

syn.1 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.1 (Scope). If $\lambda x. M$ occurs inside a term N , then the corresponding occurrence of N is the *scope* of the λx .

Definition syn.2 (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.3. 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 $FV(P)$ with a natural definition as follows:

lam:syn:fv:
def:fv **Definition syn.4 (Free variables of a term).** The set of *free variables* of a term is defined inductively by:

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

Problem syn.1. 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 $FV(\lambda x. (\lambda y. (\lambda z. xy)z)y)$

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

Lemma syn.6.

1. If $y \neq x$, then $y \in \text{FV}(\lambda x. N)$ iff $y \in \text{FV}(N)$.
2. $y \in \text{FV}(PQ)$ iff $y \in \text{FV}(P)$ or $y \in \text{FV}(Q)$.

lam:syn:fv:
lem:fv
lam:syn:fv:
lem:fv-abs
lam:syn:fv:
lem:fv-app

Proof. Exercise. □

Problem syn.2. Prove [Lemma syn.6](#).

Photo Credits

Bibliography