Typed Lambda Calculus / Calculus of Constructions

Helmut Brandl

(firstname dot lastname at gmx dot net)

Version 1.03

Abstract

In this text we describe the calculus of constructions as one of the most interesting typed lambda calculus. It is a sweet spot in the design space of typed lambda calculi because it can express an immense set of computable functions and it can express a large set of logical propositions including their proofs.

The paper is written in a textbook style. The needed concepts are introduced step by step and the proofs are layed out sufficiently detailed for newcomers to the subject. The readers who are familiar with basic concepts of lambda calculus and computer science can get a good unterstanding of typed lambda calculus.

For comments, questions, error reporting feel free to open an issue at `https://github.com/hbr/Lambda-Calculus`

Contents

1 Introduction

Computation What is a computer? What comes into your mind if you think of computers? Your laptop, your smartphone? A super-computer? ... Would you say that a human being is a computer?

Looking 100 years back, computers had not yet been invented. At least not modern electronic computers. However computers existed. A computer was a man or a woman who can carry out computations. The term *computer* in the meaning of *one who computes* had been in use from the early 17th century. The NASA and its predecessor NACA hired men and women to carry out computations.

A *computer* had no authority. He/She had to carry out computations following some fixed rules.

Carrying out computations according to fixed rules had been present in the whole history of mankind for more than 2000 years. Think of adding and multiplying numbers, using Gaussian elimination to solve a system of linear equations, using Newton's method to find the root of a nonlinear function etc.

The notion of computing i.e. carrying out some steps according to fixed rules had been intuitively clear. Whenever somebody said that he had found a method to calculate something and had written down a recipe to do it, everyone could check the method by trying to carry out the computation according to the written rules. If there are no ambiguities, then the method can be regarded as a general recipe or an *algorithm*.

Computability In the early 20th century the question came up: *What is computable?* David Hilbert, the famous german mathematician, challenged the world with his statement that everything which can be formalized in mathematics must be decidable/computable. That there is nothing *undecidable* or *uncomputable*. If a computation or decision procedure had not yet been found, then we have to try harder until we find the general method.

Looking at the question *Is there something uncomputable / undecidable?* an intuitive understanding of computation is no longer sufficient. A precise formal definition of computation becomes necessary. Nearly at the same time, three famous mathematicians came up with formal definitions of computability which could be proved to be equivalent:

- Kurt Gödel's *recursive functions*

- Alan Turing's *automatic machine* (today called *Turing machine*)
- Alonzo Church's *lambda calculus*

It can be shown that in this space of computable functions there are problems which cannot be decided. E.g. the halting problem is undecidable. There is no function which takes a valid program and its input as input and returns true if the program terminates on the input and false if the program does not terminate on the input.

Having a clear and formal definition of computability, many problems have been proved to be unsolvable by computation.

In this paper we look only into lambda calculus, because lambda calculus is not only a universal model of computation like the other two, there is much more in it.

Let's see what this *more* is. There is something unsatisfactory in lambda calculus which led to significant improvements.

Typing Since lambda calculus or more specifically the untyped lambda calculus is a universal model of computation, it is possible to do all possible computations at least theoretically in lambda calculus. Define boolean values and functions, define natural numbers and functions on natural numbers, define pairs of values, list of values, trees of values and all functions on this data. There is no limit.

However it is possible to express terms which are completely useless.

- You can feed a string to a function which expects a natural number as argument.
- You can write expressions which implement a non terminating computation.

Modern programming languages solve the first problem by adding a type to each argument of a function and a type to the result of a function. The second problem is usually not addressed in modern programming languages. In nearly all mainstream programming languages infinite loops are possible.

Computation and Logic Alonzo Church added types to his lambda calculus. But his main intention was not to avoid silly expressions. He wanted typed lambda calculus to be a model for logic. This first attempt only described propositions in the lambda calculus. More sophisticated typed lambda calculi laid the basis for the Curry-Howard isomorphism. The Curry-Howard isomorphism connects two seemingly unrelated formalisms: computations and proof systems.

The types in computations are connected to propositions in logic via the Curry-Howard isomorphism. The terms of a certain type are proofs of the corresponding proposition. A proof of an implication $A \Rightarrow B$ is a function (i.e. a computation) mapping a proof of A i.e. a term of type A to a proof of B i.e. a term of type B. The identity function is a proof of $A \Rightarrow A$.

A proof of $A \wedge (A \Rightarrow B) \Rightarrow B$ the *modus ponens rule* is nearly trivial in this computation analogy. It is a function taking two arguments. It takes an object a of type A and a function f of type $A \Rightarrow B$ and it returns an object of type B by just applying f to a.

However if we want lambda calculus to be a model for logic and proof systems, then *termination* becomes crucial. Proofs must be finite. An infinite proof makes no sense. Nobody can check the correctness of an infinite proof.

In the computational world the definition of a function directly in terms of itself e.g. $f(x : A) : B := f(x)$ is well-typed but useless. Calling f with an object a of type A ends up in an infinite recursion.

The counterpart of this nonterminating function f in logic is a proof of $A \Rightarrow B$ which uses a proof of $A \Rightarrow B$. This *circular* logic is not allowed. Proofs which use circular logic are not well-formed. Anything can be proved by using circular logic.

Typed Lambda Calculi The untyped lambda calculus can be extended to many forms of typed lambda calculus which serve both as model of computation and a model of logic i.e. they avoid silly terms and they guarantee termination.

We come from untyped lambda calculus to typed lambda calculus by adding type annotations. Type annotations are necessary only to check that terms are well-typed. After type checking, types can be thrown away. This is called *type erasure*. We distinguish between computational objects and types (i.e. non computational objects).

In this paper we treat the *Calculus of Constructions* as a typed lambda calculus which is a good model of computation and logic at the same time. The way from untyped lambda calculus to the calculus of constructions can be seen as having four steps:

1. *Simply typed lambda calculus*: The types have no structure. A type is a type variable. We have type variables $\{U, V, W, \ldots\}$

and arbitrary functions formed over the type variables $\{ U \to U, U \to V, U \to (V \to W), \dots \}$. The computational power of simply typed lambda calculus is fairly limited. But it is already a model for the implicational fragment of natural deduction.

2. *Polymorphic functions (System F)*: In the simply typed lambda calculus it is not possible to express the identity function which works for arbitrary types. Each type needs its own identity function. *Girard*'s System F allows types as arguments for functions. Now it is possible to express a polymorphic identity function. It is a function receiving two arguments. A type and term of this type. The body of the function just returns the second argument.

   This addition of polymorphic functions makes system F substantially more powerful than simply type lambda calculus. Functions operating on booleans, natural numbers, lists of values, pairs, trees etc. can be expressed in the calculus. As a logic it can expresses a proof system for second order intutionistic predicate calculus.

3. *Polymorphic types i.e. computing types (System F_ω)*: System F already allows polymorphic functions which operate on lists of a certain element type, pairs of two elements of two different types, trees of certain element types etc.

   However a list of integers and a list of booleans are different types. In system F it is not possible to define functions which take type arguments and return type results. E.g. it is not possible to define a function L which takes an element type A as argument and returns the type LA of a list where the elements are of type A.

   System F_ω adds this possibility to compute types. This adds the necessity to add types of types. It is necessary to express how many arguments a type function takes. The arguments might be types or type functions. The Haskell programming language which is based to some extent on system F_ω adds kinds. The kind $*$ is the type of a type. The kind $* \to *$ is the type of a type function which takes a type argument a returns a type. The kind $(* \to *) \to *$ takes a type function and returns a type etc.

4. *Dependent types (calculus of constructions)*: We need another dimension to express interesting logical propositions. We want to express the proposition that a certain number is a prime number. Propositions are types in the Curry-Howard isomorphism,

therefore this proposition is a type. However a number is certainly a computational object. In order to express the predicate *is prime* we need functions which map a number to a proposition i.e. a type.

Now we can express the proposition which states that all objects x of a certain type A have a certain property P i.e. the proposition $\forall(x : A).Px$. Px is a type which depends on the computational object x. Therefore it is called a dependent type.

A proof of the proposition $\forall(x : A).Px$ is a function which takes an object x of type A and returns an object of type Px. In the calculus of construction we express the proposition $\forall(x : A).Px$ as the type $\Pi x^A.Px$.

The calculus of constructions has enormous computational power and is very expressive as a logic and proof system. Therefore it is a sweet spot in the design space of typed lambda calculi.

This Paper In this paper we introduce the calculus of constructions.

- Section *Basic Mathematics* 2 describes the basic mathematics needed to understand the following chapters. It is important to understand the used logic notation and the form of induction proofs. In type theory induction proofs on inductively defined sets and relations are used excessively. Therefore a special layout has been chosen to present such proofs and figure out the induction hypotheses clearly.
- In section *The Calculus* 3 the calculus of constructions is explained. It defines the terms of the calculus, the contexts to give types to free variables, the basic computation steps and term equivalence.
- Section *Confluence* 4 proves an important property of the calculus: Uniqueness of results. The computation steps in the calculus are nondeterministic. At each state of the computation which is not a final state different steps might be possible. The property of confluence guarantees that all computation paths starting from a term can be joined and that the final result (if it exists) is unique.
- Section *Typing* 5 introduces the typing relation and proves important properties about this relation. The typing relation de-

fines the well-typed terms. Terms in the calculus formed according to section *The Calculus* 3 are well-formed, but not necessarily well-typed. In order to be well-typed they have to be valid terms in the typing relation.

- Section *Proof of Strong Normalization* 6: This section contains the proof that all well-typed terms in the calculus of constructions are strongly normalizing. I.e. Every well-typed term reduces in a finite number of computation steps to a normal form which is the result of the computation.

  The proof of strong normalization is rather involved and requires a lot of machinery to go through. All needed concepts and theorems are explained in detail. Therefore this chapter is the longest in the paper.

  Strong normalization implies consistency when regarded as a model of a proof system for logic. Consistency in logic means that it is free of contradictions. A type system is consistent as a logic if there are types which are uninhabited in the empty context. In the Curry-Howard isomorphism an uninhabited type corresponds to a proposition which is impossible to prove.

  The proof of consistency is the last subsection in this chapter.

2 Basic Mathematics

2.1 Sets and Relations

It is assumed that the reader is familiar with the following concepts:

1. Basic set notation: Elements of a set $a \in A$, Subsets $A \subseteq B$ defined as $a \in A$ implies $a \in B$.
2. Endorelations: An (endo-) relation r over the set A is the set of pairs (a, b) (i.e. $r \subseteq A \times A$) where a and b are drawn from the set A. We write $(a, b) \in r$ more pictorially as $a \xrightarrow{r} b$.
3. n-ary Relations: An endorelation is a binary relation where the domain and the codomain are the same. An n-ary relation over the domains $A_1, A_2, \ldots, A_n$ is a subset from the cartesian product $A_1 \times A_2 \times \ldots \times A_n$. In this paper we use a ternary relation as the typing relation.
4. An intuitive understanding of the reflexive transitive closure r^* of a relation r, the reflexivity, symmetry and transitivity of a relation and similar concepts.
5. Logic: The logical connectives of conjunction $a \wedge b$, disjunction $a \vee b$, implication $a \Rightarrow b$ and existential $\exists x.p(x)$ and universal quantication $\forall x.p(x)$. We use the symbol $\bot$ for falsity i.e. a logical statement which can never be proved (a contradiction).
6. An intuitive understanding of mathematical functions from the set A to the set B. I.e. $A \to B$ is the set of functions mapping each element of the set A to a unique element of the set B.

2.2 Logic

We often have to state that some premises $p_1, p_2, \ldots p_n$ have a certain conclusion c

$$p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow c$$

In many cases this notation with logical connectives is clumsy and not very readable. We use the rule notation

$$\frac{\begin{array}{c} p_1 \\ p_2 \\ \ldots \\ p_n \end{array}}{c}$$

to express the same fact.

If some variables in logical statements are not quantified, then universal quantification is assumed.

$$\frac{a \in A}{a \in B} \quad \text{means} \quad \forall a. \left[\frac{a \in A}{a \in B} \right]$$

Assume that a logical statement has more than one variable. The universal quantification can be expressed at the highest level or pushed done to the first appearance of the variable. Moreover logical conjunction is commutative. Therefore the following interpretations are equivalent.

$$\frac{\begin{array}{c} a \in A \\ b \in A \end{array}}{a \xrightarrow{r} b} \quad \text{means} \quad \begin{array}{l} \forall ab. \left[\frac{\begin{array}{c} a \in A \\ b \in A \end{array}}{a \xrightarrow{r} b} \right] \\ \forall a. \left[\frac{a \in A}{\forall b. \left[\frac{b \in A}{a \xrightarrow{r} b} \right]} \right] \\ \forall b. \left[\frac{b \in A}{\forall a. \left[\frac{a \in A}{a \xrightarrow{r} b} \right]} \right] \end{array}$$

All three settings of the universal quantifiers are logically equivalent.

2.3 Inductively Defined Sets

Sets can be defined inductively by rules. E.g. we can define the set of even numbers by the rules

1. 0 is an even number
2. If n is an even number, then $n + 2$ is an even number as well.

We write this more formally:

The set of even numbers E is defined by the rules:

1.
$$0 \in E$$

2.
$$\frac{n \in E}{n + 2 \in E}$$

We say that E is the smallest set which satisfies the above rules.

Whenever there is an element $m \in E$ it can be in the set only if it is in the set because of one of the two rules which define the set. 0 is in E because of the first rule. 2 is in E because of second rule where $n = 0$ and 0 is in the set. 4 is in the set because of the second rule where $n = 2$ and 2 is in the set.

A number $m \in E$ can be arbitrarily big, but it can be in the set only because of finitely many applications of the rules which define the set.

2.4 Induction Proofs

With inductively defined sets we can do induction proofs. Let's say that we want to prove that all numbers n in E satisfy a certain property $p(n)$ i.e. we want to prove

$$\frac{n \in E}{p(n)}$$

We prove this statement by induction on $n \in E$. Similar to induction proofs on natural numbers we have to prove

1. 0 satisfies p, i.e. $p(0)$
2. Every n that satisfies p (i.e. $p(n)$) must imply that $n+2$ satisfies p as well (i.e. $p(n+2)$).

Because the second rule is recursive we get an induction hypothesis.

We write all induction proofs on an inductively defined set in the following manner:

Theorem: *All even numbers $n \in E$ satisfy $p(n)$.*

$$\frac{n \in E}{p(n)}$$

Proof: By induction on $n \in E$.

1. $$0 \in E \mid p(0)$$

   $p(0)$ is valid because ...

2.

$$\begin{array}{l|l} n \in E & p(n) \\ \hline n+2 \in E & p(n+2) \end{array}$$

In order to prove $p(n+2)$ in the lower right corner we assume $n \in E$, $p(n)$, $n+2 \in E$. $p(n+2)$ is valid because ...

For each rule in the inductive definition of the set we write a matrix. The left part of the matrix is just the rule. The goal is always in the lower right corner of the matrix. For each recursive premise (i.e. that some other elements are already in the set we get an induction hypothesis.

In order to prove the goal in the lower right corner we can assume all other statements in the matrix.

This schema helps to layout all premises and all induction hypotheses precisely.

As an example we prove that for all even numbers $n \in E$ there exists a number m such that $n = 2m$ using the above schema.

Theorem: *For all even numbers $n \in E$ there exists a number m with $n = 2m$.*

$$\frac{n \in E}{\exists m.n = 2m}$$

Proof: By induction on $n \in E$.

1.

$$\begin{array}{l|l} 0 \in E & \exists m.0 = 2m \end{array}$$

$m = 0$ satisfies the goal.

2.

$$\begin{array}{l|l} n \in E & \exists i.n = 2i \\ \hline n+2 \in E & \exists m.n+2 = 2m \end{array}$$

By the induction hypothesis we get a number i which satisfies $n = 2i$. Therefore $n+2 = 2i+2 = 2(i+1)$. The number $m = i+1$ satisfies the goal in the lower right corner.

Note that the name of the bound variable in existential and universal quantification is arbitrary. It is good practice to choose names which do not interfere. In the case for the recursive rule we have chosen the different names i and m in the existential quantification. This makes the prove more readable for humans.

2.5 Inductively Defined Relations

An (endo-) relation r over the set A is just a subset of the cartesian product $A \times A$. I.e. a relation is just a set which can be defined inductively. Since inductively defined relations are used excessively in the paper we explain inductive definitions and induction proofs on relations as well.

Note that the following is not restricted to binary endorelations over a carrier set A. It is possibly to define n-ary relations over different carrier sets as subset of the cartesian product $A_1 \times A_2 \times \ldots \times A_n$ as well. E.g. the typing relation in the calculus of constructions (see section 5) is a ternary relation.

As an example we define the reflexive transitive closure of a relation r inductively.

> *The reflexive transitive closure r^* of the relation r is defined by the rules*
>
> 1.
> $$a \overset{r^*}{\to} a$$
>
> 2.
> $$\frac{\begin{array}{c} a \overset{r^*}{\to} b \\ b \overset{r}{\to} c \end{array}}{a \overset{r^*}{\to} c}$$

It should be intuitively clear that r^* is reflexive and transitive. The first rule guarantees reflexivity and the second rule guarantees transitivity. While reflexivity is expressed explicitely in the first rule, transitive is expressed only indirectly in the second rule therefore we need a proof of transitivity.

> Theorem: *The reflexive transitive closure r^* of the relation r is transitive*
> $$\frac{\begin{array}{c} a \overset{r^*}{\to} b \\ b \overset{r^*}{\to} c \end{array}}{a \overset{r^*}{\to} c}$$
>
> Proof: Assume $a \overset{r^*}{\to} b$ and do induction on $b \overset{r^*}{\to} c$.
>
> 1.
> $$b \overset{r^*}{\to} b \;\Big|\; a \overset{r^*}{\to} b$$
>
> The goal $a \overset{r^*}{\to} b$ is trivial because it is an assumption.

2.

$$\begin{array}{c|c} b \overset{r^*}{\rightarrow} c_0 & a \overset{r^*}{\rightarrow} c_0 \\ c_0 \overset{r}{\rightarrow} c & \\ \hline b \overset{r^*}{\rightarrow} c & a \overset{r^*}{\rightarrow} c \end{array}$$

By the induction hypothesis we have $a \overset{r^*}{\rightarrow} c_0$. Together with the second premise $c_0 \overset{r}{\rightarrow} c$ and the second rule in the definition of r^* we get the goal in the lower right corner.

2.6 Term Grammar

It is possible to define inductive sets by a grammar. E.g. we can define the set of natural numbers $\mathbb{N}$ by the grammar

$$\begin{array}{lcll} n & ::= & 0 & \text{the number } \textit{zero} \\ & | & n' & \text{the successor of } n \end{array}$$

where n ranges over natural numbers and 0 is a constant.

This grammar defines the natural numbers $\{0, 0', 0'', \dots\}$ as strings over the alphabet $\{0, '\}$.

A definition with a grammar is just a special form of an inductively defined set.

The set of natural numbers $\mathbb{N}$ *is defined by the rules*

1.

$$0 \in \mathbb{N}$$

2.

$$\frac{n \in \mathbb{N}}{n' \in \mathbb{N}}$$

If we want to prove that all natural numbers n satisfy a certain property p we have to prove that 0 satisfies the property (i.e. $p(0)$) and if n satisfies the property then n' has to satisfy the property as well i.e. $p(n) \Rightarrow p(n')$.

This is the classical induction principle on natural numbers. Note that this is just a special case of the more general induction proof scheme for inductively defined sets.

An induction proof over terms defined by a term grammar has the following form:

Theorem: *All natural numbers n have the property $p(n)$.*

Proof: By induction on the structure of n:

1. $p(0)$: The goal $p(0)$ is valid because ...
2. $p(n')$: By the induction hypothesis $p(n)$ is valid. This implies $p(n')$ because ...

Note that such a proof corresponds one to one to a proof on inductive sets shown above.

Furthermore note that a grammar rule in the term grammar might be recursive in more than one subterm. In that case we get more than one induction hypothesis.

2.7 Recursive Functions

In the previous section we have shown that for each definition of a set with a term grammar there is a corresponding inductive definition. However with terms defined via a term grammar we can define recursive functions.

We use the natural numbers defined via a term grammar and define the recursive function $p(n, m)$ which computes the sum of the numbers n and m.

$$p(n,m) := \left\{ \begin{array}{lcl} p(n,0) & := & n \\ p(n,m') & := & p(n,m)' \end{array} \right.$$

The term grammar tells us that we can construct a natural number by using the number 0 and apply zero or more times the successor function $'$ to it. The recursive function deconstructs the string generated by the term grammar. Since every term has a finite length the recursion is guaranteed to terminate.

3 The Calculus

In this section we introduce the terms of the calculus of constructions and the way to do computations with them.

Lambda terms in untyped lambda calculus are either variables $x, y, z, \ldots$, abstractions $\lambda x.e$ or applications ab.

The calculus of constructions is a typed lambda calculus where terms and types are expressed within the same syntax. Since all well-typed terms must have a type we need types of types. The types of types in the calculus of constructions are the two sorts $\mathcal{P}$ and $\mathcal{U}$.

Since the calculus is typed, all variables must have a type. Therefore all variables in binders like abstractions have types $\lambda x^A.e$. In order to assign types to free variables contexts are introduced.

Furthermore it is necessary to express functions $A \to B$ from one type A to another type B. The calculus of constructions includes dependent types, i.e. the result type B of a function might depend on the argument. In order to express that a product type of the form $\Pi x^A.B$ is used which describes the type of a function from an argument of type A to the result of type B which might depend on the argument.

After introducing the terms of the calculus on constructions we define *free variables*, *substitution*, *beta reduction*, *beta equivalence* and prove certain theorems which state some interesting properties about these definitions.

In this section we do not yet define what it means for a term to be *welltyped*. This is done in the chapter typing 5.

3.1 Sorts

In the calculus of constructions terms and types are in the same syntactic category. All welldefined terms have types and therefore types must have types as well. In order to have types of types we start with the introduction of sorts which are the types of types.

Definition 3.1. *Sorts:* There are the two sorts $\mathcal{P}$ and $\mathcal{U}$ in the calculus of constructions.

Sorts or universes are the types of types. Sorts are usually abbreviated by the variable s.

In many texts about typed lambda calculus like e.g. Barendregt1993 [1] and Geuvers1994 [2] the symbol $*$ is used instead of $\mathcal{P}$ and the symbol $\Box$ is used instead of $\mathcal{U}$. This has found its way into the Haskell programming language where $*, * \to *, (* \to *) \to *, \ldots$ are *kinds*.

We use the symbol $\mathcal{P}$ in order to emphasize the Curry-Howard correspondence of *propositions as types*. Many interesting types T in the calculus of constructions have type $\mathcal{P}$ i.e. $T : \mathcal{P}$. Within the Curry-Howard correspondence these types are propositions.

The symbol $\mathcal{U}$ is just a higher *universe* than $\mathcal{P}$. A more general lambda calculus the *extended calculus of constructions* has an infinite hierarchy of universes $\mathcal{P}, \mathcal{U}_0, \mathcal{U}_1, \ldots$ where $\mathcal{P}$ is the impredicative universe and $\mathcal{U}_i$ are the predicative universes.

In the calculus of constructions the term $\mathcal{P}$ has type $\mathcal{U}$ and the term $\mathcal{U}$ is not welltyped. In the extended calculus of constructions the term $\mathcal{U}_i$ has type $\mathcal{U}_{i+1}$ i.e. all terms of the form $\mathcal{U}_i$ for any i are welltyped.

The decision to use the symbols $\mathcal{P}$ and $\mathcal{U}$ instead of $*$ and $\Box$ is just a matter of taste.

3.2 Terms

Definition 3.2. The *terms* are defined by the grammar where s ranges over sorts, x ranges over some countably infinite set of variables and t ranges over terms.

$$\begin{array}{rcll} t & ::= & s & \text{sorts} \\ & | & x & \text{variable} \\ & | & \Pi x^t.t & \text{product} \\ & | & \lambda x^t.t & \text{abstraction} \\ & | & tt & \text{application} \end{array}$$

3.3 Free Variables

Definition 3.3. The set of *free variables* $\mathrm{FV}(t)$ of a term t is defined by the function

$$\mathrm{FV}(t) := \begin{cases} \mathrm{FV}(s) & := & \emptyset \\ \mathrm{FV}(x) & := & \{x\} \\ \mathrm{FV}(ab) & := & \mathrm{FV}(a) \cup \mathrm{FV}(b) \\ \mathrm{FV}(\lambda x^A.e) & := & \mathrm{FV}(A) \cup (\mathrm{FV}(e) - \{x\}) \\ \mathrm{FV}(\Pi x^A.B) & := & \mathrm{FV}(A) \cup (\mathrm{FV}(B) - \{x\}) \end{cases}$$

where s ranges over sorts, x ranges over variables and a, b, e, A and B range over arbitrary terms.

A variable which is not free is called a bound variable. E.g. if x is a free variable in the term e, it is no longer free in $\lambda x^A.e$. Therefore we call $\lambda x^A.e$ a binder, because it makes the variable x bound. The same applies to the term $\Pi x^A.B$ where the variable x is bound, but can appear free in B.

Note that the binders $\lambda x^A.e$ and $\Pi x^A.B$ make the variable x bound only in the subterms e and B, but not in the subterm A.

It is possible to rename bound variables within a term. The renaming of a bound variable does not change the term. We consider two terms which only differ in the name of bound variables as identical. Examples of some identical terms:

$$\begin{array}{rcl} \lambda x^{\mathcal{P}}.x & = & \lambda y^{\mathcal{P}}.y \\ \Pi x^z.x & = & \Pi y^z.y \end{array}$$

3.4 Contexts

All bound variables get their types by their corresponding binders. For free variables we need types as well. In order to assign types to free variables we define contexts.

Definition 3.4. A *context* is a sequence of variables and their corresponding types. Contexts are usually abbreviated by upper greek letters and types are terms which are usually abbreviated by uppercase letters.

Contexts are defined by the grammar

$$\begin{array}{rcll} \Gamma & := & [] & \text{empty context} \\ & | & \Gamma, x^A & \text{one more variable } x \text{ with its type } A \end{array}$$

3.5 Substitution

Definition 3.5. *Substitution:* The term $t[y := u]$ is the term t where the term u is substituted for all free occurrences of the variable y. It is defined as a recursive function which iterates over all subterms until variables or sorts are encountered.

$$t[y := u] :=$$
$$\begin{cases} s[y := u] & := \quad s \\ x[y := u] & := \quad \begin{cases} u & y = x \\ x & y \neq x \end{cases} \\ (ab)[y := u] & := \quad a[y := u]b[y := u] \\ (\lambda x^A.e)[y := u] & := \quad \lambda x^{A[y:=u]}.e[y := u] \quad\quad y \neq x, x \notin \mathrm{FV}(u) \\ (\Pi x^A.B)[y := u] & := \quad \Pi x^{A[y:=u]}.B[y := u] \quad y \neq x, x \notin \mathrm{FV}(u) \end{cases}$$

Remark: The conditions $y \neq x$ and $x \notin \mathrm{FV}(u)$ are not a restriction because the bound variable x can always be renamed to another variable which is different from y and does not occur free in the term u.

