3.2 Semantics of the Lambda Calculus

Operational Semantics: Reference interpreter which evaluates Lambda Terms.

Denotational Semantics: Mapping of expressions to mathematical objects.

The operational semantics of the Lambda Calculus can also be used for an implementation of Haskell if one can translate Haskell to corresponding Lambda Terms (see Sect. 3.3).

The operational semantics of the Lambda Calculus is given by several reduction rules, denoted by Greek letters.

\( \alpha \)-reduction: renaming of bound variables

\( \alpha \)-reduction

\( \text{Def. 3.2.} \quad (\alpha \text{-reduction}) \)

The relation \( \rightarrow_\alpha \subseteq \Lambda \times \Lambda \) is the smallest relation with:

- \( \lambda x. t \rightarrow_\alpha \lambda y. t [x/y] \) if \( y \notin \text{free}(t) \)

- If \( t_1 \rightarrow_\alpha t_2 \), then \( (t_1 \ y) \rightarrow_\alpha (t_2 \ y) \)

\( (y \ t_1) \rightarrow_\alpha (y \ t_2) \)

and \( (\lambda v. t) \rightarrow_\alpha (\lambda v. t) \)
and \((\lambda y. t_1) \rightarrow^\beta (\lambda y. t_2)\)

for all \(r \in A, y \in V\).

Ex: \(\lambda x. x y \rightarrow^\alpha \lambda x y'. x y' \rightarrow^\alpha \lambda x' y'. x' y' \lambda x. (\lambda y. x y) \rightarrow^\alpha \lambda x y. x' y' \rightarrow^\alpha \lambda x y. x y\)

\(\lambda x. x y \nrightarrow^\alpha \lambda x. x y\)

\(\beta\)-reduction: Evaluates the application of \(\lambda\)-abstractions.

\textbf{Def. 3.2.2 } (\(\beta\)-reduction)

The relation \(\rightarrow^\beta \subseteq A \times A\) is the smallest relation with:

- \((\lambda x. t) r \rightarrow^\beta t[x/r]\)
- If \(t_1 \rightarrow^\beta t_2\), then \((t_1 r) \rightarrow^\beta (t_2 r)\)
- \((r t_1) \rightarrow^\beta (r t_2)\)
- \((\lambda y. t_1) \rightarrow^\beta (\lambda y. t_2)\)
for all \( x \in A, y \in U \).

\[
\text{Ex: } (\lambda x. x) \text{ Zero } \rightarrow_B x \left[ x / \text{ Zero} \right] = \text{ Zero}
\]

\[
(\lambda x. y. x y) \ y \rightarrow_B (\lambda y. x y) \left[ x / y \right] = \lambda y. y y
\]

\[
\lambda x. (\lambda y. x y)
\]

\[
(\lambda x. \text{ plus} \times x) \ (\lambda y. \text{ times} \ y \ y) \ 3
\]

\[
\rightarrow_B \quad \text{ B}
\]

\[
\text{ plus} \ ((\lambda y. \text{ times} \ y \ y) \ 3) \ 1 \quad (\lambda x. \text{ plus} \times x) \ (\text{ times} \ 3 \ 3)
\]

\[
\rightarrow_B \quad \text{ B}
\]

\[
\text{ plus} \ (\text{ times} \ 3 \ 3) \ 1
\]

There can be several possibilities to apply \( \beta \)-reduction.

Do they always yield the same result?

This is the question of "Confluence".

Def. 3.2.3 (Transitive-reflexive closure, Normal Form, Confluence)
Let $\rightarrow$ be a relation on a set $N$.

(a) The **transitive-reflexive closure** of $\rightarrow$ on a set $N$ is the smallest relation $\rightarrow^*$ such that the following holds for all $t_1, t_2, t_3 \in N$:

- if $t_1 \rightarrow t_2$, then $t_1 \rightarrow^* t_2$ \hspace{1cm} ($\rightarrow \subseteq \rightarrow^*$)
- if $t_1 \rightarrow t_2 \rightarrow^* t_3$, then $t_1 \rightarrow^* t_3$ \hspace{1cm} (transitive)
- $t_n \rightarrow^* t_n$ \hspace{1cm} (reflexive)

In other words:

$t_0 \rightarrow^* t_n$ iff there exist $t_1, \ldots, t_{n-1}$ such that $t_0 \rightarrow t_1 \rightarrow t_2 \rightarrow \ldots \rightarrow t_{n-1} \rightarrow t_n$, $n \geq 0$

(b) An object $q \in N$ is a **$\rightarrow$-normal form** iff there is no $q' \in N$ with $q \rightarrow q'$.

An object $q$ is a **normal form of an object $t$** iff $t \rightarrow^* q$ and $q$ is a normal form.

(c) The relation $\rightarrow$ is **confluent** iff for all $t, q_1, q_2 \in N$ the following holds:

If $t \rightarrow^* q_1$ and $t \rightarrow^* q_2$,
then there exists a $q \in N$ with $q_1 \rightarrow^* q$ and $q_2 \rightarrow^* q$. 

If the green part is possible.
Confluence is important, because it guarantees that the results of computations are unique.

**Lemma 3.2.4** (Confluence implies unique normal forms)

Let \( \rightarrow \) be a confluent relation over a set \( N \). Then every \( t \in N \) has at most one normal form.

**Proof:**

Let \( q_1, q_2 \) be normal forms of \( t \):

\[
\begin{array}{c}
  t \\
  \downarrow \\
  q_1 \rightarrow^* q_2
\end{array}
\]

As \( \rightarrow \) is confluent, there must be a \( q \) such that

\[
\begin{array}{c}
  q_1 \\
  \downarrow \\
  q
\end{array} \quad 
\begin{array}{c}
  q_2 \\
  \downarrow \\
  q
\end{array}
\]

But as \( q_1 \) and \( q_2 \) are normal forms, we have \( q_1 = q = q_2 \).

