Computing with \(\beta\)-Terms

How does the \(\beta\)-calculus relate to real programming languages such as Miranda?

- \(\beta\)-abstraction corresponds to defining functions with formal parameters, cf.:
  \[
  \lambda x. \lambda y. x
  \]

- function application works in the same way, through substitution:
  \[
  \lambda x. \lambda y. x M N \Rightarrow M (\text{\(\lambda\)} y. x) M N \Rightarrow M
  \]

Functions and Values

How can we represent values in the \(\beta\)-calculus?

In Miranda we can write

\[
\text{twice} \; x = 2 \times x
\]

but we can’t (yet) write

\[
\lambda x. 2 \times x
\]

since \(2 \times x\) isn’t a \(\beta\)-term.

Representing Booleans

Booleans can be represented in the \(\beta\)-calculus as follows:

- True is represented by \(\lambda x. \lambda y. x\)
- False is represented by \(\lambda x. \lambda y. y\)

Case-Analysis

Given a \(\beta\)-term \(B\) which is either True or False, a case-analysis

\[
\text{if } B \text{ then } M \text{ else } N,
\]

where \(M\) and \(N\) are arbitrary \(\beta\)-terms, is represented by the \(\beta\)-term \((B \; M) \; N\) :

- if \(B\) is True: \((B \; M) \; N = ((\lambda x. \lambda y. x) \; M) \; N \Rightarrow_M M\)
- if \(B\) is False: \((B \; M) \; N = ((\lambda x. \lambda y. y) \; M) \; N \Rightarrow_M N\)

Representing Numbers

Natural numbers can be represented in the \(\beta\)-calculus as follows:

- 0 is represented by \(\lambda x. \lambda s. x\)
- \(\text{Succ} \; 0\) is represented by \(\lambda x. \lambda s. s \; x\)
- \(\text{Succ} \; (\text{Succ} \; 0)\) is represented by \(\lambda x. \lambda s. s \; (s \; x)\)
- \(n\) is represented by \(\lambda x. \lambda s. (s \; (s \; \ldots \; (s \; x) \ldots))\)
  (where \(s\) is applied \(n\) times to \(x\))
Representing Constructors

A natural number \( n \) is represented by a \( \lambda \)-term \( N \):
\[
\lambda z. \lambda s. s (s \ldots (s z) \ldots)
\]
(with \( s \) applied \( n \) times to \( z \))

If we apply \( N \) to \( \lambda \)-terms \( Z \) and \( S \), the effect is to replace \( z \) with \( Z \) and each \( s \) with \( S \):

\[
N Z S \Rightarrow \lambda s. S (S \ldots(S Z)\ldots)
\]

Applying \( S \) to each side gives

\[
S (N Z S) \Rightarrow \lambda s. S (S (S \ldots(S Z)\ldots))
\]
(with \( n+1 \) \( S \)'s)

Therefore \( \lambda z. \lambda s. s (N z s) \) represents \( n+1 \), and \( \text{Succ} \) is represented by the \( \lambda \)-term

\[
\lambda n. \lambda z. \lambda s. s (n z s)
\]

Working with Numbers

Given \( M \) and \( N \):

\[
M Z S \Rightarrow \lambda s. S (S \ldots(S Z)\ldots)
\]
(with \( m \) \( S \)'s)

\[
N Z S \Rightarrow \lambda s. S (S \ldots(S Z)\ldots)
\]
(with \( n \) \( S \)'s)

If we replace \( Z \) in the former by \( N Z S \), we get

\[
M (N Z S) S \Rightarrow \lambda s. S (S (S \ldots(S Z)\ldots))
\]
(with \( m+n \) \( S \)'s)

Therefore \( M+N \) is represented by \( \lambda z. \lambda s. M (N z s) \) and addition is the \( \lambda \)-term

\[
\lambda m. \lambda n. \lambda z. \lambda s. M (N z s)
\]