Lambda Calculus: A Case for Inductive Definitions

Ralph Matthes
Institut für Informatik der Universität München
Oettingenstraße 67, 80538 München
matthes@informatik.uni-muenchen.de

May 30, 2000

Abstract

This preliminary draft intends to introduce to the subject of lambda calculus and types. A special focus is on the use of inductive definitions.

Contents

1 Overview

2 Introduction to Inductive Definitions

In theoretical computer science, inductive definitions are ubiquitous. Mostly, they appear in the disguise of formal grammars which are idealizations of nat-

ural language grammars. Therefore, this introduction to inductive definitions concentrates on examples of (even context-free) grammars. In [ASU86, Example 4.8] two equivalent grammars for arithmetic expressions are studied for the purpose of illustrating the elimination of left recursion. The first one is (with **id** a set of identifiers):

$$\begin{array}{lcl} E & \to & E+T \mid T \\ T & \to & T*F \mid F \\ F & \to & (E) \mid \mathbf{id} \end{array}$$

The intuition of the non-terminals is given by:

- E: expression = a sum of terms
- T: term = a product of factors
- F: factor = a parenthesized expression or an identifier

Thus we model that in arithmetic expressions, $*$ has a higher precedence than $+$, and that parentheses enforce grouping. The grammar is left recursive and therefore cannot be treated by top-down parsing methods: One runs into the loop $E \to E+T \to E+T+T \to E+T+T+T \to \ldots$

The second grammar is as follows (with empty word ϵ):

$$\begin{array}{lcl} E & \to & TH \\ H & \to & +TH \mid \epsilon \\ T & \to & FK \\ K & \to & *FK \mid \epsilon \\ F & \to & (E) \mid \mathbf{id} \end{array}$$

The idea is to decompose an expression into a term and a string of the form $+T+T+\ldots+T$ (expressed by the auxiliary non-terminal H) while in the first grammar, the decomposition would be into a list of T's with $+$'s inbetween. The same idea is used for the terms.

The claim is that both grammars are equivalent, i. e., that the same expressions can be derived from the non-terminal E. How can we prove that with full mathematical rigour? Answer: By induction. More concretely: By induction on the generation of the expressions according to these grammars. This generation process may be made precise as follows: We simultaneously define the sets $\mathcal{E}$, $\mathcal{T}$ and $\mathcal{F}$ of strings derivable in the first grammar from the non-terminals E, T and F, respectively. This will be done by means of an inductive definition to be expressed by the following rules:

(1) If $e \in \mathcal{E}$ and $t \in \mathcal{T}$ then $e+t \in \mathcal{E}$.

(2) If $t \in \mathcal{T}$ then $t \in \mathcal{E}$.

(3) If $t \in \mathcal{T}$ and $f \in \mathcal{F}$ then $t*f \in \mathcal{T}$.

(4) If $f \in \mathcal{F}$ then $f \in \mathcal{T}$.

(5) If $e \in \mathcal{E}$ then $(e) \in \mathcal{F}$.

(6) If $i \in \mathbf{id}$ then $i \in \mathcal{F}$.

These rules have to be interpreted as describing the generation of the sets $\mathcal{E}$, $\mathcal{T}$ and $\mathcal{F}$, hence the three sets will have all the properties expressed by the rules, and an element only enters one of the three sets if this is possible by one of the six rules. E. g., $e + t$ only enters $\mathcal{E}$ by the first rule, if already $e \in \mathcal{E}$ and $t \in \mathcal{T}$.

Clearly, we are only interested in the set $\mathcal{E}$ of expressions, but we have to define sets for every non-terminal in the grammar.

Analogously, we inductively define five sets $\mathcal{E}', \mathcal{H}', \mathcal{T}', \mathcal{K}', \mathcal{F}'$, corresponding to the five non-terminals of the second grammar:

(a) If $t \in \mathcal{T}'$ and $h \in \mathcal{H}'$ then $th \in \mathcal{E}'$.

(b) If $t \in \mathcal{T}'$ and $h \in \mathcal{H}'$ then $+th \in \mathcal{H}'$.

(c) $\epsilon \in \mathcal{H}'$.

(d) If $f \in \mathcal{F}'$ and $k \in \mathcal{K}'$ then $fk \in \mathcal{T}'$.

(e) If $f \in \mathcal{F}'$ and $k \in \mathcal{K}'$ then $*fk \in \mathcal{K}'$.

(f) $\epsilon \in \mathcal{K}'$.

(g) If $e \in \mathcal{E}'$ then $(e) \in \mathcal{F}'$.

(h) If $i \in \mathbf{id}$ then $i \in \mathcal{F}'$.

Lemma 1 (Equivalence) $\mathcal{E} = \mathcal{E}'$.

Proof It is obvious that we cannot prove $\mathcal{E} = \mathcal{E}'$ in isolation. We first prove $\mathcal{E} \subseteq \mathcal{E}'$ and simultaneously prove $\mathcal{T} \subseteq \mathcal{T}'$ and $\mathcal{F} \subseteq \mathcal{F}'$ by induction on the simultaneous inductive definition of $\mathcal{E}$, $\mathcal{T}$ and $\mathcal{F}$. Proving by induction means arguing on the generation of all the strings in $\mathcal{E}$, $\mathcal{T}$ and $\mathcal{F}$ by help of the rules (1) to (6). Since the rules specify which strings have to be in $\mathcal{E}$, $\mathcal{T}$ or $\mathcal{F}$ before, we may assume that for those strings we already have that they are in $\mathcal{E}'$, $\mathcal{T}'$ and $\mathcal{F}'$, respectively. This is always called the induction hypothesis.

We go through (1) to (6):

(1) If $e + t \in \mathcal{E}$ has been concluded from $e \in \mathcal{E}$ and $t \in \mathcal{T}$, we have to show that $e + t \in \mathcal{E}'$. By induction hypothesis, $e \in \mathcal{E}'$ and $t \in \mathcal{T}'$. Since there is only one rule for $\mathcal{E}'$, we conclude that $e = t'h$ with $t' \in \mathcal{T}'$ and $h \in \mathcal{H}'$. Now, we first prove an auxiliary statement: If $h \in \mathcal{H}'$ and $t \in \mathcal{T}'$ then $h + t \in \mathcal{H}'$. This is proved by induction on $\mathcal{H}'$: If $h \in \mathcal{H}'$ due to rule (b) then $h = +t'h'$ with $t' \in \mathcal{T}'$ and $h' \in \mathcal{H}'$ which entered $\mathcal{H}'$ before h. By induction hypothesis, $h' + t \in \mathcal{H}'$, hence $h + t = +t'h' + t \in \mathcal{H}'$ by rule (b) again. If $h \in \mathcal{H}'$ due to rule (c) then $h = \epsilon$. By rules (c) and (b), $h + t = \epsilon + t = +t = +t\epsilon \in \mathcal{H}'$. By help of this statement, we infer that $h + t \in \mathcal{H}'$, hence $e + t = t'h + t \in \mathcal{E}'$ by rule (a).

(2) If $t \in \mathcal{E}$ has been concluded from $t \in \mathcal{T}$, we have to show that $t \in \mathcal{E}'$, and the induction hypothesis is $t \in \mathcal{T}'$. We are done by rules (c) and (a).

(3) We need another auxiliary lemma: If $k \in \mathcal{K}'$ and $f \in \mathcal{F}'$ then $k * f \in \mathcal{K}'$. The proof and the whole case are similar to (1).

(4) Similar to (2).

(5) If $(e) \in \mathcal{F}$ has been concluded from $e \in \mathcal{E}$, we have to show that $(e) \in \mathcal{F}'$, and the induction hypothesis is $e \in \mathcal{E}'$. So were are done by rule (g).

(6) If $i \in \mathcal{F}$ has been concluded from $i \in \mathbf{id}$, we have to show that $i \in \mathcal{F}'$. This holds by rule (h).

Let us prove $\mathcal{E}' \subseteq \mathcal{E}$. Since the definitions of $\mathcal{E}'$, $\mathcal{H}'$, $\mathcal{T}'$, $\mathcal{K}'$ and $\mathcal{F}'$ are entangled, we need to prove something for all of those. Therefore, we inductively define two auxiliary sets $\mathcal{H}$ and $\mathcal{K}$. $\mathcal{H}$ is defined by mimicking rules (b) and (c):

(b+) If $t \in \mathcal{T}$ and $h \in \mathcal{H}$ then $+th \in \mathcal{H}$.

(c+) $\epsilon \in \mathcal{H}$.

Analogously, $\mathcal{K}$ is defined after the model of (e) and (f):

(e+) If $f \in \mathcal{F}$ and $k \in \mathcal{K}$ then $*fk \in \mathcal{K}$.

(f+) $\epsilon \in \mathcal{K}$.

Now, we can simultaneously prove $\mathcal{E}' \subseteq \mathcal{E}$, $\mathcal{H}' \subseteq \mathcal{H}$, $\mathcal{T}' \subseteq \mathcal{T}$, $\mathcal{K}' \subseteq \mathcal{K}$ and $\mathcal{F}' \subseteq \mathcal{F}$ by induction on the simultaneous inductive definition of $\mathcal{E}'$, $\mathcal{H}'$, $\mathcal{T}'$, $\mathcal{K}'$ and $\mathcal{F}'$. In order to do this, we first prove that (a) to (h) hold when all the primed entities $\mathcal{X}'$ are replaced by the $\mathcal{X}$, for $\mathcal{X} \in \{\mathcal{E}, \mathcal{H}, \mathcal{T}, \mathcal{K}, \mathcal{F}\}$.

(a) Show that if $t \in \mathcal{T}$ and $h \in \mathcal{H}$ then $th \in \mathcal{E}$. We need an auxiliary lemma: If $e \in \mathcal{E}$ and $e' \in \mathcal{E}$ then $e + e' \in \mathcal{E}$. It is proved by induction on the generation of $e' \in \mathcal{E}$: If $e' \in \mathcal{E}$ has been concluded by rule (1) then $e' = e'' + t$ for some previously found $e'' \in \mathcal{E}$ and some $t \in \mathcal{T}$. By induction hypothesis, $e + e'' \in \mathcal{E}$, hence by rule (1), $e + e' = e + e'' + t \in \mathcal{E}$. If $e' \in \mathcal{E}$ has been concluded by rule (2) from $e' \in \mathcal{T}$ then $e + e' \in \mathcal{E}$ by (1).

Now, the claim is proved for arbitrary $t \in \mathcal{T}$ by induction on $h \in \mathcal{H}$: If $h \in \mathcal{H}$ stems from rule (b+) then $h = +t'h'$ for some $t \in \mathcal{T}$ and some previously generated $h' \in \mathcal{H}$. By induction hypothesis, $t'h' \in \mathcal{E}$. Since $t \in \mathcal{E}$, the auxiliary lemma yields $th = t + t'h' \in \mathcal{E}$. If $h \in \mathcal{H}$ by rule (c+) then $h = \epsilon$ and therefore $th = t \in \mathcal{E}$.

(b) By rule (b+): If $t \in \mathcal{T}$ and $h \in \mathcal{H}$ then $+th \in \mathcal{H}$.

(c) $\epsilon \in \mathcal{H}$ by rule (c+).

