Technical Report

UCAM-CL-TR-297
ISSN 1476-2986

Number 297

UNIVERSITY OF
CAMBRIDGE
Computer Laboratory

Categorical abstract machines for higher-order typed lambda calculi

Eike Ritter

April 1993

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

https://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted by the author for the degree of Doctor of Philosophy to the University of Cambridge, Trinity College.

ISSN 1476-2986

Contents

Chapter 1

Introduction

1.1 Categorical Abstract Machines

Formal relations between the syntax of a programming language, its semantics and an appropriate logic enable mathematical methods to be used for reasoning about programs and for designing programming languages. An example of the former application of this relation is program verification according to Floyd-Hoare [Hoa69] [Hoa70], where the meaning of a program is expressed in terms of first-order logic, and an example of the latter is the design of ML, where the intended semantics of the language determines the type system [MH88].

The propositions-as-types analogy [How80] provides a very important means for formally specifying links between type theory, its semantics and certain intuitionistic logics (in the sequel, "logic" will always mean "intuitionistic logic"). It says that types of a type theory, objects of a certain category and propositions in an appropriate logic correspond to each other as well as terms, morphisms and proofs:

Type theory	Category theory	Logic
Type	Object	Proposition
Term	Morphism	Proof

The relation of category theory to logic was established by Lawvere's idea of using certain adjunctions for representing logical concepts [Law69] [Law70]. The link between a type theory and an appropriate category is given by a categorical semantics, i.e. an interpretation of the type theory in the category. As an example, this analogy establishes the following correspondence between simply typed λ-calculus, cartesian closed categories and propositional logic [LS85]:

Typed λ-calculus	Cartesian Closed Categories	Propositional Logic
Product types	Products	Conjunction
Function spaces	Exponentiation	Implication

This analogy has found applications in several areas in computer science. Firstly, it is used to show properties of type theories. An example is the proof that the polymorphic λ-calculus (PLC) has no nontrivial model in classical set theory [RPar], but does have one in intuitionistic set theory [Pit87]. Secondly, the analogy supports the design of programming languages. For example, second-order existential quantification can be used for designing a module concept in the PLC [MP88]. The restrictions of the quantification are exactly the normal side conditions for the existential quantification in the logic. Another example is the construction of a category of modules for a programming language as Grothendieck's construction of the category corresponding to the programming language [Mog89].

A third application, which is the topic of this dissertation, is concerned with categorical abstract machines modelling the reduction of expressions of a typed λ-calculus. The derivation of such machines starts with the choice of an appropriate categorical structure. This structure must have an equational presentation, the so-called categorical combinators. These yield a variable-free presentation of the calculus together with an explicit substitution mechanism. Next, one chooses an inference system for the reduction of combinators corresponding to closures. Such a system finally yields an abstract machine in a natural way. This approach leads to simple and conceptually clean combinators that in turn give rise to a simple and conceptually clean instruction set together with an easy correctness proof. The explicit substitution operation makes it possible to postpone substitutions during reduction. This considerably improves the efficiency of the machines. Furthermore, the machines have a modular structure, i.e. an extension of the language corresponds to an addition of categorical concepts and machine instructions.

The earliest categorical abstract machine is the CAM, constructed by Curien, Cousineau and Mauny [CCM87]. It handles reduction of closed terms of the untyped λ-calculus to weak head normal form according to an eager strategy. The relation of the CAM to cartesian closed categories is somewhat problematic because several concepts that are important for the design of abstract machines are not properly modelled in CCC's:

- Environments and terms are both represented by morphisms although they are conceptually different.
- Composition has two roles, namely substitution in a term with respect to environments and application of a function to an argument.
- Product types and contexts are both modelled by products in the CCC, so again two separate issues are merged into one construction.

Jacobs [Jac92] describes a way of turning a CCC into an indexed category that solves the last of these problems. In that approach however, the representation of environments is still unsatisfactory for the design of abstract machines. The reason is that the products in the base category, which model the environments, impose a tree structure on environments although a list structure is sufficient. The $\lambda\rho$-calculus [Cur91] adds an explicit notion of environment to the simply typed λ-calculus to overcome the above three problems. A generalization, the $\lambda\sigma$-calculus [ACCL90], uses explicit substitutions to derive first an extension of Krivine's machine describing reduction to normal form for the untyped λ-calculus and second a type checker for the typed and the second-order λ-calculus. For the case of the simply typed

λ-calculus, the handling of environments in this version of the $\lambda\sigma$-calculus turns out to be quite close to the one described in the approach below. Crégut [Cré90] uses the variant given in [Cur91], which is linked to multicategories, to construct an abstract machine with a different handling of global variables. Asperti [Asp92] uses the retraction between A and A^1 in a CCC to give a categorical description of an extension of Krivine's machine for reduction to normal form and of a variant of the CAM implementing a λ-calculus with call-by-value and call-by-name as parameter passing modes. We propose *split D-categories* [Ehr88a], which are particular indexed categories, as an appropriate categorical framework for abstract machines, not just for the simply typed λ-calculus, but also for higher-order typed λ-calculi. In this thesis we will concentrate on the Calculus of Constructions. Split D-categories achieve the separation of terms and environments in a very natural way because terms correspond to morphisms in the fibres and environments to morphisms in the base category. This automatically leads to different combinators for substitution (which is modelled by the reindexing functor) and function application (which corresponds to composition in the fibre). The D-categories also model cartesian products and contexts differently: the former are handled by left adjoints to weakening and the latter by a right adjoint to the terminal object functor.

There are two groups of abstract machines based on these D-categories. The first uses eager environments, i.e. environments that contain only canonical combinators, whereas in the second an environment may contain arbitrary expressions. Further design decisions yield machines that, if restricted to the combinators corresponding to the simply typed λ-calculus, can be transformed directly into the CAM (eager case) or Krivine's machine (lazy case); see chapter 4 for details. These abstract machines have a modular structure, i.e. those for the Calculus of Constructions contain those for the typed λ-calculus as submachines. It is also possible to mix eager and lazy constructions, for example one can construct an eager machine with lazy products. This is another advantage of separating products from environments.

The strong normalization of the reduction rules for the combinators is an open problem, even in the case of combinators for the simply typed λ-calculus. The explicit substitution destroys the reducibility approach because there seems to be no way of showing that every reduction path of the combinator corresponding to $((\lambda x{:}\, A.t)s)[y\backslash t']$ contains a contractum of the combinator corresponding to $t[x\backslash s][y\backslash t']$. There are only combinatorial arguments showing that the reductions for substitutions alone are strongly normalizing [HL86] [CHR92]. We will demonstrate here that only finite reduction sequences can arise from any reduction strategy that reduces a combinator first to one corresponding to a weak head-normal form and only then pushes substitution under λ-abstraction to reach a normal form. This restriction excludes all reduction paths that violate the above property for the combinator $((\lambda x{:}\, A.t)s)[y\backslash t']$; therefore the reducibility method produces a proof that all those sequences are finite. Because all strategies that are commonly used for the derivation of abstract machines satisfy this limitation it does not play any role in the relation between the combinators and the abstract machines.

The original version of the Calculus of Constructions (CC) adds a special type Prop of propositions, a type of proofs of a proposition and an impredicative universal quantification over propositions to the simply typed λ-calculus. This requires the use of dependent types, so a function space becomes a special case of a dependent product. Coquand and Huet [CH88] invented this calculus with the propositions-as-types analogy in mind as a language for formalizing mathematical proofs where a proposition is valid iff the type of its proofs is inhabited. Luo [Luo90] adds strong sums to model a program together with its specification. The presence of dependent types in the Calculus of Constructions implies that type checking of CC-expressions may include a reduction of terms. As an example, let p be any proposition and take a proof of a proposition $\forall x\colon A.p$, i.e. a term t of type $\mathsf{Proof}(\forall x\colon A.p)$. If it is claimed that the term ta, where a is of type A, is a proof of a proposition q, then it must be checked whether the two types $p[x \leftarrow a]$ and q are convertible. This is done by reducing them to a suitable normal form. It turns out that such a convertibility test is only necessary when a dependent base type results or when an abstraction or a projection is type-checked. This yields a type checking algorithm that can be turned directly into an abstract machine. Local bindings cause a problem during type checking because only the result of the substitution given by a binding may be well-formed. This is the reason why in the $\lambda\sigma$-calculus the typing of a substitution is abandoned in favour of noting only the number of terms in an environment. On the other hand the typing of environments is well suited for dependent types. Therefore the above algorithm retains the notion of a type of a context morphism and mimics the substitutions necessary for handling local bindings.

The earliest implementation by Coquand and Huet [CH85] of the Calculus of Constructions represents the syntax of the calculus as a tree with bound variables coded by their de Bruijn-index and defines a parser, a pretty-printer and a type checker based on this representation. The theorem prover Coq [DFH$^+$91] is based on the so-called constructive engine by Huet [Hue89]. This engine implements the operations on proofs as manipulations of the corresponding λ-expressions. One of its basic instructions is the type-checking of a given term. Coq uses a normal-order strategy for the reduction during type checking. The LEGO theorem prover written by Pollack [Tay89] uses Luo's version of the Calculus of Constructions. Harper and Pollack [HP91] give an algorithm for this, which is the basis for type checking in LEGO. De Bruijn-indices are used, and expansion of definitions is delayed as much as possible during type checking. The reduction of λ-expressions is done according to a normal-order strategy, as in Coq. In addition, LEGO implements a sharing mechanism between the substituted term and the original pattern, which uses the exceptions of SML. In this way the re-evaluation of an argument is avoided.

1.2 Summary of the Thesis

Chapter 2

We review the Calculus of Constructions and present a formulation using de Bruijn-numbers instead of variables. This version makes it easier to establish a direct correspondence between the calculus and Ehrhard's D-categories and the categorical combinators derived from them. Furthermore we justify why this categorical structure is the only sensible choice for the derivation of combinators.

Chapter 3

This chapter discusses reduction rules and reduction strategies for the categorical combinators. We present a strongly normalizing and confluent notion of reduction that reduces a combinator first to one corresponding to a weak head-normal form in the calculus and only then pushes substitution under binding operations like λ and Π to reach the normal form. The confluence makes type information in the combinators for an application redundant, so we can simplify the combinators accordingly. Finally we discuss reduction strategies corresponding to eager and lazy reduction in the calculus.

Chapter 4

We demonstrate how the eager and lazy reduction strategies lead directly to abstract machines that generalize the CAM and Krivine's machine respectively. The correctness proof of the machine is an induction over the definition of the reduction strategies. All the hard work in establishing the properties of the reduction strategies has already been done in the previous chapter.

Chapter 5

We explain why type checking of combinators involves reduction and present an algorithm for it. The ideas presented in the previous chapter are applied to turn this algorithm into an abstract machine for type checking that uses the reduction machines described earlier.

Chapter 6

We describe an implementation of the abstract machines in ML and compare it with LEGO and Coq.

Chapter 7

We summarize what has been achieved and mention directions for further work.

Chapter 2

Categorical Combinators

This chapter presents categorical combinators for the Calculus of Constructions. It starts with the definition of the syntax, first with variables and then with de Bruijn numbers. The latter version is more suitable for establishing the correspondence between the syntax and the category theory. Next, we describe the categorical structure used for the derivation of the combinators. This is done in several stages corresponding to the steps leading from the simply typed λ-calculus to a calculus with dependent types and finally to the Calculus of Constructions. Afterwards we derive categorical combinators and establish the correspondence between the calculus and the categorical structure. Finally we compare them to the combinators Ehrhard gives in his thesis [Ehr88b].

2.1 The Calculus of Constructions

There are several versions of the Calculus of Constructions in the literature. We choose here a version with explicitly typed application and equations defined with respect to contexts, similar to [HP89] [Str89]. This has the advantage that a categorical semantics of the calculus can be given directly by an induction over the structure of the derivation. The definition of such a semantics for a version with equality defined on raw expressions and with untyped application [Luo90] [CH88] is only possible if the reduction of a term does not change its type and if the missing type information in an application can be deduced uniquely. The proof of these properties relies on the confluence of such a version. It can be surprisingly difficult to prove the latter, for example the confluence for the version with η-reduction has been shown only recently [Geu92]. Because these properties imply the equivalence of the two versions, the second one may be regarded as a convenient abbreviation for the first with respect to the categorical semantics.