Remark: Many authors in the literature use $t[u/y]$ as a notation for $t[y := u]$. Both notations mean the same thing.

Lemma 3.6. *Substitution to sort lemma:* If the result of a substitution is a sort then either the term in which the substitution occurs is the sort or the term is the variable to be replaced and the substitution term is the sort.

$$\frac{a[x := b] = s}{a = s \vee (a = x \wedge b = s)}$$

Proof. From the definition of the substitution it is evident that only the first two cases can result in a sort. All other cases cannot syntactically result in a sort. The first two cases of the definition prove exactly the goal. □

Lemma 3.7. *Double substitution lemma* If we change the order of two subsequent substitutions, then it is necessary to introduce a correction term.

$$\frac{\begin{array}{c} x \neq y \\ y \notin \mathrm{FV}(a) \end{array}}{t[y := b][x := a] = t[x := a][y := b[x := a]]}$$

Proof. The intuitive proof is quite evident. If we substitute b for the variable y in the term t and then substitute a for the variable x in the result we replace in the second substitution not only all variables x originally contained in t but all occurrences of x in b as well.

If we do the substitution $[x := a]$ on t, then all occurrences of x in b have not yet been substituted. Therefore the correction term $b[x := a]$ is necessary if the order is changed.

The formal proof goes by induction on the structure of t.

1. If t is a sort then the equality is trivial because a sort does not have any variables.
2. If t is an application, an abstraction or a product, then the goal follows immediately from the induction hypotheses.
3. The only interesting case is when t is a variable. Let's call the variable z. Then we have to distinguish the cases that z is x or z is y or z is different from x and y.
   (a) Case $z = x$:
   $$\begin{array}{lcll} x[y := b][x := a] & = & x[x := a] & \\ & = & a & \\ & & & \\ x[x := a][y := b[x := a]] & = & a[y := b[x := a]] & \\ & = & a & y \notin \mathrm{FV}(a) \end{array}$$
   (b) Case $z = y$:
   $$\begin{array}{lcl} y[y := b][x := a] & = & b[x := a] \\ & & \\ y[x := a][y := b[x := a]] & = & y[y := b[x := a]] \\ & = & b[x := a] \end{array}$$
   (c) Case $z \neq x \wedge z \neq y$:
   $$\begin{array}{lcl} z[y := b][x := a] & = & z \\ & & \\ z[x := a][y := b[x := a]] & = & z \end{array}$$

□

The validity of the *double substitution lemma* 3.7 is needed to make sure that *beta reduction* (see definition in the next section) is confluent. E.g. if we have the term $(\lambda x^A.(\lambda y^B.t)b)a$ then we can decide whether we reduce first the inner redex and then the outer redex or the other way round. Because of confluence both possibilities shall have the same result.

$$\begin{array}{lcl} (\lambda x^A.(\lambda y^B.t)b)a & \overset{\beta}{\to} & (\lambda x^A.t[y := b])a \\ & \overset{\beta}{\to} & t[y := b][x := a] \\ & & \\ (\lambda x^A.(\lambda y^B.t)b)a & \overset{\beta}{\to} & (\lambda y^B.t[x := a])b[x := a] \\ & \overset{\beta}{\to} & t[x := a][y := b[x := a]] \end{array}$$

3.6 Beta Reduction

Like in untyped lambda calculus, computation is done via *beta reduction.* A beta redex has the form $(\lambda x^A.e)a$ which reduces to the reduct $e[x := a]$. Beta reduction can be done in any subterm of a term.

The intuitive meaning of beta reduction is quite clear. The term $\lambda x^A.e$ is a function with a formal argument x of type A. The *body* e is the *implementation* of this function which can use the variable x. As with any programming language which support functions the name of the formal argument is irrelevant. An arbitrary name can be chosen and the variable is only used *internally* and is not visible to the *outside world.* We can apply the function to an argument a i.e. form the term $(\lambda x^A.e)a$. In order to compute the result we use the *implementation* e and substitute the actual argument a for the formal argument x i.e. we form $e[x := a]$.

Definition 3.8. *Beta reduction* is a binary relation $a \overset{\beta}{\to} b$ where the term a reduces to the term b. It is defined by the rules

1. Redex:
$$(\lambda x^A.e)a \overset{\beta}{\to} e[x := a]$$

2. Reduce function:
$$\frac{f \overset{\beta}{\to} g}{fa \overset{\beta}{\to} ga}$$

3. Reduce argument:
$$\frac{a \overset{\beta}{\to} b}{fa \overset{\beta}{\to} fb}$$

4. Reduce abstraction argument type:
$$\frac{A \overset{\beta}{\to} B}{\lambda x^A.e \overset{\beta}{\to} \lambda x^B.e}$$

5. Reduce abstraction inner term:
$$\frac{e \overset{\beta}{\to} f}{\lambda x^A.e \overset{\beta}{\to} \lambda x^A.f}$$

6. Reduce product argument type:
$$\frac{A \overset{\beta}{\to} B}{\Pi x^A.C \overset{\beta}{\to} \Pi x^B.C}$$

7. Reduce product result type

$$\frac{B \overset{\beta}{\to} C}{\Pi x^A.B \overset{\beta}{\to} \Pi x^A.C}$$

Note that beta reduction is not deterministic. There might be two possibilities to reduce an application, a product and an abstraction. And there might be more ambiguous subterms contained. In section *Confluence* 4 we prove that the choice of the redex does not affect the final result.

Lemma 3.9. *Reduction to sort lemma*: A term which reduces to a sort must be a redex where the sort is the body or the abstraction is the identity function and the argument is the sort.

$$\frac{t \overset{\beta}{\to} s}{\big(\exists A a. t = (\lambda x^A.s)a\big) \vee \big(\exists A. t = (\lambda x^A.x)s\big)}$$

Proof. All rules except the redex rule reduce to something which cannot be syntactically a sort. Therefore the term has to be a redex which in general has the form $(\lambda x^A.e)a$. The redex reduces to $e[x := a]$ which by the substitution to sort lemma 3.6 proves the goal. □

Theorem 3.10. *Substitute Reduction* A reduction remains valid if we do the same substitution before and after the reduction.

$$\frac{t \overset{\beta}{\to} u}{t[y := v] \overset{\beta}{\to} u[y := v]}$$

(Note that iterated application of this lemma results in the more general statement $t \overset{\beta^*}{\to} u \Rightarrow t[y := v] \overset{\beta^*}{\to} u[y := v]$.)

Proof. Proof by induction on $t \overset{\beta}{\to} u$.

1. Redex: We have to prove the goal

$$((\lambda x^A.e)a)[y := v] \overset{\beta}{\to} e[x := a][y := v]$$

We can see this by the sequence

$$\begin{array}{rcl} ((\lambda x^A.e)a)[y := v] & = & (\lambda x^{A[y:=v]}.e[y := v])a[y := v] \\ & \overset{\beta}{\to} & e[y := v][x := a[y := v]] \\ & = & \text{by Double substitution lemma 3.7} \\ & & e[x := a][y := v] \end{array}$$

2. Reduce function: We have to prove the goal

$$\begin{array}{c|c} f \overset{\beta}{\to} g & f[y := v] \overset{\beta}{\to} g[y := v] \\ \hline fa \overset{\beta}{\to} ga & (fa)[y := v] \overset{\beta}{\to} (ga)[y := v] \end{array}$$

   The validity of the final goal in the right lower corner can be seen by the following reasoning

$$\begin{array}{rcl} (fa)[y := v] & = & f[y := v]a[y := v] \\ & \overset{\beta}{\to} & g[y := v]a[y := v] \\ & = & (ga)[y := v] \end{array}$$

3. Other rules: All other rules follow the same pattern as the proof of the rule *reduce function*.

□

Theorem 3.11. *Reduction in substitution term.* A reduction in the substitution term makes the corresponding substituted terms reducing in zero or more steps as well.

$$\frac{a \overset{\beta}{\to} b}{t[x := a] \overset{\beta^*}{\to} t[x := b]}$$

Proof. Intuitively it might be clear that the substituted term reduces in zero or more steps, because the variable x might be contained in the term t zero or more times.

We prove the statement by induction on the structure of t.

1. Sort: Trival.
2. Variable: We have to distinguish the cases that the variable is the same as x or is different from x. In both case the goal is trivial.
3. Abstraction: We have to prove the goal

$$(\lambda y^B.e)[x := a] \overset{\beta^*}{\to} (\lambda y^B.e)[x := b]$$

   The goal is a consequence of the induction hypotheses $B[x := a] \overset{\beta^*}{\to} B[x := b]$ and $e[x := a] \overset{\beta^*}{\to} e[y := b]$.
4. Product: Same as abstraction.

5. Application: Same as abstraction.

□

Lemma 3.12. *Products and abstractions are preserved under beta reduction*

1. $$\frac{\Pi x^A.B \overset{\beta}{\to} t}{\left(\exists C.A \overset{\beta}{\to} C \land t = \Pi x^C.B\right) \lor \left(\exists D.B \overset{\beta}{\to} D \land t = \Pi x^A.D\right)}$$

2. $$\frac{\lambda x^A.e \overset{\beta}{\to} t}{\left(\exists B.A \overset{\beta}{\to} B \land t = \lambda x^B.e\right) \lor \left(\exists f.e \overset{\beta}{\to} f \land t = \lambda x^A.f\right)}$$

Proof. The proofs for product and abstraction follow the same pattern. Therefore we prove only the preservation of products.

Assume $\Pi x^A.B \overset{\beta}{\to} t$ and do induction on it. Only the two product rules are syntactically possible. Each one guarantees one alternative of the goal. □

3.7 Beta Equivalence

In arithmetics of numbers the terms $3+4$, 7 and $2+5$ are equivalent. Syntactically the terms are different, however we regard the terms equivalent because they represent the same value. In arithmetics the equivalence is so strong that we even consider the terms being equal and write $3+4 = 2+5 = 7$.

In lambda calculus we only mark two terms as equal if they are syntactically the same terms except for irrelevant renamings of bound variables. In the literature of lambda calculus this equality is usually called α-equivalence. We say that α-equivalent terms represent the *same* term.

In lambda calculus we call terms which represent the same value β-equivalent terms. E.g. the terms $(\lambda x^A.e)a$ and $e[x := a]$ are called (*beta-*) equivalent because they represent the same value. The former before the compuation step and the latter after the computation step.

Beta equivalence is just the reflexive, symmetric and transitive closure of the beta reduction.

Definition 3.13. We define *beta equivalence* as a binary relation $a \overset{\beta}{\sim} b$ between the terms a and b inductively by the rules

1. Reflexive:
$$a \overset{\beta}{\sim} a$$

2. Forward:
$$\begin{array}{c} a \overset{\beta}{\sim} b \\ b \overset{\beta}{\to} c \\ \hline a \overset{\beta}{\sim} c \end{array}$$

3. Backward:
$$\begin{array}{c} a \overset{\beta}{\sim} b \\ c \overset{\beta}{\to} b \\ \hline a \overset{\beta}{\sim} c \end{array}$$

In other words the beta equivalence relation $\overset{\beta}{\sim}$ is the smallest equivalence relation which contains beta reduction $\overset{\beta}{\to}$.

Theorem 3.14. *Beta equivalence is transitive.*

$$\begin{array}{c} a \overset{\beta}{\sim} b \\ b \overset{\beta}{\sim} c \\ \hline a \overset{\beta}{\sim} c \end{array}$$

Proof. Assume $a \overset{\beta}{\sim} b$. We prove the goal by induction on $b \overset{\beta}{\sim} c$.

1. Reflexive: Trivial.

2. Forward:
$$\begin{array}{c|c} b \overset{\beta}{\sim} c & a \overset{\beta}{\sim} c \\ c \overset{\beta}{\to} d & \\ \hline b \overset{\beta}{\sim} d & a \overset{\beta}{\sim} d \end{array}$$
The goal in the lower right corner is proved by the induction hypothesis and applying the forward rule.

3. Backward:
$$\begin{array}{c|c} b \overset{\beta}{\sim} c & a \overset{\beta}{\sim} c \\ d \overset{\beta}{\to} c & \\ \hline b \overset{\beta}{\sim} d & a \overset{\beta}{\sim} d \end{array}$$
The goal in the lower right corner is proved by the induction hypothesis and applying the backward rule.

□

Theorem 3.15. *Beta equivalence is symmetric.*

$$\frac{a \overset{\beta}{\sim} b}{b \overset{\beta}{\sim} a}$$

Proof. By induction on $a \overset{\beta}{\sim} b$.

1. Reflexive: Trivial
2. Forward:

$$\begin{array}{l|l} a \overset{\beta}{\sim} b_0 & b_0 \overset{\beta}{\sim} a \\ b_0 \overset{\beta}{\to} b & \\ \hline a \overset{\beta}{\sim} b & b \overset{\beta}{\sim} a \end{array}$$

   First we use the reflexive rule to derive $b \overset{\beta}{\sim} b$ and then the second premise $b_0 \overset{\beta}{\to} b$ and the backward rule to derive $b \overset{\beta}{\sim} b_0$.

   Then we use the induction hypothesis $b_0 \overset{\beta}{\sim} a$ and the transitivity of beta equivalence 3.14 to derive the goal $b \overset{\beta}{\sim} a$.

3. Backward:

$$\begin{array}{l|l} a \overset{\beta}{\sim} b_0 & b_0 \overset{\beta}{\sim} a \\ b \overset{\beta}{\to} b_0 & \\ \hline a \overset{\beta}{\sim} b & b \overset{\beta}{\sim} a \end{array}$$

   Similar reasoning as with the forward rule. The second premise implies $b \overset{\beta}{\sim} b_0$ and the induction hypothesis and transitivity imply the goal.

□

Theorem 3.16. *The same substitution applied to beta equivalent terms results in beta equivalent terms.*

$$\frac{t \overset{\beta}{\sim} u}{t[x := a] \overset{\beta}{\sim} u[x := a]}$$

Proof. By induction on $t \overset{\beta}{\sim} u$:

The reflexive case is trivial, because t and u are identical.

For the other two cases the goal is a consequence of the induction hypothesis and the transitivity of beta equivalence 3.14. □

4 Confluence

4.1 Overview

We want to be able to use the calculus of constructions for computation. The basic computation step is beta reduction. A computation in the calculus ends if no more reduction is possible i.e. if the term has been reduced to a normal form.

However beta reduction is ambiguous. There might be more than one possibility to make a beta reduction.

Example 4.1.

$$\begin{tikzcd}
 & (\lambda x.fxx)((\lambda y.y)z) \arrow[dl, "\beta"] \arrow[dr, "\beta"] & \\
f((\lambda y.y)z)((\lambda y.y)z) \arrow[d, "\beta"] & & (\lambda x.fxx)z \arrow[ddl, "\beta"] \\
fz((\lambda y.y)z) \arrow[dr, "\beta"] & & \\
 & fzz &
\end{tikzcd}$$

In the left path we reduce first the outer redex. This reduction duplicates the redex in the argument. Therefore we need three reduction steps to reach the final term fzz.

In the right path we reduce first the redex in the argument of the application and then the outer redex. Therefore we need only two reduction steps to reach the final term fzz.

In that specific example we have shown that both paths end up in the same result. However we have to prove that this is always the case. Otherwise the calculus of constructions would be useless as a calculus.

It turns out that *confluence* is a key property in the proof of uniqueness of results. We define a relation as *confluent* if going an arbitrary number of steps in two directions there are always continuations of the paths to join them. We want to prove confluence of beta reduction.

Before proving confluence of beta reduction we define the confluence of a relation by defining first a diamond property of a relation. The diamond property of a relation is something like a *one step confluence*. Then we show that the confluence of a relation can be proved by finding a diamond between it and its reflexive transitive closure.

In a next step we define a parallel beta reduction, prove that it is a diamond between beta reduction and the reflexive transitive closure of beta reduction. This proves the confluence of beta reduction.

Having the confluence of beta reduction it is easy to prove that beta equivalent terms always have a common reduct, normal forms are unique and some other interesting properties of binders.

4.2 Diamonds and Confluence

Definition 4.2. *Diamond Property* A relation r has the diamond property (or is a diamond) if $a \overset{r}{\to} b$ and $a \overset{r}{\to} c$ implies the existence of some d such that $b \overset{r}{\to} d$ and $c \overset{r}{\to} d$ are valid.

$$\frac{\begin{array}{c} a \overset{r}{\to} b \\ a \overset{r}{\to} c \end{array}}{\exists d. b \overset{r}{\to} d \wedge c \overset{r}{\to} d}$$

We can state the diamond property of a relation r more pictorially as

$$\begin{tikzcd} a \arrow[r, "r"] \arrow[d, "r"] & b \arrow[d, "r"] \\ c \arrow[r, "r"] & \exists d \end{tikzcd}$$

which looks like a diamond if tilted by 45° clockwise.

Definition 4.3. *Confluence* A relation r is confluent if its reflexive transitive closure r^* is a diamond.

Theorem 4.4. *Let r and s be two relations. If s is between r and its reflexive transitive closure i.e. $r \subseteq s \subseteq r^*$ then both closures are the same i.e. $r^* = s^*$.*

Proof. Note that $r \subseteq s$ is defined by $\forall ab. \left[\dfrac{a \overset{r}{\to} b}{a \overset{s}{\to} b} \right]$.

Assume $r \subseteq s \subseteq r^*$. In order to prove $r^* = s^*$ we have to prove $r^* \subseteq s^*$ and $s^* \subseteq r^*$.

1. $r^* \subseteq s^*$: We have to prove the goal

$$\frac{a \overset{r^*}{\to} b}{a \overset{s^*}{\to} b}$$

We use induction on $a \overset{r^*}{\to} b$.

(a) $a = b$: Trivial since $a \xrightarrow{s*} a$ is valid by definition.

(b)

$$\begin{array}{c|c} a \xrightarrow{r^*} b & a \xrightarrow{s^*} b \\ b \xrightarrow{r} c & \\ \hline a \xrightarrow{r^*} c & a \xrightarrow{s^*} c \end{array}$$

In order to prove the final goal in the lower right corner we start from the induction hypothesis $a \xrightarrow{s^*} b$.
Since $r \subseteq s$ we have $b \xrightarrow{s} c$ by definition of $\subseteq$ and the second premise.
Then by definition of the reflexive transitive closure we conclude the final goal.

2. $s^* \subseteq r^*$: We have to prove the goal

$$\frac{a \xrightarrow{s^*} b}{a \xrightarrow{r^*} b}$$

We use induction on $a \xrightarrow{s^*} b$.

(a) $a = b$: Trivial since $a \xrightarrow{r*} a$ is valid by definition.

(b)

$$\begin{array}{c|c} a \xrightarrow{s^*} b & a \xrightarrow{r^*} b \\ b \xrightarrow{s} c & \\ \hline a \xrightarrow{s^*} c & a \xrightarrow{r^*} c \end{array}$$

Start with the induction hypothesis $a \xrightarrow{r^*} b$.

Since $s \subseteq r^*$ is given we can infer $b \xrightarrow{r^*} c$ from the second premise.

r^* is transitive and therefore $a \xrightarrow{r^*} c$ must be valid.

□

Lemma 4.5. If r is a diamond then

$$\begin{tikzcd} a \arrow[r, "r^*"] \arrow[d, "r"] & b \arrow[d, "r"] \\ c \arrow[r, "r^*"] & \exists d \end{tikzcd}$$

is valid.

Proof. By induction on $a \xrightarrow{r^*} b$.

1. In the reflexive case we have $a = b$. We choose $d = c$ which satisfies the required properties.
2. $$\begin{array}{l|l} a \xrightarrow{r^*} b & \forall d. \left[\begin{tikzcd} a \arrow[r, "r^*"] \arrow[d, "r"] & b \arrow[d, "r"] \\ d \arrow[r, "r^*"] & \exists e \end{tikzcd} \right] \\ b \xrightarrow{r} c & \\ \hline a \xrightarrow{r^*} c & \forall d. \left[\begin{tikzcd} a \arrow[r, "r^*"] \arrow[d, "r"] & c \arrow[d, "r"] \\ d \arrow[r, "r^*"] & \exists f \end{tikzcd} \right] \end{array}$$

   To prove the goal in the lower right corner we assume $a \xrightarrow{r} d$ and try to find some f with the required properties.

   From the induction hypothesis we find some e which satisfies $b \xrightarrow{r} e$ and $d \xrightarrow{r^*} e$.

   Since r is a diamond there exists some f with $\begin{tikzcd} b \arrow[r, "r"] \arrow[d, "r"] & c \arrow[d, "r"] \\ e \arrow[r, "r"] & \exists f \end{tikzcd}$. By glueing the boxes together we see that f satisfies the required properties.

   $$\begin{tikzcd} a \arrow[r, "r^*"] \arrow[d, "r"] & b \arrow[r, "r"] \arrow[d, "r"] & c \arrow[d, "r"] \\ d \arrow[r, "r^*"] & e \arrow[r, "r"] & f \end{tikzcd}$$

□

Lemma 4.6. *If r is a diamond, then r^* is a diamond as well.*

$$\begin{tikzcd} a \arrow[r, "r^*"] \arrow[d, "r^*"] & b \arrow[d, "r^*"] \\ c \arrow[r, "r^*"] & \exists d \end{tikzcd}$$

Proof. We assume that r is a diamond and $a \xrightarrow{r^*} b$ and do induction on $a \xrightarrow{r^*} c$

1. In the reflexive case we have $a = c$. We use $d = b$ which satisfies the required properties.

2.

$$\begin{array}{c|c} a \xrightarrow{r^*} c & \begin{tikzcd} a \arrow[r, "r^*"] \arrow[d, "r^*"'] & b \arrow[d, "r^*"] \\ c \arrow[r, "r^*"] & \exists e \end{tikzcd} \\ c \xrightarrow{r} d & \\ \hline a \xrightarrow{r^*} d & \begin{tikzcd} a \arrow[r, "r^*"] \arrow[d, "r^*"'] & b \arrow[d, "r^*"] \\ d \arrow[r, "r^*"] & \exists f \end{tikzcd} \end{array}$$

We have to prove the goal in the lower right corner i.e. find some f with the required properties.

From the induction hypothesis we find some e. Using the previous lemma 4.5 we find some f such that we can glue the boxes together in order to see that f satisfies the required properties.

$$\begin{tikzcd} a \arrow[r, "r^*"] \arrow[d, "r^*"'] & b \arrow[d, "r^*"] \\ c \arrow[r, "r^*"] \arrow[d, "r"'] & e \arrow[d, "r"] \\ d \arrow[r, "r^*"] & f \end{tikzcd}$$

□

Now we can prove the basic theorem of this subsection:

Theorem 4.7. *In order to prove the confluence of a relation it is sufficient to find a diamond between it and its reflexive transitive closure.*

$$\begin{array}{c} r \subseteq s \subseteq r^* \\ s \text{ is a diamond} \\ \hline r \text{ is confluent} \end{array}$$

Proof. Assume $r \subseteq s \subseteq r^*$ and s is a diamond.

By 4.4 we know that both reflexive transitive closures are the same i.e. $r^* = s^*$.

From 4.6 we conclude that s^* is a diamond. This implies that r^* is a diamond as well and proves the fact that r is confluent by definition of confluence. □

4.3 Parallel Reduction Relation

Looking at the example 4.1 it can be seen that beta reduction is not a diamond. Let's analyze the example a little bit to see why beta reduction is not a diamond. Assume there is a redex like

$$(\lambda x^A.e)a$$

where the subterm a contains redexes as well i.e. there is some b with $a \overset{\beta}{\to} b$. Then the following two reduction paths are possible.

$$\begin{array}{ccccc} (\lambda x^A.e)a & \overset{\beta}{\to} & (\lambda x^A.e)b & \overset{\beta}{\to} & e[x := b] \\ (\lambda x^A.e)a & \overset{\beta}{\to} & e[x := a] & \overset{\beta^*}{\to} & e[x := b] \end{array}$$

In the first path we reach the term $e[x := b]$ in two reduction steps. In the second path we can reach the term $e[x := b]$ as well. But the needed number of steps depends on how often the variable x is present in the term e. If x is contained in the term e n times we need $n+1$ reduction steps in total because we have to reduce the term a in $e[x := a]$ to the term b n times. If beta reduction were a diamond, it would be required that n is always 1 which is not the case in general.

In order to prove the confluence of beta reduction we have to find a diamond between beta reduction and its reflexive transitive closure. Let's call this relation β_p. This relation is like beta reduction β but must allow more reductions steps to remedy the situation that in a redex $(\lambda x^A.e)a$ the variable x might be contained zero or more times in the subterm e.

- In order to remedy the situation that the variable x is not contained in the subterm e we make the relation β_p reflexive.
- In order to remedy the situation that the variable x is contained two or more times in the subterm e we allow that β_p reductions can happen in any subterm of a term. Therefore we call β_p *parallel* beta reduction because the reduction can happen in the subterms in parallel.

Definition 4.8. The *parallel reduction relation* $a \overset{\beta_p}{\to} b$ is defined inductively by the rules

1. Reflexive

$$a \overset{\beta_p}{\to} a$$

2. Redex

$$\begin{array}{c} a \overset{\beta_p}{\to} b \\ e \overset{\beta_p}{\to} f \\ \hline (\lambda x^A.e)a \overset{\beta_p}{\to} f[x := b] \end{array}$$

3. Product

$$\begin{array}{c} A \overset{\beta_p}{\to} C \\ B \overset{\beta_p}{\to} D \\ \hline \Pi x^A.B \overset{\beta_p}{\to} \Pi x^C.D \end{array}$$

4. Abstraction

$$\begin{array}{c} A \overset{\beta_p}{\to} B \\ e \overset{\beta_p}{\to} f \\ \hline \lambda x^A.e \overset{\beta_p}{\to} \lambda x^B.f \end{array}$$

5. Application

$$\begin{array}{c} a \overset{\beta_p}{\to} c \\ b \overset{\beta_p}{\to} d \\ \hline ab \overset{\beta_p}{\to} cd \end{array}$$

Lemma 4.9. *Parallel beta reduction is a superset of beta reduction*

Proof. This fact is trivial, because all rules of beta reduction are contained as special cases within the rules of parallel beta reduction. □

Lemma 4.10. *Parallel beta reduction is a subset of the transitive closure of beta reduction*

Proof. All rules of parallel beta reduction are satisfied by the transitive closure of beta reduction. Since an inductive relation defined by some rules is the smallest relation which satisfies the rules, it is evident that parallel beta reduction must be smaller than the transitive closure of beta reduction. □

Lemma 4.11. *Basic compatibility of parallel reduction and substitution*

$$\begin{array}{c} t \overset{\beta_p}{\to} u \\ \hline a[x := t] \overset{\beta_p}{\to} a[x := u] \end{array}$$

Proof. By induction on the structure of a.

