lambda y.\, x(\lambda w.\, vwx)[(uv)/x]$
2. $\lambda y.\, x(\lambda x.\, x)[(\lambda y.\, xy)/x]$
3. $y(\lambda v.\, xv)[(\lambda y.\, vy)/x]$

lam:syn:sub:thm:notinfv

Theorem syn.14. *If $x \notin \mathrm{FV}(M)$, then $\mathrm{FV}(M[N/x]) = \mathrm{FV}(M)$, if the left-hand side is defined.*

Proof. By induction on the formation of M.

1. M is a variable: exercise.

2. M is of the form (PQ): exercise.

3. M is of the form $\lambda y.\, P$, and since $\lambda y.\, P[N/x]$ is defined, it has to be $\lambda y.\, P[N/x]$. Then $P[N/x]$ has to be defined; also, $x \neq y$ and $x \notin \mathrm{FV}(Q)$. Then:

$$\begin{aligned}
&\mathrm{FV}(\lambda y.\, P[N/x]) = \\
&\qquad = \mathrm{FV}(\lambda y.\, P[N/x]) && \text{by (4)} \\
&\qquad = \mathrm{FV}(P[N/x]) \setminus \{y\} && \text{by Definition syn.10(2)} \\
&\qquad = \mathrm{FV}(P) \setminus \{y\} && \text{by inductive hypothesis} \\
&\qquad = \mathrm{FV}(\lambda y.\, P) && \text{by Definition syn.10(2)}
\end{aligned}$$

□

Problem syn.7. Complete the proof of Theorem syn.14.

Theorem syn.15. *If $x \in \mathrm{FV}(M)$, then $\mathrm{FV}(M[N/x]) = (\mathrm{FV}(M) \setminus \{x\}) \cup \mathrm{FV}(N)$, provided the left hand is defined.* *lam:syn:sub:thm:infv*

Proof. By induction on the formation of M.

1. M is a variable: exercise.

2. M is of the form PQ: Since $(PQ)[N/y]$ is defined, it has to be $(P[N/x])(Q[N/x])$ with both substitution defined. Also, since $x \in \mathrm{FV}(PQ)$, either $x \in \mathrm{FV}(P)$ or $x \in \mathrm{FV}(Q)$ or both. The rest is left as an exercise.

3. M is of the form $\lambda y.\, P$. Since $\lambda y.\, P[N/x]$ is defined, it has to be $\lambda y.\, P[N/x]$, with $P[N/x]$ defined, $x \neq y$ and $y \notin \mathrm{FV}(N)$; also, since $y \in \mathrm{FV}(\lambda x.\, P)$, we have $y \in \mathrm{FV}(P)$ too. Now:

$$\begin{aligned}
&\mathrm{FV}((\lambda y.\, P)[N/x]) = \\
&\quad = \mathrm{FV}(\lambda y.\, P[N/x]) \\
&\quad = \mathrm{FV}(P[N/x]) \setminus \{y\} \\
&\quad = ((\mathrm{FV}(P) \setminus \{y\}) \cup (\mathrm{FV}(N) \setminus \{x\}) && \text{by inductive hypothesis} \\
&\quad = (\mathrm{FV}(P) \setminus \{x, y\}) \cup \mathrm{FV}(N) && x \notin \mathrm{FV}(N) \\
&\quad = (\mathrm{FV}(\lambda y.\, P) \setminus \{x\}) \cup \mathrm{FV}(N)
\end{aligned}$$

□

Problem syn.8. Complete the proof of Theorem syn.15.

Theorem syn.16. *$x \notin \mathrm{FV}(M[N/x])$, if the right-hand side is defined and $x \notin \mathrm{FV}(N)$.* *lam:syn:sub:thm:clr*

Proof. Exercise. □

Problem syn.9. Prove Theorem syn.16.

lam:syn:sub:thm:inv **Theorem syn.17.** *If $M[y/x]$ is defined and $y \notin \mathrm{FV}(M)$, then $M[y/x][x/y] = M$.*

Proof. By induction on the formation of M.