(d) Show that if $f \in \mathcal{F}$ and $k \in \mathcal{K}$ then $fk \in \mathcal{T}$. In analogy to (a), we first prove $t, t' \in \mathcal{T} \Rightarrow t * t' \in \mathcal{T}$ by induction on $t' \in \mathcal{T}$ and then do induction on $k \in \mathcal{K}$.

(e) By rule (e+): If $f \in \mathcal{F}$ and $k \in \mathcal{K}$ then $*fk \in \mathcal{K}$.

(f) $\epsilon \in \mathcal{K}$ by rule (f+).

(g) By rule (5): If $e \in \mathcal{E}$ then $(e) \in \mathcal{F}$.

(h) If $i \in \mathbf{id}$ then $i \in \mathcal{F}$ by (6).

Therefore, the unprimed entities fulfill the defining clauses of the primed entities. Since the primed entities are assumed to be the smallest sets with those properties (elements only enter if one of the rules applies), we conclude that the primed entities are all included in the unprimed ones, respectively.

Readers not being familiar with this order-theoretic arguments may simply do an inductive proof of $x \in \mathcal{X}' \Rightarrow x \in \mathcal{X}$ for $\mathcal{X} \in \{\mathcal{E}, \mathcal{H}, \mathcal{T}, \mathcal{K}, \mathcal{F}\}$ in traditional style. We only consider this for rule (a): Assume that $th \in \mathcal{E}'$ has been concluded from $t \in \mathcal{T}'$ and $h \in \mathcal{H}'$. By induction hypothesis, $t \in \mathcal{T}$ and $h \in \mathcal{H}$. Therefore, by the validity of (a) with $\mathcal{T}'$, $\mathcal{H}'$ and $\mathcal{E}'$ replaced by $\mathcal{T}$, $\mathcal{H}$ and $\mathcal{E}$, respectively, we get $th \in \mathcal{E}$ and are done. □

zu tun: **das funktionale Programm!!**

3 Lambda Calculus

We introduce λ-calculus in its simplest form: There is only λ-abstraction and application, and no typing whatsoever. Nevertheless, in the motivating examples, a richer signature will freely be used.

3.1 Motivation

Imagine how in mathematical texts you will express that the function f is the squaring function. The easiest way to do that is by saying that $f(x) = x^2$ for all x. Others prefer to write $f := (\cdot)^2$, using the anonymous dot instead of the variable name x. But how would we denote the two-place function g which forms the sum of the squares of the arguments? We could write $g(x, y) := x^2 + y^2$ with the variables x and y. Is $g := (\cdot)^2 + (*)^2$ acceptable? It does not indicate which is the first and which is the second argument. Hence we prefer to have variable names.

If we want to speak about the sum of two squares, why should we first introduce some name g for that, instead of directly writing a mathematical description? In λ-calculus, we would write $\lambda x \lambda y.x^2 + y^2$.

How would we express the first derivative $\partial_2 g$ of $g := \lambda x \lambda y.x^2 + y^2$ w.r.t. the second argument (often written $\frac{\partial g}{\partial y}$)? It is again a function of two arguments:

$$\partial_2 g := \lambda x \lambda y. \lim_{\to 0} \lambda h. \frac{g(x, y + h) - g(x, y)}{h}.$$

In order to use λ-notation as much as possible, we even have written

$$\lim_{\to 0} \lambda h. \frac{g(x, y + h) - g(x, y)}{h}$$

instead of the more usual

$$\lim_{h \to 0} \frac{g(x, y + h) - g(x, y)}{h}.$$

Since the construction does not depend on the concrete definition of g, we may again do an abstraction and define ∂_2 as an operation which takes a two-place function g and returns $\partial_2 g$, i. e., we define

$$\partial_2 := \lambda g \lambda x \lambda y. \lim_{\to 0} \lambda h. \frac{g(x, y + h) - g(x, y)}{h}.$$

Note that the limit need not exist for every g and that therefore, strictly speaking, ∂_2 is not well-defined. So, do not take this example too serious. A much more important issue: We do not need to have the name ∂_2 at hand in order to be able to express our concept of forming the derivative. We may simply say that it is given by

$$\lambda g \lambda x \lambda y. \lim_{\to 0} \lambda h. \frac{g(x, y + h) - g(x, y)}{h}.$$

Let us now calculate $\lambda x.\partial_2(\lambda x \lambda y.x^2 + y^2)\, x\, 2$, i. e., the function taking any argument x to the partial derivative of our previous g w.r.t. y, evaluated at the point $(x, 2)$. By expanding the abbreviation, we get

$$\lambda x. \Big(\lambda g \lambda x \lambda y. \lim_{\to 0} \lambda h. \frac{g(x, y + h) - g(x, y)}{h}\Big)(\lambda x \lambda y.x^2 + y^2)\, x\, 2.$$

Clearly, we now want to replace the formal parameters g, x, y of ∂_2 by the expressions $\lambda x \lambda y.x^2 + y^2, x, 2$, respectively. This yields

$$\lambda x. \lim_{\to 0} \lambda h. \frac{(\lambda x \lambda y.x^2 + y^2)(x, 2 + h) - (\lambda x \lambda y.x^2 + y^2)(x, 2)}{h}.$$

Again we replace the formal parameters x, y of g by the actual arguments. (For this example, we do not distinguish between two subsequent arguments and a pair of arguments.) We get

$$\lambda x. \lim_{\to 0} \lambda h. \frac{(x^2 + (2 + h)^2) - (x^2 + 2^2)}{h}.$$

It is now a matter of algebra to see that the numerator of the fraction has the same value as $4h + h^2$. And $\lim_{\to 0} \lambda h.4 + h$ will certainly be 4 since it is sufficient for that to evaluate $\lambda h.4 + h$ at the argument 0, hence replacing the formal parameter h by 0 in $4 + h$, yielding $4 + 0$ which, by algebra, has the same value as 4. Hence, we have transformed $\lambda x.\partial_2(\lambda x \lambda y.x^2 + y^2)\, x\, 2$ into $\lambda x.4$ which does not allow any further simplification. So we may say that the constant function returning 4 on any input, is the result of our calculation.

In pure λ-calculus, we only model the bare bones of this example: There will be neither a reference to algebraic manipulations nor even to the concept of a limit. So, there will be no squares, no sums, no fractions, no subtractions. There are even no pairs (no tuples). But we may freely λ-abstract "formal" variables and may always apply one expression to another with the intuition that the first expression represents a function and the second one an argument to it. And the only mechanism for "calculation" will be the replacement of formal parameters by actual arguments.

3.2 Pure Untyped Lambda Calculus

Let an infinite set $\mathcal{V}$ of identifiers be given. The identifiers serve as names for variables. We usually denote elements of $\mathcal{V}$ by x, y, z. The identifiers themselves do not matter and will never appear in the presentation. The possibility of having different sets $\mathcal{V}$ is not exploited. The most basic λ-calculus only models functionals (since those functions may take as well functions as arguments, it is preferred to refer to them as functionals). This is done by giving a simple grammar for them—more precisely, by the following inductive definition of terms:

Definition 1 (Terms) *The set $\mathcal{T}$ of terms is inductively given by:*

- *If $x \in \mathcal{V}$ then $x \in \mathcal{T}$.*
- *If $x \in \mathcal{V}$ and $r \in \mathcal{T}$ then $\lambda x r \in \mathcal{T}$.*
- *If $r \in \mathcal{T}$ and $s \in \mathcal{T}$ then $(rs) \in \mathcal{T}$.*

The intention is that $\lambda x r$ models the function $x \mapsto r(x)$ in general mathematical language, where $r(x)$ is just r with the dependency on x indicated (hence only a metasyntactically blown up notation for r). In mathematics one would perhaps prefer to write $r(s)$ instead of the λ-calculus notation (rs) for an application of r to s.

By definition, terms are strings consisting of identifiers in $\mathcal{V}$, parentheses and the greek letter λ. Since we do not aim at studying parsing issues, we will view terms as trees, i. e., we identify a term with its inductive generation by the above definition. And as long as it is clear which tree is meant, we leave out parentheses. We also assume that application associates to the left and use the dot notation: A dot hides a pair of parentheses, which opens at the dot and closes as far to the right as is syntactically possible.

Examples 1 xx *is* (xx). xxx *is* $((xx)x)$. $\omega := \lambda x.xx$ *is* $\lambda x(xx)$. $\Omega := \omega\omega$ *is* $(\lambda x(xx)\lambda x(xx))$.

zu tun: **Bilder von diesen Termen als Bäume!!** Note that all of the examples are quite counterintuitive since x is applied to x itself, hence x is on one hand viewed as a function and on the other as an argument to that function. By typing restrictions to be introduced later, those bizarre terms will be ruled out. Nevertheless, the theorems on pure λ-calculus always also hold for them.

To give at least some intuition for Ω, think of functions as predicates, i. e., functions with boolean range. Then fx means that f holds true of the argument x. Moreover, $\lambda x r$ is the predicate which holds true of x iff $r(x)$ is true. In this way, λ-abstraction becomes set comprehension. In a set-theoretic notation, ω would then correspond to $M := \{x \mid x \in x\}$ and Ω to the assertion that $M \in M$.[1]

[1] Bertrand Russell used the set $M := \{x \mid x \notin x\}$ and the assertion $M \in M$ for his famous set-theoretic paradox. As a remedy he proposed typed systems. The cure for unpleasant behaviour of λ-terms will also be types, i. e., the restriction to terms which follow some typing discipline.

Definition 2 (Free variables) *Define the set $FV(r)$ of variables occurring free in r by recursion on r:*

- $FV(x) := \{x\}$.
- $FV(\lambda x r) := FV(r) \setminus \{x\}$.
- $FV(rs) := FV(r) \cup FV(s)$.

Obviously, $\mathrm{FV}(r)$ is always a finite subset of $\mathcal{V}$. As long as a term has free variables, its intended meaning depends on the assignment of values for the free variables. Hence, it is important to know the names of the variables. On the contrary, the variables occurring in r which are not free (the formal parameters or *bound variables*) are only a means of pointing to the places where to bind the actual argument to the formal parameter. Hence, there is no difference at all between λxx and λyy. More generally, we will never make a distinction between terms differing only in the names of their bound variables as long as the internal references are the same.[2] So for us, $(\lambda x.xy)x \equiv (\lambda z.zy)x$ where $\equiv$ denotes syntactic equality. Surely, $(\lambda z.zy)x \not\equiv (\lambda y.yy)x$.

A word of caution: In the passage from $\lambda x r$ to r, x may become a free variable, and consequently its name matters. Hence in arguments by induction on terms, the case of an abstraction typically reads as follows: "Case $\lambda x r$. By renaming of the bound variable x, we may assume that x does not appear in the set M [given beforehand]. By induction hypothesis for r, ...". This means that although an arbitrary $\lambda x r$ has to be studied, we feel free to rename x in it. We even assume this has already been done and led to the choice x. Then we fix the variable name and break up $\lambda x r$ to yield r with possible free occurrences of x. We may now apply the induction hypothesis to r or rename other bound variables in r, etc.