1. a is a sort: Trivial, because substitution does not change a sort and parallel beta reduction is reflexive.
2. a is a variable, let's say y: In the case $x = y$ the goal is implied by the premise. In the case $x \neq y$ the goal is implied by reflexivity.
3. a is the product $\Pi y^B.C$: We have to prove the goal $(\Pi y^B.C)[x := t] \overset{\beta_p}{\to} (\Pi y^B.C)[x := u]$ from the premise $t \overset{\beta_p}{\to} u$ and the induction hypotheses $B[x := t] \overset{\beta_p}{\to} B[x := u]$ and $C[x := t] \overset{\beta_p}{\to} C[x := u]$. The validity of the goal can be seen from the following derivation.

$$\begin{array}{lcll}
(\Pi y^B.C)[x := t] & = & \Pi y^{B[x:=t]}.C[x := t] & \text{definition of substitution} \\
 & \overset{\beta_p}{\to} & \Pi y^{B[x:=u]}.C[x := u] & \text{induction hypothesis} \\
 & = & (\Pi y^B.C)[x := u] & \text{definition of substitution}
\end{array}$$

4. a is the abstraction $\lambda y^B.e$: Same reasoning as with product.
5. a is the application $f a$: Same reasoning as with product.

□

Lemma 4.12. *Full compatibility of parallel reduction and substitution*

$$\begin{array}{l}
a \overset{\beta_p}{\to} b \\
t \overset{\beta_p}{\to} u \\
\hline
a[x := t] \overset{\beta_p}{\to} b[x := u]
\end{array}$$

Proof. By induction on $a \overset{\beta_p}{\to} b$.

1. Reflexive: In that case we have $a = b$. The goal is an immediate consequence of lemma 4.11
2. Redex:

$$\begin{array}{l|l}
a \overset{\beta_p}{\to} b & a[x := t] \overset{\beta_p}{\to} b[x := u] \\
e \overset{\beta_p}{\to} f & e[x := t] \overset{\beta_p}{\to} f[x := u] \\
\hline
(\lambda y^A.e)a \overset{\beta_p}{\to} f[y := b] & ((\lambda y^A.e)a)[x := t] \overset{\beta_p}{\to} f[y := b][x := u]
\end{array}$$

The validity of the goal in the lower right corner can be seen from the following reasoning:

$$\begin{array}{lcll}
((\lambda y^A.e)a)[x := t] & = & (\lambda y^{A[x:=t]}.e[x := t])a[x := t] & \text{definition of substitution} \\
 & \overset{\beta_p}{\to} & f[x := u][y := b[x := u]] & \text{induction hypotheses} \\
 & = & f[y := b][x := u] & \text{double substitution 3.7}
\end{array}$$

3. Product, abstraction and application: Some reasoning as with *redex*. The lemma 3.7 is not needed in these cases.

□

Lemma 4.13. *The product and abstraction are preserved under parallel reduction*

1. $$\frac{\lambda x^A.e \overset{\beta_p}{\to} c}{\exists B f. c = \lambda x^B.f \wedge A \overset{\beta_p}{\to} B \wedge e \overset{\beta_p}{\to} f}$$

2. $$\frac{\Pi x^A.B \overset{\beta_p}{\to} c}{\exists C D. c = \Pi x^C.D \wedge A \overset{\beta_p}{\to} C \wedge B \overset{\beta_p}{\to} D}$$

Proof. By induction on the premise. In both cases only one rule is syntactically possible which guarantees the existence of the corresponding terms. □

Theorem 4.14. *Parallel reduction is a diamond*

$$\begin{tikzcd}
a \arrow[r, "\beta_p"] \arrow[d, "\beta_b"'] & b \arrow[d, "\beta_b"] \\
c \arrow[r, "\beta_p"] & \exists d
\end{tikzcd}$$

Proof. By induction on $a \overset{\beta_p}{\to} b$. Note that we keep the variable c in the following universally quantified.

1. Reflexive

$$a \overset{\beta_p}{\to} a \;\Bigg|\; \forall c. \begin{bmatrix} a & \overset{\beta_p}{\to} & a \\ \downarrow_{\beta_b} & & \downarrow_{\beta_b} \\ c & \overset{\beta_p}{\to} & \exists d \end{bmatrix}$$

To prove the goal on the right side we assume $a \overset{\beta_p}{\to} c$. Then we use c for d which satisfies the required property of d trivially.

2. Redex

$$\begin{array}{l|l}
e \overset{\beta_p}{\to} f & \forall g. \left[\begin{array}{ccc} e & \overset{\beta_p}{\to} & f \\ \downarrow_{\beta_b} & & \downarrow_{\beta_b} \\ g & \overset{\beta_p}{\to} & \exists h \end{array}\right] \\
a \overset{\beta_p}{\to} b & \forall c. \left[\begin{array}{ccc} a & \overset{\beta_p}{\to} & b \\ \downarrow_{\beta_b} & & \downarrow_{\beta_b} \\ c & \overset{\beta_p}{\to} & \exists d \end{array}\right] \\ \hline
(\lambda x^A.e)a \overset{\beta_p}{\to} f[x := b] & \forall k. \left[\begin{array}{ccc} (\lambda x^A.e)a & \overset{\beta_p}{\to} & f[x := b] \\ \downarrow_{\beta_b} & & \downarrow_{\beta_b} \\ k & \overset{\beta_p}{\to} & \exists n \end{array}\right]
\end{array}$$

To prove the goal in the lower right corner we assume $(\lambda x^A.e)a \overset{\beta_p}{\to} k$ and do a case split on the construction of this relation.

(a) Reflexive: In that case $k = (\lambda x^A.e)a$. We use $n = f[x := b]$ which has the required property.

(b) Redex: In that case $k = g[x := c]$ for some g and c with the properties $e \overset{\beta_p}{\to} g$ and $a \overset{\beta_p}{\to} c$.

We have to find a term n with $f[x := b] \overset{\beta_p}{\to} n \wedge g[x := c] \overset{\beta_p}{\to} n$. The term

$$n = h[x := d]$$

with the terms h and d which exist by the induction hypotheses. It is easy to see that the properties

$$\begin{array}{lcl} f[x := b] & \overset{\beta_p}{\to} & h[x := d] \\ g[x := c] & \overset{\beta_p}{\to} & h[x := d] \end{array}$$

are satisfied because of the induction hypotheses and lemma 4.12

(c) Product: This case is syntactically impossible because $(\lambda x^A.e)a$ cannot be a product.

(d) Abstraction: This case is syntactically impossible because $(\lambda x^A.e)a$ cannot be an abstraction.

(e) Application: In that case $k = (\lambda x^B.g)c$ for some B, g and c with $A \overset{\beta_p}{\to} B$, $e \overset{\beta_p}{\to} g$ and $a \overset{\beta_p}{\to} c$. Because of lemma 4.13 we have chosen the more specific term $\lambda x^B.g$ instead of a more general term.

We have to find a term n which satisfies $(\lambda x^B.g)c \overset{\beta_p}{\to} n$ and $f[x := b] \overset{\beta_p}{\to} n$.

We use the term

$$n = h[x := d]$$

with the terms h and d which exist by the induction hypotheses satisfying

$$\begin{array}{ccc} f & \overset{\beta_p}{\to} & h \\ g & \overset{\beta_p}{\to} & h \\ b & \overset{\beta_p}{\to} & d \\ c & \overset{\beta_p}{\to} & d \end{array}$$

Therefore the goals

$$\begin{array}{ccc} (\lambda x^B.g)c & \overset{\beta_p}{\to} & h[x := d] \\ f[x := b] & \overset{\beta_p}{\to} & h[x := d] \end{array}$$

are satisfied

3. Product

$$\begin{array}{c|c}
A \overset{\beta_p}{\to} C & \forall E. \left[\begin{array}{ccc} A & \overset{\beta_p}{\to} & C \\ \downarrow_{\beta_b} & & \downarrow_{\beta_b} \\ E & \overset{\beta_p}{\to} & \exists H \end{array}\right] \\
B \overset{\beta_p}{\to} D & \forall F. \left[\begin{array}{ccc} B & \overset{\beta_p}{\to} & D \\ \downarrow_{\beta_b} & & \downarrow_{\beta_b} \\ F & \overset{\beta_p}{\to} & \exists J \end{array}\right] \\
\hline
\Pi x^A.B \overset{\beta_p}{\to} \Pi x^C.D & \forall EF. \left[\begin{array}{ccc} \Pi x^A.B & \overset{\beta_p}{\to} & \Pi x^C.D \\ \downarrow_{\beta_b} & & \downarrow_{\beta_b} \\ \Pi x^E.F & \overset{\beta_p}{\to} & \exists n \end{array}\right]
\end{array}$$

In order to prove the goal in the lower right corner we assume $\Pi x^A.B \overset{\beta_p}{\to} \Pi x^E.F$.

Because of lemma 4.13 which says that parallel reduction preserves products we have chosen the more specific $\Pi x^E.F$ which satisfies $A \overset{\beta_p}{\to} E$ and $B \overset{\beta_p}{\to} F$ instead of a more general term.

From the induction hypotheses we conclude the existence of the terms H and J such that

$$n = \Pi x^H.J$$

satisfies the required properties.

4. Abstraction: Same reasoning as with product.
5. Application

$$
\begin{array}{c|c}
a \overset{\beta_p}{\to} c & \forall e. \begin{bmatrix} a & \overset{\beta_p}{\to} & c \\ \downarrow_{\beta_b} & & \downarrow_{\beta_b} \\ e & \overset{\beta_p}{\to} & \exists g \end{bmatrix} \\
b \overset{\beta_p}{\to} d & \forall f. \begin{bmatrix} b & \overset{\beta_p}{\to} & d \\ \downarrow_{\beta_b} & & \downarrow_{\beta_b} \\ f & \overset{\beta_p}{\to} & \exists h \end{bmatrix} \\
\hline
ab \overset{\beta_p}{\to} cd & \forall k. \begin{bmatrix} ab & \overset{\beta_p}{\to} & cd \\ \downarrow_{\beta_b} & & \downarrow_{\beta_b} \\ k & \overset{\beta_p}{\to} & \exists n \end{bmatrix}
\end{array}
$$

To prove the goal in the lower right corner we assume $ab \overset{\beta_p}{\to} k$ and do a case split on the construction of this relation.

(a) Reflexive: In that case $k = ab$. We use $n = cd$ which satisfies the required properties.

(b) Redex: In this case ab has to be a redex, let's say $(\lambda x^A.m)b$. Therefore and because of lemma 4.13 $ab \overset{\beta_p}{\to} cd$ becomes $(\lambda x^A.m)b \overset{\beta_p}{\to} (\lambda x^B.o)d$ for some B and o with $A \overset{\beta_p}{\to} B$ and $m \overset{\beta_p}{\to} o$.
k has to be the reduct $p[x := f]$ for some p, f with $m \overset{\beta_p}{\to} p$ and $b \overset{\beta_p}{\to} f$.
We have to find some term n which satisfies

$$
\begin{array}{lcl}
(\lambda x^B.o)d & \overset{\beta_p}{\to} & n \\
p[x := f] & \overset{\beta_p}{\to} & n
\end{array}
$$

From the first induction hypothesis and lemma 4.13 we postulate the existence of q with $o \overset{\beta_p}{\to} q$ and $p \overset{\beta_p}{\to} q$.
From the second induction hypothesis we conclude the existence of some h with with $d \overset{\beta_p}{\to} h$ and $f \overset{\beta_p}{\to} h$.
Therefore the term

$$n = q[x := h]$$

satisfies the requirement.

(c) Product: This case is syntactically impossible because ab cannot be a product.

(d) Abstraction: This case is syntactically impossible because ab cannot be an abstraction.

(e) Application: In that case $k = ef$ for some terms e, f which satisfy $a \overset{\beta_p}{\to} e$ and $b \overset{\beta_p}{\to} f$. We have to find some term n which satisfies $cd \overset{\beta_p}{\to} n$ and $ef \overset{\beta_p}{\to} n$.
By the induction hypotheses there exist some terms g and h satisfying $c \overset{\beta_p}{\to} g$, $e \overset{\beta_p}{\to} g$, $d \overset{\beta_p}{\to} h$ and $f \overset{\beta_p}{\to} h$ such that the term

$$n = gh$$

satisfies the requirement.

□

Theorem 4.15. *Church Rosser theorem: Beta reduction is confluent.*

Proof. With the parallel beta reduction we have found a relation which is

1. a diamond (4.14)
2. between beta reduction and its reflexive transitive closure (4.9, 4.10)

I.e. with parallel beta reduction we have found a diamond between beta reduction and its reflexive transitive closure which implies by 4.7 that beta reduction is confluent. □

4.4 Equivalent Terms

One of the most important consequences of the Church Rosser theorem (i.e. confluence of beta reduction) is the fact that beta equivelent terms have a common reduct. I.e. for two beta equivalent terms a and b there exists always a term c to which both reduce

$$\begin{tikzcd}
a \arrow[dr, "\beta^*"'] & \overset{\beta}{\sim} & b \arrow[dl, "\beta^*"] \\
 & \exists c &
\end{tikzcd}$$

Theorem 4.16. *Equivalent terms have a common reduct.*

$$\frac{a \overset{\beta}{\sim} b}{\exists c. a \overset{\beta^*}{\to} c \land b \overset{\beta^*}{\to} c}$$

Proof. By induction on $a \overset{\beta}{\sim} b$.

1. Reflexive: In that case $a = b$. We use $c = a$ which satisfies the required properties trivially.

2. Forward:

$$\begin{array}{r|l} a \overset{\beta}{\sim} b & \exists d. a \overset{\beta^*}{\to} d \land b \overset{\beta^*}{\to} d \\ b \overset{\beta}{\to} c & \\ \hline a \overset{\beta}{\sim} c & \exists e. a \overset{\beta^*}{\to} e \land c \overset{\beta^*}{\to} e \end{array}$$

   In order to prove the goal in the lower right corner we construct terms d and e which satisfy the following diagram and therefore e satisfies the required properties.

$$\begin{tikzcd}
a \arrow[r, phantom, "\overset{\beta}{\sim}"] \arrow[dr, "\beta^*"] & b \arrow[r, "\beta"] \arrow[d, "\beta^*"] & c \arrow[d, "\beta^*"] \\
 & d \arrow[r, "\beta^*"] & e
\end{tikzcd}$$

   d exists by the induction hypothesis and e exists by confluence 4.15

3. Backward:

$$\begin{array}{r|l} a \overset{\beta}{\sim} b & \exists d. a \overset{\beta^*}{\to} d \land b \overset{\beta^*}{\to} d \\ c \overset{\beta}{\to} b & \\ \hline a \overset{\beta}{\sim} c & \exists d. a \overset{\beta^*}{\to} d \land c \overset{\beta^*}{\to} d \end{array}$$

   In order to prove the goal in the lower right corner we construct term d which satisfies the following diagram and therefore the required properties.

$$\begin{tikzcd}
a \arrow[r, phantom, "\overset{\beta}{\sim}"] \arrow[dr, "\beta^*"] & b \arrow[d, "\beta^*"] & c \arrow[l, "\beta"'] \arrow[dl, "\beta^*"] \\
 & d &
\end{tikzcd}$$

   The term d exists by the induction hypothesis.

□

4.5 Uniqueness of Normal Forms

Theorem 4.17. *The normal form of a lambda term is unique.*

$$\frac{\begin{array}{c} t \overset{\beta^*}{\to} u \\ t \overset{\beta^*}{\to} v \\ u \text{ and } v \text{ are in normal form} \end{array}}{u = v}$$

Proof. u and v are beta equivalent. According to the theorem 4.16 both have to reduce to a common reduct, say w. Since both are in normal form, the common reduct has to be reached in zero reduction steps (no reduction step is possible in normal form) which is possible only if u and v are the same term. □

4.6 Equivalent Binders

Theorem 4.18. *In beta equivalent binders i.e. terms of the form* $\lambda x^A.e$ *or* $\Pi x^A.B$ *corresponding subterms are beta equivalent.*

$$\frac{\Pi x^A.B \overset{\beta}{\sim} \Pi x^C.D}{A \overset{\beta}{\sim} C \wedge B \overset{\beta}{\sim} D} \qquad \frac{\lambda x^A.e \overset{\beta}{\sim} \lambda x^B.f}{A \overset{\beta}{\sim} B \wedge e \overset{\beta}{\sim} f}$$

Proof. We only prove the theorem for products (the proof for abstractions is practically the same).

Since both products are beta equivalent they have by 4.16 a common reduct. By 3.12 products (and abstractions) are preserved under beta reduction. Therefore the common reduct must have the form $\Pi x^E.F$ with $A \overset{\beta^*}{\to} E$, $B \overset{\beta^*}{\to} F$, $C \overset{\beta^*}{\to} E$ and $D \overset{\beta^*}{\to} F$.

Since reduction implies beta equivalence and beta equivalence is transitive and symmetric we get $A \overset{\beta}{\sim} C$ and $B \overset{\beta}{\sim} D$. □

4.7 Binders are not Equivalent to Variables or Sorts

Theorem 4.19. *Binders (products or abstractions) cannot be beta equivalent to variables or sorts.*

Proof. We only prove

$$\frac{\Pi x^A.B \overset{\beta}{\sim} s}{\perp}$$

because the proofs of the other variants follow the same pattern.

Assume $\Pi x^A.B \overset{\beta}{\sim} s$. Both terms must have a common reduct by 4.16 which must be the sort s since a sort is already in normal form. I.e. we get $\Pi x^A.B \overset{\beta^*}{\to} s$.

Since reduction preserves the form of binders by 3.12 $\Pi x^A.B \overset{\beta^*}{\to} s$ is not possible and we get the desired contradiction. □

5 Typing

This section describes welltyped terms and basic properties of welltyped terms. It is based on Henk Barendregt's paper *Lambda Calculi with Types* [1].

What is needed to state that a term t is welltyped? First we have to notice that the term t might contain free variables. As opposed to bound variables, free variables don't have a type expressed within a term. Therefore we need a context Γ which assigns types to the free variables in the term t. Furthermore we need a type T. We write the statement *The term t has type T in the context Γ as*

$$\Gamma \vdash t : T$$

We need a typing relation which is a subset of

$$\text{Contexts} \times \text{Terms} \times \text{Terms}$$

In this section we define the typing relation inductively.

Since terms are either sorts, variables, products, abstractions or applications, the typing relation has one introduction rule for each possible form of a term which states how to introduce a welltyped term of that form.

Furthermore there are two structural rules which state

- If we add a welltyped variable to a context, then every welltyped term in the context is welltyped in the augmented context in which it has the same type.
- If a term t has type T and the type U is beta equivalent to T and a welltyped type, then the term t has type U in the same context.

The last rule states that beta equivalent types are really equivalent in their role as types. This allows to do computations (i.e. beta reductions) in types without changing the meaning of a type.

Besides the definition of the typing relation the main part of this section is to prove important properties of welltyped terms. The most important nontrivial theorems/lemmas are:

1. Generation Lemmata: If a term t is welltyped (i.e. it has a type T in a certain context Γ), then there is an equivalent type of a certain form for each possible form the term (sort, variable,

product, abstraction, application). E.g. each welltyped abstraction $\lambda x^A.e$ has a type of the form $\Pi x^A.B$ which is beta equivalent to T.

The generation lemmata describe important properties which are used in the proofs on many other theorems.

2. Type of types: If T is the type of a welltyped term in a certain context, then T is either the sort $\mathcal{U}$ or T is welltyped in the same context and its type is a sort (i.e. either $\mathcal{P}$ or $\mathcal{U}$).

   This theorem justifies the introduction of sorts as *types of types*.

3. Subject reduction: Reduction (i.e. computation) does not change the type of a term. Or in other words: If a term has a certain type and we compute the term, then we really get an object of that type. I.e. computation does what it promises to do.

   Subject reduction is valid in all typed programming languages. If you define a function with a certain result type, then the actual execution of that function returns an object of that type (provided that the computation terminates).

4. Uniqueness of types: If a term is welltyped in a certain context, then all its possible types are beta equivalent. I.e. each welltyped term has a unique type modulo beta equivalence. I.e. we can regard the equivalence class as the unique type of a term.

   Together with subject reduction this theorem guarantees that beta equivalent welltyped terms have the same unique type modulo beta equivalence.

In the last subsection of this section we introduce *kinds*. Since we have the two sorts $\mathcal{P}$ and $\mathcal{U}$ and sorts are the types of types, there are two types of types. Types of type $\mathcal{P}$ and types of type $\mathcal{U}$. The types of type $\mathcal{U}$ are called *kinds*. *Kinds* have the special property that they are recognizable by pure syntactic analysis of the term which represents the type.

Semantically *kinds* are the types of n-ary type functions where $n = 0$ is allowed as a corner case. Therefore if we know that the type of a term is a kind, then we know that the term will return a type, if applied to sufficient arguments (of the correct type of course).

The kinds play an important role in the proof of strong normalization i.e. in the proof of consistency of the calculus of constructions.

5.1 Typing Relation

Definition 5.1. The ternary *typing relation* $\Gamma \vdash t : T$ which says that in the context Γ then term t has type T is defined inductively by the rules

1. Introduction rules:

   (a) Axiom:
   $$[] \vdash \mathcal{P} : \mathcal{U}$$

   (b) Variable:
   $$\frac{\begin{array}{l}\Gamma \vdash A : s \\ x \notin \Gamma\end{array}}{\Gamma, x^A \vdash x : A}$$

   (c) Product:
   $$\frac{\begin{array}{l}\Gamma \vdash A : s_1 \\ \Gamma, x^A \vdash B : s_2\end{array}}{\Gamma \vdash \Pi x^A.B : s_2}$$

   (d) Abstraction:
   $$\frac{\begin{array}{l}\Gamma \vdash \Pi x^A.B : s \\ \Gamma, x^A \vdash e : B\end{array}}{\Gamma \vdash (\lambda x^A.e) : \Pi x^A.B}$$

   (e) Application:
   $$\frac{\begin{array}{l}\Gamma \vdash f : \Pi x^A.B \\ \Gamma \vdash a : A\end{array}}{\Gamma \vdash fa : B[x := a]}$$

2. Structural rules:

   (a) Weaken:
   $$\frac{\begin{array}{l}\Gamma \vdash t : T \\ \Gamma \vdash A : s \\ x \notin \Gamma\end{array}}{\Gamma, x^A \vdash t : T}$$

   (b) Type equivalence:
   $$\frac{\begin{array}{l}\Gamma \vdash t : T \\ \Gamma \vdash U : s \\ T \overset{\beta}{\sim} U\end{array}}{\Gamma \vdash t : U}$$

Remarks:

- There is no introduction rule for $\mathcal{U}$. Therefore there can never be a valid type of $\mathcal{U}$ i.e. the term $\mathcal{U}$ is not welltyped. Its only purpose is to be the type of some types like $\mathcal{P}$ and it can only appear in the type position of a typing judgement $\Gamma \vdash t : T$.
- The variable introduction rule requires a welltyped type A for the introduced variable (i.e. $A : s$ for some sort s). Since $\mathcal{U}$ is not welltyped, it is impossible for a variable to have type $\mathcal{U}$.
- The product type $\Pi x^A.B$ is the type of functions mapping objects of type A to objects of type B where the result type B might contain the variable x. The introduction rule for products requires that both A and B are welltyped types. Therefore neither of them can be $\mathcal{U}$. Therefore a function cannot receive arguments of type $\mathcal{U}$ nor return results of type $\mathcal{U}$.

  It is possible to compute with types but it is impossible to compute with sorts (or kinds in general) as arguments and results. This is an important difference to the *extended calculus of constructions* which allows variables of type $\mathcal{U}_i$ because $\mathcal{U}_i$ has type $\mathcal{U}_{i+1}$. The extended calculus of constructions can compute with kinds, the calculus of constructions not.

5.2 Basic Definitions

Definition 5.2. Basic Definitions:

1. $\Gamma \vdash t : T : U$ is defined as $\Gamma \vdash t : T$ and $\Gamma \vdash T : U$.
2. $\Gamma \vdash \Delta$ for the contexts Γ and Δ is defined as $\forall x^A \in \Delta \Rightarrow \Gamma \vdash x : A$.
3. Γ is a *valid context* if $\Gamma \vdash t : T$ is valid for some terms t and T.
4. A term t is welltyped in a context Γ if $\Gamma \vdash t : T$ is valid for some term T.
5. A term t is welltyped if there is a context in which it is welltyped.
6. A term is valid in a context if it is either welltyped in the context or it is $\mathcal{U}$.
7. A term is valid if there is a context in which it is valid.
8. A term T is a valid type of sort s in a context Γ if $\Gamma \vdash T : s$ is valid.
9. A term is a valid type of some sort if it is a valid type of this sort in some context.

10. A term is a proposition or a proper type if it is a valid type of sort $\mathcal{P}$.
11. A term is a kind if it is a valid type of sort $\mathcal{U}$.

5.3 Start Lemma

Theorem 5.3. For any valid context Γ the following two typing judgements are valid:

$$\begin{array}{l} \Gamma \vdash \mathcal{P} : \mathcal{U} \\ x^A \in \Gamma \Rightarrow \Gamma \vdash x : A \end{array}$$

Proof. A context Γ is valid by definition if $\Gamma \vdash t : T$ is valid for some terms t and T. We prove

$$\begin{array}{l} \Gamma \vdash \mathcal{P} : \mathcal{U} \\ \wedge \\ \forall x^A \in \Gamma \Rightarrow \Gamma \vdash x : A \end{array}$$

by induction on $\Gamma \vdash t : T$.

For all rules except the axiom, the variable rule and the weakening rule the goal is an immediate consequence of the induction hypothesis for the same context. The axiom, the variable and the weakening rule are treated separately.

1. Axiom $[] \vdash \mathcal{P} : \mathcal{U}$: The first part is trivially valid. The second part is vacuously valid, because the empty context does not have any variables.

2. Variable:

$$\begin{array}{l|l} \Gamma \vdash B : s & \Gamma \vdash \mathcal{P} : \mathcal{U} \wedge (\forall x^A \in \Gamma \Rightarrow \Gamma \vdash x : A) \\ y \notin \Gamma & \\ \hline \Gamma, y^B \vdash y : B & \Gamma, x^B \vdash \mathcal{P} : \mathcal{U} \wedge (\forall x^A \in (\Gamma, x^B) \Rightarrow \Gamma, y^B \vdash x : A) \end{array}$$

   The first part of the goal in the lower right corner is a consequence of the induction hypothesis and the weaking rule.