1. M is a variable z: Exercise.
2. M is of the form (PQ). Then:
$$\begin{aligned}(PQ)[y/x][x/y] &= ((P[y/x])(Q[y/x]))[x/y] \\ &= (P[y/x][x/y])(Q[y/x][x/y]) \\ &= (PQ) \text{ by inductive hypothesis}\end{aligned}$$
3. M is of the form $\lambda z.\, N$. Because $\lambda z.\, N[y/x]$ is defined, we know that $z \neq y$. So:
$$\begin{aligned}&(\lambda z.\, N)[y/x][x/y] \\ &\qquad = (\lambda z.\, N[y/x])[x/y] \\ &\qquad = \lambda z.\, N[y/x][x/y] \\ &\qquad = \lambda z.\, N \text{ by inductive hypothesis}\end{aligned}$$

□

Problem syn.10. Complete the proof of Theorem syn.17.

syn.6 α-Conversion

lam:syn:alp:sec What is the relation between $\lambda x.\, x$ and $\lambda y.\, y$? They both represent the identity function. They are, of course, syntactically different terms. They differ only in the name of the bound variable, and one is the result of "renaming" the bound variable in the other. This is called *α-conversion*.

Definition syn.18 (Change of bound variable, $\xrightarrow{\alpha}$). If a term M contains an occurrence of $\lambda x.\, N$, $y \notin \mathrm{FV}(N)$, and $N[y/x]$ is defined, then replacing this occurrence by
$$\lambda y.\, N[y/x]$$
resulting in M' is called a *change of bound variable*, written as $M \xrightarrow{\alpha} M'$.

Definition syn.19 (Compatibility of relation). A relation R on terms is said to be *compatible* if it satisfies following conditions:

1. If RNN' then $R\lambda x.\, N\lambda x.\, N'$
2. If RPP' then $R(PQ)(P'Q)$
3. If RQQ' then $R(PQ)(PQ')$

Thus let's rephrase the definition:

Definition syn.20 (Change of bound variable, $\xrightarrow{\alpha}$). *Change of bound variable* ($\xrightarrow{\alpha}$) is the smallest compatible relation on terms satisfying following condition:

$$\lambda x.\, N \xrightarrow{\alpha} \lambda y.\, N[y/x] \qquad \text{if } x \neq y,\ y \notin \mathrm{FV}(N) \text{ and } N[y/x] \text{ is defined}$$

"Smallest" here means the relation contains only pairs that are required by compatibility and the additional condition, and nothing else. Thus this relation can also be defined as follows:

Definition syn.21 (Change of bound variable, $\xrightarrow{\alpha}$). *Change of bound variable* ($\xrightarrow{\alpha}$) is inductively defined as follows: lam:syn:alp: defn:aconvone

1. If $N \xrightarrow{\alpha} N'$ then $\lambda x.\, N \xrightarrow{\alpha} \lambda x.\, N'$ lam:syn:alp: defn:aconvone1
2. If $P \xrightarrow{\alpha} P'$ then $(PQ) \xrightarrow{\alpha} (P'Q)$ lam:syn:alp: defn:aconvone2
3. If $Q \xrightarrow{\alpha} Q'$ then $(PQ) \xrightarrow{\alpha} (PQ')$ lam:syn:alp: defn:aconvone3
4. If $x \neq y$, $y \notin \mathrm{FV}(N)$ and $N[y/x]$ is defined, then $\lambda x.\, N \xrightarrow{\alpha} \lambda y.\, N[y/x]$. lam:syn:alp: defn:aconvone4

The definitions are equivalent, but we leave the proof as an exercise. From now on we will use the inductive definition.

Definition syn.22 (α-conversion, $\xrightarrow{\alpha}\!\!\!\!\rightarrow$). *$\alpha$-conversion* ($\xrightarrow{\alpha}\!\!\!\!\rightarrow$) is the smallest reflexitive and transitive relation on terms containing $\xrightarrow{\alpha}$.

As above, "smallest" means the relation only contains pairs required by transitivity, and $\xrightarrow{\alpha}$, which leads to the following equivalent definition:

Definition syn.23 (α-conversion, $\xrightarrow{\alpha}\!\!\!\!\rightarrow$). *$\alpha$-conversion* ($\xrightarrow{\alpha}\!\!\!\!\rightarrow$) is inductively defined as follows: lam:syn:alp: defn:aconv

1. If $P \xrightarrow{\alpha}\!\!\!\!\rightarrow Q$ and $Q \xrightarrow{\alpha}\!\!\!\!\rightarrow R$, then $P \xrightarrow{\alpha}\!\!\!\!\rightarrow R$. lam:syn:alp: defn:aconv1
2. If $P \xrightarrow{\alpha} Q$, then $P \xrightarrow{\alpha}\!\!\!\!\rightarrow Q$. lam:syn:alp: defn:aconv2
3. $P \xrightarrow{\alpha}\!\!\!\!\rightarrow P$. lam:syn:alp: defn:aconv3

Example syn.24. $\lambda x.\, fx$ α-converts to $\lambda y.\, fy$, and conversely. Informally speaking, they are both functions that accept an argument and return f of that argument, refering to the environment variable f.