2.1.1 The Syntax of the Calculus

We present first the definition of the calculus with variables. There are three kinds of raw expressions, which are defined as follows:

Definition 2.1 (Raw expressions) *The set of raw types E, of raw terms t and of raw contexts Γ are defined by the following BNF-expressions, where x denotes an element of an infinite set of variables:*

$$\begin{array}{rcl} \Gamma & ::= & [\,] \mid (\Gamma, x{:}\,E) \\ E & ::= & \Pi x{:}\,E.E \mid \Sigma x{:}\,E.E \mid \mathsf{Prop} \mid \mathsf{Proof}(t) \mid \\ t & ::= & x \mid \lambda x{:}\,E.t \mid \mathsf{App}(x.E, E, t, t) \mid \forall x{:}\,E.t \mid \\ & & \mid \mathsf{Pair}(x.E, E, t, t) \mid \pi_1(t) \mid \pi_2(t) \end{array}$$

We identify terms which are equivalent under α-conversion (the binding operations being Π, λ, $\forall$, App and Pair). Moreover, it is assumed that the variable x in $(\Gamma, x{:}\,E)$ is distinct from all the variables occurring in Γ. The length $|\Gamma|$ of a context Γ is defined inductively as follows:

$$\begin{array}{rcl} |[\,]| & = & 0 \\ |(\Gamma, x{:}\,E)| & = & |\Gamma| + 1 \end{array}$$

The substitution of a term t for a variable x in a raw expression e is denoted by $e[x\backslash t]$ and is defined in the usual way. The term $e[x_i\backslash t_i]$ denotes the result of the simultaneous substitution of t_i for x_i in e.

The following kinds of judgements are used in the type theory:

$\vdash \Gamma$ ctxt	Γ is a valid context
$\Gamma = \Gamma'$	Γ and Γ' are equal contexts
$\Gamma \vdash A$ type	A is a well-formed type in context Γ
$\Gamma \vdash A = B$	A and B are equal types in context Γ
$\Gamma \vdash t{:}\,A$	t has type A in context Γ
$\Gamma \vdash t = s{:}\,A$	t and s are equal terms in context Γ

The rules for valid judgements must be defined in one huge inductive definition because the mutual dependencies between types, terms and context do not allow to define well-formed contexts, types and terms separately. However, one can split the definition into several parts, namely the parts dealing with contexts, general rules concerning judgements, rules concerning the dependent product and the rules for propositions.

The first part of the rules is concerned with the formation of contexts and variables:

$$\begin{array}{lll} \text{Empty} & \dfrac{}{\vdash [\,]\ \text{ctxt}} & \\[2ex] \text{Cont} - \text{Intro} & \dfrac{\Gamma \vdash A\ \text{type}}{\vdash (\Gamma, x{:}\,A)\ \text{ctxt}} & (x \notin \mathsf{FV}(\Gamma)) \\[2ex] \text{Var} & \dfrac{\vdash (\Gamma, x{:}\,A, \Gamma')\ \text{ctxt}}{(\Gamma, x{:}\,A, \Gamma') \vdash x{:}\,A} & \end{array}$$

The next part lists the usual rules for equality:

$$\text{Cequ} \quad \frac{\vdash (\Gamma, x{:}\, A, \Gamma')\ \text{ctxt} \qquad \Gamma \vdash A = B}{(\Gamma, x{:}\, A, \Gamma') = (\Gamma, x{:}\, B, \Gamma')}$$

$$\text{Refl} \quad \frac{\vdash \Gamma\ \text{ctxt}}{\Gamma = \Gamma} \quad \frac{\Gamma \vdash A\ \text{type}}{\Gamma \vdash A = A} \quad \frac{\Gamma \vdash t{:}\, A}{\Gamma \vdash t = t{:}\, A}$$

$$\text{Symm} \quad \frac{\Gamma = \Gamma'}{\Gamma' = \Gamma} \quad \frac{\Gamma \vdash A = B}{\Gamma \vdash B = A} \quad \frac{\Gamma \vdash t = s{:}\, A}{\Gamma \vdash s = t{:}\, A}$$

$$\text{Trans} \quad \frac{\Gamma = \Gamma' \qquad \Gamma' = \Gamma''}{\Gamma = \Gamma''} \quad \frac{\Gamma \vdash A = B \qquad \Gamma \vdash B = C}{\Gamma \vdash A = C}$$

$$\frac{\Gamma \vdash t = t'{:}\, A \qquad \Gamma \vdash t' = t''{:}\, A}{\Gamma \vdash t = t''{:}\, A}$$

$$\text{Conv} \quad \frac{\Gamma \vdash A = B \qquad \Gamma \vdash t{:}\, A}{\Gamma \vdash t{:}\, B} \quad \frac{\Gamma \vdash A = B \qquad \Gamma \vdash t = s{:}\, A}{\Gamma \vdash t = s{:}\, B}$$

The third part defines the rules for the dependent product:

$$\Pi - \text{form} \quad \frac{(\Gamma, x{:}\, A) \vdash B\ \text{type}}{\Gamma \vdash \Pi x{:}\, A.B\ \text{type}}$$

$$\Pi - \text{Equ} \quad \frac{\Gamma \vdash A = A' \qquad (\Gamma, x{:}\, A) \vdash B = B'}{\Gamma \vdash \Pi x{:}\, A.B = \Pi x{:}\, A'.B'}$$

$$\Pi - \text{Intro} \quad \frac{(\Gamma, x{:}\, A) \vdash t{:}\, B}{\Gamma \vdash (\lambda x{:}\, A.t){:}\, \Pi x{:}\, A.B}$$

$$\xi - \text{rule} \quad \frac{(\Gamma, x{:}\, A) \vdash t = t'{:}\, B \qquad \Gamma \vdash A = A'}{\Gamma \vdash \lambda x{:}\, A.t = \lambda x{:}\, A'.t'{:}\, \Pi x{:}\, A.B}$$

$$\Pi - \text{elim} \quad \frac{\Gamma \vdash t{:}\, \Pi x{:}\, A.B \qquad \Gamma \vdash s{:}\, A \qquad (\Gamma, x{:}\, A) \vdash B\ \text{type}}{\Gamma \vdash \mathsf{App}(x.A, B, t, s){:}\, B[x\backslash s]}$$

$$\Pi - \text{elimequ1} \quad \frac{\Gamma \vdash A = A' \qquad (\Gamma, x{:}\, A) \vdash B = B'}{\Gamma \vdash \mathsf{App}(x{:}\, A, B, t, s) = \mathsf{App}(x.A', B', t, s){:}\, B'[x\backslash s]}$$

$$\Pi - \text{elimequ2} \quad \frac{\Gamma \vdash t = t'{:}\, \Pi x{:}\, A.B \qquad \Gamma \vdash s = s'{:}\, A}{\Gamma \vdash \mathsf{App}(x.A, B, t, s) = \mathsf{App}(x{:}\, A, B, t', s'){:}\, B[x\backslash s]}$$

$$\beta - \text{rule} \quad \frac{(\Gamma, x{:}\, A) \vdash t{:}\, B \qquad \Gamma \vdash s{:}\, A \qquad (\Gamma, x{:}\, A) \vdash B\ \text{type}}{\Gamma \vdash \mathsf{App}(x.A, B, \lambda x{:}\, A.t, s) = t[x\backslash s]{:}\, B[x\backslash s]}$$

$$\eta - \text{rule} \quad \frac{\Gamma \vdash t{:}\, \Pi x{:}\, A.B \qquad (\Gamma, x{:}\, A) \vdash B\ \text{type}}{\Gamma \vdash \lambda x{:}\, A.\mathsf{App}(x.A, B, t, x) = t{:}\, \Pi x{:}\, A.B} \ (x \text{ not free in } t)$$

The fourth part defines the rules for the dependent sums:

$$\Sigma - \text{form} \quad \frac{(\Gamma, x{:}\, A) \vdash B\ \text{type}}{\Gamma \vdash \Sigma x{:}\, A.B\ \text{type}}$$

$$\Sigma - \mathrm{Equ} \qquad \frac{\Gamma \vdash A = A' \qquad (\Gamma, x\colon A) \vdash B = B'}{\Gamma \vdash \Sigma x\colon A.B = \Sigma x\colon A'.B'}$$

$$\Sigma - \mathrm{Intro} \qquad \frac{\Gamma \vdash t_1\colon A \qquad (\Gamma, x\colon A) \vdash B \text{ type} \qquad \Gamma \vdash t_2\colon B[x\backslash t_1]}{\Gamma \vdash \mathsf{Pair}(x.A, B, t_1, t_2)\colon \Sigma x\colon A.B}$$

$$\Sigma - \mathrm{Introequ1} \qquad \frac{\Gamma \vdash A = A' \qquad (\Gamma, x\colon A) \vdash B = B'}{\Gamma \vdash \mathsf{Pair}(x.A, B, t, s) = \mathsf{Pair}(x.A', B', t, s)\colon \Sigma x\colon A.B}$$

$$\Sigma - \mathrm{Introequ2} \qquad \frac{\Gamma \vdash t = t'\colon A \qquad \Gamma \vdash s = s'\colon B[x\backslash t]}{\Gamma \vdash \mathsf{Pair}(x.A, B, t, s) = \mathsf{Pair}(x.A', B', t', s')\colon \Sigma x\colon A.B}$$

$$\Sigma - \mathrm{Elim1} \qquad \frac{\Gamma \vdash t\colon \Sigma x\colon A.B}{\Gamma \vdash \pi_1(t)\colon A}$$

$$\Sigma - \mathrm{Elim2} \qquad \frac{\Gamma \vdash t\colon \Sigma x\colon A.B}{\Gamma \vdash \pi_2(t)\colon B[x\backslash \pi_1(t)]}$$

$$\Sigma - \mathrm{Elimequ1} \qquad \frac{\Gamma \vdash t_1 = t_2\colon \Sigma x\colon A.B}{\Gamma \vdash \pi_1(t_1) = \pi_1(t_2)\colon A}$$

$$\Sigma - \mathrm{Elimequ2} \qquad \frac{\Gamma \vdash t_1 = t_2\colon \Sigma x\colon A.B}{\Gamma \vdash \pi_2(t_1) = \pi_2(t_2)\colon B[x\backslash \pi_1(t_1)]}$$

$$(\sigma_1) \qquad \frac{\Gamma \vdash \mathsf{Pair}(x.A, B, t, s)\colon \Sigma x\colon A.B}{\Gamma \vdash \pi_1(\mathsf{Pair}(x.A, B, t, s)) = t\colon A}$$

$$(\sigma_2) \qquad \frac{\Gamma \vdash \mathsf{Pair}(x.A, B, t, s)\colon \Sigma x\colon A.B}{\Gamma \vdash \pi_2(\mathsf{Pair}(x.A, B, t, s)) = s\colon B[x\backslash t]}$$

$$(surj) \qquad \frac{\Gamma \vdash t\colon \Sigma x\colon A.B}{\Gamma \vdash \mathsf{Pair}(x.A, B, \pi_1(t), \pi_2(t)) = t}$$

The last part gives the rules for propositions:

$$\mathsf{Prop} \qquad \frac{\vdash \Gamma \text{ ctxt}}{\Gamma \vdash \mathsf{Prop} \text{ type}}$$

$$\mathsf{Proof} \qquad \frac{\Gamma \vdash p\colon \mathsf{Prop}}{\Gamma \vdash \mathsf{Proof}(p) \text{ type}}$$

$$\mathsf{Prop} - \mathrm{equ} \qquad \frac{\Gamma \vdash p = p'\colon \mathsf{Prop}}{\Gamma \vdash \mathsf{Proof}(p) = \mathsf{Proof}(p')}$$

$$\forall - \mathrm{Intro} \qquad \frac{(\Gamma, x\colon A) \vdash p\colon \mathsf{Prop}}{\Gamma \vdash \forall x\colon A.p\colon \mathsf{Prop}}$$

$$\forall - \mathrm{equ} \qquad \frac{(\Gamma, x\colon A) \vdash p = p'\colon \mathsf{Prop} \qquad \Gamma \vdash A = A'}{\Gamma \vdash \forall x\colon A.p = \forall x\colon A'.p'\colon \mathsf{Prop}}$$

$$\forall - \mathrm{elim} \qquad \frac{(\Gamma, x\colon A) \vdash p\colon \mathsf{Prop}}{\Gamma \vdash \mathsf{Proof}(\forall x\colon A.p) = \Pi x\colon A.\mathsf{Proof}(p)}$$

This completes the definition of the Calculus of Constructions.

