Connections with Other Courses

1. Foundations of Computer Science
2. Computation Theory
3. Semantics of Programming Languages
4. Logic and Proof
5. Types
6. Denotational Semantics

Summary of Course

The λ-calculus, Hindley and Seldin

equality and reduction

encoding of data

recursive functions

λ-calculus and computation theory

type theory

Foundations of Functional Programming

Philippa Gardner

This course is based on lecture notes by Larry Paulson.

Introducing Notation

Mathematical Notation

$$f(x) = x \times x \qquad f(2) = 2 \times 2 = 4$$

ML notation

```
fun f (x:int) = x*x;        f(2);
                            > 4:int
```

λ-calculus notation

$$f = \lambda x.\, x \times x \qquad f2 = (\lambda x.\, x \times x)\ 2 = 2 \times 2 = 4$$

Definition of the λ-terms

The terms of the λ-calculus, known as **λ-terms**, are constructed inductively from a given set of variables V. They may take one of the following forms:

x variable;

$\underbrace{(\lambda x.M)}_{x \text{ bound}}$ abstraction, where M is a term;

$(M\,N)$ application, where M and N are terms.

Notation $M \equiv N$ means that M is identical to N.

Definition The **set of all bound variables** in M, denoted by BV(M), is defined inductively by

$$\begin{aligned}
\mathsf{BV}(x) &= \emptyset \\
\mathsf{BV}(\lambda x.M) &= \mathsf{BV}(M) \cup \{x\} \\
\mathsf{BV}(M\,N) &= \mathsf{BV}(M) \cup \mathsf{BV}(N)
\end{aligned}$$

Definition The **set of all free variables** in M, denoted by FV(M), is defined inductively by

$$\begin{aligned}
\mathsf{FV}(x) &= \{x\} \\
\mathsf{FV}(\lambda x.M) &= \mathsf{FV}(M) - \{x\} \\
\mathsf{FV}(M\,N) &= \mathsf{FV}(M) \cup \mathsf{FV}(N)
\end{aligned}$$

Implementation methods

ISWIM and SECD machine, Field and Harrison

Combinators and graph reduction, Hindley and Seldin, Field and Harrison

Bracketing

We use the following abbreviations:

1. $\lambda y.M$ for $(\lambda y.M)$ and $M\,N$ for $(M\,N)$;
2. $\lambda y.M\,N$ for $\lambda y.(M\,N)$;
3. $\lambda x.\lambda y.M$ for $(\lambda x.(\lambda y.M))$;
4. $M\,N\,L$ for $((M\,N)\,L)$.

Structural Induction on λ-terms

To show property $P(M)$ for all λ-terms M, it suffices to prove

the **base case** $P(x)$ for arbitrary variable x;

two induction steps

$P(M)$ implies $P(\lambda x.M)$ for arbitrary x and M;

$P(M)$ and $P(N)$ implies $P(M\,N)$ for arbitrary M and N.

Substitution

Definition

The term $M[L/x]$, the result of substituting L for all free occurrences of y in M, is defined inductively by

$$
\begin{aligned}
x[L/y] &\equiv \begin{cases} L & \text{if } x \equiv y \\ x & \text{otherwise} \end{cases} \\
(\lambda x.M)[L/y] &\equiv \begin{cases} (\lambda x.M) & \text{if } x \equiv y \\ (\lambda x.M[L/y]) & \text{otherwise} \end{cases} \\
(MN)[L/y] &\equiv (M[L/y]N[L/y])
\end{aligned}
$$

Structural Induction on Lists

To show property $P(\mathtt{xs})$ for all lists $\mathtt{xs}$, it suffices to prove

the **base case** $P([\,])$

the **induction step** $P(\mathtt{ys})$ implies $P(\mathtt{x} :: \mathtt{ys})$ for arbitrary $\mathtt{x}$ and $\mathtt{ys}$

Theorem

For any λ-term M, the set FV(M) is finite.

α-conversion relation

The α-conversion relation $\equiv_\alpha$ can be defined by the following axioms and rules:

$$\lambda x.M \equiv_\alpha \lambda y.(M[y/x])^* \qquad M \equiv_\alpha M$$

$$\frac{M \equiv_\alpha M'}{M' \equiv_\alpha M} \qquad \frac{M \equiv_\alpha M' \quad M' \equiv_\alpha M''}{M \equiv_\alpha M''}$$

$$\frac{M \equiv_\alpha M'}{\lambda x.M \equiv_\alpha \lambda x.M'} \qquad \frac{M_1 \equiv_\alpha M_1' \quad M_2 \equiv_\alpha M_2'}{M_1 M_2 \equiv_\alpha M_1' M_2'}$$

$*y$ is not free in M.

Definition

The term $M[N/y]$ is **safe** provided the bound variables of M are disjoint from the free variables of N.