$\lambda x.\, fx$ does not α-convert to $\lambda x.\, gx$. Informally speaking, they refer to the environment variables f and g respectively, and this makes them different functions: they behave differently in environments where f and g are different.

Problem syn.11. Are the following pairs of terms α-convertible?

1. $\lambda x.\, \lambda y.\, x$ and $\lambda y.\, \lambda x.\, y$
2. $\lambda x.\, \lambda y.\, x$ and $\lambda c.\, \lambda b.\, a$
3. $\lambda x.\, \lambda y.\, x$ and $\lambda c.\, \lambda b.\, a$

lam:syn:alp: lem:fv-one

Lemma syn.25. *If* $P \xrightarrow{\alpha} Q$ *then* $\mathrm{FV}(P) = \mathrm{FV}(Q)$.

Proof. By induction on the derivation of $P \xrightarrow{\alpha} Q$.

1. If the last rule is (4), then P is of the form $\lambda x.\, N$ and Q of the form $\lambda y.\, N[y/x]$, with $x \neq y$, $y \notin \mathrm{FV}(N)$ and $N[y/x]$ defined. We distinguish cases according to whether $x \in \mathrm{FV}(N)$:

   a) If $x \in FV(N)$, then:

   $$\begin{aligned}\mathrm{FV}(\lambda y.\, N[y/x]) &= \mathrm{FV}(N[y/x]) \setminus \{y\} \\ &= ((\mathrm{FV}(N) \setminus \{x\}) \cup \{y\}) \setminus \{y\} \qquad \text{by Theorem syn.15} \\ &= \mathrm{FV}(N) \setminus \{x\} \\ &= \mathrm{FV}(\lambda x.\, N)\end{aligned}$$

   b) If $x \notin FV(N)$, then:

   $$\begin{aligned}\mathrm{FV}(\lambda y.\, N[y/x]) &= FVN[y/x] \setminus \{y\} \\ &= \mathrm{FV}(N) \setminus \{x\} \qquad \text{by Theorem syn.14} \\ &= \mathrm{FV}(\lambda x.\, N).\end{aligned}$$

2. The other three cases are left as exercises. □

Problem syn.12. Complete the proof of Lemma syn.25.

lam:syn:alp: lem:inv

Lemma syn.26. *If* $P \xrightarrow{\alpha} Q$ *then* $Q \xrightarrow{\alpha} P$.

Proof. Induction on the derivation of $P \xrightarrow{\alpha} Q$.

1. If the last rule is (4), then P is of the form $\lambda x.\, N$ and Q of the form $\lambda y.\, N[y/x]$, where $x \neq y$, $y \notin \mathrm{FV}(N)$ and $N[y/x]$ defined. First, we have $y \notin \mathrm{FV}(N[y/x])$ by Theorem syn.16. By Theorem syn.17 we have that $N[y/x][x/y]$ is not only defined, but also equal to N. Then by (4), we have $\lambda y.\, N[y/x] \xrightarrow{\alpha} \lambda x.\, N[y/x][x/y] = \lambda x.\, N$. □

Problem syn.13. Complete the proof of Lemma syn.26

Theorem syn.27. *α-Conversion is an equivalence relation on terms, i.e., it is reflexive, symmetric, and transitive.*

Proof. 1. For each term M, M can be changed to M by *zero* changes of bound variables.

2. If P is α-converts to Q by a series of changes of bound variables, then from Q we can just inverse these changes (by Lemma syn.26) in opposite order to obtain P.

3. If P α-converts to Q by a series of changes of bound variables, and Q to R by another series, then we can change P to R by first applying the first series and then the second series. □

From now on we say that M and N are *α-equivalent*, $M \stackrel{\alpha}{=} N$, iff M α-converts to N (which, as we've just shown, is the case iff N α-converts to M).