**Fundamental Theorem of Lambda Calculus:**

\( \rightarrow \) is confluent up to \( \alpha \)-reduction.
$ightarrow_{\beta}$ is confluent up to $\alpha$-reduction, i.e., up to renaming of bound variables.

**Thm. 3.2.5** *(Confluence of Lambda Calculus with $\beta$-reduction)*

$ightarrow_{\beta}$ is confluent (up to renaming of bound variables) i.e.: if $t \rightarrow_{\beta}^* q_1$ and $t \rightarrow_{\beta}^* q_2$, then there exist $q, q' \in \Lambda$ with $q_1 \rightarrow_{\beta}^* q$, $q_2 \rightarrow_{\beta}^* q'$, and $q \rightarrow_{\alpha}^* q'$.

This non-trivial classical theorem of the Lambda Calculus was proved by Church + Rosser

(Confluence $\equiv$ Church-Rosser Property)

To implement Haskell via the Lambda Calculus, we also need to evaluate pre-defined functions.

Constants of the Lambda Calculus correspond to
Constants of the Lambda Calculus correspond to

- data constructors like Zero, Cons, O, 1, -, 1,...
  which are not evaluated further

- defined function symbols like +, *, map, is_q, cons,
  arg, Cons, sel, i, ...,

which are pre-defined and have to be evaluated

\( \Delta \text{-reduction} \): Reduction rules for terms like

\[
\frac{C \ t_1 \ ... \ t_n}{\text{where } C \in C} \quad (C \ t_1) \ t_2 \ ... \ t_n
\]

The goal is to restrict the rules for \( \Delta \)-reduction
in such a way that confluence is preserved.
Therefore, the rules for \( \Delta \)-reduction have to satisfy
 certain syntactic restrictions which guarantee confluence.

**Def. 3.26** (\( \Delta \)-reduction)

A set \( \Delta \) of rules of the form \( C \ t_1 \ ... \ t_n \rightarrow r \)
with \( C \in C, \ t_1, ..., t_n, r \in A \) is a **Delta-rule-set** if
it satisfies the following conditions:

- \( t_1, ..., t_n, r \) are closed terms (i.e., without free
  variables)

  Otherwise: \( C \ O \rightarrow O \)

  \( r \ x \rightarrow 1 \)
Otherwise: \[ c \ C \rightarrow O \]
\[ c \ x \rightarrow 1 \]

Then: \[ c \ O \]
\[ 0 \ 1 \]

\[ t_1, \ldots, t_n \] are in \( \beta \)-normal form and they do not contain the left-hand side of a rule from \( \delta \)

Otherwise: \[ c \left( (\lambda x.0)1 \right) \rightarrow O \]
\[ c \ 0 \rightarrow 1 \]

Then: \[ c \left( (\lambda x.0)1 \right) \]
\[ 0 \ 1 \]

\[ \delta \] does not contain different rules \( c \ t_1 \ldots t_n \rightarrow r \) and \( c \ t_1 \ldots t_n \ldots t_m \rightarrow r' \) with \( m \geq n \)

Otherwise: \[ c \ O \rightarrow O \]
\[ (c \ O)1 \rightarrow 1 \]

Then: \[ (c \ O)1 \]
\[ 1 \ 0 \ 1 \]

For a Delta-rule-set \( \delta \), we define \( \rightarrow n \) as the
For a Delta-rule-set \( \Delta \) we define \( \rightarrow_\Delta \) as the smallest relation such that

- \( l \rightarrow_\Delta r \) for all rules \( l \rightarrow r \in \Delta \)
- if \( t_1 \rightarrow_\Delta t_2 \), then (\( t_1 \), \( r \)) \( \rightarrow_\Delta \) (\( t_2 \), \( r \)),
  \( (r \cdot t_1) \rightarrow_\Delta (r \cdot t_2) \),
  \( (\lambda y. t_1) \rightarrow_\Delta (\lambda y. t_2) \)
  for all \( r \in \Lambda \), \( y \in V \).

The combination of \( \beta \)- and \( \delta \)-reduction is:

\[ \rightarrow_{\beta \delta} = \rightarrow_{\beta} \cup \rightarrow_{\delta} \]

An example for a Delta-rule-set is:

\[ \delta = \{ \text{isa}_\text{Succ} \ (\text{Succ} \ t) \rightarrow \text{True} \mid t \in \Lambda, \ t \text{ closed and} \ i \in \rightarrow_{\beta} \text{-normal form} \}
\]

\[ \cup \{ \text{isa}_\text{Succ} \ \text{zero} \rightarrow \text{False} \} \]

**Theorem 3.2.7** (Confluence of the Lambda Calculus with \( \beta \)- and \( \delta \)-rules)

\( \rightarrow_{\beta \delta} \) is confluent, i.e.,

if \( t \rightarrow_{\beta} q_1 \) and \( t \rightarrow_{\delta} q_2 \)
if \( t \xrightarrow{\beta} q_1 \) and \( t \xrightarrow{\beta} q_2 \),
then there exist \( q, q' \) with
\[
q_1 \xrightarrow{\beta} q, \quad q_2 \xrightarrow{\beta} q, \quad \text{and} \quad q \xrightarrow{\alpha} q'.
\]

With \( \xrightarrow{\beta} \), we have defined the (operational) semantics of the Lambda Calculus (due to Church).

It took more than 30 years until the denotational semantics of the Lambda Calculus was developed (D. Scott). Key idea: Continuous functions