   For the second part we have to distinguish two cases:

   - $x^A \in \Gamma$: In that case the second part of the goal is a consequence of the second part of the induction hypothesis and the weakening rule.
   - $x^A = y^B$: In that case the second part of the goal is identical with the lower left corner.

3. Weakening:

$$\begin{array}{l|l} \Gamma \vdash t : T & \\ \Gamma \vdash B : s & \Gamma \vdash \mathcal{P} : \mathcal{U} \wedge (\forall x^A \in \Gamma \Rightarrow \Gamma \vdash x : A) \\ y \notin \Gamma & \\ \hline \Gamma, y^B \vdash t : B & \Gamma, x^B \vdash \mathcal{P} : \mathcal{U} \wedge (\forall x^A \in (\Gamma, x^B) \Rightarrow \Gamma, y^B \vdash x : A) \end{array}$$

The reasoning is nearly the same as with the variable case. Except the second part of the goal for $x^A = y^B$ is proved by the variable introduction rule by using $\Gamma \vdash B : s$ and $y \notin \Gamma$.

□

5.4 Thinning Lemma

Theorem 5.4. *Thinning Lemma*: Let Δ be a valid context with $\Gamma \subseteq \Delta$. Then

$$\frac{\Gamma \vdash t : T}{\Delta \vdash t : T}$$

Proof. By induction on $\Gamma \vdash t : T$.

1. Sort:

$$[] \vdash \mathcal{P} : \mathcal{U}$$

Since Δ is a valid context we get $\Delta \vdash \mathcal{P} : \mathcal{U}$ by the start lemma 5.3.

2. Variable:

$$\begin{array}{l|l} \Gamma \vdash A : s & \\ x \notin \Gamma & \\ \hline \Gamma, x^A \vdash x : A & \forall \Delta. \left[\begin{array}{c} \Delta \text{ valid} \\ \Gamma, x^A \subseteq \Delta \\ \hline \Delta \vdash x : A \end{array}\right] \end{array}$$

In order to prove the goal in the lower right corner we assume a valid context Δ which is a superset of Γ, x^A. Because of that it has an entry x^A. By the start lemma 5.3 we infer the goal.

3. Product:

$$
\begin{array}{l|l}
\Gamma \vdash A : s_1 & \forall \Delta. \left[\begin{array}{l} \Delta \text{ valid} \\ \Gamma \subseteq \Delta \\ \hline \Delta \vdash A : s_1 \end{array}\right] \\
\Gamma, x^A \vdash B : s_2 & \forall \Delta'. \left[\begin{array}{l} \Delta' \text{ valid} \\ \Gamma, x^A \subseteq \Delta' \\ \hline \Delta' \vdash B : s_2 \end{array}\right] \\
\hline
\Gamma \vdash \Pi x^A.B : s_2 & \forall \Delta. \left[\begin{array}{l} \Delta \text{ valid} \\ \Gamma \subseteq \Delta \\ \hline \Delta \vdash \Pi x^A.B : s_2 \end{array}\right]
\end{array}
$$

   In order to prove the goal in the lower right corner we assume a valid context Δ which is a superset of Γ and derive the following facts:

   (a) $\Delta \vdash A : s_1$: This is an immediate consequence of the first induction hypothesis.

   (b) $\Delta, x^A \vdash B : s_2$: We can assume $x \notin \Delta$. Otherwise we rename the variable such that the condition is satisfied. The context Δ, x^A is a valid context because of $\Delta \vdash A : s_1$ and the variable introduction rule. By the second induction hypothesis we get the subgoal.

   (c) $\Delta \vdash \Pi x^A.B : s_2$: This fact can be derived from the previous two subgoals and the product introduction rule.

   The last fact proves the goal.

4. Abstraction:

   Similar reasoning as in *product*

5. Application:

$$
\begin{array}{l|l}
\Gamma \vdash f : \Pi x^A.B & \forall \Delta. \left[\begin{array}{l} \Delta \text{ valid} \\ \Gamma \subseteq \Delta \\ \hline \Delta \vdash f : \Pi x^A.B \end{array}\right] \\
\Gamma \vdash a : A & \forall \Delta. \left[\begin{array}{l} \Delta \text{ valid} \\ \Gamma \subseteq \Delta \\ \hline \Delta \vdash a : A \end{array}\right] \\
\hline
\Gamma \vdash f a : B[x := a] & \forall \Delta. \left[\begin{array}{l} \Delta \text{ valid} \\ \Gamma \subseteq \Delta \\ \hline \Delta \vdash f a : B[x := a] \end{array}\right]
\end{array}
$$

We assume a valid context Δ which is a superset of Γ. By the two induction hypotheses and the application introduction rule we infer the goal.

6. Weaken:

$$\begin{array}{l|l}
\Gamma \vdash t : T & \forall \Delta. \left[\begin{array}{l} \Delta \text{ valid} \\ \Gamma \subseteq \Delta \\ \hline \Delta \vdash t : T \end{array}\right] \\
\Gamma \vdash A : s & \\
x \notin \Gamma & \\
\hline
\Gamma, x^A \vdash t : T & \forall \Delta. \left[\begin{array}{l} \Delta \text{ valid} \\ \Gamma, x^A \subseteq \Delta \\ \hline \Delta \vdash t : T \end{array}\right]
\end{array}$$

Let's assume a valid context Δ which is a superset of Γ, x^A. This implies that it is a superset of Γ as well. The goal follows immediately from the first induction hypothesis.

7. Type reduction:

$$\begin{array}{l|l}
\Gamma \vdash t : T & \forall \Delta. \left[\begin{array}{l} \Delta \text{ valid} \\ \Gamma \subseteq \Delta \\ \hline \Delta \vdash t : T \end{array}\right] \\
\Gamma \vdash U : s & \forall \Delta. \left[\begin{array}{l} \Delta \text{ valid} \\ \Gamma \subseteq \Delta \\ \hline \Delta \vdash U : s \end{array}\right] \\
T \overset{\beta}{\sim} U & \\
\hline
\Gamma \vdash t : U & \forall \Delta. \left[\begin{array}{l} \Delta \text{ valid} \\ \Gamma \subseteq \Delta \\ \hline \Delta \vdash t : U \end{array}\right]
\end{array}$$

Let's assume a valid context Δ which is a superset of Γ. From the two induction hypotheses we get $\Delta \vdash t : T$ and $\Delta \vdash U : s$. We conclude the final goal by applying the type equivalence rule.

□

5.5 Generation Lemmata

Theorem 5.5. The following generation lemmata are valid:

1. Sort:
$$\frac{\Gamma \vdash s : T}{s = \mathcal{P} \land T \overset{\beta}{\sim} \mathcal{U}}$$

2. Variable:
$$\frac{\Gamma \vdash x : T}{\exists A, s. \begin{bmatrix} \Gamma \vdash A : s \\ x^A \in \Gamma \\ A \overset{\beta}{\sim} T \end{bmatrix}}$$

3. Product:
$$\frac{\Gamma \vdash \Pi x^A.B : T}{\exists s_1, s_2. \begin{bmatrix} \Gamma \vdash A : s_1 \\ \Gamma, x^A \vdash B : s_2 \\ s_2 \overset{\beta}{\sim} T \end{bmatrix}}$$

4. Abstraction:
$$\frac{\Gamma \vdash \lambda x^A.e : T}{\exists B, s. \begin{bmatrix} \Gamma \vdash \Pi x^A.B : s \\ \Gamma, x^A \vdash e : B \\ \Pi x^A.B \overset{\beta}{\sim} T \end{bmatrix}}$$

5. Application:
$$\frac{\Gamma \vdash f a : T}{\exists A, B. \begin{bmatrix} \Gamma \vdash f : \Pi x^A.B \\ \Gamma \vdash a : A \\ B[x := a] \overset{\beta}{\sim} T \end{bmatrix}}$$

Proof. The premise in all lemmata is the validity of $\Gamma \vdash t : T$ where t is either a sort, a variable, a product, an abstraction or an application. The goal in all lemmata is that some other terms exist and the type T is beta equivalent (or identical) to some other type. All lemmata can be proved by induction on $\Gamma \vdash t : T$.

Note that the betaequivalence $X \overset{\beta}{\sim} T$ in all lemmata is essential. Otherwise the proof for the type equivalence rule cannot be passed.

1. Introduction rules: For all generation lemmata only one introduction rule is syntactically possible. The corresponding introduction rule proves the goal trivially. As an example we prove the generation lemma for products.

Only the introduction rule for products is syntactically possible.

$$\begin{array}{l|l}
\Gamma \vdash A : s_1 & \\
\Gamma, x^A \vdash B : s_2 & \\
\hline
\Gamma \vdash \Pi x^A.B : s_2 & \exists s_a, s_b. \left[\begin{array}{l} \Gamma \vdash A : s_a \\ \Gamma, x^A \vdash B : s_b \\ s_2 \overset{\beta}{\sim} s_b \end{array}\right]
\end{array}$$

The goal in the lower right corner is proved by using $s_a = s_1$ and $s_b = s_2$.

2. Structural rules: For each of the generation lemmata and structural rule the prove is straightforward. Here we prove the generation lemma for products as an example.

   (a) Weaken:

$$\begin{array}{l|l}
\Gamma \vdash \Pi x^A.B : T & \exists s_1, s_2. \left[\begin{array}{l} \Gamma \vdash A : s_1 \\ \Gamma, x^A \vdash B : s_2 \\ T \overset{\beta}{\sim} s_2 \end{array}\right] \\
\Gamma \vdash C : s & \\
z \notin \Gamma & \\
\hline
\Gamma, z^C \vdash \Pi x^A.B : T & \exists s_a, s_b. \left[\begin{array}{l} \Gamma, z^C \vdash A : s_a \\ \Gamma, z^C, x^A \vdash B : s_b \\ T \overset{\beta}{\sim} s_b \end{array}\right]
\end{array}$$

   By the induction hypothesis there are some s_1 and s_2 which satisfy the conditions in the square brackets. In order to prove the goal in the lower right corner and use $s_a = s_1$ and $s_b = s_2$ and have to prove the following three subgoals:

   i. $\Gamma, y^C \vdash A : s_1$: Since Γ, y^C is a valid context with $\Gamma \subseteq \Gamma, y^C$ we can prove the subgoal by the thinning lemma 5.4.

   ii. $\Gamma, z^C, x^A \vdash B : s_2$: The previous subgoal and the variable introduction rule ensures that Γ, y^C, x^A is a valid context with $\Gamma, x^A \subseteq \Gamma, y^C, x^A$. By the thinning lemma 5.4 and the induction hypothesis we prove the current subgoal.

   iii. $T \overset{\beta}{\sim} s_2$: This subgoal is identical to the third proposition in the induction hypothesis.

(b) Type equivalence:

$$
\begin{array}{l|l}
\Gamma \vdash \Pi x^A.B : T & \exists s_1, s_2. \begin{bmatrix} \Gamma \vdash A : s_1 \\ \Gamma, x^A \vdash B : s_2 \\ T \overset{\beta}{\sim} s_b \end{bmatrix} \\
\Gamma \vdash U : s & \\
T \overset{\beta}{\sim} U & \\
\hline
\Gamma \vdash \Pi x^A.B : U & \exists s_a, s_b. \begin{bmatrix} \Gamma \vdash A : s_a \\ \Gamma, x^A \vdash B : s_b \\ U \overset{\beta}{\sim} s_b \end{bmatrix}
\end{array}
$$

By the induction hypothesis there exist some s_1 and s_2 which satisfy the propositions in the square brackets, especially $T \overset{\beta}{\sim} s_2$. To prove the goal in the lower right corner we use $s_a = s_1$ and $s_b = s_2$ and use the fact $T \overset{\beta}{\sim} U$ and the transitivity of beta equivalence.

□

Corollary 5.6. *All types in a valid context are welltyped.*

Proof. According to the start lemma 5.3 $\Gamma \vdash x : A$ is valid for each $x^A \in \Gamma$. By the generation lemma 5.5 for variables there is a sort s with $\Gamma \vdash A : s$. Therefore A is welltyped. □

Corollary 5.7. *The term $\mathcal{U}$ is not welltyped.*

Proof. Let's assume it is welltyped. Then by definition there exists a context Γ and a term T such that $\Gamma \vdash \mathcal{U} : T$ is valid. By the generation lemma 5.5 for sorts we get $\mathcal{U} = \mathcal{P}$ which is not possible. □

Corollary 5.8. *A valid context has no variable with the type $\mathcal{U}$.*

Proof. By 5.6 all types in a context are welltyped and by 5.7 the term $\mathcal{U}$ is not welltyped. Therefore $\mathcal{U}$ cannot be the type of a variable in a valid context. □

Corollary 5.9. *All subterms of a welltyped term are welltyped.*

Proof. We prove this corollary by proving that any direct subterm of a welltyped term is welltyped by induction on the structure of the welltyped term.

For each possible form of a welltyped term the generation lemma 5.5 for this form states the existence of a context and a type of each direct subterm, i.e. the direct subterms are welltyped.

Repeating the argument proves that all indirect subterms of a welltyped subterm are welltyped as well. □

Corollary 5.10. *$\mathcal{U}$ cannot be a subterm of a welltyped term.* This implies that terms like $\lambda x^{\mathcal{U}}.e$, $\lambda x^A.\mathcal{U}$, $\Pi x^{\mathcal{U}}.B$ or $\Pi x^A.\mathcal{U}$ cannot be welltyped, because they have $\mathcal{U}$ as a subterm.

Proof. Assume that $\mathcal{U}$ is a subterm of a welltyped term. Then by the corollary 5.9 $\mathcal{U}$ must be welltyped. This contradicts corollary 5.7. □

5.6 Substitution Lemma

Theorem 5.11. Substitution (cut) theorem.

$$\begin{array}{l}
\Gamma \vdash a : A \\
\Gamma, x^A, \Delta \vdash t : T \\
\hline
\Gamma, \Delta[x := a] \vdash t[x := a] : T[x := a]
\end{array}$$

Proof. We assume $\Gamma \vdash a : A$ and prove this theorem by induction on $\Gamma, x^A, \Delta \vdash t : T$.

In order to express the proof more compactly we introduce the abbreviation $t' := t[x := a]$.

1. Axiom: This case is syntactically impossible, because the context is not empty.
2. Variable: We have to prove the goal

$$\begin{array}{l|l}
\Gamma, x^A, \Delta \vdash B : s & \Gamma, \Delta' \vdash B' : s \\
\hline
\Gamma, x^A, \Delta, y^B \vdash y : B & \Gamma, \Delta', y^{B'} \vdash y : B'
\end{array}$$

The final goal in the lower right corner follows directly from the induction hypothesis and the variable introduction rule.

3. Product:

$$\begin{array}{l|l}
\Gamma, x^A, \Delta \vdash B : s_1 & \Gamma, \Delta' \vdash B' : s_1 \\
\Gamma, x^A, \Delta, y^B \vdash C : s_2 & \Gamma, \Delta', y^{B'} \vdash C' : s_2 \\
s_2 = \mathcal{P} \vee s_1 = s2 & \\
\hline
\Gamma, x^A, \Delta \vdash \Pi y^B.C : s_2 & \Gamma, \Delta' \vdash \Pi y^{B'}.C' : s_2
\end{array}$$

The final goal follows from the induction hypotheses and the product introduction rule.

4. Abstraction:

$$\begin{array}{l|l}
\Gamma, x^A, \Delta \vdash \Pi y^B.C : s & \Gamma, \Delta' \vdash \Pi y^{B'}.C' : s \\
\Gamma, x^A, \Delta, y^B \vdash e : C & \Gamma, \Delta', y^{B'} \vdash e' : C' \\
\hline
\Gamma, x^A, \Delta \vdash \lambda y^B.e : \Pi y^B.C & \Gamma, \Delta' \vdash \lambda y^{B'}.e' : \Pi y^{B'}.C'
\end{array}$$

   Same reasoning as above.

5. Application:

$$\begin{array}{l|l}
\Gamma, x^A, \Delta \vdash f : \Pi y^B.C & \Gamma, \Delta' \vdash f' : \Pi y^{B'}.C' \\
\Gamma, x^A, \Delta \vdash b : B & \Gamma, \Delta' \vdash b' : B' \\
\hline
\Gamma, x^A, \Delta \vdash fb : C[y := b] & \Gamma, \Delta' \vdash f'b' : (C[y := b])'
\end{array}$$

   From the induction hypotheses and the application introduction rule we conclude

$$\Gamma, \Delta' \vdash f'b' : C'[y := b']$$

   and get the final goal by observing

$$C'[y := b'] = (C[y := b])'$$

   by using the double substition lemma 3.7.

6. Structural rules:

   (a) Weakening:

$$\begin{array}{l|l}
\Gamma, x^A, \Delta \vdash t : T & \Gamma, \Delta' \vdash t' : T' \\
\Gamma, x^A, \Delta \vdash B : s & \Gamma, \Delta' \vdash B' : s' \\
\hline
\Gamma, x^A, \Delta, y^B \vdash t : T & \Gamma, \Delta', y^{B'} \vdash t' : T'
\end{array}$$

   The goal in the lower right corner can be proved by the induction hypotheses and the weakening rule.

   (b) Type equivalence:

$$\begin{array}{l|l}
\Gamma, x^A, \Delta \vdash t : T & \Gamma, \Delta' \vdash t' : T' \\
\Gamma, x^A, \Delta \vdash U : s & \Gamma, \Delta' \vdash U' : s' \\
T \overset{\beta}{\sim} U & \\
\hline
\Gamma, x^A, \Delta \vdash t : U & \Gamma, \Delta' \vdash t' : U'
\end{array}$$

   From the theorem 3.16 we conclude that T' and U' are beta equivalent. The goal in the lower right corner is an immediate consequence of the induction hypotheses and the type equivalence rule.

□

5.7 Type of Types

Theorem 5.12. *A term in the type position of a context is either $\mathcal{U}$ or it is a valid type of some sort.*

$$\frac{\Gamma \vdash t : T}{T = \mathcal{U} \lor \exists s.\Gamma \vdash T : s}$$

Proof. By induction on $\Gamma \vdash t : T$:

1. Sort: Trivial
2. Variable: Trivial by the premise of the variable introduction rule.
3. Product: Easy, because there are only two sorts. If the sort is $\mathcal{P}$ then by the start lemma 5.3 $\Gamma \vdash \mathcal{P} : \mathcal{U}$ is valid in any valid context. If the sort is $\mathcal{U}$ then the goal is trivial.
4. Abstraction:

   $$\begin{array}{l|l} \Gamma \vdash \Pi x^A.B : s_0 & \\ \Gamma, x^A \vdash e : B & \\ \hline \Gamma \vdash \lambda x^A.e : \Pi x^A.B & \exists s.\Gamma \vdash \Pi x^A.B : s \end{array}$$

   Take $s = s_0$.
5. Application:

   $$\begin{array}{l|l} \Gamma \vdash f : \Pi x^A.B & \exists s_0.\Gamma \vdash \Pi x^A.B : s_0 \\ \Gamma \vdash a : A & \\ \hline \Gamma \vdash f a : B[x := a] & \exists s.\Gamma \vdash B[x := a] : s \end{array}$$

   Remark: Since $\mathcal{U} \neq \Pi x^A.B$ only the second alternative is interesting.

   From the induction hypothesis and the generation lemma 5.5 for products we conclude the existence of a sort s_2 such that $\Gamma, x^A \vdash B : s_2$ is valid.

   By applying the substitution lemma 5.11 we get $\Gamma \vdash B[x := a] : s_2$ which proves the goal.
6. Weaken: The goal is easy to prove by the induction hypothesis and the weakening rule.
7. Type Equivalence:

   The goal is a immediate consequence of one of the premises of the type equivalence rule.

□

5.8 Subject Reduction

Theorem 5.13. *Subject reduction lemma* Reduction of a term does not change its type.

$$\frac{\begin{array}{c}\Gamma \vdash t : T\\ t \overset{\beta}{\to} u\end{array}}{\Gamma \vdash u : T}$$

Proof. In order to prove the subject reduction lemma we prove the more general lemma

$$\frac{\Gamma \vdash t : T}{\left(\forall u. \frac{t \overset{\beta}{\to} u}{\Gamma \vdash u : T}\right) \wedge \left(\forall \Delta. \frac{\Gamma \overset{\beta}{\to} \Delta}{\Delta \vdash t : T}\right)}$$

where $\Gamma \overset{\beta}{\to} \Delta$ means that Δ is Γ with one of the variable types replaced by a reduced type.

We prove the more general lemma by induction on $\Gamma \vdash t : T$.

1. Introduction rules:
   (a) Sort: If Γ is empty and t is a sort, then the goal is vacuously true because neither the empty context nor a sort can reduce to anything (they are in normal form).
   (b) Variable:

$$\begin{array}{c|c}
\Gamma \vdash A : s & \left(\forall B. \frac{A \overset{\beta}{\to} B}{\Gamma \vdash B : s}\right) \wedge \left(\forall \Delta_0. \frac{\Gamma \overset{\beta}{\to} \Delta_0}{\Delta_0 \vdash A : s}\right) \\
\hline
\Gamma, x^A \vdash x : A & \left(\forall u. \frac{x \overset{\beta}{\to} u}{\Gamma, x^A \vdash u : A}\right) \wedge \left(\forall \Delta. \frac{\Gamma, x^A \overset{\beta}{\to} \Delta}{\Delta \vdash x : A}\right)
\end{array}$$

   The left part of the goal in the lower right corner is vacously true because a variable is in normal form and there is no term to which it reduces.
   For the right part we assume $\Gamma, x^A \overset{\beta}{\to} \Delta$. There are two possibilities:
   i. $\Delta = \Delta_0, x^A$ where $\Gamma \overset{\beta}{\to} \Delta_0$ for some Δ_0:
   In that case we get $\Delta_0 \vdash A : s$ from the induction hypothesis which implies the goal $\Delta_0, x^A \vdash x : A$.
   ii. $\Delta = \Gamma, x^B$ where $A \overset{\beta}{\to} B$:
   In that case we get $\Gamma \vdash B : s$ from the induction hypothesis which implies the goal $\Gamma, x^B : x \vdash B$.

(c) Product:

$$\begin{array}{l|l}
\Gamma \vdash A : s_1 & \left(\forall C.\frac{A \overset{\beta}{\to} C}{\Gamma \vdash C : s_1}\right) \vee \left(\forall \Delta.\frac{\Gamma \overset{\beta}{\to} \Delta}{\Delta \vdash A : s_1}\right) \\
\Gamma, x^A \vdash B : s_2 & \left(\forall D.\frac{B \overset{\beta}{\to} D}{\Gamma, x^A \vdash D : s_2}\right) \vee \left(\forall \Delta'.\frac{\Gamma, x^A \overset{\beta}{\to} \Delta'}{\Delta' \vdash B : s_2}\right) \\
\hline
\Gamma \vdash \Pi x^A.B : s_2 & \left(\forall t.\frac{\Pi x^A.B \overset{\beta}{\to} t}{\Gamma \vdash t : s_2}\right) \wedge \left(\forall \Delta.\frac{\Gamma \overset{\beta}{\to} \Delta}{\Delta \vdash \Pi x^A.B : s_2}\right)
\end{array}$$

i. Left part: We assume $\Pi x^A.B \overset{\beta}{\to} t$.
Since products are preserved under reduction (lemma 3.12) we have either $t = \Pi x^C.B$ where $A \overset{\beta}{\to} C$ for some C or $t = \Pi x^A.D$ where $B \overset{\beta}{\to} D$ for some D.
In both cases we can derive from the induction hypotheses either $\Gamma \vdash C : s_1$ or $\Gamma, x^A \vdash D : s_2$. Therefore $\Gamma \vdash \Pi x^C.B : s_2$ or $\Gamma \vdash \Pi x^A.D : s_2$ is valid trivially.

ii. Right part: Assume $\Gamma \overset{\beta}{\to} \Delta$. From the first induction hypothesis we get $\Delta \vdash A : s_1$. From the second induction hypothesis we get $\Delta, x^A \vdash B : s_2$ where we use $\Delta' = \Delta, x^A$. These facts imply $\Delta \vdash \Pi x^A.B : s_2$.

(d) Abstraction: Same reasoning as with product.

(e) Application:

$$\begin{array}{l|l}
\Gamma \vdash f : \Pi x^A.B & \left(\forall g.\frac{f \overset{\beta}{\to} g}{\Gamma \vdash g : \Pi x^A.B}\right) \wedge \left(\forall \Delta.\frac{\Gamma \overset{\beta}{\to} \Delta}{\Delta \vdash f : \Pi x^A.B}\right) \\
\Gamma \vdash a : A & \left(\forall b.\frac{a \overset{\beta}{\to} b}{\Gamma \vdash b : A}\right) \wedge \left(\forall \Delta.\frac{\Gamma \overset{\beta}{\to} \Delta}{\Delta \vdash a : A}\right) \\
\hline
\Gamma \vdash fa : B[x := a] & \left(\forall t.\frac{fa \overset{\beta}{\to} t}{\Gamma \vdash t : B[x := a]}\right) \wedge \left(\forall \Delta.\frac{\Gamma \overset{\beta}{\to} \Delta}{\Delta \vdash fa : B[x := a]}\right)
\end{array}$$

i. Left part: Assume $fa \overset{\beta}{\to} t$. We have three cases to consider.

   A. $fa \overset{\beta}{\to} ga$ where $f \overset{\beta}{\to} g$:
   From the first induction hypothesis we get that g has the same type as f and therefore $ga : B[x := a]$ is valid.

B. $fa \overset{\beta}{\to} fb$ where $a \overset{\beta}{\to} b$:
From the second induction hypothesis we get $\Gamma \vdash b : A$ and therefore $\Gamma \vdash fa : B[x := b]$.
Since $B[x := a]$ is a valid type and because of lemma 3.10 we have $B[x := a] \overset{\beta}{\to} B[x := b]$ and therefore $B[x := b] \leq B[x := a]$. The subtype rule let us derive $\Gamma \vdash fb : B[x := a]$ which is identical to the final goal.