Theorem syn.28. *If $M \stackrel{\alpha}{=} N$, then $\mathrm{FV}(M) = \mathrm{FV}(N)$.* *lam:syn:alp: thm:fv*

Proof. Immediate from Lemma syn.25. □

Lemma syn.29. *If $R \stackrel{\alpha}{=} R'$ and $M[R/y]$ is defined, then $M[R'/y]$ is defined and α-equivalent to $M[R/y]$.* *lam:syn:alp: lem:sub:R*

Proof. Exercise. □

Problem syn.14. Prove Lemma syn.29.

Recall that in section syn.5, substitution is undefined in some cases; however, using α-conversion on terms, we can make substitution always defined by renaming bound variables. The result preserves α-equivalence, as shown in this theorem:

Theorem syn.30. *For any M, R, and y, there exists M' such that $M \stackrel{\alpha}{=} M'$ and $M'[R/y]$ is defined. Moreover, if there is another pair $M'' \stackrel{\alpha}{=} M$ and R'' where $M''[R''/y]$ is defined and $R'' \stackrel{\alpha}{=} R$, then $M'[R/y] \stackrel{\alpha}{=} M''[R''/y]$.* *lam:syn:alp: thm:sub*

Proof. By induction on the formation of M:

1. M is a variable z: Exercise.

2. Suppose M is of the form $\lambda x.\, N$. Select a variable z other than x and y and such that $z \notin \mathrm{FV}(N)$ and $z \notin \mathrm{FV}(R)$. By inductive hypothesis, we there is N' such that $N' \stackrel{\alpha}{=} N$ and $N'[z/x]$ is defined. Then $\lambda x.\, N \stackrel{\alpha}{=} \lambda x.\, N'$ too, by Definition syn.21(1). Now $\lambda x.\, N' \stackrel{\alpha}{=} \lambda z.\, N'[z/x]$ by Definition syn.21(4). We can do this because $z \neq x$, $z \notin FV(N')$ and $N'[z/x]$ is defined. Finally, $\lambda z.\, N'[z/x][R/y]$ is defined, because $z \neq y$ and $z \notin FV(R)$.

Moreover, if there is another N'' and R'' satisfying the same conditions,

$$\begin{aligned}(\lambda z.\, N''[z/x])[R''/y] &= \\ &= \lambda z.\, N''[z/x][R''/y] \\ &= \lambda z.\, N''[z/x][R/y] && \text{by Lemma syn.29} \\ &= \lambda z.\, N'[z/x][R/y] && \text{by inductive hypothesis} \\ &= (\lambda z.\, N'[z/x])[R/y]\end{aligned}$$

3. M is of the form (PQ): Exercise. □

Problem syn.15. Complete the proof of Theorem syn.30.

lam:syn:alp:cor:sub **Corollary syn.31.** *For any M, R, and y, there exists a pair of M' and R' such that $M' \stackrel{\alpha}{=} M$, $R \stackrel{\alpha}{=} R'$ and $M'[R'/y]$ is defined. Moreover, if there is another pair $M'' \stackrel{\alpha}{=} M$ and R'' with $M'[R'/y]$ defined, then $M'[R'/y] \stackrel{\alpha}{=} M''[R''/y]$.*

Proof. Immediate from Theorem syn.30. □

syn.7 The De Bruijn Index

lam:syn:deb:sec α-Equivalence is very natural, as terms that are α-equivalent "mean the same." In fact, it is possible to give a syntax for lambda terms which does not distinguish terms that can be α-converted to each other. The best known replaces variables by their *De Bruijn index*.

When we write $\lambda x.\, M$, we explicitly state that x is the parameter of the function, so that we can use x in M to refer to this parameter. In the de Bruijn index, however, parameters have no name and reference to them in the function body is denoted by a number denoting the levels of abstraction between them. For example, consider the example of $\lambda x.\, \lambda y.\, yx$: the outer abstraction is on binds the variable x; the inner abstraction binds the variable is y; the sub-term yx lies in the scope of the inner abstraction: there is no abstraction between y and its abstract λy, but one abstract between x and its abstract λx. Thus we write $0\,1$ for yx, and $\lambda.\, \lambda.\, 0\,1$ for the entire term.

Definition syn.32. De Bruijn terms are inductively defines as follows:

1. n, where n is any natural number.
2. PQ, where P and Q are both De Bruijn terms.
3. $\lambda.\, N$, where N is a De Bruijn term.