Remark The properties of weakening and substitution are given by the following judgements, in which $\Gamma \Rightarrow J$ is an abbreviation for either $\vdash \Gamma$ ctxt, $\Gamma = \Gamma'$, $\Gamma \vdash A$ type, $\Gamma \vdash A = B$, $\Gamma \vdash t: A$ or $\Gamma \vdash t = s$.

$$\text{Thin} \quad \frac{(\Gamma, \Gamma') \Rightarrow J \qquad \Gamma \vdash A \text{ type}}{(\Gamma, x: A, \Gamma') \Rightarrow J} \quad (x \notin \mathsf{FV}(\Gamma) \cup \mathsf{FV}(\Gamma'))$$

$$\text{Sub} \quad \frac{(\Gamma, x: A, \Gamma') \Rightarrow J \qquad \Gamma \vdash t: A}{(\Gamma, \Gamma'[x\backslash t]) \vdash J[x\backslash t]} \quad (x \text{ free for } t \text{ in } \Gamma')$$

$$\text{Sub} - \text{Equ} \quad \frac{\Gamma \vdash t = t': A \qquad (\Gamma, x: A, \Gamma') \Rightarrow J}{(\Gamma, \Gamma'[x\backslash t]) \vdash J[x\backslash t] = J[x\backslash t']} \quad (x \text{ free for } t \text{ in } \Gamma')$$

The judgements can be derived from the other ones by an induction over the derivations which is omitted here.

2.1.2 de Bruijn Numbers

As in the case of the simply typed λ-calculus [Cur86] and in the second-order λ-calculus [CE87] the relation between the syntax and the categorical combinators is easier to establish if variables are replaced by de Bruijn numbers. This applies also to the Calculus of Constructions, as section 2.3 shows. At the beginning of this subsection we define the raw expressions of the Calculus of Constructions in de Bruijn form and give a translation of the raw expressions of the ordinary calculus into the latter. Afterwards we give an adapted version of the rules concerning well-formedness.

The raw expressions are as follows:

Definition 2.2 (Raw types and terms, raw contexts) *The set of raw types E, the set of raw terms t and the set of raw contexts Γ of the Calculus of Constructions in de Bruijn form are defined by the following BNF-expressions:*

$$\begin{array}{rcl} \Gamma & ::= & [\,] \mid (\Gamma, E) \\ E & ::= & \Pi E.E \mid \Sigma x: E.E \mid \mathsf{Prop} \mid \mathsf{Proof}(t) \\ t & ::= & n \mid \lambda E.t \mid \mathsf{App}(E, E, t, t) \mid \forall E.t \mid \mathsf{Pair}(E, E, t, t) \mid \pi_1(t) \mid \pi_2(t) \end{array}$$

Furthermore, the length $|\Gamma|$ of a context Γ is defined inductively as follows:

$$\begin{array}{rcl} |[\,]| & = & 0 \\ |(\Gamma, E)| & = & |\Gamma| + 1 \end{array}$$

The translation of raw expressions of the calculus with variables into the calculus with de Bruijn numbers is given next. It is only possible if raw expressions in raw contexts are considered, because a variable x is translated to the number k indicating the position of the variable in the context $\Gamma = (x_{n-1}: A_{n-1}, \ldots, x_0: A_0)$. The definition is as follows:

Definition 2.3 *The* translation $\mathsf{db}(\Gamma, e)$ *of a raw expression with variables to a raw expression in de Bruijn form with respect to a context* $\Gamma = (x_{n-1}\colon A_{n-1}, \ldots, x_0\colon A_0)$ *is defined by induction over the structure of the raw expression* e *as follows:*

(i) *on contexts:*

$$\begin{array}{rcl}\mathsf{db}([\,]) & = & [\,]\\ \mathsf{db}((\Gamma, x\colon E)) & = & (\mathsf{db}(\Gamma), \mathsf{db}(\Gamma, E))\end{array}$$

(ii) *on types:*

$$\begin{array}{rcl}\mathsf{db}(\Gamma, \Pi x\colon A.B) & = & \Pi\mathsf{db}(\Gamma, A).\mathsf{db}((\Gamma, x\colon A), B))\\ \mathsf{db}(\Gamma, \Sigma x\colon A.B) & = & \Sigma\mathsf{db}(\Gamma, A).\mathsf{db}((\Gamma, x\colon A), B))\\ \mathsf{db}(\Gamma, \mathsf{Prop}) & = & \mathsf{Prop}\\ \mathsf{db}(\Gamma, \mathsf{Proof}(t)) & = & \mathsf{Proof}(\mathsf{db}(\Gamma, t))\end{array}$$

(iii) *on terms*

$$\begin{array}{rcl}\mathsf{db}(\Gamma, x_k) & = & k\ (0 \leq k \leq n-1)\\ \mathsf{db}(\Gamma, \lambda x\colon A.t) & = & \lambda\mathsf{db}(\Gamma, A).\mathsf{db}((\Gamma, x\colon A), t)\\ \mathsf{db}(\Gamma, \mathsf{App}(x.A, B, s, t)) & = & \mathsf{App}(\mathsf{db}(\Gamma, A), \mathsf{db}((\Gamma, x\colon A), B), \mathsf{db}(\Gamma, s), \mathsf{db}(\Gamma, t))\\ \mathsf{db}(\Gamma, \forall x\colon A.t) & = & \forall\mathsf{db}((\Gamma, A).\mathsf{db}((\Gamma, x\colon A), t)\\ \mathsf{db}(\Gamma, \mathsf{Pair}(x.A, B, t, s)) & = & \mathsf{Pair}(\mathsf{db}(\Gamma, A), \mathsf{db}((\Gamma, x\colon A), B), \mathsf{db}(\Gamma, t), \mathsf{db}(\Gamma, s))\\ \mathsf{db}(\Gamma, \pi_1(t)) & = & \pi_1(\mathsf{db}(\Gamma, t))\\ \mathsf{db}(\Gamma, \pi_2(t)) & = & \pi_2(\mathsf{db}(\Gamma, t))\end{array}$$

The operations of weakening and substitution in de Bruijn form, which are to be defined next, are more complex than those in the previous case and will therefore be given explicitly. The intended meaning of the weakening operation U_i^m is that if for types A and $A_0, \ldots, A_{m-1}$ and contexts Γ and Γ' with $|\Gamma'| = i$

$$\begin{array}{rcl}(\Gamma, \Gamma') & \vdash & A \text{ type}\\ (\Gamma, A_{m-1}, \ldots, A_{j+1}) & \vdash & A_j \text{ type } (0 \leq j \leq m-1)\end{array}$$

are valid judgements, then the judgement

$$(\Gamma, A_{m-1}, \ldots, A_0, \mathrm{U}_i^m(\Gamma')) \vdash \mathrm{U}_i^m(A)$$

is valid. A similar claim applies for a term t instead of a type A.

Definition 2.4 (Weakening) *The weakening of pure types, terms and contexts is given by the operation* U_i^m*, which is defined as follows:*

(i) *On contexts:*

$$\begin{array}{rcll}
\mathsf{U}_i^m([\,]) & = & [\,] & \\
\mathsf{U}_i^m((\Gamma, E)) & = & (\mathsf{U}_{i-1}^m(\Gamma), \mathsf{U}_i^m(E)) & (i > 0) \\
\mathsf{U}_i^m(\Gamma) & = & \Gamma & (i = 0)
\end{array}$$

(ii) *On types:*

$$\begin{array}{rcl}
\mathsf{U}_i^m(\Pi A.B) & = & \Pi \mathsf{U}_i^m(A).\mathsf{U}_{i+1}^m(B) \\
\mathsf{U}_i^m(\Sigma A.B) & = & \Sigma \mathsf{U}_i^m(A).\mathsf{U}_{i+1}^m(B) \\
\mathsf{U}_i^m(\mathsf{Prop}) & = & \mathsf{Prop} \\
\mathsf{U}_i^m(\mathsf{Proof}(t)) & = & \mathsf{Proof}(\mathsf{U}_i^m(t))
\end{array}$$

(iii) *On terms:*

$$\begin{array}{rcl}
\mathsf{U}_i^m(k) & = & \begin{cases} k & k < i \\ k + m & k \geq i \end{cases} \\
\mathsf{U}_i^m(\lambda E.t) & = & \lambda \mathsf{U}_i^m(E).\mathsf{U}_{i+1}^m(t) \\
\mathsf{U}_i^m(\mathsf{App}(A, B, t, s)) & = & \mathsf{App}(\mathsf{U}_i^m(A), \mathsf{U}_{i+1}^m(B), \mathsf{U}_i^m(t), \mathsf{U}_i^m(s)) \\
\mathsf{U}_i^m(\forall A.t) & = & \forall \mathsf{U}_i^m(E).\mathsf{U}_{i+1}^m(t) \\
\mathsf{U}_i^m(\mathsf{Pair}(A, B, t, s)) & = & \mathsf{Pair}(\mathsf{U}_i^m(A), \mathsf{U}_{i+1}^m(B), \mathsf{U}_i^m(t), \mathsf{U}_i^m(s)) \\
\mathsf{U}_i^m(\pi_1(t)) & = & \pi_1(\mathsf{U}_i^m(t)) \\
\mathsf{U}_i^m(\pi_2(t)) & = & \pi_2(\mathsf{U}_i^m(t))
\end{array}$$

In the sequel we will abbreviate $\mathsf{U}_0^1(e)$ *by* $(e)\uparrow$.

The second definition concerns the substitution of a term s for the nth variable in an expression e, which is denoted by $e[n\backslash s]$. The precise definition is as follows:

Definition 2.5 (Substitution)

(i) *In contexts*

$$\begin{array}{rcll}
[\,][n\backslash s] & = & [\,] & \\
(\Gamma, E)[n\backslash s] & = & (\Gamma[n-1\backslash s], E[n\backslash s]) & (n > 0) \\
\Gamma[n\backslash s] & = & \Gamma & (n = 0)
\end{array}$$

(ii) *In types*

$$\begin{array}{rcl}
(\Pi A.B)[n\backslash s] & = & \Pi A[n\backslash s].B[n+1\backslash s] \\
(\Sigma A.B)[n\backslash s] & = & \Sigma A[n\backslash s].B[n+1\backslash s] \\
\mathsf{Prop}[n\backslash s] & = & \mathsf{Prop} \\
\mathsf{Proof}(t)[n\backslash s] & = & \mathsf{Proof}(t[n\backslash s])
\end{array}$$

(iii) *In terms*

$$\begin{aligned}
k[n\backslash s] &= \begin{cases} k & k<n \\ \mathsf{U}_0^n(s) & k=n \\ k-1 & k>n \end{cases} \\
(\lambda A.t)[n\backslash s] &= \lambda A[n\backslash s].t[n+1\backslash s] \\
\mathsf{App}(A,B,s_1,s_2)[n\backslash s] &= \mathsf{App}(A[n\backslash s],B[n+1\backslash s],s_1[n\backslash s],s_2[n\backslash s]) \\
(\forall A.t)[n\backslash s] &= \forall A[n\backslash s].t[n+1\backslash s] \\
\mathsf{Pair}(A,B,t_1,t_2)[n\backslash s] &= \mathsf{Pair}(A[n\backslash s],B[n+1\backslash s],t_1[n\backslash s],t_2[n\backslash s]) \\
\pi_1(t)[n\backslash s] &= \pi_1(t[n\backslash s]) \\
\pi_2(t)[n\backslash s] &= \pi_2(t[n\backslash s])
\end{aligned}$$

The adaptation of the rules describing well-formedness is given here only for the rules involving weakening and substitution, because all other rules are merely rewritten. However, the complete set of rules can be found in the appendix. The adapted rules look as follows:

1. Variable rule

$$\text{Var}\quad \frac{\vdash (\Gamma,A,\Gamma')\ \text{ctxt}}{(\Gamma,A,\Gamma')\vdash |\Gamma'|\colon \mathsf{U}_0^{|\Gamma'|+1}(A)}$$

3. Rules for dependent product

$$\beta-\text{rule}\quad \frac{(\Gamma,A)\vdash t\colon B \qquad \Gamma\vdash s\colon A \qquad (\Gamma,A)\vdash B\ \text{type}}{\Gamma\vdash \mathsf{App}(A,B,t,s)=t[0\backslash s]\colon B[0\backslash s]}$$

$$\eta-\text{rule}\quad \frac{\Gamma\vdash t\colon \Pi A.B) \qquad (\Gamma,A)\vdash B\ \text{type}}{\Gamma\vdash \lambda A.\mathsf{App}((A)\uparrow,\mathsf{U}_1^1(B),(t)\uparrow,0)=t\colon \Pi A.B}$$

The rules for weakening and substitution are as follows:

$$\text{Thin}\quad \frac{(\Gamma,\Gamma')\Rightarrow J \qquad \Gamma\vdash A\ \text{type}}{(\Gamma,A,\mathsf{U}_{|\Gamma'|}^1(\Gamma'))\Rightarrow \mathsf{U}_{|\Gamma'|+1}^1(J)}$$

$$\text{Sub}\quad \frac{(\Gamma,A,\Gamma')\Rightarrow J \qquad \Gamma\vdash t\colon A}{(\Gamma,\Gamma'[|\Gamma'|\backslash t])\vdash J[|\Gamma'|\backslash t]}$$

$$\text{Sub}-\text{Equ}\quad \frac{(\Gamma,A,\Gamma')\Rightarrow J \qquad \Gamma\vdash t=t'\colon A}{(\Gamma,\Gamma'[|\Gamma'|\backslash t])\vdash J[|\Gamma'|\backslash t]=J[|\Gamma'|\backslash t']}$$

2.2 The CC-Category

As already mentioned in the introduction, we propose split D-categories as an appropriate categorical framework for abstract machines. Curien and Ehrhard describe those categories in [Cur89] as an extension of Cartmell's categories with attributes

[Car86]. Ehrhard considers in his thesis D-categories, which are based on fibrations rather than on indexed categories. He shows that an extra condition, the so-called fullness condition, is necessary to show the equivalence between the category of display maps [Tay86] [HP89] and D-categories. This condition ensures that the morphisms in the fibres are uniquely determined by their global sections. It is applied here to establish the correspondence between the categorical combinators and the calculus. As mentioned in Jacob's thesis [Jac91], it is also necessary to describe strong sums as left adjoints to weakening.