C. $(\lambda x^A.e)a \overset{\beta}{\to} e[x := a]$:
In that case we have to prove

$$\begin{array}{l} \Gamma \vdash \lambda x^A.e : \Pi x^A.B \\ \Gamma \vdash a : A \\ \hline \Gamma \vdash e[x := a] : B[x := a] \end{array}$$

We assume $\Gamma \vdash \lambda x^A.e : \Pi x^A.B$ and $\Gamma \vdash a : A$ and prove the final goal.
Since $\Pi x^A.B$ is not $\mathcal{U}$ the type of types lemma 5.12 states the existence of a sort s such that $\Gamma \vdash \Pi x^A.B : s$ is valid and then by the generation lemma 5.5 for products we get the existence of some s_B such that $\Gamma, x^A \vdash B : s_B$ is valid which implies by the substitution lemma 5.11 $\Gamma \vdash B[x := a] : s_B$.
According to the generation lemma 5.5 for abstractions there are some B_0 and s_0 with

$$\begin{array}{l} \Gamma \vdash \Pi x^A.B_0 : s_0 \\ \Gamma, x^A \vdash e : B_0 \\ \Pi x^A.B_0 \overset{\beta}{\sim} \Pi x^A.B \end{array}$$

The second one together with $\Gamma \vdash a : A$ and the substitution lemma 5.11 gives us $\Gamma \vdash e[x := a] : B_0[x := a]$.
The third one with the Church Rosser theorem 4.16 give $B \overset{\beta}{\sim} B_0$ which by the theorem 3.16 results in $B[x := a] \overset{\beta}{\sim} B_0[x := a]$.
Finally we can convert $\Gamma \vdash e[x := a] : B_0[x := a]$, $\Gamma \vdash B[x := a] : s_B$ and $B[x := a] \overset{\beta}{\sim} B_0[x := a]$ via the type equivalence rule into the final goal.

ii. Right part: Immediate consequence of the induction hypotheses.

2. Structural rules:

   (a) Weaken:

$$\begin{array}{l|l}
\Gamma \vdash t : T & \left(\forall u. \dfrac{t \overset{\beta}{\to} u}{\Gamma \vdash u : T}\right) \wedge \left(\forall \Delta_0. \dfrac{\Gamma \overset{\beta}{\to} \Delta_0}{\Delta_0 \vdash t : T}\right) \\
\Gamma \vdash A : s & \left(\forall B. \dfrac{A \overset{\beta}{\to} B}{\Gamma \vdash B : s}\right) \wedge \left(\forall \Delta_0. \dfrac{\Gamma \overset{\beta}{\to} \Delta_0}{\Delta_0 \vdash A : s}\right) \\
x \notin \Gamma & \\
\hline
\Gamma, x^A \vdash t : T & \left(\forall u. \dfrac{t \overset{\beta}{\to} u}{\Gamma, x^A \vdash u : T}\right) \wedge \left(\forall \Delta. \dfrac{\Gamma, x^A \overset{\beta}{\to} \Delta}{\Delta \vdash t : T}\right)
\end{array}$$

   i. Left part: Assume $t \overset{\beta}{\to} u$. From the first induction hypothesis we get $\Gamma \vdash u : T$ which derives the final goal by applying the variable introduction rule.

   ii. Right part: We assume $\Gamma, x^A \overset{\beta}{\to} \Delta$ and have to distinguish two cases.

   A. $\Delta = (\Delta_0, x^A) \wedge \Gamma \overset{\beta}{\to} \Delta_0$:
   In that case we get $\Delta_0 \vdash t : T$ and $\Delta_0 \vdash A : s$ from the induction hypotheses which imply the final goal $\Delta_0, x^A \vdash t : T$ by application of the weakening rule.

   B. $\Delta = (\Gamma, x^B) \wedge A \overset{\beta}{\to} B$:
   In that case we get $\Gamma \vdash B : s$ from the second induction hypothesis which implies the final goal $\Gamma, x^B \vdash t : T$ by application of the weakening rule.

   (b) Type equivalence:

$$\begin{array}{l|l}
\Gamma \vdash t : T & \left(\forall u. \dfrac{t \overset{\beta}{\to} u}{\Gamma \vdash u : T}\right) \wedge \left(\forall \Delta. \dfrac{\Gamma \overset{\beta}{\to} \Delta}{\Delta \vdash t : T}\right) \\
\Gamma \vdash U : s & \ldots \wedge \left(\forall \Delta. \dfrac{\Gamma \overset{\beta}{\to} \Delta}{\Delta \vdash U : s}\right) \\
T \overset{\beta}{\sim} U & \\
\hline
\Gamma \vdash t : U & \left(\forall u. \dfrac{t \overset{\beta}{\to} u}{\Gamma \vdash u : U}\right) \wedge \left(\forall \Delta. \dfrac{\Gamma \overset{\beta}{\to} \Delta}{\Delta \vdash t : U}\right)
\end{array}$$

   i. Left part: Assume $t \overset{\beta}{\to} u$. We get $\Gamma \vdash u : T$ by the first induction hypothesis. The final goal $\Gamma \vdash u : U$ is obtained by applying the type equivalence rule.

ii. Right part: Assume $\Gamma \overset{\beta}{\to} \Delta$. From the first induction hypothesis we derive $\Delta \vdash t : T$ and from the second induction hypothesis we derive $\Delta \vdash U : s$. The final goal $\Delta \vdash t : U$ is obtained by applying the type equivalence rule.

□

Corollary 5.14. *No welltyped term can reduce to $\mathcal{U}$.*

$$\frac{\begin{array}{c} A \overset{\beta^*}{\to} \mathcal{U} \\ \Gamma \vdash A : T \end{array}}{\bot}$$

Proof. Assume $\Gamma \vdash A : T$. By the subject reduction theorem 5.13 (applied zero or more times) we get $\Gamma \vdash \mathcal{U} : T$ which contradicts the fact that $\mathcal{U}$ is not welltyped 5.7. □

Corollary 5.15. *No welltyped term can be beta equivalent to $\mathcal{U}$.*

$$\frac{\begin{array}{c} \Gamma \vdash A : T \\ A \overset{\beta}{\sim} \mathcal{U} \end{array}}{\bot}$$

Proof. Assume $\Gamma \vdash A : T$. Since A and $\mathcal{U}$ are beta equivalent, by 4.16 they must have a common reduct and since $\mathcal{U}$ cannot reduce to anything (it is in normal form) the common reduct must be $\mathcal{U}$.

Because of the previous theorem 5.14 A cannot reduce to $\mathcal{U}$ and we get the desired contradiction. □

5.9 Type Uniqueness

Theorem 5.16. *All types of a term are beta equivalent*

$$\frac{\begin{array}{c} \Gamma \vdash t : T \\ \Gamma \vdash t : U \end{array}}{T \overset{\beta}{\sim} U}$$

Proof. By induction on the structure of t using the generation lemmata 5.5. The generation lemma for each form of t guarantees the existence of a term V for which $V \overset{\beta}{\sim} T$ and $V \overset{\beta}{\sim} U$ is valid. By transitivity of beta equivalence $T \overset{\beta}{\sim} U$ is implied. □

Theorem 5.17. *All types of beta equivalent terms are beta equivalent*

$$\frac{\begin{array}{c} t \overset{\beta}{\sim} u \\ \Gamma \vdash t : T \\ \Gamma \vdash u : U \end{array}}{T \overset{\beta}{\sim} U}$$

Proof. $t \overset{\beta}{\sim} u$ implies by the Church Rosser Theorem 4.16 the existence of a term v such that $t \overset{\beta^*}{\to} v$ and $u \overset{\beta^*}{\to} v$ are valid.

By repeated application of the subject reduction theorem we get $\Gamma \vdash v : T$ and $\Gamma \vdash v : U$.

This implies by the previous theorem 5.16 the validity of $T \overset{\beta}{\sim} U$. □

Corollary 5.18. *Two types T and U of beta equivalent terms t and u are either both $\mathcal{U}$ or they are types of the same sort.*

$$\frac{\begin{array}{c} t \overset{\beta}{\sim} u \\ \Gamma \vdash t : T \\ \Gamma \vdash u : U \end{array}}{T = U = \mathcal{U} \lor \exists s. \left[\begin{array}{c} \Gamma \vdash T : s \\ \Gamma \vdash U : s \end{array} \right]}$$

Proof. By 5.17 we get $T \overset{\beta}{\sim} U$ and by the type of types theorem 5.12 both are either $\mathcal{U}$ or types of some sort.

The mixed cases are not possible since a welltyped term cannot be beta equivalent to $\mathcal{U}$ by 5.15.

It remains to prove that both being welltyped implies that they are types of the same sort.

Since both T and U are beta equivalent, their sorts have to be beta equivalent as well by 5.17. The mixed case that one sort is $\mathcal{P}$ and the other is $\mathcal{U}$ is not possible, because $\mathcal{P}$ is welltyped (it has type $\mathcal{U}$ in any valid context) and a welltyped term cannot be beta equivalent to $\mathcal{U}$ by 5.15. Therefore both sorts are either $\mathcal{P}$ or $\mathcal{U}$. □

5.10 Kinds

In the section 5.7 we saw that any term T in the type position of a typing judgement $\Gamma \vdash t : T$ is either $\mathcal{U}$ or is a welltyped term whose type is a sort. This justifies the statement that sorts are the types of types.

However the calculus of constructions has dependent types i.e. there are type valued functions F which when applied to arguments return a type i.e. FTa might be a type where T represents a type and a represents some other welltyped term e.g. a natural number. In that case F has a type of the form $\Pi X^{\mathcal{P}} y^{A}.\mathcal{P}$. We call such terms F *type functions* and their types *kinds*. A *kind* is a product where the final type is a sort. We include in our definition of *kinds* products with arity zero i.e. sorts. Type functions with arity zero are types.

Definition 5.19. The set of *syntactical kinds* $\mathbb{K}$ is defined as the set of terms generated by the grammar

$$K ::= s \mid \Pi x^{A}.K \mid \Pi x^{K}.K$$

where s ranges over sorts, K ranges over kinds and A ranges over terms which are not kinds.

It is also possible to use the equivalent definition: The set $\mathbb{K}$ is an inductive set defined by the rules

1.
$$s \in \mathbb{K}$$

2.
$$\frac{\begin{array}{l} A \notin \mathbb{K} \\ K \in \mathbb{K} \end{array}}{\Pi x^{A}.K \in \mathbb{K}}$$

3.
$$\frac{\begin{array}{l} A \in \mathbb{K} \\ K \in \mathbb{K} \end{array}}{\Pi x^{A}.K \in \mathbb{K}}$$

Theorem 5.20. *Every term K which has type $\mathcal{U}$ is a syntactical kind*

$$\frac{\Gamma \vdash K : \mathcal{U}}{K \in \mathbb{K}}$$

Proof. By induction on $\Gamma \vdash K : \mathcal{U}$.

1. Sort:
$$[] \vdash \mathcal{P} : \mathcal{U}$$
Trival, because $\mathcal{P} \in \mathbb{K}$.

2. Variable: The variable rule deriving $\Gamma, x^{\mathcal{U}} \vdash x : \mathcal{U}$ is not applicable, because it would require $\Gamma \vdash \mathcal{U} : s$ which contradicts the first generation lemma 5.5.

3. Product:

$$\begin{array}{l|l} \Gamma \vdash A : s & \\ \Gamma, x^A \vdash B : \mathcal{U} & B \in \mathbb{K} \\ \hline \Gamma \vdash \Pi x^A.B : \mathcal{U} & \Pi x^A.B \in \mathbb{K} \end{array}$$

   The goal in the lower right corner is a consequence of the induction hypothesis.

4. Abstraction:

   The abstraction rule deriving $\Gamma \vdash \lambda x^A.e : \Pi x^A.B$ is syntactically not possible, because $\Pi x^A.B \neq \mathcal{U}$.

5. Application: In order to apply the rule

$$\begin{array}{l} \Gamma \vdash f : \Pi x^A.B \\ \Gamma \vdash a : A \\ \hline \Gamma \vdash fa : B[x := a] \end{array}$$

   the validity of $B[x := a] = \mathcal{U}$ would be necessary. This would require either $B = \mathcal{U}$ or $B = x \wedge a = \mathcal{U}$ by lemma 3.6 and both cases lead to a contradiction.

   - $B = \mathcal{U}$: The generation lemma 5.5 for a product requires the existence of a sort s with $\Gamma, x^A \vdash \mathcal{U} : s$ which requires by the generation lemma for sorts that $\mathcal{U} = \mathcal{P}$ which is not possible.
   - $B = x \wedge a = \mathcal{U}$: The generation lemma for sorts would require in that case $\mathcal{U} = \mathcal{P}$ which is not possible.

6. Weaken: Trivial, because the goal is immediately implied by the induction hypothesis.

7. Type equivalence: $\Gamma \vdash K : \mathcal{U}$ cannot be derived by the type equivalence rule because the premise $\Gamma \vdash \mathcal{U} : s$ is not satisfiable ($\mathcal{U}$ is not welltyped by 5.7).

□

Corollary 5.21. *Every type T of sort $\mathcal{P}$ is not a kind.*

$$\begin{array}{l} \Gamma \vdash T : \mathcal{P} \\ \hline T \notin \mathbb{K} \end{array}$$

Proof. Assume $\Gamma \vdash T : \mathcal{P}$ and by way of contradiction $T \in \mathbb{K}$.

This implies by the previous theorem 5.23 $\Gamma \vdash T : \mathcal{U}$ and by theorem 5.16 $\mathcal{P} \overset{\beta}{\sim} \mathcal{U}$ which is contradictory. □

Corollary 5.22. *Every welltyped type which is not a kind has type $\mathcal{P}$.*

$$\frac{\begin{array}{l} T \notin \mathbb{K} \\ \Gamma \vdash T : s \end{array}}{\Gamma \vdash T : \mathcal{P}}$$

Proof. Assume $T \notin \mathbb{K}$ and $\Gamma \vdash T : s$.

There are only two sorts, therefore either $s = \mathcal{P}$ or $s = \mathcal{U}$ has to be valid. The first case proves the goal. The second case implies by the theorem 5.20 $T \in \mathbb{K}$ which contradicts the assumption $T \notin \mathbb{K}$. □

Theorem 5.23. *Every welltyped syntactical kind K has type $\mathcal{U}$.*

$$\frac{\begin{array}{l} K \in \mathbb{K} \\ \Gamma \vdash K : T \end{array}}{\Gamma \vdash K : \mathcal{U}}$$

Proof. By induction on the structure of K

1. Sort s: The validity of $\Gamma \vdash s : T$ implies by the generation lemma 5.5 for sorts $s = \mathcal{P}$ and $T \overset{\beta}{\sim} \mathcal{U}$. By the type of types lemma 5.12 we know that T is either $\mathcal{U}$ or it is welltyped and by the lemma 5.15 we know that no welltyped term can be beta equivalent to $\mathcal{U}$. This implies $T = \mathcal{U}$ and therefore the goal.
2. $\Pi x^A.K$: We assume $\Gamma \vdash \Pi x^A.K : T$ and have to prove $\Gamma \vdash \Pi x^A : K : \mathcal{U}$. The generation lemma 5.5 for products guarantees the existence of a sort s with $\Gamma, x^A \vdash K : s$. This together with the induction hypotheses implies $\Gamma, x^A \vdash K : \mathcal{U}$. By the introduction rule for products we get $\Gamma \vdash \Pi x^A.K : \mathcal{U}$.

□

Theorem 5.24. *Every syntactical kind in the type position of a typing judgement either is $\mathcal{U}$ or has type $\mathcal{U}$.*

$$\frac{\begin{array}{l} K \in \mathbb{K} \\ \Gamma \vdash A : K \end{array}}{K = \mathcal{U} \vee \Gamma \vdash K : \mathcal{U}}$$

Proof. By 5.12 either $K = \mathcal{U}$ or $\Gamma \vdash K : s$ must be valid for some sort s. In the first case the goal is trivial. In the second case the goal can be inferred from theorem 5.23. □

6 Proof of Strong Normalization

In this section we prove the strong normalization of the calculus of constructions i.e. we prove that all welltyped terms have no infinite reduction sequence which implies that all welltyped terms can be reduced to a normal form. The presented proof is based on the proof of Herman Geuvers in his paper *A short and flexible proof of strong normalization for the calculus of constructions* [2].

The article of Herman Geuvers is a scientific paper which assume some good knowledge of type theory. Here we present the proof in a text book style which makes the proof more accessible to readers who are no experts in type theory.

In the last subsection of this chapter we use the proof of strong normalization to prove that the calculus of constructions is consistent as a logic i.e. that it is impossible to derive a contradiction in the calculus of constructions.

The proof of strong normalization is not trivial and spans nearly the whole part of this chapter. Why is the proof nontrivial?

A naive attempt to prove the strong normalization would be to prove

$$\frac{\Gamma \vdash t : T}{t \in \mathrm{SN}}$$

by induction on $\Gamma \vdash t : T$ (where SN is the set of all strongly normalizing terms).

This attempt passes a lot of rules of the typing relation. But it fails at the rule for application. In the rule for application we have to prove the strong normalization of fa by using the induction hypotheses that the function term f and the argument term a are strongly normalizing. The impossibility of this proof step can be seen by looking at a counterexample.

The term $\lambda x^A.xx$ is strongly normalizing because it is in normal form. If we apply the term to any argument we get the reduction

$$(\lambda x^A.xx)a \overset{\beta}{\to} aa$$

If we apply the term to itself we get

$$(\lambda x^A.xx)(\lambda x^A.xx) \overset{\beta}{\to} (\lambda x^A.xx)(\lambda x^A.xx)$$

It is obvious that no normal form can be reached because the reduction steps can be carried out forever.

The problem with this naive approach is that the goal $t \in \mathrm{SN}$ gives induction hypotheses which are too weak. The set of strongly normalizing terms SN is not connected to the type T of the term t.

- *Type Interpretation*

  It is better to find a subset of the strongly normalizing terms $[\![T]\!] \subseteq \mathrm{SN}$ which reflects a set of strongly normalizing terms which can represent terms of type T. We call $[\![T]\!]$ an *interpretation* of the type T. The interpretation of a type shall be chosen in a way such that

  $$\frac{\begin{array}{l} f \in [\![\Pi x^A.B]\!] \\ a \in [\![A]\!] \end{array}}{fa \in [\![B[x := a]]\!]}$$

  is valid where $\Pi x^A.B$ is the type of f and A is the type of a.

- *Saturated Sets*

  In the proof on strong normalization we use *saturated sets* as interpretations of types. Saturated sets are sets of strongly normalizing terms which are closed in the sense that they contain all base terms (strongly normalizing terms of the form $xa_1a_2 \ldots a_n$) and that they contain all strongly normalizing terms which reduce by beta reduction of a redex in the leftmost position of the term to a term in the set (key reductions).

  These closure conditions are important in the proof. The set of saturated set will be denoted by SAT. Note that SAT is a set of sets of strongly normalizing terms.

- *Lambda Function Space*

  Having a type interpretation $[\![A]\!]$ for the type A and a type interpretation $[\![B]\!]$ it is possible to form a saturated set

  $$[\![A]\!] \overset{\lambda}{\to} [\![B]\!]$$

  which is the set of lambda terms f which whenever applied to a term $a \in [\![A]\!]$ guarantees that $fa \in [\![A]\!] \overset{\lambda}{\to} [\![B]\!]$ is valid. Note that $[\![A]\!] \overset{\lambda}{\to} [\![B]\!]$ is a set of strongly normalizing lambda terms and not a relation. We call such a set of stronly normalizing terms a *lambda function space*. We use sets in the form of a lambda function space as type interpretations of types of the form $\Pi x^A.B$.

This lambda function space solves the problem of proving $fa \in [\![B]\!]$ from the induction hypotheses $f \in [\![A]\!] \overset{\lambda}{\to} [\![B]\!]$ and $a \in [\![A]\!]$. In the previous paragraph we have ignored the detail that the variable x might be contained in the type B within the product $\Pi x^A.B$ and that each different value of the variable x might generate a different interpretation $[\![B]\!]$ of the type B. In the detailed proof we will see that this fact is important when the type A of the variable x is a kind. In that case we have to form the lambda function space

$$[\![A]\!] \overset{\lambda}{\to} \bigcap_x [\![B]\!]$$

where for the purposes of this overview we understand that $\bigcap_x [\![B]\!]$ is the intersection of all possible interpretations $[\![B]\!]$ for all possible values of x. In the detailed proof we give a precise definition of this lambda function space. Here we get a first hint why the closure conditions in the definition of saturated sets are important. They guarantee that any intersection of saturated sets is a saturated set.

- *Type Functions*

  It is not sufficient to have interpretations $[\![A]\!]$ of types A. E.g. a term which represents list L in the calculus of constructions is not a type. It is a type function of type $\Pi X^{\mathcal{P}}.\mathcal{P}$. We have to find an interpretation for L as well. But its interpretation cannot be a saturated set. Its interpretation $[\![L]\!]$ has to be a mathematical function which maps saturated sets to saturated sets.

  In that case the term LT which represents the type of a list of elements of type T where $[\![T]\!] \in \text{SAT}$ and $[\![LT]\!] \in \text{SAT}$ are valid. Therefore the interpretation of L must be a mathematical function which maps saturated sets to saturated sets i.e. $[\![L]\!] \in \text{SAT} \to \text{SAT}$. Note that $\text{SAT} \to \text{SAT}$ is a set of mathematical functions.

  Since type functions can be arbitrarily nested, interpretations for type functions can be drawn e.g from SAT for types i.e. 0-ary type functions, $\text{SAT} \to \text{SAT}$ for unary type functions, $(\text{SAT} \to \text{SAT}) \to \text{SAT}$ for unary type functions having a unary type function as argument, $\text{SAT} \to \text{SAT} \to \text{SAT}$ for binary type functions (e.g. pairs), ... to arbitrary depth.

- *Models and Context Interpretations*

  In the chapter *Typing* 5 we have shown that *kinds* are the types of n-ary type functions where the corner case $n = 0$ is allowed. Based on this it is possible to define a function ν which maps any kind K to its appropriate set of models $\nu(K)$ which is the set of possible interpretations of terms of type K.

  Kinds have the advantage that they can be recognized by pure syntactical analysis. This makes the definition of the function ν easy.

  By looking at the types in a context Γ it is decidable whether the type A of a variable x is a kind or not. A *context interpretation* $\xi = [x_1^{M_1}, \ldots, x_n^{M_n}]$ is a sequence of models for all variables in the context whose type is a kind. We say that ξ is a valid interpretation of a valid context Γ which we denote by $\xi \models \Gamma$ if ξ associates to each variable in the context which is a type function a model from the corresponding set $\nu(K)$ where K is the type of the variable (which has to be a kind).

- *Type Interpretation Function*

  Based on a valid context interpretation $\xi \models \Gamma$ it is possible to define a function $[\![F]\!]_{\xi\Gamma}$ which maps each welltyped type function F (i.e. not only the variables which are type functions) into a type interpretation which is in the set $\nu(K)$ where K is the type of F.

  In the definition of the type interpretation function it is crucial to map all types to saturated sets i.e. that all types are interpreted by staturated sets i.e. sets of strongly normalizing terms.

  The definition of the type interpretation function is nontrivial because it has to be shown in each case that the return value is in the correct set.

  Furthermore it has to be proved that the type interpretation function returns for equivalent types the same interpretation. This is important because beta equivalent types are practically the same types. Therefore it is not allowed that they have different interpretations. In order to prove this fact we have to show that substitutions which are the basis of beta reduction and beta equality are treated in a consistent manner. This prove is nontrivial either.

- *Term Interpretations and Context Models*

In addition to type interpretations ξ we add term interpretations ρ. Term interpretations or better variable interpretations are a mapping from variables to terms which are an element of the type interpretation of the corresponding type i.e. strongly normalizing terms.

The term interpretation $(\!|t|\!)_\rho$ replaces in the term t all free occurrences of a variable by its corresponding variable interpretation.

A context model $\rho\xi \vDash \Gamma$ is a variable interpretation ρ and a type variable interpretation ξ which are consistent.

The additional complexity of term interpretations $(\!|t|\!)_\rho$ is necessary to be able to enter binders like $\lambda x^A.e$ and $\Pi x^A B$ and get sufficiently strong induction hypotheses.

- *Soundness Theorem*

  The whole machinery culminates in the proof of the soundness theorem which states that for all context models with $\rho\xi \vDash \Gamma$ and all welltyped term t with $\Gamma \vdash t : T$ we can prove $(\!|t|\!)_\rho \in [\![T]\!]_{\xi\Gamma}$.

  Since all type interpretations of types are saturated sets i.e. sets of strongly normalizing terms and there is a canonical interpretation of type variables and the term interpretation which is the identity function is possible (base terms include variables and are in all saturated sets) we easily conclude that all welltyped terms are strongly normalizing.

6.1 Strong Normalization

We start the proof of strong normalization with a definition of the set of *normal forms* NF and the set of *strongly normalizing terms* SN and prove some properties based on these definitions.

Definition 6.1. The set NF of terms in *normal form* is the set of all terms which do not reduce (i.e. which do not have any redex).

$$\frac{\forall b.\left[\dfrac{a \overset{\beta}{\to} b}{\bot}\right]}{a \in \mathrm{NF}}$$

Definition 6.2. The set SN of *strongly normalizing* terms is defined inductively by the rule

$$\frac{\forall b.\left[\dfrac{a \overset{\beta}{\to} b}{b \in \mathrm{SN}}\right]}{a \in \mathrm{SN}}$$

In words: A term a is strongly normalizing if all its reducts b are strongly normalizing.

We can view the strongly normalizing terms more intuitively in the following manner.

- A term which is already in normalform i.e. it has no reducts is strongly normalizing. In hat case all its reducts (there are none) are vacuously strongly normalizing. I.e. terms in normal form form the 0-th generation of strongly normalizing terms.
- The 1st generation of strongly normalizing terms are terms which reduce only to terms in normal form.
- The $(n+1)$th generation of strongly normalizing terms are terms which reduce only to terms of the nth generation of strongly normalizing terms.
- ...

This intuitive definition defines the terms which are guaranteed to reduce in at most n steps to terms in normal form. The strongly normalizing terms are terms which are guaranteed to reduce in n steps to terms in normal form for some natural number n.

With this intuitive definition we could prove properties of strongly normalizing terms by doing induction on the maximal number of steps n needed to reduce to normal form.

However the formal definition is better suited for doing induction proofs. In order to prove that some term a which is strongly normalizing has some property $p(a)$ one can assume that all its reducts have this property. I.e. we can use an induction scheme similar to rule based induction.

$$\begin{array}{l|l}
\forall b.\left[\dfrac{a \overset{\beta}{\to} b}{b \in \mathrm{SN}}\right] & \forall b.\left[\dfrac{a \overset{\beta}{\to} b}{p(b)}\right] \\
\hline
a \in \mathrm{SN} & p(a)
\end{array}$$

In order to prove the goal $p(a)$ in the lower right corner we can assume all the other statements especially the induction hypothesis in the upper right corner.

Theorem 6.3. *All subterms of a strongly normalizing term are strongly normalizing.*

Proof. Sorts and variables don't have subterms. A product $\Pi x^A.B$ has the subterms A and B, an abstraction $\lambda x^A.e$ has the subterms A and e and an application ab has the subterms a and b. All proofs that subterms of strongly normalizing terms are strongly normalizing follow the same pattern. Here we prove $\dfrac{ab \in SN}{a \in \mathrm{SN}}$ which we transform into the equivalent statement

$$\frac{t \in \mathrm{SN}}{\forall ab. \left[\dfrac{t = ab}{a \in \mathrm{SN}} \right]}$$

and prove by induction on $t \in \mathrm{SN}$.

$$\begin{array}{l|l} \forall u. \left[\dfrac{t \overset{\beta}{\to} u}{u \in \mathrm{SN}} \right] & \forall u a_1 b. \left[\dfrac{\begin{array}{c} t \overset{\beta}{\to} u \\ u = a_1 b \end{array}}{a_1 \in \mathrm{SN}} \right] \\ \hline t \in \mathrm{SN} & \forall ab. \left[\dfrac{t = ab}{a \in \mathrm{SN}} \right] \end{array}$$

In order to prove the goal in the lower right corner we assume $t = ab$ and $a \overset{\beta}{\to} a_1$ and prove $a_1 \in \mathrm{SN}$.

