A: Lambda calculus

This appendix briefly reviews lambda calculus. It is not a general or comprehensive introduction to the topic. The material covered relates to the main body of the text and they are used in it frequently.

Lambda terms (equivalently, λ-terms) are well-formed lambda expressions. They are recursively defined as follows.

λ-words are constructed from the alphabet

> $x, y, z, \cdots$ for variables,
> $1, 2, a', b'$ for invariables (constants),
> λ for abstractor (lambda binding),
> (,) for grouping (parentheses).

λ-terms are the set Λ such that

> variables and invariables are in Λ,
> if $M \in \Lambda$ then $(\lambda x.M) \in \Lambda$ where x is an arbitrary variable,
> if $M, N \in \Lambda$ then $(MN) \in \Lambda$.

By convention, we write multiple lambda bindings with a single dot: $\lambda x.\lambda y.xy$ is written as $\lambda x\lambda y.xy$.

Also by convention, lambda bindings associate to the right, and juxtaposition associates to the left. $\lambda x\lambda y\lambda z.xyz$ is same as $\lambda x(\lambda y(\lambda z((xy)z)))$.

A variable is **free** if it is not in the scope of its lambda binding, **bound** otherwise. For example, x is free in $x+2$, $\lambda y.x$ and $(\lambda x(a'b'))x$. It is bound in $\lambda x.x$ and in $\lambda x\lambda y.xy$. Within the inner body of the last lambda term, xy, both variables are said to have **free occurrences** because there is no lambda binding in the body.

Lambda conversions are operations that denote equivalences among lambda terms. When used in the direction of eliminating a lambda binding, they are called **reductions**. If a lambda is introduced they are called **abstractions**.

The conversions rely on the property of substitution for bound variables. Eta conversion shows the behavioral equivalence of the typed objects with and without variables. Beta conversion is the main mechanism to establish function application and function abstraction as two sides of the same coin. Alpha conversion shows the equivalence of bound variables under substitution. Together they define equivalence in the function treatment of lambda calculus.

substitution $M[a/x]$ stands for substituting a for free occurrences of x in M.

η-conversion $\lambda x.fx =_\eta f$, if x is not free in f.

β-conversion $\lambda x.M(a) =_\beta M[a/x]$

α-conversion $\lambda x.M =_\alpha \lambda y.(M[y/x])$, if scopes of variables in $\lambda x.M$ and $\lambda y.(M[y/x])$ are the same.

equivalence $M = N$ iff $Ma =_{\alpha,\beta,\eta} Na$, for all lambda terms M, N, a.

Read '=' as 'behaves the same', not as 'identical'. From substitution and beta reduction, we get $\lambda x.fx(a) =_\beta fx[a/x]$, which is the same as fa, hence the association of beta with function application and abstraction. By equivalence, $\lambda x.fx = f$ too, hence the same behavior when f is supplied with a. The condition on eta conversion ensures that we do not change the behavior of objects; $\lambda x.(\lambda y.yx)x$, in which x is free in $\lambda y.yx$, is not equivalent to $\lambda y.yx$. Similarly, the condition on alpha conversion avoids an accidental capture of the same names, for example $\lambda x.y \neq_\alpha \lambda y.y$, and $\lambda x \lambda y.xy \neq_\alpha \lambda y \lambda y.yy$.

Normalization refers to the successive application of a conversion until it no longer applies. For example, the beta normalization of $(\lambda x \lambda y.f'yx)(a')(b')$ is two applications of beta reduction giving $f'b'a'$. The eta normalization of $\lambda x \lambda y.f'yx$ is f'. Some lambda terms have no normal forms because the process may not always terminate: $(\lambda x.xx)(\lambda x.xx)$ has no beta normal form.

Normal-order evaluation of a lambda term is the application of beta reduction to the leftmost outermost reducible expression (**redex**) first. In $(\lambda x.x)((\lambda y.y)a')$ there are two redexes, and normal-order chooses to reduce it to $(\lambda y.y)a'$, i.e. the application of the second one, without evaluation, to the leftmost redex. The Church-Rosser theorem establishes the result that two distinct sequences of reductions from the same lambda term will yield the same normal form if there is one. For the example above, it is $a.'$