We define the higher-order full split D-categories, which we will call CC-categories for short, in three steps. First, we explain those parts that correspond to the simply typed λ-calculus over a set of ground types, then we add the properties required for modelling dependent types, and finally we extend this framework to the Calculus of Constructions.

2.2.1 Definition and Connection to the Syntax

The simply typed λ-calculus

We assume familiarity with basic concepts of category theory, especially with indexed categories and fibrations. An introduction into these topics can be found in [Mac71] [BW90]. The so-called *Grothendieck construction* plays an important role in the definition of a D-category below. It takes an indexed category $E\colon \mathcal{B}^{op} \rightarrow \mathbf{Cat}$ and produces a fibration, which is denoted by $Gr(E) \xrightarrow{p} \mathcal{B}$. The objects of $Gr(E)$ are pairs (Γ, A), where Γ is an object of $\mathcal{B}$ and A is an object of $E(\Gamma)$. A morphism from (Γ, A) to (Δ, B) is a pair (f, t) of morphisms with f a morphism from Γ to Δ in $\mathcal{B}$ and t a morphism from A to $E(f)(B)$ in $E(\Gamma)$. The functor p is the projection to the first component, which maps an object (Γ, A) in $Gr(E)$ to Γ and every morphism (f, t) to f.

The part of the categorical structure, the so-called *constant split D-category*, that models the simply typed λ-calculus with a set of ground types $\mathcal{G}$ is described first. The framework is given by a certain indexed category:

- A category $\mathcal{B}$ with a terminal object $[\,]$. The morphism from any object Γ to $[\,]$ is denoted by $\langle\rangle$.
- An indexed category $E\colon \mathcal{B}^{op} \rightarrow \mathbf{Cat}$. For any morphism f in $\mathcal{B}$ we will write $f^*(-)$ for $E(f)(-)$.
- For every object Γ in $\mathcal{B}$ the fibre $E(\Gamma)$ has a terminal object 1 that is preserved on the nose by every functor f^*, e.g. these terminal objects are chosen in such a way that $f^*(1) = 1$.
- For every object Γ in $\mathcal{B}$ the category $E(\Gamma)$ has the same set $\mathcal{T}$ of objects with $f^*(A) = A$ for every morphism f in $\mathcal{B}$ and element A of $\mathcal{T}$.

The connection to the syntax is as follows:

Category	Syntax
Object Γ of the base category $\mathcal{B}$	Context Γ
Terminal object $[\,]$ in the base	Empty context
Morphism in the base	Environment, i.e. list of terms
Object A in the fibre over Γ	Type A
Morphism $t: 1 \rightarrow A \in E(\Gamma)$	Term of type A in context Γ
"Pulling back" of a morphism t in $E(\Gamma)$ along $f: \Gamma \rightarrow \Gamma'$, i.e. $f^*(t)$	Substitution of f in t (see below)

Note that the morphisms in the base category have no counterpart in the calculus. They correspond to an environment, i.e. a list of terms, and show how to interpret it as a function sending one context to another. These morphisms are therefore called *context morphisms.*

The formation of new contexts is captured by the following adjunction:

- The functor $I: \mathcal{B} \rightarrow Gr(E)$, given by

$$\begin{aligned} I(\Gamma) &= (\Gamma, 1) \\ I(f) &= (f, \mathsf{Id}) \end{aligned}$$

  has a right adjoint $G: Gr(E) \rightarrow \mathcal{B}$. The object $G((\Gamma, A))$ is abbreviated $\Gamma \cdot A$ in the sequel. Furthermore $(\mathsf{Fst}, \mathsf{Snd}): (\Gamma \cdot A, 1) \rightarrow (\Gamma, A)$ denotes the counit of this adjunction. The natural isomorphism between $\mathrm{Hom}_{Gr(E)}((-,1),(-,A))$ and $\mathrm{Hom}_{\mathcal{B}}(-, - \cdot A)$ is denoted by $\langle -, - \rangle$.

The functor G models the formation of new contexts from old ones, i.e. $\Gamma \cdot A$ corresponds to the context (Γ, A). This adjunction is also used for the modelling of substitution. If t is a term of type A in context Γ and u is a term of type B in context (Γ, A), then the morphism $\langle \mathsf{Id}, t \rangle^* u: 1 \rightarrow B$ in $E(\Gamma)$ corresponds to the substitution of the variable with de Bruijn-number 0 in the term u by t. The weakening operation, which is also necessary for the definition of substitution (cf. section 2.1), is also modelled by the adjunction $I \dashv G$. More precisely, it corresponds to the functor $\mathsf{Fst}_A^*: E(\Gamma) \rightarrow E(\Gamma \cdot A)$. We will abbreviate $G((f, t))$ to $f \cdot t$.

As already mentioned, we need the so-called *fullness condition:*

- For every pair of morphisms $g: \Gamma \cdot A \rightarrow \Delta \cdot B$ and $f: \Gamma \rightarrow \Delta$ such that the diagram

$$\begin{tikzcd} \Gamma \cdot A \arrow[r, "g"] \arrow[d, "\mathsf{Fst}"'] & \Delta \cdot B \arrow[d, "\mathsf{Fst}"] \\ \Gamma \arrow[r, "f"'] & \Delta \end{tikzcd}$$

  commutes, there exists a unique morphism $t: A \rightarrow B$ in $E(\Gamma)$ such that $g = f \cdot t$.

The intuition behind this condition becomes clear if we reformulate it. Because every morphism $(f, t): (\Gamma, A) \rightarrow (\Delta, B)$ in $Gr(E)$ is equal to $(\mathsf{Id}, t); (f, \mathsf{Id})$, it is enough to require fullness only in the special case $f = \mathsf{Id}$. In this case it means that any morphism $t: A \to B$ in $E(\Gamma)$ corresponds uniquely to the morphism $t' = \mathsf{Snd}; \mathsf{Fst}^* t: 1 \to B$ in $E(\Gamma \cdot A)$. It therefore shows how to translate any morphism $t: A \to B$ in $E(\Gamma)$ in a unique way to a term of type B in context (Γ, A).

Function types and abstraction are modelled by the right adjoint Π_A to the weakening functor Fst_A^* plus Beck-Chevalley condition, i.e. the requirement that

$$f^*(\mathsf{Cur}_A(t)) \quad = \quad \mathsf{Cur}_A((f \cdot \mathsf{Id})^*(t))$$

holds for every $f: \Delta \to \Gamma$, $A \in E(\Gamma)$, $B \in E(\Gamma \cdot A)$. In these equations, Cur denotes the natural isomorphism between $\mathrm{Hom}_{E(\Gamma \cdot A)}(B, C)$ and $\mathrm{Hom}_{E(\Gamma)}(B, \Pi_A(C))$. See next subsection for the reasons why the strict version of the Beck-Chevalley-condition is used. This modelling of abstraction and functions by a right adjoint to weakening is common in categorical logic. It is also used in cartesian closed categories [LS85], and works also for generalizations like the universal quantification in the polymorphic λ-calculus[See87].

Products in the λ-calculus are modelled as a special case of a dependent sum:

- For every object $A \in E(\Gamma)$ and $B \in E(\Gamma \cdot A)$ there exists an object $\Sigma(A, B)$ in $E(\Gamma)$ such that $\Gamma \cdot A \cdot B$ and $\Gamma \cdot \Sigma(A, B)$ are naturally isomorphic.

If this condition is unravelled, it yields the existence of morphisms $\mathsf{Pair}: 1 \to \Sigma(A, B)$ in $E(\Gamma \cdot A \cdot B)$, $\pi_1: 1 \to A$ in $E(\Gamma \cdot \Sigma(A, B))$ and $\pi_2: 1 \to B$ in $E(\Gamma \cdot \Sigma(A, B))$ such that

$$\begin{aligned}
\langle \mathsf{Fst}, \mathsf{Pair} \rangle; \langle \mathsf{Fst}, \pi_1, \pi_2 \rangle &= \mathsf{Id} \\
\langle \mathsf{Fst}, \pi_1, \pi_2 \rangle; \langle \mathsf{Fst}, \mathsf{Pair} \rangle &= \mathsf{Id} \\
(f \cdot \mathsf{Id} \cdot \mathsf{Id})^* \mathsf{Pair} &= \mathsf{Pair} \\
(f \cdot \mathsf{Id})^* \pi_1 &= \pi_1 \\
(f \cdot \mathsf{Id})^* \pi_2 &= \pi_2
\end{aligned}$$

The morphisms $\langle \langle \mathsf{Id}, t \rangle, s \rangle^* \mathsf{Pair}$, $\langle \mathsf{Id}, t \rangle^* \pi_1$ and $\langle \mathsf{Id}, t \rangle^* \pi_1$ correspond to the product of the terms t and s and the first and second projection of t respectively.

Dependent Types

The categorical structure described for the simply typed λ-calculus can be generalized to interpret a λ-calculus with dependent types if the following changes are made:

- The condition that there exists a fixed set $\mathcal{T}$ of objects for all categories $E(\Gamma)$ is removed.

- The requirement $f^* A = A$ is dropped as well.

This categorical structure is called a *split D-category*. In this setting the exponents become dependent products, and the products turn into strong sums, so we obtain a Martin-Löf theory without equality.