From $a \overset{\beta}{\to} a_1$ we infer $ab \overset{\beta}{\to} a_1 b$ and get the final goal from the induction hypothesis by using $u = a_1 b$. □

Theorem 6.4. *A redex is strongly normalizing if its reduct and all its subexpressions are strongly normalizing.*

$$\frac{\begin{array}{l} e \in \mathrm{SN} \\ e[x := a] \in \mathrm{SN} \\ a \in \mathrm{SN} \\ A \in \mathrm{SN} \end{array}}{(\lambda x^A.e)a \in SN}$$

Proof. We do induction on $e \in \mathrm{SN}$, $a \in \mathrm{SN}$ and $A \in \mathrm{SN}$ to get good induction hypotheses and then prove the final goal.

- Induction on $e \in \mathrm{SN}$:

$$\begin{array}{l|l}
\forall f. \left[\dfrac{e \overset{\beta}{\to} f}{f \in \mathrm{SN}} \right] & \forall f a A. \left[\dfrac{\begin{array}{l} e \overset{\beta}{\to} f \\ f[x := a] \in \mathrm{SN} \\ a \in \mathrm{SN} \\ A \in \mathrm{SN} \end{array}}{(\lambda x^A . f) a \in SN} \right] \\
\hline
e \in \mathrm{SN} & \forall a A. \left[\dfrac{\begin{array}{l} e[x := a] \in \mathrm{SN} \\ a \in \mathrm{SN} \\ A \in \mathrm{SN} \end{array}}{(\lambda x^A . e) a \in \mathrm{SN}} \right]
\end{array}$$

- Assume $e[x := a] \in \mathrm{SN}$ and do induction on $a \in \mathrm{SN}$:

$$\begin{array}{l|l}
\forall b. \left[\dfrac{a \overset{\beta}{\to} b}{b \in \mathrm{SN}} \right] & \forall b A. \left[\dfrac{\begin{array}{l} a \overset{\beta}{\to} b \\ A \in \mathrm{SN} \end{array}}{(\lambda x^A . e) b \in SN} \right] \\
\hline
a \in \mathrm{SN} & \forall A. \left[\dfrac{A \in \mathrm{SN}}{(\lambda x^A . e) a \in \mathrm{SN}} \right]
\end{array}$$

- Induction on $A \in \mathrm{SN}$:

$$\begin{array}{l||l}
\forall B. \left[\dfrac{A \overset{\beta}{\to} B}{B \in \mathrm{SN}} \right] & \forall B. \left[\dfrac{A \overset{\beta}{\to} B}{(\lambda x^B . e) a \in SN} \right] \\
\hline
A \in \mathrm{SN} & (\lambda x^A . e) a \in \mathrm{SN}
\end{array}$$

- I.e. we have to prove the goal $(\lambda x^A . e) a \in SN$ under the following assumptions $e \in$ SN, $e[x := a] \in \mathrm{SN}$, $a \in \mathrm{SN}$, $A \in \mathrm{SN}$ and the following induction hypotheses:

  1. $\forall f a A. \left[\dfrac{\begin{array}{l} e \overset{\beta}{\to} f \\ f[x := a] \in \mathrm{SN} \\ a \in \mathrm{SN} \\ A \in \mathrm{SN} \end{array}}{(\lambda x^A . f) a \in \mathrm{SN}} \right]$
  2. $\forall b A. \left[\dfrac{\begin{array}{l} a \overset{\beta}{\to} b \\ A \in \mathrm{SN} \end{array}}{(\lambda x^A . e) b \in \mathrm{SN}} \right]$
  3. $\forall B. \left[\dfrac{A \overset{\beta}{\to} B}{(\lambda x^B . e) a \in \mathrm{SN}} \right]$

In order to prove the goal $(\lambda x^A.e)a \in SN$ we have to prove

$$\forall c. \left[\frac{(\lambda x^A.e)a \overset{\beta}{\to} c}{c \in \mathrm{SN}} \right]$$

We assume $(\lambda x^A.e)a \overset{\beta}{\to} c$. An application can reduce by definition of reduction only if the application is a redex (which is the case) or if the function term reduces or if the argument reduces. Since the function term $\lambda x^A.e$ is an abstraction it reduces either if the type of the argument A reduces or the body e reduces. I.e. we have 4 cases to consider:

1. $(\lambda x^A.e)a \overset{\beta}{\to} e[x := a]$. Trivial. The goal $e[x := a]$ is valid by assumption.

2. $(\lambda x^A.e)a \overset{\beta}{\to} (\lambda x^A.f)a$, where $e \overset{\beta}{\to} f$: In that case we have to prove the goal

   $$(\lambda x^A.f)a \in \mathrm{SN}$$

   by using $e \overset{\beta}{\to} f$ and the above assumptions. We can use the induction hypothesis 1 to prove the goal. The only thing missing is a prove of $f[x := a] \in \mathrm{SN}$. From the theorem 3.10 we infer $e[x := a] \overset{\beta}{\to} f[x := a]$. Since $e[x := a] \in \mathrm{SN}$, all its reducts must be strongly normalizing by definition of strong normalization. This completes the proof for that case.

3. $(\lambda x^A.e)a \overset{\beta}{\to} (\lambda x^B.e)a$, where $A \overset{\beta}{\to} B$: In that case we have to prove the goal

   $$(\lambda x^B.e)a \in \mathrm{SN}$$

   By using $A \overset{\beta}{\to} B$ and the induction hypothesis 3 we prove the goal.

4. $(\lambda x^A.e)a \overset{\beta}{\to} (\lambda x^A.e)b$, where $a \overset{\beta}{\to} b$: In that case we have to prove the goal

   $$(\lambda x^A.e)b \in \mathrm{SN}$$

   By using $a \overset{\beta}{\to} b$ and the induction hypothesis 2 we prove the goal.

□

6.2 Base Terms

Definition 6.5. The set of *base terms* BT is the set of all variables applied to zero or more strongly normalizing arguments i.e. terms of the form $xa_1 \dots a_n$ where $a_i \in \mathrm{SN}$ for all i. We formally define the set BT by the rules

1. $x \in \mathrm{BT}$ where x ranges over variables.
2. $\dfrac{\begin{array}{c} a \in \mathrm{BT} \\ b \in \mathrm{SN} \end{array}}{ab \in \mathrm{BT}}$

Theorem 6.6. *All base terms are strongly normalizing.*

$$\frac{a \in \mathrm{BT}}{a \in \mathrm{SN}}$$

Proof. Because a base terms begins with a variable followed by zero or more strongly normalizing arguments, it has no redex whose reduction can change the structure. I.e. only the arguments can reduce and they are by definition strongly normalizing. □

6.3 Key Reduction

Definition 6.7. The *key reduction* $a \overset{\beta_{\mathrm{k}}}{\to} b$ is a relation defined by the rules

1. $$(\lambda x^A.e)a \overset{\beta_{\mathrm{k}}}{\to} e[x := a]$$
2. $$\frac{a \overset{\beta_{\mathrm{k}}}{\to} b}{ac \overset{\beta_{\mathrm{k}}}{\to} bc}$$

I.e. a key reduction reduces only a leftmost redex in an application.

Lemma 6.8. *Let $a \overset{\beta_k}{\to} b$ and $a \overset{\beta}{\to} a_1$ where $a_1 \neq b$. Then there exists a term b_1 with $a_1 \overset{\beta_k}{\to} b_1$ and $b \overset{\beta^*}{\to} b_1$.*

$$\begin{tikzcd}
a \arrow[r, "\beta_{\mathrm{k}}"] \arrow[d, "\beta"] & b \arrow[d, "\beta^*"] \\
\underbrace{a_1}_{\neq b} \arrow[r, "\beta_{\mathrm{k}}"] & \exists b_1
\end{tikzcd}$$

Proof. By induction on $a \overset{\beta_k}{\to} b$:

1. $(\lambda y^A.e)a \overset{\beta_k}{\to} e[x := a]$:

   Assume $\lambda x^A.e \overset{\beta}{\to} a_1$ and $a_1 \neq e[x := a]$. The goal is to find b_1 such that $a_1 \overset{\beta_k}{\to} b_1$ and $e[x := a] \overset{\beta^*}{\to} b_1$.

   There are 4 cases to consider:

   (a) $(\lambda x^A.e)a \overset{\beta}{\to} e[x := a]$:
   This case is contradictory because $e[x := a] \neq e[x := a]$ is not satisfiable.

   (b) $(\lambda x^A.e)a \overset{\beta}{\to} (\lambda x^{A_1}.e)a$ where $A \overset{\beta}{\to} A_1$: In that case have $(\lambda x^{A_1}.e)a \overset{\beta_k}{\to} e[x := a]$ and we choose $b_1 = e[x := a]$.

   (c) $(\lambda x^A.e)a \overset{\beta}{\to} (\lambda x^A.f)a$ where $e \overset{\beta}{\to} f$:
   In that case we have $(\lambda x^A.f)a \overset{\beta_k}{\to} f[x := a]$ and we choose $b_1 = f[x := a]$. This is possible because $e[x := a] \overset{\beta^*}{\to} f[x := a]$ by theorem 3.10.

   (d) $(\lambda x^A.e)a \overset{\beta}{\to} (\lambda x^A.e)b$ where $a \overset{\beta}{\to} b$:
   In that case we have $(\lambda x^A.e)b \overset{\beta_k}{\to} e[x := b]$ and we choose $b_1 = e[x := b]$. This is possible because $e[x := a] \overset{\beta^*}{\to} e[x := b]$ by theorem 3.11.

2. $$\begin{array}{c|c} a \overset{\beta_k}{\to} b & \forall a_1.\left[\dfrac{a \overset{\beta}{\to} a_1}{\exists b_1.a_1 \overset{\beta_k}{\to} b_1 \wedge b \overset{\beta^*}{\to} b_1}\right] \\ \hline ac \overset{\beta_k}{\to} bc & \forall d.\left[\dfrac{ac \overset{\beta}{\to} d}{\exists d_1.d \overset{\beta_k}{\to} d_1 \wedge bc \overset{\beta^*}{\to} d_1}\right] \end{array}$$

   We prove the goal in the lower right corner by assuming $ac \overset{\beta}{\to} d$ and find some d_1 with the required properties.

   Since $a \overset{\beta_k}{\to} b$ we know that a is not an abstraction. Therefore we have two cases to consider:

   (a) $ac \overset{\beta}{\to} a_1c$ where $a \overset{\beta}{\to} a_1$:
   From the induction hypothesis we postulate the existence of b_1 such that $a_1 \overset{\beta_k}{\to} b_1$ and $b \overset{\beta^*}{\to} b_1$. We choose $d_1 = b_1c$ which satisfies $a_1c \overset{\beta_k}{\to} b_1c$ and $bc \overset{\beta^*}{\to} b_1c$.

   (b) $ac \overset{\beta}{\to} ac_1$ where $c \overset{\beta}{\to} c_1$:

We choose $d_1 = bc_1$ which satisfies $ac_1 \xrightarrow{\beta^*} bc_1$ and $bc \xrightarrow{\beta^*} bc_1$.

$\square$

Theorem 6.9. *Let's have $a \xrightarrow{\beta_k} b$ and a and bc are strongly normalizing. Then ac is strongly normalizing as well.*

$$\dfrac{\begin{array}{l} a \xrightarrow{\beta_k} b \\ a, bc \in \text{SN} \end{array}}{ac \in \text{SN}}$$

Proof. Since $bc \in \text{SN}$ implies $c \in \text{SN}$ we can prove the equivalent theorem

$$\dfrac{\begin{array}{l} a \in \text{SN} \\ a \xrightarrow{\beta_k} b \\ c \in \text{SN} \\ bc \in \text{SN} \end{array}}{ac \in \text{SN}}$$

This modified theorem has the advantage that we can do induction on $c \in \text{SN}$ in the course of the proof.

1. Induction on $a \in \text{SN}$:

$$\begin{array}{l|l}
\forall a_1 . \left[\dfrac{a \xrightarrow{\beta} a_1}{a_1 \in \text{SN}} \right] & \forall a_1 b_1 c . \left[\dfrac{\begin{array}{l} a \xrightarrow{\beta} a_1 \\ a_1 \xrightarrow{\beta_k} b_1 \\ c \in \text{SN} \\ b_1 c \in \text{SN} \end{array}}{a_1 c \in \text{SN}} \right] \\
\hline
a \in \text{SN} & \forall bc . \left[\dfrac{\begin{array}{l} a \xrightarrow{\beta_k} b \\ c \in \text{SN} \\ bc \in \text{SN} \end{array}}{ac \in \text{SN}} \right]
\end{array}$$

2. Assume $a \xrightarrow{\beta_k} b$.

3. Induction on $c \in \text{SN}$:

$$\begin{array}{l|l}
\forall c_1 . \left[\dfrac{c \xrightarrow{\beta} c_1}{c_1 \in \text{SN}} \right] & \forall c_1 . \left[\dfrac{\begin{array}{l} c \xrightarrow{\beta} c_1 \\ bc_1 \in \text{SN} \end{array}}{ac_1 \in \text{SN}} \right] \\
\hline
c \in \text{SN} & \dfrac{bc \in \text{SN}}{ac \in \text{SN}}
\end{array}$$

We prove the goal in the lower right corner by assuming $bc \in \mathrm{SN}$ and prove the final goal $ac \in \mathrm{SN}$.

4. In order to prove $ac \in \mathrm{SN}$ we assume $ac \overset{\beta}{\to} d$ and prove $d \in \mathrm{SN}$ for all d.

   Since $a \overset{\beta_\mathrm{k}}{\to} b$ we know that a is not an abstraction. Therefore there are only three cases to consider.

   (a) $ac \overset{\beta}{\to} bc$ where $a \overset{\beta}{\to} b$: In that case we have to prove the goal
   $$d = bc \in \mathrm{SN}$$
   This is trivial since $bc \ \in \mathrm{SN}$ is an assumption.

   (b) $ac \overset{\beta}{\to} a_1c$ where $a \overset{\beta}{\to} a_1$ and $a_1 \neq b$: We have to prove the goal
   $$d = a_1c \in \mathrm{SN}$$
   Since we have $a \overset{\beta_\mathrm{k}}{\to} b$ by lemma 6.8 there exists a b_1 such that $a_1 \overset{\beta_\mathrm{k}}{\to} b_1$ and $b \overset{\beta^*}{\to} b_1$. Because of $bc \in \mathrm{SN}$ we have $b_1c \in \mathrm{SN}$ as well. Therefore all premises of the induction hypothesis of step 1 are satisfied and we get $a_1c \in \mathrm{SN}$ from it.

   (c) $ac \overset{\beta}{\to} ac_1$ where $c \overset{\beta}{\to} c_1$: We have to prove the goal
   $$d = ac_1 \in \mathrm{SN}$$
   Because of $bc \in \mathrm{SN}$ we have $bc_1 \in \mathrm{SN}$ as well. Therefore the preconditions of the induction hypothesis of step 3 are satisfied and the goal is an immediate consequence of the induction hypothesis.

□

6.4 Saturated Sets

Definition 6.10. A *saturated set* S is a set of strongly normalizing terms (i.e. $S \subseteq \mathrm{SN}$) which is closed under the rules

1. All base terms are in a saturated set:
$$\frac{b \in \mathrm{BT}}{b \in S}$$

2. All strongly normalizing terms which keyreduce to a term in the saturated set are in the saturated set as well:

$$\frac{\begin{array}{l} a \in \mathrm{SN} \\ b \in S \\ a \overset{\beta_k}{\to} b \end{array}}{a \in S}$$

The set of all saturated sets is abbreviated by SAT.

Theorem 6.11. *An arbitrary intersection of saturated sets is a saturated set.*

$$\frac{C \subseteq \mathrm{SAT}}{\bigcap C \in \mathrm{SAT}}$$

Proof. We have to prove three things:

1. All terms in $\bigcap C$ are strongly normalizing:

   Since all sets in C contain only strongly normalizing terms, the intersection contains only strongly normalizing terms as well.

   Note the cornercase $\bigcap \emptyset = \mathrm{SN}$ since SN is the base set.

2. $\bigcap C$ contains all base terms:

   Since all sets in C contain all base terms, the intersection of all sets in C contain all base terms as well.

3. $\bigcap C$ contains all strongly normalizing key redexes which reduce to a term in it:

   Assume that the term b is in $\bigcap C$. Then by definition of intersection $b \in S$ for all $S \in C$. Since all $S \in C$ are saturated any strongly normalizing term a with $a \overset{\beta_k}{\to} b$ is in all sets S as well. Therefore a has to be in the intersection $\bigcap C$.

□

Remark:

For those interested in lattice theory. The set of all subsets of the set of strongly normalizing terms (i.e. the powerset of SN) is a complete lattice with intersection and union as the meet and join operations. The subset relation induces a partial order.

The function which maps any set of strongly normalizing terms into a saturated set (i.e. which adds all base terms and all strongly normalizing key redexes) is monotonic, increasing and idempotent. I.e. it is a closure map. The saturated sets are fixpoints of that function.

The fixpoints of such a map form in general a complete lattice which is closed with respect to intersection and union.

6.5 Lambda Function Space

Definition 6.12. *If A and B are sets of lambda terms we define the lambda functions space $A \overset{\lambda}{\to} B$ as the set of lambda terms f such that whenever a is in A then fa is in B.*

$$A \overset{\lambda}{\to} B := \{f \mid \forall a.a \in A \Rightarrow fa \in B\}$$

Theorem 6.13. *The lambda function space between saturated sets is a saturated set.*

$$\frac{\begin{array}{l} A \in \mathrm{SAT} \\ B \in \mathrm{SAT} \end{array}}{A \overset{\lambda}{\to} B \in \mathrm{SAT}}$$

Proof. We have to prove three things:

1. All terms in $A \overset{\lambda}{\to} B$ are strongly normalizing: Assume $f \in A \overset{\lambda}{\to} B$. Then by definition $fa \in B \subseteq \mathrm{SN}$ for all $a \in A$. Therefore fa is strongly normalizing. Since all subterms of a strongly normalizing term are strongly normalizing as well 6.3 f is strongly normalizing.

2. $A \overset{\lambda}{\to} B$ contains all base terms:

   We have to prove two things according to the definition of base terms:

   (a) All variables are in $A \overset{\lambda}{\to} B$:
   Since B is saturated, it contains all terms of the form xa where $a \in A$, because xa is a base term. Therefore $x \in A \overset{\lambda}{\to} B$ for all variables x.

   (b) If $c \in A \overset{\lambda}{\to} B$ where c is a baseterm, then $cd \in A \overset{\lambda}{\to} B$ for all strongly normalizing terms d:
   Since B is saturated it contains all terms of the form cda for $a \in A \subseteq \mathrm{SN}$, because cda is a base term. Therefore by definition of the lambda function space $cd \in A \overset{\lambda}{\to} B$.

3. $A \overset{\lambda}{\to} B$ contains all strongly normalizing key redexes which reduce to a term in it:

   Assume $d \in A \overset{\lambda}{\to} B$, $c \overset{\beta_k}{\to} d$ and $c \in \text{SN}$. We have to prove that $c \in A \overset{\lambda}{\to} B$.

   By definition of keyreduction we have $ca \overset{\beta_k}{\to} da$ for all $a \in A$ and by definition of the lambda function space $da \in B \subseteq \text{SN}$.

   Since B is saturated, it contains ca provided that ca is strongly normalizing. ca is strongly normalizing by theorem 6.9 since $ca \overset{\beta_k}{\to} da$ and $da \in \text{SN}$.

   Therefore by definition of the lambda function space $c \in A \overset{\lambda}{\to} B$.

□

6.6 Model Set

Definition 6.14. *Model Set* For $K \in \mathbb{K}$ we define the model set $\nu(K)$ by

$$\nu(K) := \begin{cases} \nu(s) & := & \text{SAT} & s \text{ is a sort} \\ \nu(\Pi x^A.B) & := & \nu(A) \to \nu(B) & A, B \in \mathbb{K} \\ \nu(\Pi x^A.B) & := & \nu(B) & A \notin \mathbb{K} \wedge B \in \mathbb{K} \end{cases}$$

Definition 6.15. *Canonical Model* For $K \in \mathbb{K}$ we define the canonical model $\nu^c(K)$ by

$$\nu^c(K) := \begin{cases} \nu^c(s) & := & \text{SN} & s \text{ is a sort} \\ \nu^c(\Pi x^A.B) & := & \bullet \mapsto \nu^c(B) & A, B \in \mathbb{K} \\ \nu^c(\Pi x^A.B) & := & \nu^c(B) & A \notin \mathbb{K} \wedge B \in \mathbb{K} \end{cases}$$

where $\bullet \mapsto v$ is the constant function which maps any argument to the value v.

The following property of ν is convenient:

$$\frac{\begin{array}{l} K \in \mathbb{K} \\ \Gamma \vdash F : K \\ \Gamma \vdash F : T \end{array}}{T \in \mathbb{K} \wedge \nu(K) = \nu(T)}$$

In order to prove the property we first prove a similar lemma for kinds.

Lemma 6.16. *The model set of a welltyped kind is the same as the model set of any beta equivalent welltyped type.*

$$\frac{\begin{array}{l} K \in \mathbb{K} \\ \Gamma \vdash K : \mathcal{U} \\ \Gamma \vdash T : s_T \\ K \overset{\beta}{\sim} T \end{array}}{T \in \mathbb{K} \wedge \nu(K) = \nu(T)}$$

Proof. By induction on the structure of K.

General observation: Since K and T are betaequivalent we get from theorem 5.18 $s_T = \mathcal{U}$ and by 5.20 that T is a syntactical kind. This is valid for all induction hypotheses.

For the induction proof we distiguish three cases:

1. $K = s$: In that case K has to be $\mathcal{P}$, otherwise it would not be welltyped.

   Using the general observation above we conclude that T is a syntactical kind. $\mathcal{P}$ is the only possible syntactical kind beta equivalent to $\mathcal{P}$. Therefore $T = \mathcal{P}$ which implies the goal $\nu(T) = \nu(\mathcal{P})$ trivially.

2. $K = \Pi x^{K_1}.K_2$: Since T is a syntactical kind betaequivalent to K it must have the form of a product (a sort is not possible). I.e. $T = \Pi x^A.B$ for some types A and B.

   By theorem 4.18 we get the equivalences $K_1 \overset{\beta}{\sim} A$ and $K_2 \overset{\beta}{\sim} B$.

   From both induction hypotheses we conclude $\nu(K_1) = \nu(A)$ and $\nu(K_2) = \nu(B)$.

3. $K = \Pi x^A.K_2$ (where $A \notin \mathbb{K}$): By the same reasoning as above we get $T = \Pi x^{A_T}.B$ for some types A_T and B with $A \overset{\beta}{\sim} A_T$ and $K_2 \overset{\beta}{\sim} B$.

   From the induction hypothesis we get $\nu(K_2) = \nu(B)$.

   Since both A and A_T are valid types and A is not a syntactical kind, we conclude by using 5.18 that A_T cannot be a syntactical kind either.

   Therefore we get

   $$\nu(K) = \nu(K_2) = \nu(B) = \nu(T).$$

□

Theorem 6.17. *If a type function F is welltyped (i.e. its type is a kind), then the model sets of all its possible types are the same.*

$$\begin{array}{l}
K \in \mathbb{K} \\
\Gamma \vdash F : K \\
\Gamma \vdash F : T \\
\hline
T \in \mathbb{K} \wedge \nu(K) = \nu(T)
\end{array}$$

Proof. From 5.16 we infer $T \overset{\beta}{\sim} U$ and from 5.18 we infer that either both are $\mathcal{U}$ or both are types of the same sort.

If both are $\mathcal{U}$, then $\nu(K) = \nu(T)$ is valid trivially.

Assume both are types of the same sort i.e. we have $\Gamma \vdash K : s$ and $\Gamma \vdash T : s$. Since K is a syntactical kind $s = \mathcal{U}$ is valid.

Therefore the assumptions of lemma 6.16 are valid and we infer the goal by applying the lemma. □

As a next step we prove the fact that the model set of a kind is not affected by a welltyped substitution.

Theorem 6.18. *A type correct variable substitution does not affect the model set of a kind.*

$$\begin{array}{l}
K \in \mathbb{K} \\
\Gamma, x^A \vdash K : \mathcal{U} \\
\Gamma \vdash a : A \\
\hline
\nu(K) = \nu(K[x := a])
\end{array}$$

Proof. By induction on $K \in \mathbb{K}$.

1. $s \in \mathbb{K}$: Trivial
2.

$$\begin{array}{l|l}
B \notin \mathbb{K} & \\
K \in \mathbb{K} & \forall \Delta'. \left[\dfrac{\Gamma, x^A, \Delta' \vdash K : \mathcal{U}}{\nu(K) = \nu((K)[x := a])} \right] \\
\hline
\Pi y^B.K \in \mathbb{K} & \forall \Delta. \left[\dfrac{\Gamma, x^A, \Delta \vdash \Pi y^B.K : \mathcal{U}}{\nu(\Pi y^B.K) = \nu((\Pi y^B.K)[x := a])} \right]
\end{array}$$

We assume $\Gamma, x^A, \Delta \vdash \Pi y^B.K : \mathcal{U}$ and derive the goal $\nu(\Pi y^B.K) = \nu((\Pi y^B.K)[x := a])$.

From the generation lemma 5.5 for products we postulate the existence of s_B and s_K such that

$$\begin{array}{lcl} \Gamma, x^A, \Delta & \vdash & B : s_B \\ \Gamma, x^A, \Delta, y^B & \vdash & K : s_K \\ s_K \overset{\beta}{\sim} \mathcal{U} & & \end{array}$$

This implies $s_B = \mathcal{P}$ (because $B \notin \mathbb{K}$ and corollary 5.22) and $s_K = \mathcal{U}$.

Applying the substitution lemma 5.11 we get

$$\Gamma, x^A, \Delta[x := a] \vdash B[x := a] : \mathcal{P}$$

which by theorem 5.23 implies $B[x := a] \notin \mathbb{K}$.

We use $\Delta' = \Delta, y^B$ and derive $\nu(K) = \nu(K[x := a])$ from the induction hypothesis.