A formalized translation from ordinary lambda terms to De Bruijn indexed terms is as follows:

Definition syn.33.

$$\begin{aligned} F_\Gamma(x) &= \Gamma(x) \\ F_\Gamma(PQ) &= F_\Gamma(P)F_\Gamma(Q) \\ F_\Gamma(\lambda x.\, N) &= \lambda.\, F_{x,\Gamma}(N) \end{aligned}$$

where Γ is a list of variables indexed from zero, and $\Gamma(x)$ denotes the position of the variable x in Γ. For example, if Γ is x, y, z, then $\Gamma(x)$ is 0 and $\Gamma(z)$ is 2.

x, Γ denotes the list resulted from pushing x to the head of Γ; for instance, continuing the Γ in last example, w, Γ is w, x, y, z.

Recovering a standard lambda term from a de Bruijn term is done as follows:

Definition syn.34.

$$\begin{aligned} G_\Gamma(n) &= \Gamma[n] \\ G_\Gamma(PQ) &= G_\Gamma(P)G_\Gamma(Q) \\ G_\Gamma(\lambda.\, N) &= \lambda x.\, G_{x,\Gamma}(N) \end{aligned}$$

where Γ is again a list of variables indexed from zero, and $\Gamma[n]$ denotes the variable in position n. For example, if Γ is x, y, z, then $\Gamma[1]$ is y.

The variable x in last equation is chosen to be any variable that not in Γ.

Here we give some results without proving them:

Proposition syn.35. *If $M \xrightarrow{\alpha} M'$, and Γ is any list containing $\mathrm{FV}(M)$, then $F_\Gamma(M) \equiv F_\Gamma(M')$.*

syn.8 Terms as α-Equivalence Classes

lam:syn:tr:sec

From now on, we will consider terms up to α-equivalence. That means when we write a term, we mean its α-equivalance class it is in. For example, we write $\lambda a.\, \lambda b.\, ac$ for the set of all terms α-equivalent to it, such as $\lambda a.\, \lambda b.\, ac$, $\lambda b.\, \lambda a.\, bc$, etc.

Also, while in previous sections letters such as N, Q are used to denote a term, from now on we use them to denote a class, and it is these classes instead of terms that will be our subjects of study in what follows. Letters such as x, y continues to denote a variable.

We also adopt the notation $\underline{M}$ to denote an arbitrary element of the class M, and $\underline{M}_0, \underline{M}_1, etc.$ if we need more than one.

We reuse the notations from terms to simplify our wording. We have following definition on classes:

Definition syn.36.

1. $\lambda x.\, N$ is defined as the class containing $\lambda x.\, \underline{N}$.
2. PQ is defined to be the class containing $\underline{P}\,\underline{Q}$.

It is not hard to see that they are well defined, because α-conversion is compatible.

lam:syn:tr: def:fv **Definition syn.37.** The *free variables* of an α-equivalence class M, or $FV(M)$, is defined to be $FV(\underline{M})$.

This is well defined since $FV(\underline{M}_0) = FV(\underline{M}_1)$, as shown in Theorem syn.28.

We also reuse the notation for substition into classes:

lam:syn:tr: def:sub **Definition syn.38.** The *substitution* of R for y in M, or $M[R/y]$, is defined to be $\underline{M}_{\underline{R}/y}$, for any $\underline{M}$ and $\underline{R}$ making the substition defined.

This is also well defined as shown in Corollary syn.31.

Note how this definition significantly simplifies our reasoning. For example:

lam:syn:tr: eq:1 lam:syn:tr: eq:2

$$\begin{aligned} \lambda x.\, x[y/x] &= && \text{(syn.1)}\\ &= \lambda z.\, z[y/x] && \text{(syn.2)}\\ &= \lambda z.\, z[y/x] && \text{(syn.3)}\\ &= \lambda z.\, z && \text{(syn.4)} \end{aligned}$$

eq. (syn.1) is undefined if we still regard it as substitution on terms; but as mentioned earlier, we now consider it a substitution on classes, which is why eq. (syn.2) can happen: we can replace $\lambda x.\, x$ with $\lambda z.\, z$ because they belong to the same class.

For the same reason, from now on we will assume that the representatives we choose always satisfy the conditions needed for substitution. For example, when we see $\lambda x.\, N[R/y]$, we will assume the representative $\lambda x.\, N$ is chosen so that $x \neq y$ and $x \notin FV(R)$.