The Calculus of Constructions

The Calculus of Constructions is a special kind of Martin-Löf theory with a type Prop of propositions, a type of all proofs of a proposition and an impredicative universal quantification. The corresponding categorical structure, a so-called *CC-category*, is therefore a generalization of the split D-categories. First we require that

- there exist an object Ω in $E([\,])$ and T in $E([\,]\cdot\Omega)$.

modelling the first two additions respectively. The object T corresponds to the type $\mathsf{Proof}(0)$, so that for any proposition p in context Γ the type $\mathsf{Proof}(p)$ is represented by the object $\langle\langle\rangle,p\rangle^*T$ in $E(\Gamma)$. A dependent product over proofs of propositions is already defined, so in order to model the universal quantification we only have to ensure that for every proposition p, i.e. a morphism $p\colon 1\to\Omega$, there is a proposition corresponding to any object $\Pi(A,p)$ and that this correspondence respects substitution. This leads to the following two conditions:

- For every morphism $t\colon 1\to\langle\rangle^*(\Omega)$ in $E(\Gamma\cdot A)$ there exists a morphism $\forall(A,t)\colon 1\to\langle\rangle^*\Omega$ in $E(\Gamma)$ and the naturality condition
$$h^*\forall(A,t)=\forall(h^*A,(h\cdot\mathsf{Id})^*t)$$
holds for every $h\colon\Gamma'\to\Gamma$.
- For every object A in $E(\Gamma)$ and any morphism $t\colon 1\to\Omega$ in $E(\Gamma\cdot A)$, the following *coherence condition* holds:
$$\langle\langle\rangle,\forall(A,t)\rangle^*(T)=\Pi(A,\langle\langle\rangle,t\rangle^*(T))$$

If we put all parts of the definition together, we obtain the following definition of a CC-category:

Definition 2.6 *Let $\mathcal{B}$ be a category with a terminal object $[\,]$, let $\langle\rangle$ be the morphism from any object Γ of $\mathcal{B}$ to $[\,]$ and let $E\colon\mathcal{B}^{op}\to\mathbf{Cat}$ be an indexed category over $\mathcal{B}$. We will write f^* for $E(f)$. E is a* CC-category *if it satisfies*

(i) *For every object Γ of $\mathcal{B}$ there exists a terminal object 1 in the category $E(\Gamma)$, which is preserved on the nose by every functor f^*, i.e. the terminal objects are chosen in such a way that $f^*(1)=1$.*

(ii) *There exists a right adjoint G to the functor $I\colon\mathcal{B}\to Gr(E)$, where I is defined by*
$$\begin{array}{rcll} I(\Gamma) & = & (\Gamma,1) & (\Gamma\in\mathrm{Obj}(\mathcal{B}))\\ I(f) & = & (f,\mathsf{Id}) & (f\in\mathrm{Hom}(\Gamma,\Gamma')),\end{array}$$
and $Gr(E)$ is the category obtained by applying the Grothendieck construction to E. We abbreviate $G((\Gamma,A))$ to $\Gamma\cdot A$ and $G((f,t))$ to $f\cdot t$. Furthermore $(\mathsf{Fst},\mathsf{Snd})\colon(\Gamma\cdot A,1)\to(\Gamma,A)$ *denotes the counit of this adjunction. The natural isomorphism between* $\mathrm{Hom}_{Gr(E)}((-,1),(-,A))$ *and* $\mathrm{Hom}_{\mathcal{B}}(-,-\cdot A)$ *is denoted by* $\langle -,-\rangle$.

(iii) *For every pair of morphisms* $g\colon \Gamma \cdot A \to \Delta \cdot B$ *and* $f\colon \Gamma \to \Delta$ *such that the diagram*

$$\begin{tikzcd}
\Gamma \cdot A \arrow[r, "g"] \arrow[d, "\mathsf{Fst}"'] & \Delta \cdot B \arrow[d, "\mathsf{Fst}"] \\
\Gamma \arrow[r, "f"'] & \Delta
\end{tikzcd}$$

commutes, there exists a unique morphism $t\colon 1 \to \mathsf{Fst}^* B$ *in* $E(\Gamma \cdot A)$ *such that* $g = f \cdot t$.

(iv) *For every object* Γ *of* $\mathcal{B}$ *and* A *of* $E(\Gamma)$, *the functor*

$$\mathsf{Fst}^*_A\colon E(\Gamma) \to E(\Gamma \cdot A)$$

has a right adjoint

$$\Pi_A\colon E(\Gamma \cdot A) \to E(\Gamma).$$

We will write Cur *for the natural isomorphism between* $\mathrm{Hom}_{E(\Gamma \cdot A)}(\mathsf{Fst}^*_A(B), C)$ *and* $\mathrm{Hom}_{E(\Gamma)}(B, \Pi_A(C))$.

(v) *The Beck-Chevalley-condition for the adjunctions* $\mathsf{Fst}^*_A \vdash \Pi_A$ *is satisfied in the strict sense, i.e. the equations*

$$\begin{aligned}
f^*(\Pi_A(B)) &= \Pi_{f^*(A)}((f \cdot \mathsf{Id})^*(B)) \\
f^*(\mathsf{Cur}_A(t)) &= \mathsf{Cur}_{f^*(A)}((f \cdot \mathsf{Id})^*(t))
\end{aligned}$$

hold for every $f\colon \Delta \to \Gamma$, $A \in E(\Gamma)$, $B \in E(\Gamma \cdot A)$.

(vi) *For every object* $A \in E(\Gamma)$ *and* $B \in E(\Gamma \cdot A)$ *there exists an object* $\Sigma(A, B)$ *in* $E(\Gamma)$ *such that* $\Gamma \cdot A \cdot B$ *and* $\Gamma \cdot \Sigma(A, B)$ *are naturally isomorphic.*

(vii) *There exists an object* Ω *in the category* $E([\,])$ *and an object* T *in the category* $E([\,] \cdot \Omega)$.

(viii) *For every object* Γ *in* $\mathcal{B}$, *every object* A *in* $E(\Gamma)$ *and every morphism* $t\colon 1 \to \Omega$ *in* $E(\Gamma \cdot A)$ *there exists a morphism* $\forall(A, t)\colon 1 \to \Omega$ *in* $E(\Gamma)$ *such that the equations*

$$\begin{aligned}
h^*(\forall(A, t)) &= \forall(h^* A, (h \cdot \mathsf{Id})^*(t)) \\
\langle\langle\rangle, \forall(A, t)\rangle^*(T) &= \Pi_A(\langle\langle\rangle, t\rangle^*(T))
\end{aligned}$$

are satisfied.

The formulation of the combinators relies on some isomorphisms involving the terminal object in the fibre. These are given in the following lemma:

Lemma 2.7 *Let $E : \mathcal{B}^{op} \to \mathbf{Cat}$ be any CC-category. For any object Γ of $\mathcal{B}$ and objects A of $E(\Gamma)$ and B of $E(\Gamma \cdot 1)$ we have with $\langle \mathsf{Id}, \mathsf{Id} \rangle : \Gamma \to \Gamma \cdot 1$ the following isomorphisms:*

(i) $\Gamma \cdot 1 \cong \Gamma$

(ii) $\Pi(1, B) \cong \langle \mathsf{Id}, \mathsf{Id} \rangle^* B$ *and* $\Pi(A, 1) \cong 1$

(iii) $\Sigma(1, B) \cong \langle \mathsf{Id}, \mathsf{Id} \rangle^* B$ *and* $\Sigma(A, 1) \cong A$

Proof

(i) We have

$$\mathsf{Fst}; \langle \mathsf{Id}, \mathsf{Id} \rangle = \langle \mathsf{Fst}, \mathsf{Id} \rangle = \langle \mathsf{Fst}, \mathsf{Snd} \rangle = \mathsf{Id} : \Gamma \cdot 1 \to \Gamma \cdot 1$$

and

$$\langle \mathsf{Id}, \mathsf{Id} \rangle ; \mathsf{Fst} = \mathsf{Id} : \Gamma \to \Gamma$$

.

(ii) There are natural bijections

$$\begin{array}{c} \mathsf{Fst}^* C \longrightarrow B \text{ in } E(\Gamma \cdot 1) \\ \hline\hline C \longrightarrow \langle \mathsf{Id}, \mathsf{Id} \rangle^* B \text{ in } E(\Gamma) \end{array}$$

and

$$\begin{array}{c} \mathsf{Fst}^* C \xrightarrow{!} 1 \text{ in } E(\Gamma \cdot A) \\ \hline\hline C \xrightarrow{!} 1 \text{ in } E(\Gamma) \end{array}$$

so the claim follows from the uniqueness up to isomorphism of $\Pi(1, B)$ and $\Pi(A, 1)$.

(iii) By (i), we have

$$\Gamma \cdot \Sigma(1, B) \cong \Gamma \cdot 1 \cdot B \cong \Gamma \cdot \langle \mathsf{Id}, \mathsf{Id} \rangle^* B$$

and

$$\Gamma \cdot \Sigma(A, 1) \cong \Gamma \cdot A \cdot 1 \cong \Gamma \cdot A$$

□

Remark The fullness is necessary to show that a natural isomorphism between $\Gamma \cdot A \cdot B$ and $\Gamma \cdot \Sigma(A, B)$ yields a left adjoint to weakening. With fullness we get a natural isomorphism between $\mathsf{Hom}_{E(\Gamma\cdot A)}(B, \mathsf{Fst}^*C)$ and $\mathsf{Hom}_{E(\Gamma)}(\Sigma(A,B), C)$, which characterizes such an adjunction:

$$\begin{array}{rrcll}
 & B & \rightarrow & \mathsf{Fst}^*C & \text{in } E(\Gamma\cdot A) \\
\text{(Fullness)} & \hline\hline & & & \\
 & 1 & \rightarrow & (\mathsf{Fst};\mathsf{Fst})^*C & \text{in } E(\Gamma\cdot A\cdot B) \\
\text{(Nat. Iso)} & \hline\hline & & & \\
 & 1 & \rightarrow & \mathsf{Fst}^*C & \text{in } E(\Gamma\cdot\Sigma(A,B)) \\
\text{(Fullness)} & \hline\hline & & & \\
 & \Sigma(A,B) & \rightarrow & C & \text{in } E(\Gamma)
\end{array}$$

As an example of a category with a natural isomorphism between $\Gamma\cdot\Sigma(A,B)$ and $\Gamma\cdot A\cdot B$, but no left adjoint to weakening, take as base category $\mathcal{B}$ the category with two objects and one non-identity morphism

$$\Gamma \xrightarrow{\langle\rangle} [\,]$$

and define an indexed category $E\colon \mathcal{B}^{op}\rightarrow\mathbf{Cat}$ by

$$E([\,]):\quad A \overset{t_1}{\underset{t_2}{\rightrightarrows}} A \xrightarrow{!} 1, \quad t_1;! = t_2;! =!$$

and let $E(\Gamma)$ be the category with one object 1 and the identity morphism together with

$$\langle\rangle^*A = 1, \quad \langle\rangle^*t_1 = \langle\rangle^*t_2 = \langle\rangle^*! \; = \mathsf{Id}$$

It is easy to verify that E is an indexed category with a terminal object in each fibre. If we define

$$\begin{array}{rcl}
[\,]\cdot 1 & \stackrel{\text{def}}{=} & [\,] \\
[\,]\cdot A & \stackrel{\text{def}}{=} & \Gamma \\
\Gamma\cdot 1 & \stackrel{\text{def}}{=} & \Gamma
\end{array}$$

we obtain an adjunction $I \dashv G$. The adjunction $\mathsf{Fst} \dashv \Pi$ is already given: define $\Pi(A,1) = \Pi(1,1) = 1$. If we let furthermore $\Sigma(A,1)$ be A and $\Sigma(1,1)$ be 1 then the identities between $[\,]\cdot 1\cdot A = \Gamma$ and $[\,]\cdot A = \Gamma$ and $[\,]\cdot 1\cdot 1 = [\,]$ and $[\,]\cdot 1 = [\,]$ yield obviously natural isomorphisms. So we have a split D-category, but it is not full because there are two morphisms, namely t_1 and t_2, that correspond to the morphism $1 \xrightarrow{\mathsf{Id}} 1 = \mathsf{Fst}^*A$ in $E([\,]\cdot A)$. There is also no left adjoint to weakening because the sets $\mathsf{Hom}_{E([\,])}(\Sigma(A,1) = A, A)$ and $\mathsf{Hom}_{E([\,]\cdot A)}(1, \mathsf{Fst}^*A = 1)$ are not isomorphic.

2.2.2 Objectives of the Definition of a CC-Category

The CC-category is used in the next section for the derivation of categorical combinators. This yields two principles for the definition of a CC-category:

- Turn all canonical isomorphisms into identities whenever possible
- Use those constructions that give rise to the simplest relation to the syntax

The application of the first principle makes extra combinators and equations for the canonical isomorphisms superfluous.

A D-category is defined using fibrations, whereas the definition of a CC-category uses indexed categories and the Grothendieck construction instead. This has several advantages:

- All canonical isomorphisms that are involved in the definition of fibrations become identities.
- The Grothendieck construction has an intuitive meaning ((Γ, A) denotes the type A in context Γ), which is hidden in the fibration.
- No pullbacks are necessary for the definition of a CC-category. They are implicitly part of the definition of a fibration (cf. [Ehr88a, Prop. 2]).

Also the strict version of the Beck-Chevalley-condition is adopted because this eliminates further canonical isomorphisms. All these advantages lead to a much simpler system of combinators.

Other approaches [HP89][Pit89] use so-called *display maps.* They consider a category $\mathcal{B}$ with finite products, where the fibre over an object Γ of $\mathcal{B}$ is given by a collection of distinguished objects in $\mathcal{B}/\Gamma$, the display maps. For every object Γ in $\mathcal{B}$ the full subcategory $\mathcal{B}/\Gamma$ of display maps is denoted by $E(\Gamma)$. They are not used here for two reasons. Firstly, the pullbacks involved yield quite complicated combinators and secondly I do not know how to capture the role of display maps as distinguished morphisms in the base category. Streicher's extension of contextual categories [Str89] imposes a tree structure on the objects to capture the notion of type-in-context. This leads to complex categorical combinators, whereas in the CC-category terms-in-contexts are modelled by an adjunction that allows the derivation of simple combinators in a standard way.

