Übungsblatt 6.5

Abgabe der Lösungen: So, 23.06., 23:59 per E-Mail an die Tutoren

De-Bruijn-Indizes

Die Darstellung von λ-Abstraktion mittels Variablennamen hat einen erheblichen Nachteil, wenn man sie aus der Sicht eines Programmierers sieht: dadurch, dass Terme modulo α-Äquivalenz betrachtet werden, gibt es keine kanonische Form; die Gleichheitsprüfung zweier Terme gestaltet sich dadurch kompliziert; zudem muss z.B. für die Implementierung der β-Reduktion auch auf Variableneinfang geachtet werden.

Alternative Darstellung. Um dieses Problem zu umgehen, wird bei Implementierungen des λ-Kalküls oft eine Darstellung von Termen mittels sogenannter *De-Bruijn-Indizes* gewählt. Man verzichtet hier vollständig auf Benennung von Variablen; stattdessen verwendet man als Variablen *natürliche Zahlen*, welche den zugehörigen λ-Operator adressieren.

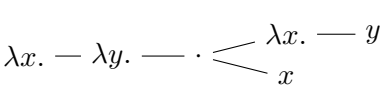
Freie Variablen sind dann schlichtweg natürliche Zahlen, die größer sind als die Anzahl an λ-Operatoren oberhalb im Termbaum.

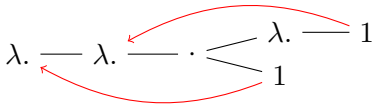
Beispiele:

Notation mit Namen	De Bruijn
$\lambda x.\, \lambda y.\, x$	$\lambda.\, \lambda.\, 1$
$\lambda x.\, \lambda y.\, y$	$\lambda.\, \lambda.\, 0$
$\lambda x.\, \lambda y.\, z\ (\lambda x.\, x)$	$\lambda.\, \lambda.\, 2\ (\lambda.\, 0)$
$\lambda x.\, \lambda y.\, (z\ w)\ (\lambda z.\, z)$	$\lambda.\, \lambda.\, (2\ 3)\ (\lambda.\, 0)$
$\lambda x.\, \lambda y.\, (\lambda x.\, y)\ x$	$\lambda.\, \lambda.\, (\lambda.\, 1)\ 1$
$\lambda x.\, \lambda y.\, (\lambda z.\, x)\ x$	$\lambda.\, \lambda.\, (\lambda.\, 2)\ 1$

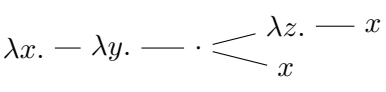
Beachten Sie, wie in den letzten beiden Beispielen ersichtlich, dass

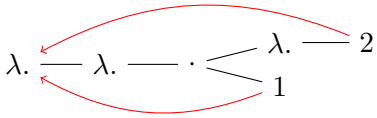
- nur λs *oberhalb im Termbaum* gezählt werden (*Applikation* gekennzeichnet durch „·“):





- dieselbe Variable von *unterschiedlichen* natürlichen Zahlen repräsentiert werden kann (und umgekehrt dieselbe Zahl verschiedene Variablen repräsentieren kann, wie gerade gesehen):





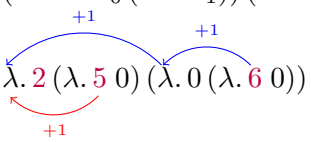
β-Reduktion. Die Durchführung einer β-Reduktion, wie sie in der Vorlesung eingeführt wurde, beruht auf Substitutionen von Variablen in Termen sowie, falls nötig, α-Konversion zur Vermei-

dung von Variableneinfang. In obiger Darstellung mittels *De-Bruijn-Indizes* lässt sich das Vorgehen schematischer beschreiben: um eine Reduktion von $(\lambda.\, s)\, t$ auszuführen, müssen wir

1. die Instanzen von Variablen (natürliche Zahlen $n_0, \ldots, n_k$) finden, die von dem λ gebunden werden;
2. alle *freien* Variablen in s dekrementieren (da ein λ entfernt wird)
3. die $n_0, \ldots, n_k$ durch t ersetzen; hierbei ist darauf zu achten, *jeweils* die *freien* Variablen in t um die Anzahl der λs unter denen das jeweilige n_i sich befindet, zu erhöhen.

Beispiel:

$$
\begin{aligned}
& (\lambda.\, \lambda.\, 3\ 1\ (\lambda.\, 0\ 2))\ (\lambda.\, 4\ 0) \\
\rightsquigarrow\ & (\lambda.\, \lambda.\, 3\ n_0\ (\lambda.\, 0\ n_1))\ (\lambda.\, 4\ 0) && \text{(Schritt 1)} \\
\rightsquigarrow\ & (\lambda.\, \lambda.\, 2\ n_0\ (\lambda.\, 0\ n_1))\ (\lambda.\, 4\ 0) && \text{(Schritt 2)} \\
\rightsquigarrow\ & \lambda.\, 2\ (\lambda.\, 5\ 0)\ (\lambda.\, 0\ (\lambda.\, 6\ 0)) && \text{(Schritt 3)}
\end{aligned}
$$



HAUSAUFGABEN

Übung 1 Ein Interpreter für den λ-Kalkül (10 Punkte)

Programmieren Sie einen Interpreter für den λ-Kalkül zur Auswertung von λ-Termen in der Darstellung mit De-Bruijn-Indizes in *normaler* sowie in *applikativer* Reihenfolge. Nutzen Sie dafür eine der auf der Übungswebseite bereitgestellten Vorlagen für *Scala*, *Java* oder *Haskell*.

Hinweise:

- In der *Java*-Vorlage sollen die vorgegebenen Klassen immutable bleiben.
- Sie dürfen zudem den `instanceOf`-Operator verwenden.
- Die `reduce*`-Methoden der Klassen sollen jeweils nur einen Schritt der Reduktion durchführen (Definitionen 3.13 und 3.14 im Skript), die vollständige Auswertung erfolgt durch `eval*` in der Oberklasse `LambdaTerm`.
- *Haskells* Auswertungsstrategie ist *lazy* (call-by-need), d.h. Terme werden erst reduziert, wenn sie benötigt werden; Sie können sich dies in der *Haskell*-Vorlage zunutze machen um `eval*` direkt rekursiv zu programmieren, ohne die Einzelschritte wie in `Java` und `Scala`; sie dürfen aber auch die Einzelschrittreduktion in Hilfsfunktionen implementieren.
- Testen Sie Ihre Implementierung, beispielsweise mittels des Terms aus Beispiel 3.12 im Skript.