Since it is a bit strange to call $\lambda x.\, x$ a "class", let's call them Λ-terms (or simply "terms" in the rest of the part) from now on, to distinguish them from λ-terms that we are familiar with.

> We cannot say goodbye to terms yet: the whole definition of Λ-terms is based on λ-terms, and we haven't provided a method to define functions on Λ-terms, which means all such functions have to be first defined on λ-terms, and then "projected" to Λ-terms, as we did for substitutions. However we assume the reader can intuitively understand how we can define functions on Λ-terms.

int.9 β-reduction

lam:int:bet: sec When we see $(\lambda m.\, (\lambda y.\, y)m)$, it is natural to conjecture that it has some connection with $\lambda m.\, m$, namely the second term should be the result of "simplifying" the first. The notion of *β-reduction* captures this intuition formally.

Definition int.39 (β-contraction, $\xrightarrow{\beta}$). *The β-contraction* ($\xrightarrow{\beta}$) is the smallest compatible relation on terms satisfying the following condition: lam:int:bet: defn:betacontr

$$(\lambda x.\, N)Q \xrightarrow{\beta} N[Q/x]$$

We say P is *β-contracted* to Q if $P \xrightarrow{\beta} Q$. A term of the form $(\lambda x.\, N)Q$ is called a *redex*.

Problem int.16. Spell out the equivalent inductive definitions of β-contraction as we did for change of bound variable in Definition syn.21. lam:int:bet: prob:def

Definition int.40 (β-reduction, $\xrightarrow{\beta}{\twoheadrightarrow}$). *$\beta$-reduction* ($\xrightarrow{\beta}{\twoheadrightarrow}$) is the smallest reflexive, transitive relation on terms containing $\xrightarrow{\beta}$. We say P is β-reduced to Q if $P \xrightarrow{\beta}{\twoheadrightarrow} Q$. lam:int:bet: defn:betared

We will write $\to$ instead of $\xrightarrow{\beta}$, and $\twoheadrightarrow$ instead of $\xrightarrow{\beta}{\twoheadrightarrow}$ when context is clear.

Informally speaking, $M \xrightarrow{\beta}{\twoheadrightarrow} N$ if and only if M can be changed to N by zero or several steps of β-contraction.

Definition int.41 (β-normal). A term that cannot be β-contracted any further is said to be *β-normal*.

If $M \xrightarrow{\beta}{\twoheadrightarrow} N$ and N is β-normal, then we say N is a *normal form* of M. One may ask if the normal form of a term is unique, and the answer is yes, as we will see later.

Let us consider some examples.

1. We have

$$\begin{aligned}(\lambda x.\, xxy)\lambda z.\, z &\to (\lambda z.\, z)(\lambda z.\, z)y\\ &\to (\lambda z.\, z)y\\ &\to y\end{aligned}$$

2. "Simplifying" a term can actually make it more complex:

$$\begin{aligned}(\lambda x.\, xxy)(\lambda x.\, xxy) &\to (\lambda x.\, xxy)(\lambda x.\, xxy)y\\ &\to (\lambda x.\, xxy)(\lambda x.\, xxy)yy\\ &\to \dots\end{aligned}$$

3. It can also leave a term unchanged:

$$(\lambda x.\, xx)(\lambda x.\, xx) \to (\lambda x.\, xx)(\lambda x.\, xx)$$

4. Also, some terms can be reduced in more than one way; for example,

$$(\lambda x.\,(\lambda y.\,yx)z)v \to (\lambda y.\,yv)z$$

by contracting the outermost application; and

$$(\lambda x.\,(\lambda y.\,yx)z)v \to (\lambda x.\,zx)v$$

by contracting the innermost one. Note, in this case, however, that both terms further reduce to the same term, zv.

The final outcome in the last example is not a coincidence, but rather illustrates a deep and important property of the lambda calculus, known as the Church–Rosser property.