Jacobs describes in his thesis [Jac91] a categorical semantics for the Calculus of Constructions using so-called full comprehension categories with units. These are given by a full and faithful functor $\mathcal{P}: \mathcal{E} \to \mathcal{B}^{\to}$, where $\mathcal{B}^{\to}$ is the arrow category of $\mathcal{B}$, such that

(i) $cod \circ \mathcal{P}: \mathcal{E} \to \mathcal{B}$ is a fibration, and

(ii) f cartesian in $\mathcal{E}$ implies $\mathcal{P}f$ is a pullback in $\mathcal{B}$. We will write it as

$$\begin{tikzcd}
\Gamma \cdot f^*A \arrow[r, "f \cdot \mathsf{Id}"] \arrow[d, "\mathsf{Fst}"'] & \Delta \cdot A \arrow[d, "\mathsf{Fst}"] \\
\Gamma \arrow[r, "f"'] & \Delta
\end{tikzcd}$$

(iii) Every fibre has a terminal object, which is stable under reindexing.

Their equational presentation is feasible only by using their equivalence with D-categories because all other formulations involve either pullbacks or yield conditional equations. The latter also happens if condition (i) is replaced by the requirement that for every morphism $f: \Gamma \to \Delta$ in $\mathcal{E}$ the operation ϕ sending any morphism $h: \Gamma \to \Gamma \cdot f^*A$ in $\mathcal{E}$ such that $h; \mathsf{Fst} = \mathsf{Id}$ to a morphism $g: \Gamma \to \Delta \cdot A$ in $\mathcal{E}$ satisfying $g; \mathsf{Fst} = f$, namely $\phi(h) = h; (f \cdot \mathsf{Id})$, is invertible.

2.3 The Combinators

In this section we define categorical combinators for the Calculus of Constructions and show how the calculus can be translated into the combinators. We also show the soundness of this translation and the equivalence of the calculus and the equational theory of categorical combinators. Finally we compare these combinators with Ehrhard's [Ehr88b].

2.3.1 The Equational Presentation

The equational theory of categorical combinators is a generalized algebraic theory in the sense of Cartmell [Car86]. The generalization concerns the sorts of the theory. Whereas in a normal multi-sorted algebraic theory the sorts are constants and may be interpreted as sets, in a generalized algebraic theory the sorts depend on a variable of a certain sort. Before giving the equational theory of a CC-category in detail, we describe its structure. There are two kinds of statements in the theory. The first is

$$T \ \text{sort}$$

and indicates that T is a well-formed sort. The second kind of statement

$$f \in T$$

where T is a well-formed sort, indicates that f is a well-formed term. The theory also contains equations between well-formed terms of the same sort. The sort of the terms and the conditions under which it can be deduced are omitted if they can be derived from the terms.

We have in the first instance the following rules concerning sorts:

$$(obj) \quad \frac{}{\text{Obj sort}}$$

$$(Hom) \quad \frac{\Gamma, \Gamma' \in \text{Obj}}{\text{Hom}(\Gamma, \Gamma') \ \text{sort}}$$

$$(Fib) \quad \frac{\Gamma \in \text{Obj}}{\text{Fib}(\Gamma) \ \text{sort}}$$

$$(Fun) \quad \frac{\Gamma \in \text{Obj} \qquad A, A' \in \text{Fib}(\Gamma)}{\text{Fun}_\Gamma(A, A') \ \text{sort}}$$

Elements Γ, $\Delta \cdots$ of type Obj and $f, g \cdots$ of sort $\mathrm{Hom}(\Gamma, \Gamma')$ correspond to objects and morphisms of the base category $\mathcal{B}$ respectively, whereas elements $A, B \cdots$ of sort $\mathrm{Fib}(\Gamma)$ and $t, s \cdots$ of sort $\mathrm{Fun}_\Gamma(A, B)$ denote objects and morphisms in the category $E(\Gamma)$ respectively.

However, the terminal object in the fibre cannot be treated simply as a combinator of sort $\mathrm{Fib}(\Gamma)$. The equation $t =!$, where ! is the combinator for the unique morphism from A to 1, is only true if t is a morphism from A to 1. When we later turn the equations into reduction rules, this means that the reduction $t \leadsto !$ is subject to a typing constraint. The abstract machines, which are based on these reduction rules, would then have to maintain type information during the reduction, which causes substantial overhead.

This can be avoided if the isomorphisms of Lemma 2.7 are treated as identities because in this case the only combinators with domain A and codomain 1 are the combinators $t;!$ if $A \neq 1$ and Id and $t;!$ if $A = 1$. Furthermore this identification streamlines the correspondence between the Calculus of Constructions and the categorical combinators significantly. The reason is that the calculus has no terminal type and therefore has no counterpart for the objects $\Gamma \cdot 1$ in the base and $1, \Pi(A, 1)$, $\Pi(1, A)$, $\Sigma(A, 1)$ and $\Sigma(1, A)$ in the fibre. The price we pay are additional sorts for the terminal object and for the morphisms having it as a domain or codomain:

$$(Termt) \quad \frac{\Gamma \in \mathrm{Obj}}{\mathrm{Termt}(\Gamma)\ \text{sort}}$$

$$(Fun) \quad \frac{\Gamma \in \mathrm{Obj} \quad A \in \mathrm{Fib}(\Gamma)}{\mathrm{Fun}_\Gamma(1, A)\ \text{sort}} \quad \frac{\Gamma \in \mathrm{Obj} \quad A \in \mathrm{Fib}(\Gamma)}{\mathrm{Fun}_\Gamma(A, 1)\ \text{sort}} \quad \frac{\Gamma \in \mathrm{Obj}}{\mathrm{Fun}_\Gamma(1, 1)\ \text{sort}}$$

We will use D, $D' \cdots$ as an abbreviation for either an element of $\mathrm{Fib}(\Gamma)$ or the combinator 1. The following notations are used:

$$\begin{array}{rcl}
\Gamma \rhd f{:}\,\Gamma' & \equiv & f \in \mathrm{Hom}(\Gamma, \Gamma') \\
\Gamma \rhd A & \equiv & A \in \mathrm{Fib}(\Gamma) \\
\Gamma \rhd 1 & \equiv & 1 \in \mathrm{Termt}(\Gamma) \\
\Gamma \rhd t{:}\, D \to D' & \equiv & t \in \mathrm{Fun}_\Gamma(D, D')
\end{array}$$

The signature of the equational theory is given by the following BNF-expressions:

$$\begin{array}{rcl}
\Gamma & ::= & [\,] \mid \Gamma \cdot A \\
f & ::= & \langle\rangle \mid \mathsf{Id} \mid f;f \mid \mathsf{Fst} \mid \langle f, t[A]\rangle \\
A & ::= & f * A \mid \Pi(A, A) \mid \Sigma(A, A) \mid \Omega \mid T \\
D & ::= & A \mid 1 \\
t & ::= & ! \mid \mathsf{Id} \mid t;t \mid f * t \mid \mathsf{Snd} \mid \mathsf{Cur}(A, t) \mid \mathsf{App}(A, A) \mid \forall(A, t) \\
 & & \mid \mathsf{Pair}(A, A) \mid \pi_1 \mid \pi_2 \mid t^f
\end{array}$$

We have the usual axioms for equality, saying that equality is an equivalence relation and that one can substitute equals for equals in every expression. They are as follows:

(Refl) $\quad \Gamma = \Gamma \quad f = f \quad A = A \quad t = t$

(Symm) $$\frac{\Gamma = \Gamma'}{\Gamma' = \Gamma} \quad \frac{f = f'}{f' = f} \quad \frac{A = A'}{A' = A} \quad \frac{t = t'}{t' = t}$$

(Trans) $$\frac{\Gamma = \Gamma' \quad \Gamma' = \Gamma''}{\Gamma = \Gamma''} \quad \frac{f = f' \quad f' = f''}{f = f''}$$

$$\frac{A = A' \quad A' = A''}{A = A''} \quad \frac{t = t' \quad t' = t''}{t = t''}$$

$(\cdot)$ $$\frac{\Gamma = \Gamma' \quad A = A'}{\Gamma \cdot A = \Gamma' \cdot A'}$$

$(*)$ $$\frac{f = f' \quad A = A'}{f * A = f' * A'} \quad \frac{f = f' \quad t = t'}{f * t = f' * t'}$$

(Π) $$\frac{A = A' \quad B = B'}{\Pi(A, B) = \Pi(A', B')} \quad \frac{t = t' \quad A = A'}{\mathsf{Cur}(A, t) = \mathsf{Cur}(A', t')}$$

$$\frac{A = A' \quad B = B'}{\mathsf{App}(A, B) = \mathsf{App}(A', B')}$$

(Σ) $$\frac{A = A' \quad B = B'}{\Sigma(A, B) = \Sigma(A', B')} \quad \frac{A = A' \quad B = B'}{\mathsf{Pair}(A, B) = \mathsf{Pair}(A', B')}$$

$(;)$ $$\frac{f = f' \quad g = g'}{f; g = f'; g'} \quad \frac{t = t' \quad u = u'}{t; u = t'; u'}$$

$(\langle -, - \rangle)$ $$\frac{f = f' \quad t = t' \quad A = A'}{\langle f, t[A] \rangle = \langle f', t'[A'] \rangle}$$

$(\forall)$ $$\frac{A = A' \quad t = t'}{\forall(A, t) = \forall(A', t')}$$

$(-^f)$ $$\frac{t = s}{t^f = s^f}$$

$(conv)$ $$\frac{\Gamma \rhd f\colon \Delta \quad \Delta = \Delta' \quad \Gamma = \Gamma'}{\Gamma' \rhd f\colon \Delta'} \quad \frac{\Gamma \rhd t\colon A \to B \quad A = A' \quad B = B'}{\Gamma \rhd t\colon A' \to B'}$$

Next we give the rules and equations for the indexed category $E\colon \mathcal{B}^{op} \to \mathbf{Cat}$. Firstly, we say that $\mathcal{B}$ is a category with a terminal object:

$([\,])$ $$\frac{}{[\,] \in \mathrm{Obj}}$$

$(\langle\rangle)$ $$\frac{\Gamma \in \mathrm{Obj}}{\Gamma \rhd \langle\rangle\colon [\,]}$$

$$(id-base)\quad \frac{\Gamma \in \mathsf{Obj}}{\Gamma \rhd \mathsf{Id}\colon \Gamma}$$

$$(;-base)\quad \frac{\Gamma \rhd f\colon \Gamma' \qquad \Gamma' \rhd g\colon \Gamma''}{\Gamma \rhd f;g\colon \Gamma''}$$

$$\begin{array}{llcl}
(\langle\rangle) & \langle\rangle & = & \mathsf{Id}\colon[\,]\to[\,] \\
(\langle\rangle) & f;\langle\rangle & = & \langle\rangle \\
(idL) & f;\mathsf{Id} & = & f \\
(idR) & \mathsf{Id};f & = & f \\
(;-assoc) & (f;g);h & = & f;(g;h)
\end{array}$$

Secondly, we express that for every object Γ in the base category $\mathcal{B}$ there is a category $E(\Gamma)$:

$$(id-fib)\quad \frac{\Gamma \rhd D}{\Gamma \rhd \mathsf{Id}\colon D \to D}$$

$$(;-fib)\quad \frac{\Gamma \rhd t\colon D \to D' \qquad \Gamma \rhd s\colon D' \to D''}{\Gamma \rhd t;s\colon D \to D''}$$

$$\begin{array}{llcl}
(idL) & t;\mathsf{Id} & = & t \\
(idR) & \mathsf{Id};t & = & t \\
(;-assoc) & (u;t);s & = & u;(t;s)
\end{array}$$