Definition 3 (Substitution) *Define the result $r[x := s]$ of replacing every free occurrence of the variable x in r by the term s recursively as follows:*

- $x[x := s] := s$
- $y[x := s] := y$ *for* $y \neq x$.
- $(\lambda y r)[x := s] := \lambda y.r[x := s]$ *where we may assume by renaming of the bound variable y that $y \notin \{x\} \cup FV(s)$.*
- $(rt)[x := s] := r[x := s]t[x := s]$.

Note in the case of an abstraction that $y = x$ would forbid any replacement since then x would not be free in $\lambda x r$. If we allowed $y \in \mathrm{FV}(s)$, this free variable of s would be *captured* by the outer λ-abstraction although there has been no functional dependency beforehand. This would be counterintuitive and also make substitution incompatible with renaming of bound variables.

[2] In the literature, this will often be called a variable convention or that terms are considered up to renaming of bound variables or modulo α-equivalence. A mathematically sound justification of this identification process is not as trivial as one might expect it to be.

Lemma 2 $r[x := s][y := t] = r[y := t][x := s[y := t]]$ *for* $x \notin \{y\} \cup FV(t)$.

Proof Induction on r. □

With the notion of substitution at hand, we are now able to define which terms are considered to be computationally equal if our computations are restricted to the replacement of formal parameters by arguments, i. e., when replacing $(\lambda x r)s$ by $r[x := s]$ in any part of the expressions.

Definition 4 (β-equality) *Let* $=_\beta$ *be the congruence relation generated from* $(\lambda x r)s =_\beta r[x := s]$, *i. e.,* $=_\beta$ *is defined inductively by:*

(β) $(\lambda x r)s =_\beta r[x := s]$ *(outer β-equality).*

(ξ) $r =_\beta r' \Rightarrow \lambda x r =_\beta \lambda x r'$ *(β-equality under an abstraction).*

(a) $r =_\beta r' \wedge s =_\beta s' \Rightarrow rs =_\beta r's'$ *(application).*

(r) $r =_\beta r$ *(reflexivity).*

(t) $r =_\beta s \wedge s =_\beta t \Rightarrow r =_\beta t$ *(transitivity).*

(s) $r =_\beta s \Rightarrow s =_\beta r$ *(symmetry).*

(ξ) and (a) are the rules of compatibility with the term formation rules, the rules (r), (t) and (s) express that $=_\beta$ *is an equivalence relation.*

Clearly, it would suffice to restrict the reflexivity rule (r) to $x =_\beta x$ since (ξ) and (a) are present in the system.

Example 2 $\Omega = (\lambda x.xx)\omega =_\beta (xx)[x := \omega] = \omega\omega = \Omega$ *by rule (β). This does not sound interesting since reflexivity would also prove* $\Omega =_\beta \Omega$.

zu tun: **Addition auf Church-Ziffern!!**

Problem 1 *How can we argue that* $\lambda x\lambda y x \neq_\beta \lambda x\lambda y y$*? Clearly, we want them to be different since they represent procedures taking two arguments and returning the first and the second argument, respectively. But how do we know that we cannot use (β) together with (s) in order to produce terms* $(\lambda x r)s$ *out of* $r[x := s]$ *in course of the hypothetical derivation of* $\lambda x\lambda y x =_\beta \lambda x\lambda y y$*?*

By leaving out the rules of equivalence relations, and by adjusting the application rule properly, we arrive at the definition of β-*reduction*.

Definition 5 (β-reduction) *Inductively define the relation* $\to_\beta$ *as follows:*

(β) $(\lambda x r)s \to_\beta r[x := s]$ *(outer β-reduction).*

(ξ) $r \to_\beta r' \Rightarrow \lambda x r \to_\beta \lambda x r'$ *(β-reduction under an abstraction).*

(r) $r \to_\beta r' \Rightarrow rs \to_\beta r's$ *(right application).*

(l) $r \to_\beta r' \Rightarrow sr \to_\beta sr'$ *(left application).*

If $r \to_\beta s$ we say that r reduces by one β-*reduction step* to s.

Example 3 $\Omega = (\lambda x.xx)\omega \to_\beta (xx)[x := \omega] = \omega\omega = \Omega$ *by rule (β). This is interesting since reflexivity has been excluded by passing from* $=_\beta$ *to* $\to_\beta$. *In fact, this example is a major nuisance which will later be removed by typing restrictions.*

Lemma 3 $r \to_\beta r' \Rightarrow FV(r') \subseteq FV(r)$.

Proof Induction on $\to_\beta$. □

For every binary relation $\to$, the *transitive reflexive closure* of $\to$ (i.,e., the least transitive and reflexive relation containing $\to$) is denoted by $\to^*$. Equivalently, $r \to^* s$ iff there are $n \in \mathbb{N}_0$ and $r_0, r_1, \ldots, r_n$ such that $r = r_0$, $\forall i \in \{1, \ldots, n\}. r_{i-1} \to r_i$, and $r_n = s$.

Corollary 4 $r \to_\beta^* r' \Rightarrow FV(r') \subseteq FV(r)$. □

It is easy to see that $\to_\beta^*$ has all the defining properties of $=_\beta$ except symmetry. But the absence of symmetry makes life much easier as will be the theme of the next section. Problem 1 will be solved.

4 Confluence

Although λ-calculus even enjoys confluence, a proof of local confluence is shown first.

Definition 6 (Local confluence) *A binary relation* $\to \subseteq M \times M$ *is locally confluent iff*

$$\forall r \in M \forall r' \in M \forall r'' \in M. r \to r' \wedge r \to r'' \Rightarrow \exists t \in M. r' \to^* t \wedge r'' \to^* t.$$

$$\begin{tikzcd} & r \arrow[dl] \arrow[dr] & \\ r' \arrow[dr, dashed, "*"] & \textit{local confluence} & r'' \arrow[dl, dashed, "*"'] \\ & t & \end{tikzcd}$$

Lemma 5 $\to_\beta$ *is locally confluent.*

Proof Let $r \to_\beta r'$ and $r \to_\beta r''$. By induction on r show that there is a term t such that $r' \to_\beta^* t$ and $r'' \to_\beta^* t$. We distinguish sixteen cases according to the four cases in the generation of $r \to_\beta r'$ and $r \to_\beta r''$, respectively. This will be indicated by a pair of names of rules taken from the definition of β-reduction.

β/β. The situation is trivial:

$$\begin{tikzcd}[column sep=small] & (\lambda x r)s \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & \\ r[x := s] \arrow[dr, equal] & & r[x := s] \arrow[dl, equal] \\ & r[x := s] & \end{tikzcd}$$

ξ/ξ. We have

$$\begin{tikzcd}[column sep=small] & \lambda x r \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & \\ \lambda x r' & & \lambda x r'' \end{tikzcd}$$

due to

$$\begin{tikzcd}[column sep=small] & r \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & \\ r' & & r'' \end{tikzcd}.$$

By induction hypothesis, we get a term t with

$$\begin{tikzcd}[column sep=small] r' \arrow[dr, "*"', "\beta"] & & r'' \arrow[dl, "*", "\beta"'] \\ & t & \end{tikzcd}.$$

Therefore, also

$$\begin{tikzcd}[column sep=small] \lambda x r' \arrow[dr, "*"', "\beta"] & & \lambda x r'' \arrow[dl, "*", "\beta"'] \\ & \lambda x t & \end{tikzcd}.$$

r/r and l/l. Similarly.

r/l. We have

$$\begin{tikzcd}[column sep=small] & rs \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & \\ r's & & rs' \end{tikzcd}$$

with $r \to_\beta r'$ and $s \to_\beta s'$.

Obviously,

$$\begin{tikzcd}[column sep=small] r's \arrow[dr, "\beta"'] & & rs' \arrow[dl, "\beta"] \\ & r's' & \end{tikzcd}.$$

l/r. Symmetric to the preceding case.

r/β. We have

$$\begin{tikzcd}[column sep=small] & (\lambda x r)s \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & \\ ts & & r[x := s] \end{tikzcd}$$

with $\lambda x r \to_\beta t$.

Hence, $t = \lambda x r'$ with $r \to_\beta r'$. Since this implies $r[x := s] \to_\beta r'[x := s]$ (see the substitutivity lemma below),

we arrive at

$$\begin{tikzcd}[column sep=small] ts \arrow[d, equal] & & r[x := s] \arrow[ddl, "\beta"] \\ (\lambda x r')s \arrow[dr, "\beta"'] & & \\ & r'[x := s] & \end{tikzcd}.$$

β/r. Symmetric to the preceding case.

l/β. We have

$$\begin{tikzcd}[column sep=small] & (\lambda x r)s \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & \\ (\lambda x r)s' & & r[x := s] \end{tikzcd}$$

with $s \to_\beta s'$.

By the compatibility lemma below, this implies $r[x := s] \to_\beta^* r[x := s']$ (with as many steps as free occurrences of x in r).

Hence,

$$\begin{tikzcd}[column sep=small] (\lambda x r)s' \arrow[dr, "\beta"'] & & r[x := s] \arrow[dl, "*", "\beta"'] \\ & r[x := s'] & \end{tikzcd}.$$

Notice that it is essential to put $\to_\beta^*$ instead of $\to_\beta$ in this diagram.

r/β. Symmetric to the preceding case.

ξ with other rules. Those six cases are impossible since the other rules need application terms. □

Two properties of $\to_\beta$ have been used in this proof which are now stated more prominently:

Lemma 6 (Substitutivity) *If $r \to_\beta r'$ then $r[x := s] \to_\beta r'[x := s]$.*

Proof Induction on $\to_\beta$. □

Lemma 7 (Compatibility) *If $s \to_\beta s'$ then $r[x := s] \to_\beta^* r[x := s']$, and $r[x := s] \to_\beta r[x := s']$ if x occurs exactly once free in r.*

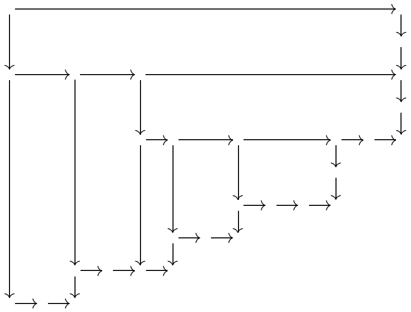
Proof Prove by induction on r that $r[x := s] \to_\beta^* r[x := s']$ with as many β-reduction steps as the number of free occurrences of x in r. □

Definition 7 (Confluence) *A binary relation $\to \subseteq M \times M$ is confluent iff*

$$\forall r \in M \forall r' \in M \forall r'' \in M.\, r \to^* r' \wedge r \to^* r'' \Rightarrow \exists t \in M.\, r' \to^* t \wedge r'' \to^* t.$$

$$\begin{tikzcd}[column sep=small] & r \arrow[dl, "*"'] \arrow[dr, "*"] & \\ r' \arrow[dr, dashed, "*"'] & \textit{confluence} & r'' \arrow[dl, dashed, "*"] \\ & t & \end{tikzcd}$$

Problem 2 *Can we get confluence of $\to$ out of its local confluence? A direct proof fails:*



This could go on with ever increasing complexity. Moreover, there is a binary relation $\to$ which is locally confluent and not confluent: It is given by the following graph on four items: $\longleftarrow\ \rightleftarrows\ \longrightarrow$.