Therefore by the equalities

$$\begin{array}{rcl} \nu(\Pi y^B.K) & = & \nu(K) \\ & = & \nu(K[x := a]) \\ & = & \nu(\Pi y^{B[x:=a]}.K[x := a]) \\ & = & \nu((\Pi y^B.K)[x := a]) \end{array}$$

we derive the desired goal.

3.

$$\begin{array}{l|l} B \in \mathbb{K} & \forall\Delta. \left[\dfrac{\Gamma, x^A, \Delta \vdash B : \mathcal{U}}{\nu(B) = \nu((B)[x := a])}\right] \\ K \in \mathbb{K} & \forall\Delta'. \left[\dfrac{\Gamma, x^A, \Delta' \vdash K : \mathcal{U}}{\nu(K) = \nu((K)[x := a])}\right] \\ \hline \Pi y^B.K \in \mathbb{K} & \forall\Delta. \left[\dfrac{\Gamma, x^A, \Delta \vdash \Pi y^B.K : \mathcal{U}}{\nu(\Pi y^B.K) = \nu((\Pi y^B.K)[x := a])}\right] \end{array}$$

We assume $\Gamma, x^A, \Delta \vdash \Pi y^B.K : \mathcal{U}$ and derive the goal $\nu(\Pi y^B.K) = \nu((\Pi y^B.K)[x := a])$.

From the generation lemma 5.5 for products we postulate the existence of s_B and s_K such that

$$\begin{array}{lcl} \Gamma, x^A, \Delta & \vdash & B : s_B \\ \Gamma, x^A, \Delta, y^B & \vdash & K : s_K \\ s_K \overset{\beta}{\sim} \mathcal{U} & & \end{array}$$

This implies $s_B = \mathcal{U}$ and $s_K = \mathcal{U}$ because of $B, K \in \mathbb{K}$ and therorem 5.23.

By using $\Delta' = \Delta, y^B$ the preconditions of both induction hypotheses are satisfied and we derive the facts

$$\begin{array}{rcl} \nu(B) & = & \nu(B[x := a]) \\ \nu(K) & = & \nu(K[x := a]) \end{array}$$

Therefore by the equalities

$$\begin{array}{rcl} \nu(\Pi y^B.K) & = & \nu(B) \to \nu(K) \\ & = & \nu(B[x := a]) \to \nu(K[x := a]) \\ & = & \nu(\Pi y^{B[x:=a]}.K[x := a]) \\ & = & \nu((\Pi y^B.K)[x := a]) \end{array}$$

we derive the desired goal.

□

6.7 Context Interpretation

Definition 6.19. *Context Interpretation*: We call $\xi = [x_1^{M_1}, x_2^{M_2}, \ldots, x_n^{M_n}]$ an interpretation of a context Γ if and only if it satisfies the relation $\xi \vDash \Gamma$ defined by the rules

1. Empty context

$$[] \vDash []$$

2. Term variable

$$\frac{\begin{array}{l} \xi \vDash \Gamma \\ \Gamma \vdash A : \mathcal{P} \\ x \notin \Gamma \end{array}}{\xi \vDash \Gamma, x^A}$$

3. Type (function) variable

$$\frac{\begin{array}{l} \xi \vDash \Gamma \\ \Gamma \vdash K : \mathcal{U} \\ F \notin \Gamma \\ M \in \nu(K) \end{array}}{\xi, F^M \vDash \Gamma, F^K}$$

Theorem 6.20. *For every valid context Γ there exists a unique canonical context interpretation $\xi^c(\Gamma)$ with $\xi^c(\Gamma) \vDash \Gamma$.*

Proof. We construct the canonical context interpretation recursively.

$$\xi^c(\Gamma) := \begin{cases} \xi^c([]) & := & [] & \\ \xi^c(\Gamma, x^A) & := & \xi^c(\Gamma) & A \notin \mathbb{K} \\ \xi^c(\Gamma, x^A) & := & \xi^c(\Gamma), x^{\nu^c(A)} & A \in \mathbb{K} \end{cases}$$

□

6.8 Type Interpretation

The goal of this section is to find a function which maps any welltyped type function F for all types K it can have in a certain context Γ into a value $[\![F]\!]_{\xi\Gamma} \in \nu(K)$ or $\mathcal{U}$ into $[\![\mathcal{U}]\!]_{\xi\Gamma} \in \mathrm{SAT}$. The function is based on a context interpretation ξ which assigns to each type variable x of type A (which is a kind) in the context a model which is an element of $\nu(A)$.

Definition 6.21. *The type interpretation function $[\![F]\!]_{\xi\Gamma}$ must satisfy the following specification.*

$$\frac{\begin{array}{l}\xi \vDash \Gamma \\ \Gamma \vdash F : K \\ K \in \mathbb{K}\end{array}}{[\![F]\!]_{\xi\Gamma} \in \nu(K)} \wedge \frac{\xi \vDash \Gamma}{[\![\mathcal{U}]\!]_{\xi\Gamma} \in \mathrm{SAT}}$$

I.e. it has the preconditions

- The context Γ has to be valid and ξ is a context interpretation (i.e. $\xi \vdash \Gamma$).
- F is either $\mathcal{U}$ or it is a welltyped type function (i.e. there exist some type $K \in \mathbb{K}$ such that $\Gamma \vdash t : K$ is valid).

In the case that F is a welltyped type function, then the typeinterpretation has to be an element of $\nu(T)$ for all possible types of F. If F is $\mathcal{U}$ then the typeinterpretation must be a saturated set.

Definition 6.22. *The type interpretation function $[\![F]\!]_{\xi\Gamma}$ is defined recursively on the structure of F.*

Note that by theorem 6.17 we can choose any type of a welltyped type function to prove the satisfaction of the specification.

1. Sort:

$$[\![s]\!]_{\xi\Gamma} := \mathrm{SN}$$

This definition satisfies the specification 6.21. There are two cases possible.

If $s = \mathcal{P}$ then it is welltyped and its type is $\mathcal{U}$ and we have $\mathrm{SN} \in \mathrm{SAT}$ and $\mathrm{SAT} = \nu(\mathcal{U})$.

If $s = \mathcal{U}$ then the specification is trivially satisfied.

2. Variable:

$$[\![x]\!]_{\xi\Gamma} := M$$

where $x^M \in \xi$.

The precondition $\xi \vDash \Gamma$ is possible only if there is a type A such that $x^A \in \Gamma$ and $\Gamma \vdash A : \mathcal{U}$ is valid. By the start lemma 5.3 $\Gamma \vdash x : A$ is valid and since ξ is a valid context interpretation for the context Γ we get $M \in \nu(A)$.

3. Product:

$$[\![\Pi x^A.B]\!]_{\xi\Gamma} := [\![A]\!]_{\xi\Gamma} \overset{\lambda}{\to} I_B$$

where

$$I_B := \begin{cases} [\![B]\!]_{\xi(\Gamma,x^A)} & \text{if } A \notin \mathbb{K} \\ \bigcap_{M\in\nu(A)} [\![B]\!]_{(\xi,x^M)(\Gamma,x^A)} & \text{if } A \in \mathbb{K} \end{cases}$$

Since a product is not $\mathcal{U}$ we have to prove the left part of the specification

$$\frac{\begin{array}{l} \xi \vDash \Gamma \\ \Gamma \vdash \Pi x^A.B : K \\ K \in \mathbb{K} \end{array}}{[\![A]\!]_{\xi\Gamma} \overset{\lambda}{\to} I_B \in \nu(K)}$$

where I_B is the type interpretation of B (see above).

From the generation lemma 5.5 for products we postulate the existence of the sorts s_A and s_B such that $\Gamma \vdash A : s_A$ and $\Gamma, x^A \vdash B : s_B$ are valid and s_B is beta equivalent to K.

Since products cannot be beta equivalent to sorts by 4.19 K must be a sort and the only sort beta equivalent to s_B is s_B. Therefore we have $K = s_B$ and $\nu(K) = \mathrm{SAT}$.

In order to prove that $[\![A]\!]_{\xi\Gamma} \overset{\lambda}{\to} I_B$ is a saturated set by theorem 6.13 it is sufficient to prove that $[\![A]\!]_{\xi\Gamma}$ and I_B are saturated sets.

(a) $[\![A]\!]_{\xi\Gamma} \in \mathrm{SAT}$: The preconditions $\xi \vDash \Gamma$, $\Gamma \vdash A : s_A$ and $s_A \in \mathbb{K}$ of the typeinterpretation function are satisfied. Therefore we can conclude the goal.

(b) $I_B \in \mathrm{SAT}$: We have to distinguish two cases:

i. $A \notin \mathbb{K}$: In that case we have $\xi \vDash \Gamma, x^A$ i.e. the preconditions for the typeinterpretation function are satisfied and we get $[\![B]\!]_{\xi(\Gamma,x^A)} \in \mathrm{SAT}$.

ii. $A \in \mathbb{K}$: By theorem 6.11 it is sufficient to prove $[\![B]\!]_{(\xi,x^M)(\Gamma,x^A)} \in \mathrm{SAT}$ for all $M \in \nu(A)$.
Assume $M \in \nu(A)$. Then $\xi, x^M \vDash \Gamma, x^A$ i.e. the preconditions of the typeinterpretation function are satisfied and we infer the goal.

4. Abstraction:

$$[\![\lambda x^A.e]\!]_{\xi\Gamma} := \begin{cases} [\![e]\!]_{\xi(\Gamma,x^A)} & \text{if } A \notin \mathbb{K} \\ M \mapsto [\![e]\!]_{(\xi,x^M)(\Gamma,x^A)} & \text{if } A \in \mathbb{K}, M \in \nu(A) \end{cases}$$

Since an abstraction is not $\mathcal{U}$ we have to prove the left part of the specification

$$\frac{\begin{array}{l}\xi \vDash \Gamma \\ \Gamma \vdash \lambda x^A.e : K \\ K \in \mathbb{K}\end{array}}{[\![\lambda x^A.e]\!]_{\xi\Gamma} \in \nu(K)}$$

By the generation lemma 5.5 for abstractions we postulate the existence of B and s such that $\Gamma \ \vdash \Pi x^A.B : s$, $\Gamma, x^A \vdash e : B$ and $K \overset{\beta}{\sim} \Pi x^A.B$ are satisfied.

By the lemma 6.17 we get $\Pi x^A.B \in \mathbb{K}$ and $\nu(K) = \nu(\Pi x^A.B)$.

In order to prove the goal $[\![\lambda x^A.e]\!]_{\xi\Gamma} \in \nu(\Pi x^A.B)$ we distinguish two cases:

(a) $A \notin \mathbb{K}$: In that case the goal is

$$[\![e]\!]_{\xi(\Gamma,x^A)} \in \nu(B)$$

Since A is not a kind we have $\xi \vDash \Gamma, x^A$ and therefore the preconditions for $[\![e]\!]_{\xi(\Gamma,x^A)}$ are satisfied and the specification of the typeinterpretation function guarantees the goal.

(b) $A \in \mathbb{K}$: In that case the goal is

$$M \mapsto [\![e]\!]_{(\xi,x^M)(\Gamma,x^A)} \in \nu(A) \to \nu(B)$$

where $M \in \nu(A)$. The function argument is in the correct domain. Because of $\xi, x^M \vDash \Gamma, x^A$ the preconditions of $[\![e]\!]_{(\xi,x^M)(\Gamma,x^A)}$ are satisfied and the specification of the type-interpretation function guarantees that the function maps its argument to a value in the correct range.

5. Application:

$$[\![Fa]\!]_{\xi\Gamma} := \left\{ \begin{array}{ll} [\![F]\!]_{\xi\Gamma} & \text{if } \Gamma \vdash a : A \text{ for some } A \notin \mathbb{K} \\ [\![F]\!]_{\xi\Gamma}([\![a]\!]_{\xi\Gamma}) & \text{if } \Gamma \vdash a : A \text{ for some } A \in \mathbb{K} \end{array} \right.$$

where $\Gamma \vdash a : A$ for some A.

Since an application is not $\mathcal{U}$ we have to prove the left part of the specification

$$\frac{\begin{array}{l} \xi \vDash \Gamma \\ \Gamma \vdash Fa : K \\ K \in \mathbb{K} \end{array}}{[\![Fa]\!]_{\xi\Gamma} \in \nu(K)}$$

- By the generation lemma 5.5 for applications we postulate the existence of A and B such that

$$\begin{array}{l} \Gamma \vdash F : \Pi x^A.B \\ \Gamma \vdash a : A \\ K \overset{\beta}{\sim} B[x := a] \end{array}$$

  are valid. By the lemma 6.17 we get $B[x := a] \in \mathbb{K}$ and $\nu(K) = \nu(B[x := a])$ i.e. we have to prove the goal

$$[\![Fa]\!]_{\xi\Gamma} \in \nu(B[x := a])$$

- Using the type of types lemma 5.12 we can derive the existence of some sort s such that $\Gamma \vdash \Pi x^A.B : s$ is valid. This implies by the generation lemma 5.5 for products the existence of the sorts s_A and s_B such that $\Gamma \ \vdash A : s_A$, $\Gamma, x^A \vdash B : s_B$ and $s \overset{\beta}{\sim} s_B$ are valid (i.e. $s = s_B$). Furthermore by the substitution theorem 5.11 we get $\Gamma \vdash B[x := a] : s$.

  This implies that K and $B[x := a]$ are welltyped and therefore cannot be $\mathcal{U}$. Since K is a kind, $s = \mathcal{U}$ must be valid. I.e. we get

$$\begin{array}{lcl} \Gamma & \vdash & A : s_A \\ \Gamma, x^A & \vdash & B : \mathcal{U} \\ \Gamma & \vdash & \Pi x^A.B : \mathcal{U} \end{array}$$

Because of theorem 5.20 we have $B \in \mathbb{K}$ and $\Pi x^A.B \in \mathbb{K}$ i.e. F is a type function.

- Since F is a type function, the preconditions for the type interpretation are satisfied and we get

$$[\![F]\!]_{\xi\Gamma} \in \nu(\Pi x^A.B)$$

- We distinguish two cases

  (a) $A \notin \mathbb{K}$: In that case we have to prove the goal

$$[\![F]\!]_{\xi\Gamma} \in \nu(B[x := a])$$

  We prove the goal by using the equivalence

$$\begin{array}{lcll} \nu(B[x := a] & = & \nu(B) & 6.18 \\ & = & \nu(\Pi x^A.B) & \text{definition of } \nu \end{array}$$

  (b) $A \in \mathbb{K}$: In that case the preconditions of the type interpretation function for a are satisfied and we get

$$[\![a]\!]_{\xi\Gamma} \in \nu(A)$$

  Furthermore we have $[\![F]\!]_{\xi\Gamma} \in \nu(A) \to \nu(B)$ and therefore $[\![F]\!]_{\xi\Gamma}([\![a]\!]_{\xi\Gamma}$ is a valid function application with

$$[\![F]\!]_{\xi\Gamma}([\![a]\!]_{\xi\Gamma}) \in \nu(B)$$

  Since typesafe substitution does not change the model we get by using 6.18 $\nu(B) = \nu(B[x := a])$ which proves the goal

$$[\![F]\!]_{\xi\Gamma)}([\![a]\!]_{\xi\Gamma}) \in \nu(B[x := a])$$

Theorem 6.23. *Type interpretation treats substitution consistently*

$$\frac{\begin{array}{l} \xi \models \Gamma \\ \Gamma \vdash a : A \\ \Gamma, x^A \vdash F : K \\ K \in \mathbb{K} \end{array}}{[\![F[x := a]]\!]_{\xi\Gamma} = \begin{cases} [\![F]\!]_{\xi(\Gamma, x^A)} & A \notin \mathbb{K} \\ [\![F]\!]_{(\xi, x^{[\![a]\!]_{\xi\Gamma}})(\Gamma, x^A)} & A \in \mathbb{K} \end{cases}}$$

Note: Due to the substitution lemma 5.11 we get $\Gamma \vdash F[x := a] : K[x := a]$. A substituted kind remains a kind (easy induction on the structure of the kind). Therefore the preconditions for the type interpretation function are satisfied for the substituted term as well.

Proof. We distinguish two cases:

- $A \in \mathbb{K}$: Assume $\xi \vDash \Gamma$ and $\Gamma \vdash a : A$ and prove the more general lemma:

  $$\forall K \Delta \eta. \left[\begin{array}{l} \Gamma, x^A, \Delta \vdash F : K \\ \xi, x^{[\![a]\!]_{\xi\Gamma}}, \eta \vDash \Gamma, x^A, \Delta \\ K \in \mathbb{K} \\ \hline [\![(F)']\!]_{(\xi,\eta)(\Gamma,\Delta')} = [\![F]\!]_{(\xi,x^{[\![a]\!]_{\xi\Gamma}},\eta)(\Gamma,x^A,\Delta)} \end{array} \right]$$

  where Y' is an abbreviation for $Y[x := a]$.

  Note: Since the model set function ν respects substitution (lemma 6.18) the fact $\xi, x^{[\![a]\!]_{\xi\Gamma}}, \eta \vDash \Gamma, x^A, \Delta$ implies the fact $\xi, \eta \vDash \Gamma, \Delta'$.

  We prove this lemma by induction on the structure of F.

  1. Sort: Trivial, becaus a substituted sort remains the same sort and the type interpretation of a sort is always SN.
  2. Variable y: We distinguish two cases

     (a) $y = x$: In that case we prove the goal by the equivalence

     $$\begin{array}{rcl} [\![x']\!]_{(\xi,\eta)(\Gamma,\Delta')} & = & [\![a]\!]_{(\xi,\eta)(\Gamma,\Delta')} \\ & = & [\![a]\!]_{\xi\Gamma} \\ & = & [\![x]\!]_{(\xi,x^{[\![a]\!]_{\xi\Gamma}},\eta)(\Gamma,x^A,\Delta)} \end{array}$$

     (b) $y \neq x$: In that case we prove the goal by the equivalence

     $$\begin{array}{rcl} [\![y']\!]_{(\xi,\eta)(\Gamma,\Delta')} & = & [\![y]\!]_{(\xi,\eta)(\Gamma,\Delta')} \\ & = & [\![y]\!]_{(\xi,x^{[\![a]\!]_{\xi\Gamma}},\eta)(\Gamma,x^A,\Delta)} \end{array}$$

     Since $y \neq x$ and its type is a kind, the type interpretation by definition depends only on ξ and η.
  3. Product $\Pi y^C.D$: We have to prove the goal

     $$\forall K \Delta \eta. \left[\begin{array}{l} \Gamma, x^A, \Delta \vdash \Pi y^C.D : K \\ \xi, x^{[\![a]\!]_{\xi\Gamma}}, \eta \vDash \Gamma, x^A, \Delta \\ K \in \mathbb{K} \\ \hline [\![(\Pi y^C.D)']\!]_{(\xi,\eta)(\Gamma,\Delta')} = [\![\Pi y^C.D]\!]_{(\xi,x^{[\![a]\!]_{\xi\Gamma}},\eta)(\Gamma,x^A,\Delta)} \end{array} \right]$$

     We can use the two induction hypotheses:

     (a) $$\forall s_C \Delta \eta. \left[\begin{array}{l} \Gamma, x^A, \Delta \vdash C : s_C \\ \xi, x^{[\![a]\!]_{\xi\Gamma}}, \eta \vDash \Gamma, x^A, \Delta \\ s_C \in \mathbb{K} \\ \hline [\![(C)']\!]_{(\xi,\eta)(\Gamma,\Delta')} = [\![C]\!]_{(\xi,x^{[\![a]\!]_{\xi\Gamma}},\eta)(\Gamma,x^A,\Delta)} \end{array} \right]$$

(b) $$\forall s_D \Delta_D \eta_D . \left[\begin{array}{l} \Gamma, x^A, \Delta_D \vdash D : s_D \\ \xi, x^{\llbracket a \rrbracket_{\xi\Gamma}}, \eta \models \Gamma, x^A, \Delta_D \\ s_D \in \mathbb{K} \\ \hline \llbracket (D)' \rrbracket_{(\xi,\eta_D)(\Gamma,\Delta'_D)} = \llbracket D \rrbracket_{(\xi,x^{\llbracket a \rrbracket_{\xi\Gamma}},\eta_D)(\Gamma,x^A,\Delta_D)} \end{array}\right]$$

We prove the goal below the line by assuming all statements above the line and use the equivalence

$$\begin{array}{lcll} \llbracket (\Pi y^C.D)' \rrbracket_{(\xi,\eta)(\Gamma,\Delta')} & = & I_{C'} \overset{\lambda}{\to} I_{D'} & \\ & = & I_C \overset{\lambda}{\to} I_D & \text{see below} \\ & = & \llbracket \Pi y^C.D \rrbracket_{(\xi,x^{\llbracket a \rrbracket_{\xi\Gamma}},\eta)(\Gamma,x^A,\Delta)} & \end{array}$$

where

$$\begin{array}{lcll} I_{C'} & = & \llbracket C' \rrbracket_{(\xi,\eta)(\Gamma,\Delta')} & \\ I_C & = & \llbracket C \rrbracket_{(\xi,x^{\llbracket a \rrbracket_{\xi\Gamma}},\eta)(\Gamma,x^A,\Delta)} & \\ I_{D'} & = & \bigcap_{N \in \nu(C)} \llbracket D' \rrbracket_{(\xi,\eta,y^N)(\Gamma,x^A,\Delta,y^{C'})} & C \in \mathbb{K} \\ I_D & = & \bigcap_{N \in \nu(C)} \llbracket D \rrbracket_{(\xi,x^{\llbracket a \rrbracket_{\xi\Gamma}},\eta,y^N)(\Gamma,x^A,\Delta,y^C)} & C \in \mathbb{K} \\ I_{D'} & = & \llbracket D' \rrbracket_{(\xi,\eta)(\Gamma,\Delta',y^{C'})} & C \notin \mathbb{K} \\ I_D & = & \llbracket D \rrbracket_{(\xi,x^{\llbracket a \rrbracket_{\xi\Gamma}},\eta)(\Gamma,x^A,\Delta,y^C)} & C \notin \mathbb{K} \end{array}$$

We still have to prove $I_{C'} = I_C$ and $I_{D'} = I_D$.
$I_{C'} = I_C$ follows immediately from the first induction hypothesis. In order to prove $I_{D'} = I_D$ we have to distinguish two cases:

(a) $C \notin \mathbb{K}$: Consequence of the second induction hypothesis by using $\eta_D = \eta$ and $\Delta_D = \Delta, y^C$.

(b) $C \in \mathbb{K}$: Consequence of the second induction hypothesis by using $\eta_D = \eta, y^N$ and $\Delta_D = \Delta, y^C$.

4. Abstraction $\lambda y^C.e$: We have to prove the goal

$$\forall K \Delta \eta . \left[\begin{array}{l} \Gamma, x^A, \Delta \vdash \lambda y^C.e : K \\ \xi, x^{\llbracket a \rrbracket_{\xi\Gamma}}, \eta \models \Gamma, x^A, \Delta \\ K \in \mathbb{K} \\ \hline \llbracket (\lambda y^C.e)' \rrbracket_{(\xi,\eta)(\Gamma,\Delta')} = \llbracket \lambda y^C.e \rrbracket_{(\xi,x^{\llbracket a \rrbracket_{\xi\Gamma}},\eta)(\Gamma,x^A,\Delta)} \end{array}\right]$$

We can use the two induction hypotheses

(a) $$\forall s_C \Delta \eta . \left[\begin{array}{l} \Gamma, x^A, \Delta \vdash C : s_C \\ \xi, x^{\llbracket a \rrbracket_{\xi\Gamma}}, \eta \models \Gamma, x^A, \Delta \\ s_C \in \mathbb{K} \\ \hline \llbracket (C)' \rrbracket_{(\xi,\eta)(\Gamma,\Delta')} = \llbracket C \rrbracket_{(\xi,x^{\llbracket a \rrbracket_{\xi\Gamma}},\eta)(\Gamma,x^A,\Delta)} \end{array}\right]$$

(b) $$\forall K_e \Delta_e \eta_e . \left[\begin{array}{l} \Gamma, x^A, \Delta_e \vdash e : K_e \\ \xi, x^{\llbracket a \rrbracket_{\xi\Gamma}}, \eta \vDash \Gamma, x^A, \Delta_e \\ K_e \in \mathbb{K} \\ \hline \llbracket (e)' \rrbracket_{(\xi,\eta_e)(\Gamma,\Delta'_e)} = \llbracket e \rrbracket_{(\xi,x^{\llbracket a \rrbracket_{\xi\Gamma}},\eta_e)(\Gamma,x^A,\Delta_e)} \end{array} \right]$$

We prove the goal below the line by assuming all statements above the line and distinguish two cases:

(a) $C \notin \mathbb{K}$:

$$\begin{array}{rcl} \llbracket (\lambda y^C . e)' \rrbracket_{(\xi,\eta)(\Gamma,\Delta')} & = & \llbracket e' \rrbracket_{(\xi,\eta)(\Gamma,\Delta')} \\ & = & \llbracket e \rrbracket_{(\xi,x^{\llbracket a \rrbracket_{\xi\Gamma}},\eta)(\Gamma,y^C,\Delta)} \\ & = & \llbracket \lambda y^C . e \rrbracket_{(\xi,x^{\llbracket a \rrbracket_{\xi\Gamma}},\eta)(\Gamma,\Delta)} \end{array}$$

In this equivalence we used the definition of type interpretation for abstractions and the second induction hypothesis with $\eta_e = \eta$ and $\Delta_e = \Delta$.

(b) $C \in \mathbb{K}$:

$$\begin{array}{rcl} \llbracket (\lambda y^C . e)' \rrbracket_{(\xi,\eta)(\Gamma,\Delta')} & = & \underbrace{N}_{\in \nu(C')} \mapsto \llbracket e' \rrbracket_{(\xi,\eta)(\Gamma,\Delta')} \\ & = & \underbrace{N}_{\in \nu(C)} \mapsto \llbracket e \rrbracket_{(\xi,x^{\llbracket a \rrbracket_{\xi\Gamma}},\eta,y^N)(\Gamma,y^C,\Delta,y^C)} \\ & = & \llbracket \lambda y^C . e \rrbracket_{(\xi,x^{\llbracket a \rrbracket_{\xi\Gamma}},\eta)(\Gamma,\Delta)} \end{array}$$

In this equivalence we used the definition of type interpretation for abstractions, $\nu(C') = \nu(C)$ by 6.18 and the second induction hypothesis with $\eta_e = \eta, y^N$ and $\Delta_e = \Delta, y^C$.

5. Application: We have to prove the goal

$$\forall K \Delta \eta . \left[\begin{array}{l} \Gamma, x^A, \Delta \vdash Gb : K \\ \xi, x^{\llbracket a \rrbracket_{\xi\Gamma}}, \eta \vDash \Gamma, x^A, \Delta \\ K \in \mathbb{K} \\ \hline \llbracket (Gb)' \rrbracket_{(\xi,\eta)(\Gamma,\Delta')} = \llbracket Gb \rrbracket_{(\xi,x^{\llbracket a \rrbracket_{\xi\Gamma}},\eta)(\Gamma,x^A,\Delta)} \end{array} \right]$$

We assume all statements above the line and prove the final goal under the line.