Thirdly, we have the following rules and equations concerning the functor $(-)^*$, which is represented by the operator $*$ in the equational theory:

$$(*-obj)\quad \frac{\Gamma \rhd f\colon \Gamma' \qquad \Gamma' \rhd A'}{\Gamma \rhd f * A'}$$

$$(*-morph)\quad \frac{\Gamma \rhd f\colon \Gamma' \qquad \Gamma' \rhd t\colon A \to A'}{\Gamma \rhd f * t\colon f * A \to f * A'} \qquad \frac{\Gamma \rhd f\colon \Gamma' \qquad \Gamma' \rhd t\colon 1 \to A}{\Gamma \rhd f * t\colon 1 \to f * A}$$

$$\frac{\Gamma \rhd f\colon \Gamma' \qquad \Gamma' \rhd t\colon A \to 1}{\Gamma \rhd f * t\colon f * A \to 1} \qquad \frac{\Gamma \rhd f\colon \Gamma' \qquad \Gamma' \rhd t\colon 1 \to 1.}{\Gamma \rhd f * t\colon 1 \to 1}$$

$$\begin{array}{rcl}
\mathsf{Id} * A & = & A \\
\mathsf{Id} * t & = & t \\
f * \mathsf{Id} & = & \mathsf{Id} \\
f * (t;s) & = & (f * t);(f * s) \\
(f;g) * A & = & f * (g * A) \\
(f;g) * t & = & f * (g * t)
\end{array}$$

After the equational theory of indexed categories we give the rules and equations for the terminal object 1 in each category $E(\Gamma)$, which is preserved on the nose by every functor $E(f)$:

$$(1)\quad \frac{}{\Gamma \rhd 1}$$

$$(!)\quad \frac{\Gamma \rhd A}{\Gamma \rhd !\colon A \to 1}$$

$$\begin{aligned} ! &= \mathsf{Id}: 1 \rightarrow 1 \\ t;! &= ! \end{aligned}$$

Remark The equation

$$f*! = !$$

can be derived using the other rules and was therefore omitted above:

$$f*! = f*!; \mathsf{Id} = f*!;! = !$$

The next part of the equational theory of CC-categories is concerned with the adjunction $I \dashv G$. The presentation is obtained using a well-known characterization of an adjunction given in [Mac71, Theorem IV.2] that is applied also in [CE87]:

Lemma 2.8 *Given the following data:*

- *two small categories $\mathcal{C}$ and $\mathcal{D}$ and a functor $F: \mathcal{C} \rightarrow \mathcal{D}$,*
- *a function G sending every object of $\mathcal{D}$ to an object of $\mathcal{C}$,*
- *a function Λ assigning each morphism $f: FA \rightarrow U$ in $\mathcal{C}$ a morphism $\Lambda(f)$: $A \rightarrow GU$ in $\mathcal{D}$, and*
- *for every object U of $\mathcal{D}$ a morphism $\epsilon(U): FGU \rightarrow U$*

then if the equations

$$\begin{array}{lrcl} (\beta) & \epsilon(U) \circ F(\Lambda(f)) & = & f \\ (nat) & \Lambda(f) \circ g & = & \Lambda(f \circ F(g)) \\ (\eta) & \Lambda(\epsilon(U)) & = & \mathsf{Id}_{GU} \end{array}$$

hold for all objects U of $\mathcal{D}$ and all morphisms $f: FA \rightarrow U$, $g: B \rightarrow A$, the function G can be extended to a functor $G: \mathcal{D} \rightarrow \mathcal{C}$, being a right adjoint to F. Furthermore, the bijection between the hom-sets of the adjunction is given by the function Λ, and ϵ is the counit of the adjunction.

Proof Define the effect of G on a morphism $g: U \rightarrow V$ by

$$G(g) := \Lambda(g \circ \epsilon(U))$$

and the inverse of Λ by

$$\Lambda^{-1}(h) := \epsilon(U) \circ F(h) \quad (h: A \rightarrow GU)$$

An easy calculation shows that G is a functor and that Λ and Λ^{-1} are inverse to each other and that the bijection between the hom-sets

$$\frac{f: FA \rightarrow U}{\overline{\Lambda(f): A \rightarrow GU}}$$

is natural in both A and U. Therefore $F \dashv G$, and ϵ is the counit of this adjunction.

□

This lemma is now applied to the adjunction $I \dashv G$ of definition 2.6 to get its equational presentation. We do not introduce a separate notation for the category $Gr(E)$, but we use instead the corresponding expressions in terms of the indexed category E. This yields the following presentation:

$$(G-obj) \qquad \frac{\Gamma \rhd A}{\Gamma \cdot A \in \mathrm{Obj}}$$

$$(\langle\rangle) \qquad \frac{\Gamma \rhd f\colon \Gamma' \qquad \Gamma' \rhd A' \qquad \Gamma \rhd t\colon 1 \to f * A'}{\Gamma \rhd \langle f, t[A']\rangle\colon \Gamma' \cdot A'}$$

$$(\mathsf{Fst}) \qquad \frac{\Gamma \rhd A}{\Gamma \cdot A \rhd \mathsf{Fst}\colon \Gamma}$$

$$(\mathsf{Snd}) \qquad \frac{\Gamma \rhd A}{\Gamma \cdot A \rhd \mathsf{Snd}\colon 1 \to \mathsf{Fst} * A}$$

$$\begin{array}{lrcl}
(\mathsf{Fst}) & \langle f, t[A']\rangle ; \mathsf{Fst} & = & f \\
(\mathsf{Snd}) & \langle f, t[A']\rangle * \mathsf{Snd} & = & t \\
(nat) & f ; \langle g, t[A']\rangle & = & \langle f ; g, f * t[A']\rangle \\
(\langle\rangle - \eta) & \langle \mathsf{Fst}, \mathsf{Snd}[A']\rangle & = & \mathsf{Id}
\end{array}$$

Remark The correspondence between the rules and equations given above and the general setting as described in Lemma 2.8 is as follows:

- $G((\Gamma, A))$ corresponds to $\Gamma \cdot A$.
- The bijection Λ of the hom-sets corresponds to the operator $\langle\rangle$. We need the extra type information in the combinator $\langle f, t[A]\rangle$ to make its type unique. The reason is that it is in general impossible to derive the type A from a type B equal to $f * A$.
- The counit ϵ, which is a morphism $(\Gamma \cdot A, 1) \to (\Gamma, A)$, is the morphism $(\mathsf{Fst}, \mathsf{Snd})$.
- The equation (β) corresponds to the equations Fst and Snd, because the equation (β) looks in this situation as follows:

  $$(\mathsf{Fst}, \mathsf{Snd}) \ \circ ((\langle f, t\rangle, \mathsf{Id}) = (f, t)$$

  which is equivalent to

  $$(\mathsf{Fst} \circ \langle f, t\rangle, \langle f, t\rangle^*(\mathsf{Snd})) = \langle f, t\rangle$$

- The equations (nat) and (η) correspond to the same equations in lemma 2.8.

We use later on the abbreviation $f \cdot t[A']$ also for the term $\langle \mathsf{Fst}; f, \mathsf{Snd}; \mathsf{Fst} * t[A']\rangle$ of the equational theory.

The equational presentation of the fullness condition cannot be derived directly from the definition because this leads to a conditional equation. Instead we use the fact that it is equivalent with the statement

- For every morphism $t: 1 \to \mathsf{Fst}^* B$ in $E(\Gamma \cdot A)$ there exists a unique morphism $t^f: A \to B$ in $E(\Gamma)$ such that $\mathsf{Snd}; \mathsf{Fst}^* t^f = t$.

This is captured by the following rules and equations:

$$(-^f) \quad \frac{\Gamma \cdot A \rhd t: 1 \to \mathsf{Fst} * B}{\Gamma \rhd t^f: A \to B}$$

$$\begin{array}{lrcl} (-^f - red) & \mathsf{Snd}; \mathsf{Fst} * t^f & = & t \\ (-^f - equ) & (\mathsf{Snd}; \mathsf{Fst} * t)^f & = & t \end{array}$$

The next part of the equational theory concerns the adjunctions $\mathsf{Fst}^*_A \dashv \Pi_A$ of Definition 2.6. Lemma 2.8 yields the following equations and rules:

$$(\Pi) \quad \frac{\Gamma \rhd A \qquad \Gamma \cdot A \rhd B}{\Gamma \rhd \Pi(A, B)}$$

$$(\mathsf{Cur}) \quad \frac{\Gamma \rhd A \qquad \Gamma \cdot A \rhd t: \mathsf{Fst} * C \to B}{\Gamma \rhd \mathsf{Cur}(A, t): C \to \Pi(A, B)}$$

$$(\mathsf{Cur}) \quad \frac{\Gamma \rhd A \qquad \Gamma \cdot A \rhd t: 1 \to B}{\Gamma \rhd \mathsf{Cur}(A, t): 1 \to \Pi(A, B)}$$

$$(\mathsf{App}) \quad \frac{\Gamma \rhd A \qquad \Gamma \cdot A \rhd B}{\Gamma \cdot A \rhd \mathsf{App}(A, B): \mathsf{Fst} * \Pi(A, B) \to B}$$

$$\begin{array}{lrcl} (\beta) & (\mathsf{Fst} * \mathsf{Cur}(A, t)); \mathsf{App}(A, B) & = & t \\ (nat) & s; \mathsf{Cur}(A, t) & = & \mathsf{Cur}(A, (\mathsf{Fst} * s); t) \\ (\eta) & \mathsf{Cur}(A, \mathsf{App}(A, B)) & = & \mathsf{Id} \end{array}$$

The Beck-Chevalley condition yields directly the following equations:

$$\begin{array}{lrcl} (\mathsf{Cur}) & f * \mathsf{Cur}(A, t) & = & \mathsf{Cur}(f * A, (f \cdot \mathsf{Id}[A]) * t) \\ (\Pi) & f * \Pi(A, B) & = & \Pi(f * A, (f \cdot \mathsf{Id}[A]) * B) \end{array}$$

We will need the following equation, which is a consequence of the Beck-Chevalley condition and the η-rule:

$$\begin{array}{lrcl} (\mathsf{App} - nat) & f * \mathsf{App}(A, B) & = & \langle \mathsf{Id}, f * \mathsf{Snd}[f; \mathsf{Fst} * A] \rangle * \mathsf{App}(f; \mathsf{Fst} * A, \\ & & & \langle \mathsf{Fst}; f; \mathsf{Fst}, \mathsf{Snd}[f; \mathsf{Fst} * A] \rangle * B) \end{array}$$

Remark The combinator $\mathsf{Cur}(A, t)$ is needed only for morphisms $\Gamma \cdot A \rhd t: 1 \to B$ as the following calculation shows for any $\Gamma \cdot A \rhd t: \mathsf{Fst} * C \to B$:

$$\begin{array}{rcl} \mathsf{Cur}(A, t) & = & (\mathsf{Snd}; \mathsf{Fst} * \mathsf{Cur}(A, t))^f \\ & = & (\mathsf{Snd}; \mathsf{Cur}(\mathsf{Fst} * A, \langle \mathsf{Fst}; \mathsf{Fst}, \mathsf{Snd}[A] \rangle))^f \\ & = & (\mathsf{Cur}(\mathsf{Fst} * A, \mathsf{Fst} * \mathsf{Snd}; \langle \mathsf{Fst}; \mathsf{Fst}, \mathsf{Snd}[A] \rangle * t))^f \end{array}$$

Using the $(\mathsf{App} - nat)$-equation we can reformulate the (η)-equation to

$$\mathsf{Cur}(A, \mathsf{Fst} * t; \mathsf{App}(A, B)) = t$$

In the sequel we will use only the modified version.