Later we will see a condition for deriving confluence from local confluence.

Theorem 1 $\to_\beta$ *is confluent.*

The proof will occupy the rest of this section.

Definition 8 (Diamond Property) *A binary relation $\to\ \subseteq M \times M$ has the diamond property iff*

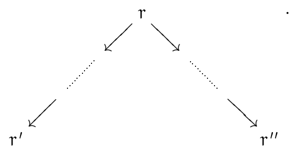
$$\forall r \in M \forall r' \in M \forall r'' \in M.\, r \to r' \wedge r \to r'' \Rightarrow \exists t \in M.\, r' \to t \wedge r'' \to t.$$

$$\begin{tikzcd}
 & r \arrow[dl] \arrow[dr] & \\
r' \arrow[dr, dashed] \arrow[rr, phantom, "\textit{diamond property}"] & & r'' \arrow[dl, dashed] \\
 & t &
\end{tikzcd}$$

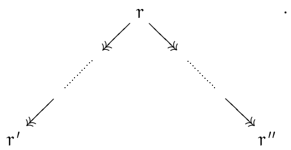
Notice that confluence is a derived concept: $\to$ is confluent iff $\to^*$ has the diamond property. Also note that the case $1/\beta$ of the proof of local confluence immediately lets us find a counterexample to the diamond property for $\to_\beta$, e. g., by starting with $(\lambda x.yxx)((\lambda zr)s)$.

The idea to prove confluence of some $\to$ is as follows: Find a binary relation $\twoheadrightarrow$ (to be read as *parallel reduction*) such that $\to\ \subseteq\ \twoheadrightarrow\ \subseteq\ \to^*$ and $\twoheadrightarrow$ has the diamond property. Confluence of $\to$ follows easily: Assume
$$\begin{tikzcd}
 & r \arrow[dl, "*"'] \arrow[dr, "*"] & \\
r' & & r''
\end{tikzcd},$$

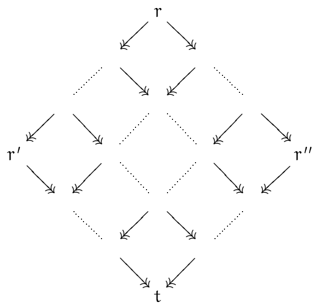
i. e.,

.

Since $\to\ \subseteq\ \twoheadrightarrow$, we have

.

Multiple uses of the diamond property for $\twoheadrightarrow$ give:



Hence, by $\twoheadrightarrow\ \subseteq\ \to^*$, $r' \to^* \ldots \to^* t$ and $r'' \to^* \ldots \to^* t$. Since $\to^*$ is transitive by definition, we finally get $r' \to^* t$ and $r'' \to^* t$. (Of course, this proof with dots could be made more precise by some inductive argument.)

How do we define such a notion $\twoheadrightarrow_\beta$ for $\to_\beta$? Reconsider the crucial case $1/\beta$ in the proof of local confluence. If we want to satisfy $\to_\beta\ \subseteq\ \twoheadrightarrow_\beta$ and the diamond property for $\twoheadrightarrow_\beta$, we have to solve
$$\begin{tikzcd}
 & (\lambda x r)s \arrow[dl, "\beta"'] \arrow[dr, "\beta"] & \\
(\lambda x r)s' \arrow[dr, "\beta"'] & & r[x := s] \arrow[dl, "\beta"] \\
 & ? &
\end{tikzcd}.$$

(Recall that $s \to_\beta s'$ is assumed in $1/\beta$.) Clearly, we want to have $r[x := s']$ as the common reduct. Therefore, our $\twoheadrightarrow_\beta$ must fulfill

$$s \to_\beta s' \Rightarrow r[x := s] \twoheadrightarrow_\beta r[x := s'].$$

This intuitively means that in $r[x := s] \twoheadrightarrow_\beta r[x := s']$, the β-reduction step from s to s' has to be carried out in parallel for each free occurrence of x in r. We will now try to find out which β-reductions on a term can be performed in one pass through it, and will later define $\twoheadrightarrow_\beta$ such that $r \twoheadrightarrow_\beta s$ iff s is the result when performing β-reductions on r in one pass through r. First we define the optimal result r^* of such an action on r.

Definition 9 *(Complete superdevelopment) By recursion on terms r define the term r^* as follows:*

$$\begin{aligned} x^* &:= x,\\ (\lambda x r)^* &:= \lambda x r^*,\\ (rs)^* &:= \begin{cases} t[x := s^*] & \text{if } r^* = \lambda x t,\\ r^* s^* & \text{otherwise.}\end{cases}\end{aligned}$$

In case of a variable, there is nothing to do. Everything we can do with an abstraction $\lambda x r$, takes place in its kernel r. Concerning an application rs, we first look what can be done with r and s. If the result r^* happens to be an abstraction, we even carry out the outer β-reduction step $(\lambda x t)s^* \to_\beta t[x := s^*]$, otherwise we simply apply r^* to s^*. (Further possibilities for β-reduction steps cannot be grasped uniformly by one pass through the term.)

Examples 4 *Consider $r := (\lambda x \lambda y.xyy)st$. In order to calculate r^* we need to know $((\lambda x \lambda y.xyy)s)^*$ which hin turn calls for $(\lambda x \lambda y.xyy)^*$. $(\lambda x \lambda y.xyy)^* = \lambda x \lambda y.xyy$ is plain. Therefore, $((\lambda x \lambda y.xyy)s)^* = (\lambda y.xyy)[x := s^*] = \lambda y.s^* yy$. Since we may assume that $y \notin FV(s)$, we finally get $r^* = (s^* yy)[y := t^*] = s^* t^* t^*$. We see that the complete superdevelopment is capable of replacing a list of formal parameters (here x and y) by the actual arguments (here the results s^* and t^*).*

The complete superdevelopment may also eliminate intervening identities, e. g., $((\lambda xx)(\lambda y r)s)^* = r^*[y := s^*]$. Notice that $(\lambda xx)(\lambda y r)s$ has hidden parentheses, as shown in $((\lambda xx)(\lambda y r))s$, preventing the β-reduction of $(\lambda y r)s$. Nevertheless, $(\lambda xx)(\lambda y r)s$ and $(\lambda y r)s$ have the same complete superdevelopment.

Finally, complete superdevelopments cannot remove every possibility for β-reduction, e. g., for $x \notin FV(s)$, we get $((\lambda x.xs)(\lambda y r))^* = (\lambda y r^*)s^*$ (use that $FV(s^*) \subseteq FV(s)$ shown below) which can further be β-reduced to $r^*[y := s^*]$ (which of course may again be β-reducible depending on r and s).

Lemma 8 $r \to^* r^*$.

Proof Induction on r. □

As a corollary, we get $FV(r^*) \subseteq FV(r)$.

Now, we want to see how $\twoheadrightarrow_\beta$ should be defined to ensure that $r \twoheadrightarrow_\beta r^*$: Since $x^* = x$, we have to require $x \twoheadrightarrow_\beta x$. Concerning $(\lambda x r)^* = \lambda x r^*$, we will already know that $r \twoheadrightarrow_\beta r^*$ and have to show that $\lambda x r \twoheadrightarrow_\beta \lambda x r^*$. This suggests to require, more generally, $r \twoheadrightarrow_\beta r' \Rightarrow \lambda x r \twoheadrightarrow_\beta \lambda x r'$. Similarly, in case $r^* = \lambda x t$, we will already know that $r \twoheadrightarrow_\beta \lambda x t$ and $s \twoheadrightarrow_\beta s^*$ and have to show that $rs \twoheadrightarrow_\beta t[x := s^*]$. This suggest to require $r \twoheadrightarrow_\beta \lambda x t \wedge s \twoheadrightarrow_\beta s' \Rightarrow rs \twoheadrightarrow_\beta t[x := s']$. Finally, if we already know $r \twoheadrightarrow_\beta r^*$ and $s \twoheadrightarrow_\beta s^*$, we want to conclude $rs \twoheadrightarrow_\beta r^* s^*$, if r^* fails to be an abstraction. This suggests to require $r \twoheadrightarrow_\beta r' \wedge s \twoheadrightarrow_\beta s' \Rightarrow rs \twoheadrightarrow_\beta r's'$, which seems more reasonable if we also allow r' to be an abstraction. These four requirements will be the definition of $\twoheadrightarrow_\beta$.

Definition 10 (Parallel β-reduction) *Define the binary relation $\twoheadrightarrow_\beta$ inductively as follows:*

- *(β)* $r \twoheadrightarrow_\beta \lambda x t \wedge s \twoheadrightarrow_\beta s' \Rightarrow rs \twoheadrightarrow_\beta t[x := s']$.
- *(ξ)* $r \twoheadrightarrow_\beta r' \Rightarrow \lambda x r \twoheadrightarrow_\beta \lambda x r'$.
- *(a)* $r \twoheadrightarrow_\beta r' \wedge s \twoheadrightarrow_\beta s' \Rightarrow rs \twoheadrightarrow_\beta r's'$.
- *(v)* $x \twoheadrightarrow_\beta x$.

Notice that for given r, there may be several terms r' such that $r \twoheadrightarrow_\beta r'$, e. g., in case $r \twoheadrightarrow_\beta \lambda x t$ and $s \twoheadrightarrow_\beta s'$, we have that $rs \twoheadrightarrow_\beta t[x := s']$ by (β) and $rs \twoheadrightarrow_\beta (\lambda x t)s'$ by (a). (Recall that there are possibilities that those terms coincide: $t = xx$ and $s' = \omega$.)

Lemma 9 $r \twoheadrightarrow_\beta r^*$.

Proof By induction on r we verify that $\twoheadrightarrow_\beta$ indeed has the property which led us to the definition. □

Lemma 10 *$\twoheadrightarrow_\beta$ is reflexive.*

Proof By induction on r show $r \twoheadrightarrow_\beta r$. This does not need rule (β). □

Corollary 11 $\to_\beta \subseteq \twoheadrightarrow_\beta$.

Proof By induction on $\to_\beta$. Since $\to_\beta$ is the smallest set with its defining properties, we simply have to show those properties for $\twoheadrightarrow_\beta$, i. e., we have to show:

- (β) $(\lambda x r)s \twoheadrightarrow_\beta r[x := s]$.
- (ξ) $r \twoheadrightarrow_\beta r' \Rightarrow \lambda x r \twoheadrightarrow_\beta \lambda x r'$.
- (r) $r \twoheadrightarrow_\beta r' \Rightarrow rs \twoheadrightarrow_\beta r's$.
- (l) $r \twoheadrightarrow_\beta r' \Rightarrow sr \twoheadrightarrow_\beta sr'$.

This uses reflexivity of $\twoheadrightarrow_\beta$ at least four times. □

Lemma 12 $\twoheadrightarrow_\beta \subseteq \to_\beta^*$.

Proof By induction on $\twoheadrightarrow_\beta$. Since $\twoheadrightarrow_\beta$ is the smallest set with its defining properties, we simply have to show those properties for $\to_\beta^*$, i. e., we have to show:

- (β) $r \to_\beta^* \lambda x t \wedge s \to_\beta^* s' \Rightarrow rs \to_\beta^* t[x := s']$.