In general, there is more than one way to β-reduce a term, thus many reduction strategies have been invented, among which the most common is the *natural strategy*. The natural strategy always contracts the *left-most* redex, where the position of a redex is defined as its starting point in the term. The natural strategy has the useful property that a term can be reduced to a normal form by some strategy iff it can be reduced to normal form using the natural strategy. In what follows we will use the natural stratuegy unless otherwise specified. digression

Definition int.42 (β-equivalence, $=$). *β-Equivalence* $(=)$ is the relation inductively defined as follows:

1. $M = M$.
2. If $M = N$, then $N = M$.
3. If $M = N$, $N = O$, then $M = O$.
4. If $M = N$, then $PM = PN$.
5. If $M = N$, then $MQ = NQ$.
6. If $M = N$, then $\lambda x.\,M = \lambda x.\,N$.
7. $(\lambda x.\,N)Q = N[Q/x]$.

The first three rules make the relation an equivalence relation; the next three make it compatible; the last ensures that it contains β-contraction.

Informally speaking, two terms are β-equivalent if and only if one of them can be changed to the other in zero or more steps of β-contraction, or "inverse" of β-contraction. The inverse of β-contraction is defined so that M inverse-β-contracts to N iff N β-contracts to M.

Besides the above rules, we will extend the relation with more rules, and denote the extended equivalence relation as $\overset{X}{=}$, where X is the extending rule.

syn.10 η-conversion

There is another relation on λ terms. In section syn.4 we used the example $\lambda x.\,(fx)$, which accepts an argument and applies f to it. In other words, it is the same function as f: $\lambda x.\,(fx)N$ and fN both reduce to fN. We use η-reduction (and η-extension) to capture this idea. lam:syn:eta:sec

Definition syn.43 (η-contraction, $\xrightarrow{\eta}$). *η-contraction* ($\xrightarrow{\eta}$) is the smallest compatible relation on terms satisfying the following condition: lam:syn:eta:defn:beredone

$$\lambda x.\,Mx \xrightarrow{\eta} M \text{ provided } x \notin FV(M)$$

Definition syn.44 ($\beta\eta$-reduction, $\xrightarrow{\beta\eta}\!\!\!\!\rightarrow$). *$\beta\eta$-reduction* ($\xrightarrow{\beta\eta}\!\!\!\!\rightarrow$) is the smallest reflexive, transitive relation on terms containing $\xrightarrow{\beta}$ and $\xrightarrow{\eta}$, i.e., the rules of reflexivity and transitive plus the following two rules: lam:syn:eta:defn:bered

1. If $M \xrightarrow{\beta} N$ then $M \xrightarrow{\beta\eta}\!\!\!\!\rightarrow N$. lam:syn:eta:defn:bered3
2. If $M \xrightarrow{\eta} N$ then $M \xrightarrow{\beta\eta}\!\!\!\!\rightarrow N$. lam:syn:eta:defn:bered4

Definition syn.45. We extend the equivalence relation $=$ with the η-conversion rule:

$$\lambda x.\,fx = f$$

and denote the extended relation as $\overset{\eta}{=}$.

η-equivalence is important because it is related to extensionality of lambda terms:

Definition syn.46 (Extensionality). We extend the equivalence relation $=$ with the (ext) rule:

$$\text{If } Mx = Nx \text{ then } M = N, \text{ provided } x \notin FV(MN).$$

and denote the extended relation as $\overset{ext}{=}$.

Roughly speaking, the rule states that two terms, viewed as functions, should be considered equal if they behave the same for the same argument.

We now prove that the η rule provides exactly the extensionality, and nothing else.

Theorem syn.47. *$M \overset{ext}{=} N$ if and only if $M \overset{\eta}{=} N$.*

Proof. First we prove that $\overset{\eta}{=}$ is closed under the extensionality rule. That is, *ext* rule doesn't add anything to $\overset{\eta}{=}$. We then have $\overset{\eta}{=}$ contains $\overset{ext}{=}$, and if $M \overset{ext}{=} N$, then $M \overset{\eta}{=} N$.

To prove $\overset{\eta}{=}$ is closed under *ext*, note that for any $M = N$ derived by the *ext* rule, we have $Mx \overset{\eta}{=} Nx$ as premise. Then we have $\lambda x.\, Mx \overset{\eta}{=} \lambda x.\, Nx$ by a rule of $=$, applying η on both side gives us $M \overset{\eta}{=} N$.

Similarly we prove that the η rule is contained in $\overset{ext}{=}$. For any $\lambda x.\, Mx$ and M with $x \notin FV(M)$, we have that $(\lambda x.\, Mx)x \overset{ext}{=} Mx$, giving us $\lambda x.\, Mx \overset{ext}{=} M$ by the *ext* rule. □

Photo Credits

Bibliography