From the generation lemma 5.5 for applications we postulate the existence of B and C such that

$$\begin{array}{lcl} \Gamma, x^A, \Delta & \vdash & G : \Pi y^B . C \\ \Gamma, x^A, \Delta & \vDash & b : B \\ C[y := b] & \overset{\beta}{\sim} & K \end{array}$$

are valid. Since $C[x := a]$ is a kind, C and $\Pi y^B.C$ are kinds as well. Therefore we get the induction hypothesis for G

$$\forall K_G \Delta\eta.\left[\begin{array}{l} \Gamma, x^A, \Delta \vdash G : K_G \\ \xi, x^{\llbracket a \rrbracket_{\xi\Gamma}}, \eta \models \Gamma, x^A, \Delta \\ K_G \in \mathbb{K} \\ \hline \llbracket (G)' \rrbracket_{(\xi,\eta)(\Gamma,\Delta')} = \llbracket G \rrbracket_{(\xi, x^{\llbracket a \rrbracket_{\xi\Gamma}}, \eta)(\Gamma, x^A, \Delta)} \end{array}\right]$$

We distinguish two cases:

(a) $B \notin \mathbb{K}$:

$$\begin{array}{rcl} \llbracket (Gb)' \rrbracket_{(\xi,\eta)(\Gamma,\Delta')} & = & \llbracket G' \rrbracket_{(\xi,\eta)(\Gamma,\Delta')} \\ & = & \llbracket G \rrbracket_{(\xi, x^{\llbracket a \rrbracket_{\xi\Gamma}}, \eta)(\Gamma, x^A, \Delta)} \\ & = & \llbracket Gb \rrbracket_{(\xi, x^{\llbracket a \rrbracket_{\xi\Gamma}}, \eta)(\Gamma, x^A, \Delta)} \end{array}$$

where we used the definition of type interpretation for applications and the induction hypothesis for G.

(b) $B \in \mathbb{K}$: In that case we get an additional induction hypothesis for b:

$$\forall B \Delta\eta.\left[\begin{array}{l} \Gamma, x^A, \Delta \vdash b : B \\ \xi, x^{\llbracket a \rrbracket_{\xi\Gamma}}, \eta \models \Gamma, x^A, \Delta \\ B \in \mathbb{K} \\ \hline \llbracket (b)' \rrbracket_{(\xi,\eta)(\Gamma,\Delta')} = \llbracket b \rrbracket_{(\xi, x^{\llbracket a \rrbracket_{\xi\Gamma}}, \eta)(\Gamma, x^A, \Delta)} \end{array}\right]$$

and prove the goal by the equivalence

$$\begin{array}{rcl} \llbracket (Gb)' \rrbracket_{(\xi,\eta)(\Gamma,\Delta')} & = & \llbracket G' \rrbracket_{(\xi,\eta)(\Gamma,\Delta')}(\llbracket b' \rrbracket_{(\xi,\eta)(\Gamma,\Delta')}) \\ & = & \llbracket G \rrbracket_{(\xi, x^{\llbracket a \rrbracket_{\xi\Gamma}}, \eta)(\Gamma, x^A, \Delta)}(\llbracket b \rrbracket_{(\xi, x^{\llbracket a \rrbracket_{\xi\Gamma}}, \eta)(\Gamma, x^A, \Delta)}) \\ & = & \llbracket Gb \rrbracket_{(\xi, x^{\llbracket a \rrbracket_{\xi\Gamma}}, \eta)(\Gamma, x^A, \Delta)} \end{array}$$

where we used the definition of type interpretation for applications and both induction hypotheses.

- $A \notin \mathbb{K}$: Assume $\xi \models \Gamma$ and $\Gamma \vdash a : A$ and prove the more general lemma

$$\forall K \Delta\eta.\left[\begin{array}{l} \Gamma, x^A, \Delta \vdash F : K \\ \xi, \eta \models \Gamma, x^A, \Delta \\ K \in \mathbb{K} \\ \hline \llbracket F' \rrbracket_{(\xi,\eta)(\Gamma,\Delta')} = \llbracket F \rrbracket_{(\xi,\eta)(\Gamma,\Delta)} \end{array}\right]$$

The proof is practically the same as the proof for the case $A \in \mathbb{K}$ except that $x^{\llbracket a \rrbracket_{\xi\Gamma}}$ is never needed, because A is not a kind and therefore a is not a type function. The variable case is even simpler, because the variable cannot be x (the type of x is not a kind).

□

Theorem 6.24. *Type interpretation respects reduction*

$$\begin{array}{l} F \overset{\beta}{\to} G \\ \xi \models \Gamma \\ \Gamma \vdash F : K \\ K \in \mathbb{K} \\ \hline [\![F]\!]_{\xi\Gamma} = [\![G]\!]_{\xi\Gamma} \end{array}$$

Proof. By induction on $F \overset{\beta}{\to} G$. The only interesting case is the case of a redex. In all the other case the reduction does not change the toplevel structure of the term and the goal can be derived from the induction hypotheses and the definition of type interpretation.

Therefore here we prove only the redex case i.e. we prove

$$\begin{array}{l} (\lambda x^A.e)a \overset{\beta}{\to} e[x := a] \\ \xi \models \Gamma \\ \Gamma \vdash \lambda x^A.e : K \\ K \in \mathbb{K} \\ \hline [\![(\lambda x^A.e)a]\!]_{\xi\Gamma} = [\![e[x := a]]\!]_{\xi\Gamma} \end{array}$$

We distinguish two cases:

1. $A \notin \mathbb{K}$: We prove the goal by the equivalence

$$\begin{array}{lcl} [\![(\lambda x^A.e)a]\!]_{\xi\Gamma} & = & [\![e]\!]_{\xi(\Gamma,x^A)} \\ & = & [\![e[x := a]]\!]_{\xi\Gamma} \end{array}$$

   We have used the definition of type interpretation for applications and abstractions and theorem 6.23.

2. $A \in \mathbb{K}$: We prove the goal by the equivalence

$$\begin{array}{lcl} [\![(\lambda x^A.e)a]\!]_{\xi\Gamma} & = & (\underbrace{M}_{M \in \nu(A)} \mapsto [\![e]\!]_{(\xi,x^M)(\Gamma,x^A)})([\![a]\!]_{\xi\Gamma}) \\ & = & [\![e]\!]_{(\xi,x^{[\![a]\!]_{\xi\Gamma}})(\Gamma,x^A)} \\ & = & [\![e[x := a]]\!]_{\xi\Gamma} \end{array}$$

   We have used the definition of type interpretation for applications and abstractions and theorem 6.23.

□

Theorem 6.25. *Equivalent type functions have the same type interpretation*

$$\frac{\begin{array}{l} F \overset{\beta}{\sim} G \\ \xi \vDash \Gamma \\ \Gamma \vdash F : K \\ \Gamma \vdash G : s \\ K \in \mathbb{K} \end{array}}{[\![F]\!]_{\xi\Gamma} = [\![G]\!]_{\xi\Gamma}}$$

Proof. By induction on $F \overset{\beta}{\sim} G$. The reflexive case is trivial. The forward and the backward cases can be proved by the corresponding induction hypothesis and theorem 6.24. □

Theorem 6.26. *The type interpretation of a type is a saturated set.*

$$\frac{\begin{array}{l} \xi \vDash \Gamma \\ \Gamma \vdash T : s \end{array}}{[\![T]\!]_{\xi\Gamma} \ \in \mathrm{SAT}}$$

Proof. The type interpretation function satisfies the specification

$$[\![T]\!]_{\xi\Gamma} \in \nu(s) = \mathrm{SAT}$$

□

6.9 Term Interpretation

Definition 6.27. *A variable interpretation ρ is a list of variables which associates to each variable a term. No duplicate variables are allowed.*

$$\rho = \{x_1^{t_1}, x_2^{t_2}, \ldots, x_n^{t_n}\}$$

Definition 6.28. *A term interpretation $(\!|u|\!)_\rho$ replaces each free variable x in the term u with the term t when $x^t \in \rho$, otherwise leaves the variable unchanged.*

$$(\!|u|\!)_\rho := \begin{cases} (\!|s|\!)_\rho & := \quad s \\ (\!|x|\!)_\rho & := \quad \begin{cases} t & x^t \in \rho \\ x & \text{otherwise} \end{cases} \\ (\!|\Pi x^A.B|\!)_\rho & := \quad \Pi x^{(\!|A|\!)_\rho}.(\!|B|\!)_{\rho,x^x} \\ (\!|\lambda x^A.e|\!)_\rho & := \quad \lambda x^{(\!|A|\!)_\rho}.(\!|e|\!)_{\rho,x^x} \\ (\!|ab|\!)_\rho & := \quad (\!|a|\!)_\rho(\!|b|\!)_\rho \end{cases}$$

A term interpretation is just a parallel substitution of the free variables in a term. In the following we use only variable interpretations which contain all variables of a context and apply it only to terms which are welltyped in a context.

6.10 Context Model

Definition 6.29. *Context Model*: We call a variable interpretation ρ and a context interpretation ξ (i.e. $\xi \vDash \Gamma$) a model of a context which we write

$$\rho\xi \vDash \Gamma$$

when ρ and Γ have the form

$$\begin{array}{rcl} \Gamma & = & [x_1^{A_1}, \ldots, x_n^{A_n}] \\ \rho & = & [x_1^{t_1}, \ldots, x_n^{t_n}] \end{array}$$

where $t_i \in [\![A_i]\!]_{\xi\Gamma}$ for all $i \in \{1, \ldots, n\}$.

6.11 Soundness Theorem

Theorem 6.30. *Let $\rho\xi$ be a model of the context Γ and $\Gamma \vdash t : T$ a valid typing judgement. Then the term interpretation $(\!|t|\!)_\rho$ is an element of the type interpretation $[\![T]\!]_{\xi\Gamma}$.*

$$\frac{\begin{array}{l}\Gamma \vdash t : T \\ \rho\xi \vDash \Gamma\end{array}}{(\!|t|\!)_\rho \in [\![T]\!]_{\xi\Gamma}}$$

Proof. By induction on the structure of t.

1. Sort: We have to prove the goal

$$\frac{\begin{array}{l}\Gamma \vdash s : T \\ \rho\xi \vDash \Gamma\end{array}}{(\!|s|\!)_\rho \in [\![T]\!]_{\xi\Gamma}}$$

From the generation lemma 5.5 for sorts we get

$$\begin{array}{l} s = \mathcal{P} \\ T \overset{\beta}{\sim} \mathcal{U} \end{array}$$

and prove the goal by

$$(\!|\mathcal{P}|\!)_\rho = \mathcal{P} \ \in \ \mathrm{SN} = [\![\mathcal{U}]\!]_{\xi\Gamma} = [\![T]\!]_{\xi\Gamma}$$

and using the fact that type interpretation respects beta equivalence 6.25

2. Variable: We have to prove the goal

$$\frac{\begin{array}{l}\Gamma \vdash x : T \\ \rho\xi \vDash \Gamma\end{array}}{(\!|x|\!)_\rho \in [\![T]\!]_{\xi\Gamma}}$$

From the generation lemma 5.5 for variables we postulate the existence of A and s with

$$\begin{array}{l}\Gamma \vdash A : s \\ x^A \in \Gamma \\ T \overset{\beta}{\sim} A\end{array}$$

and prove the goal by

$$(\!|x|\!)_\rho = t \ \in \ [\![A]\!]_{\xi\Gamma} = [\![T]\!]_{\xi\Gamma}$$

where $x^t \in \rho$ and we used the definition of $\rho\xi \vDash \Gamma$ and the fact that type interpretation respects beta equality 6.25

3. Product: We have to prove the goal

$$\frac{\begin{array}{l}\Gamma \vdash \Pi x^A.B : T \\ \rho\xi \vDash \Gamma\end{array}}{(\!|\Pi x^A.B|\!)_\rho \in [\![T]\!]_{\xi\Gamma}}$$

From the generation lemma 5.5 for products we postulate the existence of the sorts s_b and s_c with

$$\begin{array}{l}\Gamma \vdash A : s_a \\ \Gamma, x^A \vdash B : s_b \\ T \overset{\beta}{\sim} s_b\end{array}$$

We can use the induction hypotheses

(a) $\forall \Gamma T_a \rho\xi. \left[\dfrac{\begin{array}{l}\Gamma \vdash A : T_a \\ \rho\xi \vDash \Gamma\end{array}}{(\!|A|\!)_\rho \in [\![T_a]\!]_{\xi\Gamma}}\right]$

(b) $\forall \Gamma_b T_b \rho_b \xi_b . \left[\begin{array}{l} \Gamma_b \vdash B : T_b \\ \rho_b \xi_b \models \Gamma_b \\ \hline (\!| B |\!)_{\rho_b} \in \llbracket T_b \rrbracket_{\xi_b \Gamma_b} \end{array} \right]$

The final goal is $(\!| \Pi x^A . B |\!)_\rho \in \llbracket T \rrbracket_{\xi\Gamma} = \llbracket s_b \rrbracket_{\xi\Gamma} = \mathrm{SN}$ which requires the subgoals

$$\begin{array}{lcl} (\!| A |\!)_\rho & \in & \mathrm{SN} \\ (\!| B |\!)_{\rho, x^x} & \in & \mathrm{SN} \end{array}$$

The first subgoal is proved by the first induction hypothesis by using $T_a = s_a$.

The second subgoal is proved by the second induction hypothesis by using

$$\begin{array}{lcl} \Gamma_b & = & \Gamma, x^A \\ T_b & = & s_b \\ \rho_b & = & \rho, x^x \\ \xi_b & = & \left\{ \begin{array}{ll} \xi & A \notin \mathbb{K} \\ \xi, x^{\nu^c(A)} & A \in \mathbb{K} \end{array} \right. \end{array}$$

4. Abstraction: We have to prove the goal

$$\begin{array}{l} \Gamma \vdash \lambda x^A . e : T \\ \rho\xi \models \Gamma \\ \hline (\!| \lambda x^A . e |\!)_\rho \in \llbracket T \rrbracket_{\xi\Gamma} \end{array}$$

From the generation lemma 5.5 for abstractions we postulate the existence of the type B and the sort s with

$$\begin{array}{l} \Gamma \vdash \Pi x^A . B : s \\ \Gamma, x^A \vdash e : B \\ T \overset{\beta}{\sim} \Pi x^A . B \end{array}$$

We can use the induction hypotheses

(a) $\forall \Gamma T_a \rho \xi . \left[\begin{array}{l} \Gamma \vdash A : T_a \\ \rho\xi \models \Gamma \\ \hline (\!| A |\!)_\rho \in \llbracket T_a \rrbracket_{\xi\Gamma} \end{array} \right]$

(b) $\forall \Gamma_e T_e \rho_e \xi_e . \left[\begin{array}{l} \Gamma_e \vdash e : T_e \\ \rho_e \xi_e \models \Gamma_e \\ \hline (\!| e |\!)_{\rho_e} \in \llbracket T_e \rrbracket_{\xi_e \Gamma_e} \end{array} \right]$

We have to prove the final goal

$$\llparenthesis \lambda x^A.e \rrparenthesis_\rho \ \in \ \llbracket T \rrbracket_{\xi\Gamma} = \llbracket \Pi x^A.B \rrbracket_{\xi\Gamma} = \llbracket A \rrbracket_{\xi\Gamma} \overset{\lambda}{\to} I_B$$

where $I_B = \begin{cases} \llbracket B \rrbracket_{\xi(\Gamma,x^A)} & A \notin \mathbb{K} \\ \bigcap_{M\in\nu(A)} \llbracket B \rrbracket_{(\xi,x^M)(\Gamma,x^A)} & A \in \mathbb{K} \end{cases}$

By definition of $\overset{\lambda}{\to}$ we have to prove

$$\llparenthesis \lambda x^A.e \rrparenthesis_\rho a \in I_B$$

for all $a \in \llbracket A \rrbracket_{\xi\Gamma}$.

From the second induction hypothesis we infer

$$\llparenthesis e \rrparenthesis_{\rho,x^a} \in \llbracket B \rrbracket_{\xi_e(\Gamma,x^A)}$$

with $\xi_e = \begin{cases} \xi & A \notin \mathbb{K} \\ \xi, x^M & A \in \mathbb{K} \; M \in \nu(A) \end{cases}$ because $(\rho, x^A)\xi_e \vDash \Gamma, x^A$, since $a \in \llbracket A \rrbracket_{\xi\Gamma}$.

This is true for all $M \in \nu(A)$, if $A \in \mathbb{K}$. Therefore we infer from the second induction hypothesis

$$\llparenthesis e \rrparenthesis_{\rho,x^a} \in I_B$$

for all $a \in \llbracket A \rrbracket_{\xi\Gamma}$.

Since I_B is a saturated set it includes all strongly normalizing terms t with $t \overset{\beta_k}{\to} \llparenthesis e \rrparenthesis_{\rho,x^a}$. By definition of $\overset{\beta_k}{\to}$ and term interpretation we have

$$\begin{array}{rcl} \llparenthesis \lambda x^A.e \rrparenthesis_\rho a & = & (\lambda x^{\llparenthesis A \rrparenthesis_\rho}.\llparenthesis e \rrparenthesis_{\rho,x^x})a \\ & \overset{\beta_k}{\to} & \llparenthesis e \rrparenthesis_{\rho,x^x}[x := a] \\ & = & \llparenthesis e \rrparenthesis_{\rho,x^a} \end{array}$$

Therefore it remains to prove that $(\lambda x^{\llparenthesis A \rrparenthesis_\rho}.\llparenthesis e \rrparenthesis_{\rho,x^x})a$ is strongly normalizing. This can be proved by theorem 6.4 provided that

(a) $\llparenthesis A \rrparenthesis_\rho \in \text{SN}$: This can be infered from the first induction hypothesis, because $\llparenthesis A \rrparenthesis_\rho \in \llbracket T_a \rrbracket_{\xi\Gamma}$ and the type of A must be a sort whose type interpretation is the set of strongly normalizing terms.

(b) $a \in \text{SN}$: Since $a \in \llbracket A \rrbracket_{\xi\Gamma}$ and A is a type we infer from theorem 6.26 that a is in a saturated set which by definition has only strongly normalizing terms.

(c) $(\!|e|\!)_{\rho,x^x}[x := a] = (\!|e|\!)_{\rho,x^a} \in \mathrm{SN}$: We have already inferred from the second induction hypothesis $(\!|e|\!)_{\rho,x^a} \in I_B$. Since I_B is either a type interpretation of a type or an intersection of type interpretations of a type and saturated sets are closed with respect to intersection, I_B is a saturated set which by definition contains only strongly normalizing terms.

(d) $(\!|e|\!)_{\rho,x^x} \in \mathrm{SN}$: Because $[\![A]\!]_{\xi\Gamma}$ and type interpretations of types are saturated and saturated sets contain all base terms, we have $x \in [\![A]\!]_{\xi\Gamma}$. Therefore $(\rho, x^x)\xi_e \models \Gamma, x^A$ which implies the goal.

5. Application: We have to prove the goal

$$(\!|f a|\!)_\rho \in [\![T]\!]_{\xi\Gamma}$$

under the assumptions

$$\begin{array}{lcl} \Gamma & \vdash & f a : T \\ \rho\xi & \models & \Gamma \end{array}$$

From the generation lemma 5.5 for applications we postulate the existence of the types A and B such that

$$\begin{array}{lcl} \Gamma & \vdash & f : \Pi x^A.B \\ \Gamma & \vdash & a : A \\ T & \overset{\beta}{\sim} & B[x := a] \end{array}$$

Furthermore we have the following induction hypotheses available:

$$\begin{array}{lcl} (\!|f|\!)_\rho & \in & [\![A]\!]_{\xi\Gamma} \xrightarrow{\lambda} I_B \\ (\!|a|\!)_\rho & \in & [\![A]\!]_{\xi\Gamma} \end{array}$$

where

$$I_B = \begin{cases} [\![B]\!]_{\xi(\Gamma,x^A)} & A \notin \mathbb{K} \\ \bigcap_{M\in\nu(A)} [\![B]\!]_{(\xi,x^M)(\Gamma,x^A)} & A \in \mathbb{K} \end{cases}$$

By using the theorem 6.23, the theorem 6.25 and $T \overset{\beta}{\sim} B[x := a]$ we have to prove the goal

$$(\!|f|\!)_\rho(\!|a|\!)_\rho \in [\![T]\!]_{\xi\Gamma} = [\![(B[x := a])]\!]_{\xi\Gamma} = \begin{cases} [\![B]\!]_{\xi(\Gamma,x^A)} & A \notin \mathbb{K} \\ [\![B]\!]_{(\xi,x^{[\![a]\!]_{\xi\Gamma}})(\Gamma,x^A)} & A \in \mathbb{K} \end{cases}$$

We distinguish two cases:

(a) $A \notin \mathbb{K}$: The goal follows immediately from the induction hypotheses and the definition of $\overset{\lambda}{\to}$.

(b) $A \in \mathbb{K}$: From the induction hypotheses and the definition of $\overset{\lambda}{\to}$ we get

$$\forall M. \left[\frac{M \in \nu(A)}{(\!|f|\!)_\rho (\!|a|\!)_\rho \in [\![B]\!]_{(\xi, x^M)(\Gamma, x^A)}} \right]$$

and from the specification of the type interpretation we get

$$[\![a]\!]_{\xi\Gamma} \in \nu(A)$$

which finally proves the goal.

□

6.12 Strong Normalization Proof

Theorem 6.31. *All welltyped terms are strongly normalizing.*

$$\frac{\Gamma \vdash t : T}{t \in \mathrm{SN}}$$

Proof. Assume $\Gamma \vdash t : T$, i.e. $\Gamma = [x_1 : A_1, \ldots, x_n : A_n]$ is a valid context.

By theorem 6.20 $\xi := \xi^c(\Gamma)$ is a valid interpretation for the context Γ with $\xi \models \Gamma$.

We can form a context model

$$\rho\xi \models \Gamma$$

where

$$\rho = [x_1^{x_1}, \ldots, x_n^{x_n}]$$

Then we have by the soundness theorem 6.30

$$t = (\!|t|\!)_\rho \in [\![T]\!]_{\xi\Gamma} \in \mathrm{SAT}$$

which proves the goal because type interpretations of types are saturated sets 6.26 and saturated sets contain only strongly normalizing terms by definition. □

6.13 Logical Consistency

Logical consistency of the calculus of constructions means that it is not possible to prove contradictions in the calculus. Since a contradiction implies everything we can state logical consistency as *It is not possible to prove every proposition.*

Via the Curry-Howard correspondence we interpret types as propositions and terms of a type as a proof of the proposition which corresponds to that type. In the calculus of constructions terms of the type $\Pi X^{\mathcal{P}}.X$ are functions which map every type X into a proof of that type.

Therefore in the calculus of contructions logical consistency means that there does not exist a term t of type $\Pi X^{\mathcal{P}}.X$ in the empty context.

It is important to use the empty context here because a context $\Gamma = [x_1^{A_1}, \dots, x_n^{A_n}]$ is a sequence of assumptions and it is perfectly possible to make contradictory assumptions e.g. having $f^{\Pi X^{\mathcal{P}}.X}$ as one assumption. In such a context f is a term of type $\Pi X^{\mathcal{P}}.X$.

In the previous section we have proved that all welltyped terms in the calculus of constructions are strongly normalizing. I.e. all welltyped terms can be reduced to their normal form. Therefore it is sufficient to prove that in the empty context there exists no term in normal form which has the type $\Pi X^{\mathcal{P}}.X$.

Lemma 6.32. *All welltyped terms in normal form are either sorts, products, abstractions or base terms.*

Proof. We prove the goal by induction on the structure of t.

1. For sorts, variables, products and abstractions there is nothing to do, because they fall trivially into one of the alternatives.

2. It remains to prove that a welltyped application in normal form is a base term. Assume $\Gamma \vdash fa : T$ and $fa \in \text{NF}$.

   By the generation lemma 5.5 for applications we postulate the existence of the types A and B such that $\Gamma \vdash f : \Pi x^A.B$.

   The first induction hypothesis states that f falls into one of the categories.

   However it cannot be a sort nor a product, because neither of them can have a type equivalent to $\Pi x^A.B$.

   f cannot be an abstraction, otherwise fa would not be in normal form. Therefore f can only be a base term.

If f is a base term, then by definition of base terms fa is a base term as well, because $a \in \text{NF} \subseteq \text{SN}$.

□

Theorem 6.33. *There are no welltyped terms in normal form which have the type* $\Pi X^{\mathcal{P}} : X$ in the empty context.

$$\frac{\begin{array}{l}[] \vdash t : \Pi X^{\mathcal{P}}.X \\ t \in \text{NF}\end{array}}{\bot}$$

Proof. We assume $t \in \text{NF}$ and $[] \vdash t : \Pi X^{\mathcal{P}}.X$ and derive a contradiction.

1. By the previous theorem 6.32 t has to be either a sort, a product an abstraction or a base term. t cannot be a sort nor a product, because they cannot have a type equivalent to a product (the only valid type of both are sorts). It cannot be a base term, because a base term has a free variable in the head position and we are in the empty context (only closed terms).

   Therefore t must be an abstraction, say $\lambda y^B.e$.

2. By the generation lemma 5.5 for abstractions we postulate the existence of a type C such that $[X^{\mathcal{P}}] \vdash e : C$ and $\Pi X^{\mathcal{P}}.X \overset{\beta}{\sim} \Pi X^{\mathcal{P}}.C$ which imply $X \overset{\beta}{\sim} C$. Therefore we have

   $$[X^{\mathcal{P}}] \vdash e : X$$

   and e must be in normal form.

3. e cannot be a sort nor a product nor an abstraction, because the type of the first two must be a sort and the type of the last must be a product. Neither a sort nor a product can be beta equivalent to a variable.

   Therefore e must be a base term. There are two possibilities:

4. e is a variable, say y:

   In that case we have $y = X$, because X is the only free variable available. By the generation lemma 5.5 for variables we get $\mathcal{P} \overset{\beta}{\sim} X$ which is not possible.

5. e has the form $ya_1, \ldots, a_n$ with $n > 0$: We get $y = X$ as well for the same reason. Since y is in a function position its type must be beta equivalent to some product $\Pi y^B.C$. However its type is $\mathcal{P}$ which cannot beta equivalent to a product.

□

7 Bibliography

References

[1] Henk Barendregt. Lambda calculi with types. *Handbook of Logic in Computer Science*, II, 1993.

[2] Herman Geuvers. A short and flexible proof of strong normalization for the calculus of constructions. *Computing Science Report*, 9450, 1994.