(ξ) $r \to_\beta^* r' \Rightarrow \lambda x r \to_\beta^* \lambda x r'$.

(a) $r \to_\beta^* r' \wedge s \to_\beta^* s' \Rightarrow rs \to_\beta^* r's'$.

(v) $x \to_\beta^* x$.

They clearly hold. □

The only remaining task is to prove the diamond property for $\twoheadrightarrow_\beta$. A crucial case (with $r \twoheadrightarrow_\beta r'$ and $s \twoheadrightarrow_\beta s'$) will be

$$\begin{tikzcd}
 & (\lambda x r)s \arrow[dl, two heads, "\beta"'] \arrow[dr, two heads, "\beta"] & \\
(\lambda x r')s' \arrow[dr, two heads, "\beta"'] & & r[x := s] \arrow[dl, two heads, "\beta"] \\
 & ? &
\end{tikzcd}$$

.

Clearly, we want to have $r'[x := s']$ as the common reduct. This calls for the following

Lemma 13 $r \twoheadrightarrow_\beta r' \wedge s \twoheadrightarrow_\beta s' \Rightarrow r[x := s] \twoheadrightarrow_\beta r'[x := s']$.

Proof Induction on $r \twoheadrightarrow_\beta r'$. We only consider the most technical case (β): Assume that $ru \twoheadrightarrow_\beta t[y := u']$ has been derived from $r \twoheadrightarrow_\beta \lambda y t$ and $u \twoheadrightarrow_\beta u'$. By the inductive hypothesis, $r[x := s] \twoheadrightarrow_\beta (\lambda y t)[x := s'] = \lambda y.t[x := s']$ (since we may assume that $y \notin \{x\} \cup FV(s')$), and also $u[x := s] \twoheadrightarrow_\beta u'[x := s']$. Hence,

$$r[x := s]u[x := s] \twoheadrightarrow_\beta t[x := s'][y := u'[x := s']] = t[y := u'][x := s'],$$

by Lemma 2. Therefore, $(ru)[x := s] \twoheadrightarrow_\beta t[y := u'][x := s']$, as required. □

Note that this lemma comprises substitutivity and compatibility (cf. Lemma 6 and Lemma 7 for $\to_\beta$).

Lemma 14 (Maximality) *If* $r \twoheadrightarrow_\beta r'$ *then* $r' \twoheadrightarrow_\beta r^*$.

$$\begin{tikzcd}
 & r \arrow[dl, two heads, "\beta"] \\
r' \arrow[dr, dashed, two heads, "\beta"'] & \textit{maximality of } (.)^* \\
 & r^*
\end{tikzcd}$$

Thus r^* is the optimum what can be done by β-reduction in one pass, and $\twoheadrightarrow_\beta$ gives all the possibilities one has in one pass (including doing nothing since $\twoheadrightarrow_\beta$ is reflexive), and if in the passage from r to r', the optimal way has not been chosen (hence $r' \neq r^*$), r^* can be reached in another pass through r'.

Proof Induction on $r \twoheadrightarrow_\beta r'$. We only look at the interesting cases.

(β). Case $rs \twoheadrightarrow_\beta t[x := s']$ thanks to $r \twoheadrightarrow_\beta \lambda x t$ and $s \twoheadrightarrow_\beta s'$. By induction hypothesis, $\lambda x t \twoheadrightarrow_\beta r^*$ and $s' \twoheadrightarrow_\beta s^*$. By inspection of the rules of $\twoheadrightarrow_\beta$, it is clear that $\lambda x t \twoheadrightarrow_\beta r^*$ can only be derived by rule (ξ). Therefore, r^* has to be a λ-abstraction $r^* = \lambda x t'$ and $t \twoheadrightarrow_\beta t'$. Hence, $(rs)^* = t'[x := s^*]$. By the preceding lemma, $t[x := s'] \twoheadrightarrow_\beta t'[x := s^*]$.

(a). By induction hypothesis, $r' \twoheadrightarrow_\beta r^*$ and $s' \twoheadrightarrow_\beta s^*$. We have to show that $r's' \twoheadrightarrow_\beta (rs)^*$. If $r^* = \lambda x t$ then, by rule (β), $r's' \twoheadrightarrow_\beta t[x := s^*] = (rs)^*$. Otherwise, by rule (a), $r's' \twoheadrightarrow_\beta r^* s^* = (rs)^*$. □

Notice that Lemma 9 is merely the special case with $r' = r$, hence its proof is superfluous since we did not use it to prove maximality.

Corollary 15 $\twoheadrightarrow_\beta$ *has the diamond property.*

Proof Given r, we already know the term t which for any r' and r'' such that $r \twoheadrightarrow_\beta r'$ and $r \twoheadrightarrow_\beta r''$ fulfills $r' \twoheadrightarrow_\beta t$ and $r'' \twoheadrightarrow_\beta t$: It is r^*. □

To conclude, we have found a binary relation $\twoheadrightarrow_\beta$ with $\to_\beta \subseteq \twoheadrightarrow_\beta \subseteq \to_\beta^*$ and the diamond property. Hence, $\to_\beta$ is confluent.

zu tun: **Anwendung in den Übungsaufgaben!!**

5 Weak and Strong Normalization

Definition 11 *Given a binary relation* $\to \subseteq M \times M$, *an element* $r \in M$ *is* in normal form *iff there is no* $s \in M$ *such that* $r \to s$.

Lemma 16 (β-normal forms) *The set* nf *of terms in normal form w. r. t.* $\to_\beta$ *equals the set* NF, *defined by induction as follows:*

- *If* $\vec{r} \subset \mathsf{NF}$ *then* $x\vec{r} \in \mathsf{NF}$.
- *If* $r \in \mathsf{NF}$ *then* $\lambda x r \in \mathsf{NF}$.

The notation with vectors deserves a short explanation: $\vec{r}$ denotes a finite list of terms. This is not part of the syntax of λ-calculus but a metasyntactic device to communicate terms. Otherwise, one would have to consider infinitely many rules of the form: $r_1, \dots, r_n \in \mathsf{NF} \Rightarrow x r_1 \dots r_n \in \mathsf{NF}$ for every $n \in \mathbb{N}_0$.

Proof $\mathsf{nf} \subseteq \mathsf{NF}$ is proved by induction on terms: $x \in \mathsf{NF}$. Let rs be in nf, hence also $r, s \in \mathsf{nf}$ since a reduction in r or s gives rise to a reduction in rs. By induction hypothesis $r, s \in \mathsf{NF}$. If r were of the form $\lambda x t$, then certainly $rs \notin \mathsf{nf}$. Hence, $r = x\vec{s}$ for $\vec{s} \subset \mathsf{NF}$. We conclude $rs = x\vec{s}s \in \mathsf{NF}$. Let $\lambda x r$ be in nf. Hence, also $r \in \mathsf{nf}$. By induction hypothesis, $r \in \mathsf{NF}$ which yields $\lambda x r \in \mathsf{NF}$.

$\mathsf{NF} \subseteq \mathsf{nf}$ is proved by induction on the definition of NF. Let $x\vec{r}$ be in NF thanks to $\vec{r} \subset \mathsf{NF}$. By induction hypothesis, $\vec{r} \subset \mathsf{nf}$, hence also $x\vec{r} \in \mathsf{nf}$ since any possible reduction of $x\vec{r}$ has to happen in one of the $\vec{r}$. Let $\lambda x r$ be in NF due to $r \in \mathsf{NF}$. By induction hypothesis, $r \in \mathsf{nf}$. Since reductions on $\lambda x r$ can only happen by help of the rule ξ, also $\lambda x r \in \mathsf{nf}$. □

Examples 5 $\omega = \lambda x.xx \in \mathsf{NF}$. $\Omega = \omega\omega \notin \mathsf{NF}$.

Definition 12 *Given a binary relation* $\to\, \subseteq M \times M$, *an element* $r \in M$ *is* weakly normalizing *iff there is an* $s \in M$ *in normal form such that* $r \to^* s$.

Let wn be the set of terms which are weakly normalizing w. r. t. $\to_\beta$.

Examples 6 $\mathsf{nf} \subset \mathsf{wn}$. $\Omega \notin \mathsf{wn}$ *since the only term* t *such that* $\Omega \to_\beta^* t$ *is* Ω *itself.* $(\lambda x \lambda y.y)\Omega \in \mathsf{wn}$ *since its reduct* λyy *is in* nf.

Definition 13 *Given a binary relation* $\to\, \subseteq M \times M$, *an element* $r \in M$ *is* strongly normalizing *iff there is no infinite reduction sequence starting from* r, *i. e., iff there are no* $r_0, r_1, \ldots$ *such that* $r_0 = r$ *and* $r_i \to r_{i+1}$ *for every* i.

This definition has a major drawback: It is formulated as a negative statement, namely the non-existence of an infinite reduction sequence. This can be avoided and even the reference to infinity be removed in the following inductive characterization.

Definition 14 *Given a binary relation* $\to\, \subseteq M \times M$, *its* accessible part acc *is inductively defined by:*

$$\forall r \in M.(\forall s \in M.r \to s \Rightarrow s \in \mathsf{acc}) \Rightarrow r \in \mathsf{acc}.$$

In other words: An object $r \in M$ qualifies for acc if every one-step reduct s (any object $s \in M$ such that $r \to s$) already qualified for acc.

Clearly, if r is in normal form, then there is nothing to check, and r enters acc immediately.

Lemma 17 *The set* acc *is the set of strongly normalizing terms.*

Proof How can we prove that acc only contains strongly normalizing terms? By induction on the inductive definition of acc! Let r be in acc thanks to $s \in \mathsf{acc}$ for every s such that $r \to s$. Assume, we had an infinite reduction sequence starting from r, i. e., some $s_{-1}, s_0, s_1, \ldots$ such that $r = s_{-1}$ and for all $i \geq -1$: $s_i \to s_{i+1}$. Setting $s := s_0$, we have $r \to s$, and an infinite reduction sequence starting from s. This contradicts the induction hypothesis which tells us that s is already strongly normalizing (since s has entered acc before r entered it).

We now show that an $r \in M \setminus \mathsf{acc}$ is not strongly normalizing by constructing $r_0, r_1, \ldots$ such that $r_0 = r$ and for all i, $r_i \to r_{i+1}$ and $r_i \in M \setminus \mathsf{acc}$. Set $r_0 := r$. Assume that $r_0, \ldots, r_i$ have already been constructed. Since $r_i \notin \mathsf{acc}$, it did not qualify for acc, hence there has to be some one-step reduct r_{i+1} of r_i which also did not qualify for acc, i. e., some $r_{i+1} \in M$ such that $r_i \to r_{i+1}$ and $r_{i+1} \notin \mathsf{acc}$. □

Define $\mathsf{sn} := \mathsf{acc}_{\to_\beta}$. Hence, sn is the set of λ-terms which do not have an infinite sequence of β-reductions.

Example 7 $(\lambda x \lambda y.y)\Omega \notin \mathsf{sn}$ *since we get a constant infinite reduction sequence by* β*-reductions of* Ω.

Lemma 18 (Characterization of the strongly normalizing terms) *The set* sn *equals the set* SN, *defined by induction as follows:*

- *If* $\vec{r} \subset \mathsf{SN}$ *then* $x\vec{r} \in \mathsf{SN}$.
- *If* $r \in \mathsf{SN}$ *then* $\lambda xr \in \mathsf{SN}$.
- *If* $r[x := s]\vec{s} \in \mathsf{SN}$ *and* $s \in \mathsf{SN}$ *then* $(\lambda xr)s\vec{s} \in \mathsf{SN}$.

Proof The more important part is proving that $\mathsf{SN} \subseteq \mathsf{sn}$. Clearly, this has to be done by induction on the inductive definition of SN. We will profit from a deeper understanding of induction: Proving by induction on an inductively defined set I, that every inhabitant of I has some property expressed by a set M, amounts to showing that this set M has each of the defining properties of I (where—of course—the reference to I has to be replaced by M). Why is that so? The very idea of an inductive definition of I is that the rules describing its definition give all the possibilities. No element may enter I without the application of one of the rules. Hence, I becomes the smallest set with those closure properties, and if M is an arbitrary set fulfilling the closure properties, then $I \subseteq M$, which was our goal to prove.

Therefore, it suffices to prove

- If $\vec{r} \subset \mathsf{sn}$ then $x\vec{r} \in \mathsf{sn}$.
- If $r \in \mathsf{sn}$ then $\lambda xr \in \mathsf{sn}$.
- If $r[x := s]\vec{s} \in \mathsf{sn}$ and $s \in \mathsf{sn}$ then $(\lambda xr)s\vec{s} \in \mathsf{sn}$.

We first give a proof which does not argue by induction on the accessible part but by the non-existence of infinite reduction sequences. Imagine an infinite reduction sequence starting from $x\vec{r}$. In every step there has to be a reduction in one of the $\vec{r}$. Since there are only finitely many terms in $\vec{r}$, there has to be an r_i in $\vec{r}$ which faces infinitely many reduction steps. Contradiction. Assume an infinite reduction sequence with first term λxr. Since the reductions can only take place in r, we get an infinite reduction sequence starting from r. Contradiction. Imagine an infinite reduction sequence starting from $(\lambda xr)s\vec{s}$. Since $r \to_\beta r'$ implies $r[x := s] \to_\beta r'[x := s]$, and consequently $r[x := s]\vec{s} \to_\beta r'[x := s]\vec{s}$, there can neither be an infinite reduction sequence starting from r nor from s nor from any of the $\vec{s}$. Hence, there has to be a reduct of $(\lambda xr)s\vec{s}$ in the infinite reduction sequence which does no longer have the shape $(\lambda xr')s'\vec{s}\,'$ with $r \to_\beta^* r'$, $s \to_\beta^* s'$, $s_i \to_\beta^* s_i'$. Assume this were the last successive term of this shape. The next term then has to be $r'[x := s']\vec{s}\,'$, still followed by an infinite reduction sequence. From Lemma 6 and Lemma 7, we get $r[x := s]\vec{s} \to_\beta^* r'[x := s']\vec{s}\,'$. Therefore, we also get an infinite reduction sequence starting with $r[x := s]\vec{s}$. Contradiction.

Now we redo the proof for the last clause and completely avoid Lemma 17: Show that $(\lambda xr)s\vec{s} \in \mathsf{sn}$ by main induction on $s \in \mathsf{sn}$ and side induction on $r[x := s]\vec{s} \in \mathsf{sn}$. Hence, prove $t \in \mathsf{sn}$ for every t such that $(\lambda xr)s\vec{s} \to_\beta t$. The following reductions are possible.

$(\lambda x r)s\vec{s} \to_\beta (\lambda x r')s\vec{s}$. Then $r[x := s]\vec{s} \to_\beta r'[x := s]\vec{s}$ by substitutivity, hence by side induction hypothesis $(\lambda x r')s\vec{s} \in \mathsf{sn}$.

$(\lambda x r)s\vec{s} \to_\beta (\lambda x r)s'\vec{s}$. Then $r[x := s]\vec{s} \to^*_\beta r[x := s']\vec{s}$ by compatibility, hence also $r[x := s']\vec{s} \in \mathsf{sn}$. The main induction hypothesis yields $(\lambda x r)s'\vec{s} \in \mathsf{sn}$.

$(\lambda x r)s\vec{s} \to_\beta (\lambda x r)s\vec{s}\,'$. Then $r[x := s]\vec{s} \to_\beta r[x := s]\vec{s}\,'$, hence by side induction hypothesis $(\lambda x r)s\vec{s}\,' \in \mathsf{sn}$.

$(\lambda x r)s\vec{s} \to_\beta r[x := s]\vec{s} \in \mathsf{sn}$ by assumption.

In order to clarify the intricate structure of the argument we prove this once again and spell out the nesting of inductions precisely. Main induction on $s \in \mathsf{sn}$ amounts to showing the defining property of sn for the set

$$M := \{s \mid \forall r, \vec{s}.r[x := s]\vec{s} \in \mathsf{sn} \;\Rightarrow\; (\lambda x r)s\vec{s} \in \mathsf{sn}\}.$$

So assume that any one-step reduct of s is in M. For $s \in M$ we do side induction on $r[x := s]\vec{s} \in \mathsf{sn}$, i.e., we show the defining property of sn for the set

$$N := \{t' \mid t' \in \mathsf{sn} \wedge \forall r, \vec{s}.t' = r[x := s]\vec{s} \Rightarrow (\lambda x r)s\vec{s} \in \mathsf{sn}\}.$$

So assume t' is a term and each immediate reduct is in N. We have to show $t' \in N$. $t' \in \mathsf{sn}$ follows from the definition of sn, since $N \subseteq \mathsf{sn}$. If t' has the form $r[x := s]\vec{s}$ we have to show $(\lambda x r)s\vec{s} \in \mathsf{sn}$, so we prove $t \in \mathsf{sn}$ for every one-step reduct t. Henceforward, each clause of the proof in the above proof can be recast using the pending assumptions.

For the other direction $\mathsf{sn} \subseteq \mathsf{SN}$, we do induction on sn. Hence, we have to show $\forall r.(\forall r'.r \to_\beta r' \Rightarrow r' \in \mathsf{SN}) \Rightarrow r \in \mathsf{SN}$. This is done by induction on the term r. For this to work, note that every term has exactly one of the following shapes (to be proved by induction on terms): $x\vec{r}, \lambda x r, (\lambda x r)s\vec{s}$. And because we do induction on terms, we may use the induction hypothesis for r_i in the case $x\vec{r}$, for r in the case $\lambda x r$ (no surprise!) and for s in the case $(\lambda x r)s\vec{s}$. The rest is routine verification. □

6 Simple and Intersection Types

We have seen that there are terms which are not strongly normalizing and even terms which fail to be weakly normalizing. The idea was to use the unintuitive term $\omega = \lambda x.xx$ and apply it to itself. The first examples in section 3.1 dealing with derivatives of functions on the reals were modeled without using the information on the domains and ranges of the functions at hand. E. g., we never expressed that we considered the squaring function

$$f : \left\{ \begin{array}{ccc} \mathbb{R} & \to & \mathbb{R} \\ x & \mapsto & x^2 \end{array} \right.$$

This more specific information may be expressed by $\lambda x.x^2 : \mathbb{R} \to \mathbb{R}$, meaning that $\lambda x.x^2$ is considered as a function from the reals to the reals. That our

$\lambda x.x^2$ belongs to the function space $\mathbb{R} \to \mathbb{R}$ is called type information. If we again ignore the fact that limits do not always exist, we might type $\lim_{\to 0}$ with $(\mathbb{R} \to \mathbb{R}) \to \mathbb{R}$, reflecting that $\lim_{\to 0}$ takes a function from $\mathbb{R}$ to $\mathbb{R}$ and returns a real. What would be the type of ∂_2? Again by ignoring undefinedness problems, we would give it the type $(\mathbb{R} \to \mathbb{R} \to \mathbb{R}) \to \mathbb{R} \to \mathbb{R} \to \mathbb{R}$: The first argument is a two-place real-valued function on $\mathbb{R}$, the second and third arguments are reals again, and the result is again in $\mathbb{R}$.

Our semantic intuition is a world of functionals, i. e., functions taking functions and even functionals as arguments. In that *simply-typed* world, there is no place for $\omega = \lambda x.xx$. Since ω is already in normal form, it is very well-behaved w. r. t. β-reduction. So, in some sense, we have to give up too much when restricting to the simply-typed world.

If we loosen the typing concept to include *intersection types*, we also cover ω: Simply allow intersections of already formed types. If some x "lives" in $\mathbb{N} \to \mathbb{N}$ as well as in $\mathbb{N}$, there is no hesitation to assert that xx "lives" in $\mathbb{N}$. Hence, we would type ω with $((\mathbb{N} \to \mathbb{N}) \cap \mathbb{N}) \to \mathbb{N}$. Are there objects which can been seen as numbers and as number-theoretic functions? Not in the set-theoretic model we were appealing to, but in recursion-theoretic models (where recursive functions are coded by numbers). But this is not the point. We only need the soundness of the implications $x : (\mathbb{N} \to \mathbb{N}) \cap \mathbb{N} \Rightarrow x : \mathbb{N} \to \mathbb{N} \wedge x : \mathbb{N}$ and $x : \mathbb{N} \to \mathbb{N} \wedge x : \mathbb{N} \Rightarrow xx : \mathbb{N}$ to conclude a correct type for ω.

6.1 Simply-Typed Lambda Calculus

Assume a set $\mathcal{V}_\mathrm{T}$ of identifiers. These serve as names for atomic types. Their generic name will be ι. Typical examples would be nat and real, representing $\mathbb{N}$ and $\mathbb{R}$ by mere syntax.

Definition 15 (Simple types) *The set $\mathcal{T}_\mathrm{s}$ of simple types is inductively given by:*

- *If $\iota \in \mathcal{V}_\mathrm{T}$ then $\iota \in \mathcal{T}_\mathrm{s}$.*
- *If $\rho \in \mathcal{T}_\mathrm{s}$ and $\sigma \in \mathcal{T}_\mathrm{s}$ then $(\rho \to \sigma) \in \mathcal{T}_\mathrm{s}$.*

Hence, the simple types are nothing but strings of symbols built up from the given symbols in $\mathcal{V}_\mathrm{T}$ by help of the binary $\to$, syntactically representing the construction of function spaces. As for the terms, we avoid parentheses as much as possible, and therefore assume $\to$ to be right-associative (application was meant to be left-associative).

Examples 8 $\rho \to \sigma$ *is* $(\rho \to \sigma)$. $\mathsf{nat} \to \mathsf{nat} \to \mathsf{nat}$ *is* $(\mathsf{nat} \to (\mathsf{nat} \to \mathsf{nat}))$. *The second example represents the space of number-theoretic functions of two arguments and explains why $\to$ has been declared right-associative. More complicated things need parentheses:* $(\mathsf{nat} \to \mathsf{nat}) \to \mathsf{nat}$ *represents the functionals taking number-theoretic functions and returning values in $\mathbb{N}$. Here, we only suppressed the outer parentheses.*

As a further abbreviation, set $\rho_1, \dots, \rho_n \to \sigma := \rho_1 \to \dots \to \rho_n \to \sigma$ and also use this with $\vec{\rho} := \rho_1, \dots, \rho_n$, hence $\vec{\rho} \to \sigma$ is a well-defined type. By induction on types, it is easy to verify that every type uniquely decomposes into $\vec{\rho} \to \iota$ for some $\iota \in \mathcal{V}_T$ and arbitrary types $\vec{\rho}$.

Definition 16 (Simple typing) *The relation $\Gamma \vdash r : \rho$ (term r has type ρ in context Γ) is inductively defined by the following rules:*

$$\frac{}{\Gamma, x : \rho \vdash x : \rho}(V) \qquad \frac{\Gamma, x : \rho \vdash r : \sigma}{\Gamma \vdash \lambda x r : \rho \to \sigma}(\to_I) \qquad \frac{\Gamma \vdash r : \rho \to \sigma \qquad \Gamma \vdash s : \rho}{\Gamma \vdash rs : \sigma}(\to_E)$$

From these clauses, it is clear that only finite lists of variable names with types (each pair separated by a colon) may occur on the left-hand side of $\vdash$. We moreover restrict those contexts *to lists with pairwise disjoint variable names. Hence, by writing $\Gamma, x : \rho$, it is implicit that x does not occur in Γ, written $x \notin \Gamma$. We also syntactically identify contexts which are permutations of each other. This amounts to using the following exchange rule without including it into the typing system: If Δ is a permutation of the pairs in Γ and $\Gamma \vdash r : \rho$ then $\Delta \vdash r : \rho$ (Exchange 1).*

Note that our notation always suggests that we have lists on the left-hand side of $\vdash$. In fact, we consider the contexts to be sets of variable declarations without inconsistencies arising from multiple declarations for some variable. The empty context will be denoted by $\emptyset$, hence $\vdash r : \rho$ and $\emptyset \vdash r : \rho$ mean the same assertion.

Some comments on the rules: Rule (V) is nothing but lookup in the context, rule $(\to_I)$, the $\to$-introduction rule, follows our intuition: If r gets type σ under the assumption that the possibly free variable x is of type ρ, then the abstraction $\lambda x r$ gets the function type $\rho \to \sigma$. The $\to$-elimination rule $(\to_E)$ requires that the argument s to r of type $\rho \to \sigma$ has to have the domain type ρ, and states that the result of the application has the range type σ.

Lemma 19 *There are no Γ and ρ such that $\Gamma \vdash \omega : \rho$.*

Proof If there were Γ and ρ, then the rule $(\to_I)$ would have been applied last. Hence, we would have $\rho = \rho_1 \to \rho_2$ and $\Gamma, x : \rho_1 \vdash xx : \rho_2$. This can only be derived by help of $(\to_E)$. Hence, there is a type σ such that $\Gamma, x : \rho_1 \vdash x : \sigma \to \rho_2$ and $\Gamma, x : \rho_1 \vdash x : \sigma$. These can only be found by applying (V), hence $\sigma \to \rho_2 = \rho_1 = \sigma$. Contradiction (the equality is the equality of strings). □

6.2 Lambda Calculus with Intersection Types

Definition 17 (Intersection types) *The set $\mathcal{T}_i$ of intersection types is inductively given by:*

- *If $\iota \in \mathcal{V}_T$ then $\iota \in \mathcal{T}_i$.*
- *If $\rho \in \mathcal{T}_i$ and $\sigma \in \mathcal{T}_i$ then $(\rho \to \sigma) \in \mathcal{T}_i$.*
- *If $\rho \in \mathcal{T}_i$ and $\sigma \in \mathcal{T}_i$ then $(\rho \cap \sigma) \in \mathcal{T}_i$.*

Since we have more types, we also have additional rules for typing. Nevertheless, we will use the same symbol $\vdash$ for both simple typing and intersection typing.

Definition 18 (Intersection typing) *The relation $\Gamma \vdash r : \rho$ (term r has type ρ in context Γ) is inductively defined by the following rules:*

$$\frac{}{\Gamma, x : \rho \vdash x : \rho}(V) \qquad \frac{\Gamma, x : \rho \vdash r : \sigma}{\Gamma \vdash \lambda x r : \rho \to \sigma}(\to_I) \qquad \frac{\Gamma \vdash r : \rho \to \sigma \qquad \Gamma \vdash s : \rho}{\Gamma \vdash rs : \sigma}(\to_E)$$

$$\frac{\Gamma \vdash r : \rho \qquad \Gamma \vdash r : \sigma}{\Gamma \vdash r : \rho \cap \sigma}(\cap_I) \qquad \frac{\Gamma \vdash r : \rho_1 \cap \rho_2 \qquad i \in \{1, 2\}}{\Gamma \vdash r : \rho_i}(\cap_E)$$

The conventions concerning contexts are as for simple typing.

Examples 9 $x : (\rho \to \rho) \cap \rho \vdash x : \rho \to \rho$ *and* $x : (\rho \to \rho) \cap \rho \vdash x : \rho$, *hence* $x : (\rho \to \rho) \cap \rho \vdash xx : \rho$ *and finally* $\vdash \omega : ((\rho \to \rho) \cap \rho) \to \rho$. *In Lemma 24, we will see that for no Γ and ρ, $\Gamma \vdash \Omega : \rho$ holds, i. e., Ω is not typable with intersection types.*

Lemma 20 (Derived rules)

- *If $\Gamma \vdash r : \rho$ then $\Gamma, x : \sigma \vdash r : \rho$ (Weakening 1).*
- *If $\Gamma, x : \sigma \vdash r : \rho$ then $\Gamma, x : \sigma \cap \tau \vdash r : \rho$ (Weakening 2).*
- *If $\Gamma, x : \sigma \vdash r : \rho$ and $x \notin FV(r)$ then $\Gamma \vdash r : \rho$ (Strengthening).*
- *If $\Gamma, x : \rho \cap \sigma \vdash r : \tau$ then $\Gamma, x : \sigma \cap \rho \vdash r : \tau$ (Exchange 2).*
- *If $\Gamma, x : (\rho_1 \cap \rho_2) \cap \rho_3 \vdash r : \tau$ then $\Gamma, x : \rho_1 \cap (\rho_2 \cap \rho_3) \vdash r : \tau$ (Exchange 3).*

Proof By rule induction on $\vdash$. □

Lemma 21 (Substitution)

(a) *If $\Gamma, x : \rho \vdash r : \tau$ and $\Gamma \vdash s : \rho$ then $\Gamma \vdash r[x := s] : \tau$.*

(b) *If $\Gamma \vdash r[x := s] : \tau$, $x \notin FV(s) \cup \Gamma$ and $(x \notin FV(r) \Rightarrow \Gamma \vdash s : \rho_0)$ then there is a type ρ such that $\Gamma, x : \rho \vdash r : \tau$ and $\Gamma \vdash s : \rho$.*

Proof (a) has a completely straightforward proof by induction on the derivation of $\Gamma, x : \rho \vdash r : \tau$ and does not reveal anything specific to intersection types. (b) is typical of intersection types (compare with Proposition 3 in [Kri93, p. 51] which is unfortunately too weak since it always requires $\Gamma \vdash s : \rho_0$ for some type ρ_0) and is proved by induction on the derivation of $\Gamma \vdash r[x := s] : \tau$. The case $r = x$ is trivial. Let now $r \neq x$.

$(\to_I)$ Let $\Gamma \vdash r[x := s] = \lambda y t : \tau = \tau_1 \to \tau_2$ due to $\Gamma, y : \tau_1 \vdash t : \tau_2$. Since $r \neq x$, $r = \lambda y r_0$ and $t = r_0[x := s]$. We may assume that $y \notin \{x\} \cup FV(s)$. Since $x \notin FV(r_0)$ implies $x \notin FV(r)$ implies $\Gamma \vdash s : \rho_0$ implies $\Gamma, y : \tau_1 \vdash s : \rho_0$, we may apply the induction hypothesis and get a type ρ such that $\Gamma, y : \tau_1, x : \rho \vdash r_0 : \tau_2$ and $\Gamma, y : \tau_1 \vdash s : \rho$. Strengthening yields $\Gamma \vdash s : \rho$. Finally, $\Gamma, x : \rho \vdash \lambda y r_0 : \tau_1 \to \tau_2$.

($\to_E$) Let $\Gamma \vdash r[x := s] = t_1 t_2 : \tau$ be derived from $\Gamma \vdash t_1 : \sigma \to \tau$ and $\Gamma \vdash t_2 : \sigma$. Since $r \neq x$, $r = r_1 r_2$ and $t_i = r_i[x := s]$ $(i = 1, 2)$. If $x \in FV(r)$ then $x \in FV(r_1)$ or $x \in FV(r_2)$. In both cases the induction hypothesis provides a type ρ_0 such that $\Gamma \vdash s : \rho_0$. Hence, from the assumption of (b) we always have some ρ_0 with $\Gamma \vdash s : \rho_0$. By induction hypothesis, there are types ρ_1 and ρ_2 such that $\Gamma, x : \rho_1 \vdash r_1 : \sigma \to \tau$, $\Gamma \vdash s : \rho_1$, $\Gamma, x : \rho_2 \vdash r_2 : \sigma$ and $\Gamma \vdash s : \rho_2$. By Weakening 2 and Exchange 2, we get $\Gamma, x : \rho_1 \cap \rho_2 \vdash r_1 : \sigma \to \tau$ and $\Gamma, x : \rho_1 \cap \rho_2 \vdash r_2 : \sigma$, hence $\Gamma, x : \rho_1 \cap \rho_2 \vdash r_1 r_2 : \tau$. Finally, $\Gamma \vdash s : \rho_1 \cap \rho_2$.

($\cap_I$) Intersect the two types given by the induction hypothesis as in the preceding case.

($\cap_E$) Trivial from the induction hypothesis. □

The next lemma will always be needed if a given type assignment is analyzed. For its statement we need the concepts of prime types and prime factors [Kri93, p. 50]: A *prime type* is a type which is not of the form $\rho \cap \sigma$. Let $\mathcal{P}$ be the set of prime types. Every type can be written in the form $\rho_1 \cap \ldots \cap \rho_n$ with $n \geq 1$ and $\rho_i \in \mathcal{P}$ for $i \in \{1, \ldots, n\}$. Note that we do not care about parentheses since they do not change typability (neither in contexts due to Exchange 3 nor in the derived type due to ($\cap_I$) and ($\cap_E$)). The ρ_i are called *prime factors* of ρ. We now present Lemma 1 from [Kri93, p. 50]. In the sequel the notation $\cap\vec{\rho}$ will be used for $\rho_1 \cap \ldots \cap \rho_n$ with the implicit assumption that the ρ_i are prime types. $\rho \in \vec{\rho}$ clearly means that $\rho = \rho_i$ for some i.

Lemma 22 (Inversion) *Let $\Gamma \vdash r : \rho$ with $\rho \in \mathcal{P}$.*

1. *If $r = x$ then $x : \cap\vec{\rho} \in \Gamma$ with $\rho \in \vec{\rho}$.*
2. *If $r = \lambda x t$ then $\rho = \rho_1 \to \rho_2$ and $\Gamma, x : \rho_1 \vdash t : \rho_2$.*[3]
3. *If $r = ts$ then $\Gamma \vdash t : \sigma \to \cap\vec{\rho}$ and $\Gamma \vdash s : \sigma$ for some types σ and $\vec{\rho}$ such that $\rho \in \vec{\rho}$.*

Proof Consider in the derivation of $\Gamma \vdash r : \rho$ the first occurrence of some $\Gamma \vdash r : \cap\vec{\rho}$ with $\rho \in \vec{\rho}$. The rule by which this is achieved cannot be ($\cap_I$) or ($\cap_E$) since they would require an earlier occurrence of the described form. Therefore, in the first, second and third case, the rule is (V), ($\to_I$) and ($\to_E$), respectively. □

Lemma 23 (Subject reduction) *If $\Gamma \vdash r : \rho$ and $r \to_\beta r'$ then $\Gamma \vdash r' : \rho$.*

Proof By induction on $\Gamma \vdash r : \rho$. Only the case ($\to_E$) is non-trivial: We have $\Gamma \vdash rs : \rho$ due to $\Gamma \vdash r : \sigma \to \rho$ and $\Gamma \vdash s : \sigma$, and $rs \to_\beta t$. Show that $\Gamma \vdash t : \tau$.

- If $t = r's'$ by one reduction step altogether from r to r' and s to s' then we are done by the induction hypothesis.

[3] We may assume that $x \notin \Gamma$.

- In the case of an outer β-reduction $r = \lambda x r_0$ and $t = r_0[x := s]$. By the preceding lemma, $\Gamma, x : \sigma \vdash r_0 : \rho$. By Lemma 21(a), $\Gamma \vdash r_0[x := s] : \rho$. □

As a further application of Inversion, we show that Ω is not typable.

Lemma 24 *Let $\vec{\sigma} \subset \mathcal{P}$. The following is impossible:*

$$\Gamma \vdash \omega : \cap\vec{\sigma} \to \rho \text{ and } \Gamma \vdash \omega : \cap\vec{\sigma}.$$

Proof Induction on the number of different types in $\vec{\sigma}$. Assume both typings. By Inversion, $\Gamma, x : \cap\vec{\sigma} \vdash xx : \rho$. $\rho = \cap\vec{\rho}$. Therefore, $\Gamma, x : \cap\vec{\sigma} \vdash xx : \rho_1$ (the first element of $\vec{\rho}$). Again by Inversion, $\Gamma, x : \cap\vec{\sigma} \vdash x : \tau \to \cap\vec{\mu}$ and $\Gamma, x : \cap\vec{\sigma} \vdash x : \tau$ with $\rho_1 \in \vec{\mu}$. τ decomposes into prime factors as $\tau = \tau_1 \cap \ldots \cap \tau_n$, hence for every $k \in \{1, \ldots, n\}$: $\Gamma, x : \cap\vec{\sigma} \vdash x : \tau_k$. Again by Inversion, $\tau_k \in \vec{\sigma}$ for every $k \in \{1, \ldots, n\}$. Therefore, $\tau = \cap\vec{\sigma}\,'$ for some finite list $\vec{\sigma}\,'$ composed only of elements of $\vec{\sigma}$. Once more by Inversion, applied to $\Gamma, x : \cap\vec{\sigma} \vdash x : \cap\vec{\sigma}\,' \to \cap\vec{\mu}$, there is a σ_j in $\vec{\sigma}$ such that $\sigma_j = \cap\vec{\sigma}\,' \to \cap\vec{\mu}$. Hence $\sigma_j \notin \vec{\sigma}\,'$, and consequently $\vec{\sigma}\,'$ has fewer different types than $\vec{\sigma}$. From $\Gamma \vdash \omega : \cap\vec{\sigma}$ we now immediately get $\Gamma \vdash \omega : \cap\vec{\sigma}\,' \to \cap\vec{\mu}$ and $\Gamma \vdash \omega : \cap\vec{\sigma}\,'$. Contradiction by induction hypothesis. □

Corollary 25 $\Gamma \not\vdash \Omega : \rho$.

Proof If $\Gamma \vdash \Omega : \rho$ then also for some $\rho \in \mathcal{P}$. By Inversion, there is $\vec{\rho}$ and σ such that $\rho \in \vec{\rho}$ and $\Gamma \vdash \omega : \sigma \to \cap\vec{\rho}$ and $\Gamma \vdash \omega : \sigma$. Since $\sigma = \cap\vec{\sigma}$ with $\vec{\sigma} \subset \mathcal{P}$, the previous lemma applies. □

6.3 Strong Normalization of Typable Terms

Lemma 26 *If $r \in \mathsf{SN}$ then $rx \in \mathsf{SN}$.*

Proof Induction on SN. □

Lemma 27 *If $r, s \in \mathsf{SN}$ and $\Gamma \vdash s : \rho$ and $\Gamma, x : \rho \vdash r : \sigma$ then $r[x := s] \in \mathsf{SN}$.*

Proof Main induction on the maximal number of $\to$ in the types in $\vec{\rho}$ in the decomposition $\rho = \cap\vec{\rho}$, side induction on $r \in \mathsf{SN}$, case distinction according to $r \in \mathsf{SN}$. (Observe that we also have $\Gamma \vdash r[x := s] : \sigma$ by Lemma 21(a).)

Case $y\vec{r}$. This has been derived from $\vec{r} \subset \mathsf{SN}$. By multiple Inversion, we get $\Gamma, x : \rho \vdash \vec{r} : \vec{\mu}$ for suitable $\vec{\mu}$. By side induction hypothesis, $\vec{r}[x := s] \subset \mathsf{SN}$. Therefore, $y\vec{r}[x := s] \in \mathsf{SN}$. This finishes with the case $y \neq x$. So assume $y = x$. If $\vec{r}$ is empty, the claim is trivial. Otherwise, $\vec{r} = t, \vec{t}$, hence $\Gamma, x : \rho \vdash xt\vec{t} : \sigma$. $xt\vec{t} = (z\vec{t})[z := xt]$ for some "new" variable z. By Lemma 21(b), there is a type τ such that $\Gamma, x : \rho, z : \tau \vdash z\vec{t} : \sigma$ and $\Gamma, x : \rho \vdash xt : \tau$. $\tau = \cap\vec{\tau}$. Consider the element τ_k of $\vec{\tau}$. $\Gamma, x : \rho \vdash xt : \tau_k$. By Inversion, there is a type ρ_k' and types $\vec{\nu}_k$ with $\tau_k \in \vec{\nu}_k$ such that $\Gamma, x : \rho \vdash x : \rho_k' \to \cap\vec{\nu}_k$ and $\Gamma, x : \rho \vdash t : \rho_k'$. With $\rho = \cap\vec{\rho}$ and Inversion, we get $\rho_k' \to \cap\vec{\nu}_k \in \vec{\rho}$. Therefore, τ_k has a smaller number of $\to$ than the maximum of the numbers of $\to$ in $\vec{\rho}$. We continue with $k := 1$, $\rho' := \rho_1'$ and

$\vec{v} := \vec{v}_1$. Clearly, $\Gamma \vdash s : \rho' \to \cap\vec{v}$. By the previous lemma, $sz' \in \mathsf{SN}$ with a "new" variable z'. By Weakening 1, $\Gamma, z' : \rho' \vdash sz' : \cap\vec{v}$. By Lemma 21(a): $\Gamma \vdash t[x := s] : \rho'$. We apply the main induction hypothesis for type ρ' (which is smaller than ρ in the sense given by the main induction) and get $st[x := s] \in \mathsf{SN}$. By Lemma 21(a), $\Gamma, z : \tau \vdash z\vec{t}[x := s] : \sigma$ and $\Gamma \vdash st[x := s] : \tau$. Clearly, $z\vec{t}[x := s] \in \mathsf{SN}$. Hence, by the main induction hypothesis for type τ, we finally get $(x\vec{r})[x := s] = st[x := s]\vec{t}[x := s] \in \mathsf{SN}$.

Case $\lambda y r$. This comes from $r \in \mathsf{SN}$. We have $\Gamma, x : \rho \vdash \lambda y r : \sigma$. Show $(\lambda y r)[x := s] \in \mathsf{SN}$. We may assume that $y \notin \{x\} \cup \mathrm{FV}(s)$. Therefore $(\lambda y r)[x := s] = \lambda y.r[x := s]$, and it suffices to show $r[x := s] \in \mathsf{SN}$. $\sigma = \cap\vec{\sigma}$ and $\Gamma, x : \rho \vdash \lambda y r : \sigma_1$ (the first element of $\vec{\sigma}$). By Inversion, $\sigma_1 = \tau \to \sigma'$ and $\Gamma, x : \rho, y : \tau \vdash r : \sigma'$. By Weakening 1, $\Gamma, y : \tau \vdash s : \rho$. The side induction hypothesis yields $r[x := s] \in \mathsf{SN}$.

Case $(\lambda y r)s\vec{s}$. This is derived from $r[y := t]\vec{t} \in \mathsf{SN}$ and $t \in \mathsf{SN}$. We have $\Gamma, x : \rho \vdash (\lambda y r)t\vec{t} : \sigma$ and have to show that (we assume that $y \notin \{x\} \cup \mathrm{FV}(s)$) $((\lambda y r)t\vec{t})[x := s] = (\lambda y.r[x := s])t[x := s]\vec{t}[x := s] \in \mathsf{SN}$. For this we need $(r[x := s])[y := t[x := s]]\vec{t}[x := s] \in \mathsf{SN}$ and $t[x := s] \in \mathsf{SN}$. By multiple Inversion, we get $\Gamma, x : \rho \vdash t : \mu$ for some type μ. The side induction hypothesis yields $t[x := s] \in \mathsf{SN}$. By Subject Reduction, $\Gamma, x : \rho \vdash r[y := t]\vec{t} : \sigma$. Hence, again by side induction hypothesis, $(r[y := t]\vec{t})[x := s] \in \mathsf{SN}$. We are done by Lemma 2 which tells us that $(r[y := t]\vec{t})[x := s] = (r[x := s])[y := t[x := s]]\vec{t}[x := s]$. □

Corollary 28 (Main Theorem) *If* $\Gamma \vdash r : \rho$ *then* $r \in \mathsf{SN}$.

Proof By induction on $\vdash$. In the case $(\to_E)$ use the induction hypothesis, $rs = (rx)[x := s]$ and the preceding two lemmas. □

7 Monotone Inductive Types

This section is based on [Mat99].

References

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers, principles, techniques, and tools.* Addison-Wesley, 1986.

[Kri93] Jean-Louis Krivine. *Lambda-calculus, types and models.* Masson, Paris and Ellis Horwood, Hemel Hempstead, 1993. English translation of Lambda-calcul, types et modèles, Masson, 1990.

[Mat99] Ralph Matthes. Tarski's fixed-point theorem and lambda calculi with monotone inductive types. To appear in: Benedikt Löwe and Florian Rudolph, Foundations of the Formal Sciences, Refereed Papers of a Research Colloquium, Humboldt-Universität zu Berlin, May 7-9, 1999.