The natural isomorphism between $\Gamma\cdot A\cdot B$ and $\Gamma\cdot\Sigma(A,B)$ gives rise in the first instance to three combinators Pair, π_1' and π_2' corresponding to the three morphisms Pair, π_1 and π_2. We do not use the combinators π_1' and π_2' for the derivation of the abstract machines but equivalent ones obtained by the fullness condition. The reason is that the morphism $\langle\mathsf{Id},t\rangle*\pi_1$, which corresponds to the term $\pi_1(t)$, is equal to the morphism $t;\pi_1^f$. If we define the combinator π_1 to correspond to the morphism π_1^f, the abstract machines can handle the combinator $t;\pi_1$ for the first projection in the same way as the combinator $t;\langle\mathsf{Id},s[A]\rangle*\mathsf{App}(A,B)$ for application because both are translated into a composition in the fibre. In the case of the combinator π_2', we obtain only that the morphism $\langle\mathsf{Id},t\rangle*\pi_2$ is equal to the morphism $t;(\mathsf{Fst};\langle\mathsf{Id},t\rangle*\pi_2^f)$. Because the morphisms $t_1;(\mathsf{Fst};\langle\mathsf{Id},t\rangle*\pi_2^f)$ and $t_2;(\mathsf{Fst};\langle\mathsf{Id},t\rangle*\pi_2^f)$ are equal for any $t_1,t_2\colon 1\to\Sigma(A,B)$ in $E(\Gamma)$, we can define the combinator $t;\pi_2$ as an abbreviation for the morphism $t;(\mathsf{Fst};\langle\mathsf{Id},t\rangle*\pi_2^f)$. This leads to the following rules and equations:

$$(\Sigma)\quad \frac{\Gamma\rhd A\qquad \Gamma\cdot A\rhd B}{\Gamma\rhd\Sigma(A,B)}$$

$$(\mathsf{Pair})\quad \frac{\Gamma\cdot A\rhd B}{\Gamma\cdot A\cdot B\rhd \mathsf{Pair}(A,B)\colon 1\to\mathsf{Fst};\mathsf{Fst}*\Sigma(A,B)}$$

$$(\pi_1)\quad \frac{\Gamma\cdot A\rhd B}{\Gamma\rhd\pi_1\colon\Sigma(A,B)\to A}$$

$$(\pi_2)\quad \frac{\Gamma\rhd t\colon 1\to\Sigma(A,B)}{\Gamma\rhd t;\pi_2\colon 1\to\langle\mathsf{Id},t;\pi_1[A]\rangle*B}$$

$$\begin{array}{lrcl}
(\sigma_1) & \langle\mathsf{Id},t[A],s[B]\rangle*\mathsf{Pair}(A,B);\pi_1 & = & t\\
(\sigma_2) & \langle\mathsf{Id},t[A],s[B]\rangle*\mathsf{Pair}(A,B);\pi_2 & = & s\\
(surj) & \langle\mathsf{Fst},\mathsf{Snd};\mathsf{Fst}*\pi_1[A],\mathsf{Snd};\pi_2[B]\rangle*\mathsf{Pair}(A,B) & = & \mathsf{Snd}
\end{array}$$

$$\begin{array}{lrcl}
(nat-\mathsf{Pair}) & \langle f,t[A],s[B]\rangle*\mathsf{Pair}(A,B) & = & \langle\mathsf{Id},f*t[f*A],f\cdot\mathsf{Id}[A]*s[f\cdot\mathsf{Id}\\
 & & & [A]*B]\rangle*\mathsf{Pair}(f*A,f\cdot\mathsf{Id}[A]*B)\\
(nat-\pi_1) & f*\pi_1 & = & \pi_1\\
(nat-\pi_2) & f*(t;\pi_2) & = & f*t;\pi_2
\end{array}$$

Finally, the clauses (vii) and (viii) of definition 2.6 lead directly to the following additional rules and equations:

$$(\Omega)\quad \frac{}{[\,]\rhd\Omega}$$

$$(\forall)\quad \frac{\Gamma\cdot A\rhd t\colon 1\to\langle\rangle*\Omega}{\Gamma\rhd\forall(A,t)\colon 1\to\langle\rangle*\Omega}$$

$$(T)\quad \frac{}{[\,]\cdot\Omega\rhd T}$$

$$\begin{array}{lrcl}
(\forall-fun) & h*\forall(A,t) & = & \forall(h*A,(h\cdot\mathsf{Id}[A])*t)\\
(Coh) & \langle\langle\rangle,\forall(A,t)[\Omega]\rangle*T & = & \Pi(A,\langle\langle\rangle,t[\Omega]\rangle*T)
\end{array}$$

This completes the equational theory of a CC-category.

2.3.2 The Translation of the Calculus of Constructions

Now we define an interpretation of the type theory of section 2.1 in the equational presentation outlined in the previous section. This interpretation is given by a translation of raw types, terms and contexts of the type theory into the corresponding categorical combinators.

Definition 2.9 (Translation into combinators) *The translation of raw types, terms and contexts into categorical combinators is given by the following function* $[\![\text{-}]\!]$, *where* Fst^n *is an abbreviation for* $\underbrace{\mathsf{Fst}; \mathsf{Fst}; \cdots; \mathsf{Fst}}_{n\text{-times}}$*:*

(i) *on contexts*

$$\begin{aligned} [\![[\,]]\!] &= [\,] \\ [\![(\Gamma, A)]\!] &= [\![\Gamma]\!] \cdot [\![A]\!] \end{aligned}$$

(ii) *on types*

$$\begin{aligned} [\![\Pi A.B]\!] &= \Pi([\![A]\!], [\![B]\!]) \\ [\![\Sigma A.B]\!] &= \Sigma([\![A]\!], [\![B]\!]) \\ [\![\mathsf{Prop}]\!] &= \langle\rangle * \Omega \\ [\![\mathsf{Proof}(t)]\!] &= \langle\langle\rangle, [\![t]\!][\Omega]\rangle * T \end{aligned}$$

(iii) *on terms*

$$\begin{aligned} [\![n]\!] &= \mathsf{Fst}^n * \mathsf{Snd} \\ [\![\lambda A.t]\!] &= \mathsf{Cur}([\![A]\!], [\![t]\!]) \\ [\![\mathsf{App}(A, B, t, s)]\!] &= [\![t]\!]; \langle \mathsf{Id}, [\![s]\!][[\![A]\!]]\rangle * \mathsf{App}([\![A]\!], [\![B]\!]) \\ [\![\forall A.t]\!] &= \forall([\![A]\!], [\![t]\!]) \\ [\![\mathsf{Pair}(A, B, t, s)]\!] &= \langle \mathsf{Id}, [\![t]\!][[\![A]\!]], [\![s]\!][[\![B]\!]]\rangle * \mathsf{Pair}([\![A]\!], [\![B]\!]) \\ [\![\pi_1(t)]\!] &= [\![t]\!]; \pi_1 \\ [\![\pi_2(t)]\!] &= [\![t]\!]; \pi_2 \end{aligned}$$

Remarks

1. The translation shows how neat the correspondence between the Calculus of Constructions and the categorical combinators is. The raw terms of the calculus can be translated directly into raw combinators. In order to establish the soundness of the translation it is therefore enough to show that every judgement in the calculus can be made also using the combinators obtained by translating the terms. If the categorical semantics of the calculus is given directly by a translation of raw terms and types into morphisms and objects of a CC-category, the definition is more complex [Str89]. A partial translation has to be defined, and it has to be proved that this translation is defined on well-formed terms and types and respects the judgements.

2. The translation explains also why de Bruijn-variables are used in the definition of the Calculus of Constructions (cf. section 2.1). The reason is the clause concerning variables. The categorical combinator corresponding to a variable is the projection from the context, in which the variable is declared, to its type. This is exactly captured by the de Bruijn-number and therefore it is not necessary to introduce a translation of terms-in-contexts, as would be the case if normal variables were used.

3. The translations shows how the syntactical distinctions in this version of the Calculus of Constructions are motivated by corresponding distinctions in the combinators: a proposition corresponds to a morphism $\Gamma \rhd t{:}\, 1 \rightarrow \Omega$, and the type of its proofs to an object in the fibre.

2.3.3 Soundness

The soundness of the above translation can now be stated and proved in a fairly standard way.

Theorem 2.10 (Soundness) *For every judgement in the calculus there exists a corresponding judgement in the equational theory of combinators, more precisely:*

(Contexts) $\vdash \Gamma$ ctxt *implies* $[\![\Gamma]\!] \in \mathsf{Obj}$

(Types) $\Gamma \vdash A$ type *implies* $[\![\Gamma]\!] \rhd [\![A]\!]$

(Terms) $\Gamma \vdash t{:}\, A$ *implies* $[\![\Gamma]\!] \rhd [\![t]\!]{:}\, 1 \rightarrow [\![A]\!]$

(Eq-Contexts) $\Gamma = \Gamma'$ *implies* $[\![\Gamma]\!] = [\![\Gamma']\!] \in \mathsf{Obj}$

(Eq-Types) $\Gamma \vdash A = B$ *implies* $[\![\Gamma]\!] \rhd [\![A]\!] = [\![B]\!]$

(Eq-Terms) $\Gamma \vdash t = s{:}\, A$ *implies* $[\![\Gamma]\!] \rhd [\![t]\!] = [\![s]\!]{:}\, 1 \rightarrow [\![A]\!]$

Proof The proof of the theorem proceeds by induction over the derivation of the judgements in the type theory and depends crucially on the fact that weakening and substitution, which are meta-operations in the type theory, are translated into certain categorical combinators in the equational theory. This is expressed in the following two lemmata. Their proofs are routine inductions over the derivation of the judgements and are therefore omitted.

Lemma 2.11 *Weakening of an expression e well-formed in context (Γ, Γ') with $\Gamma' = B_{i-1} \cdot \cdots \cdot B_0$ with respect to types $A_{m-1}, \ldots, A_0$ corresponds to the application of the combinator*

$$\mathsf{S}^m_i := \mathsf{Fst}^m \cdot \underbrace{\mathsf{Id}[B_{i-1}] \cdot \cdots \cdot \mathsf{Id}[B_0]}_{i-\text{times}} .$$

More precisely, if we define $\mathsf{S}^m_i(\Gamma)$ as an abbreviation for

$$\begin{array}{rcll} \mathsf{S}^m_i([\,]) & = & [\,] & \\ \mathsf{S}^m_i(\Gamma \cdot A) & = & \mathsf{S}^m_{i-1}(\Gamma) \cdot \mathsf{S}^m_i(A) & (i > 0) \\ \mathsf{S}^m_i(\Gamma) & = & \Gamma & (i = 0), \end{array}$$

then for a given context $\vdash (\Gamma, \Gamma')$ ctxt with $\Gamma' = B_{i-1} \cdot \cdots \cdot B_0$, $[\![(\Gamma, \Gamma')]\!] \in \mathsf{Obj}$ and types $A_{m-1}, \ldots, A_0$ satisfying

$$(\Gamma, A_{m-1}, \ldots, A_{j+1}) \vdash A_j \text{ type} \qquad [\![(\Gamma, A_{m-1}, \ldots, A_{j+1})]\!] \rhd [\![A_j]\!] \qquad (m-1 \geq j \geq 0)$$

we have with $\Delta := (\Gamma, A_{m-1}, \ldots, A_0, \mathsf{U}_i^m(\Gamma'))$

(i) $[\![\Delta]\!] = [\![\Gamma]\!] \cdot [\![A_{m-1}]\!] \cdot \cdots \cdot [\![A_0]\!] \cdot \mathsf{S}_i^m([\![\Gamma']\!])$

(ii) $(\Gamma, \Gamma') \vdash A$ type and $[\![(\Gamma, \Gamma')]\!] \rhd [\![A]\!]$ implies
$[\![\Delta]\!] \rhd \mathsf{S}_i^m * [\![A]\!] = [\![\mathsf{U}_i^m(A)]\!]$

(iii) $(\Gamma, \Gamma') \vdash t{:}\, A$ and $[\![(\Gamma, \Gamma')]\!] \rhd [\![t]\!]{:}\, 1 \rightarrow [\![A]\!]$ implies
$[\![\Delta]\!] \rhd \mathsf{S}_i^m * [\![t]\!] = [\![\mathsf{U}_i^m(t)]\!]{:}\, 1 \rightarrow [\![\mathsf{U}_i^m(A)]\!]$

(iv) $\vdash (\Gamma, \Gamma') = E$ and $[\![(\Gamma, \Gamma')]\!] = [\![E]\!]$ implies
$[\![\Delta]\!] = \mathsf{S}_i^m([\![E]\!])$

(v) $(\Gamma, \Gamma') \vdash A = B$ and $[\![(\Gamma, \Gamma')]\!] \rhd [\![A]\!] = [\![B]\!]$ implies
$[\![\Delta]\!] \rhd \mathsf{S}_i^m * [\![A]\!] = \mathsf{S}_i^m * [\![B]\!]$

(vi) $(\Gamma, \Gamma') \vdash s = t{:}\, A$ and $[\![(\Gamma, \Gamma')]\!] \rhd [\![t]\!] = [\![s]\!]{:}\, 1 \rightarrow [\![A]\!]$ implies
$[\![\Delta]\!] \rhd \mathsf{S}_i^m * [\![s]\!] = \mathsf{S}_i^m * [\![t]\!]$

The next lemma is concerned with substitution, which is also modelled by an operator in the equational theory.

Lemma 2.12 *Substitution of a term s of type A in an expression e well-formed in a context* (Γ, Γ')*, where* $\Gamma' = B_{n-1} \cdot \cdots \cdot B_0$*, for the de Bruijn- variable n corresponds to the application of the combinator*

$$\mathsf{Su}_n([\![s]\!]) := \langle \mathsf{Id}, [\![s]\!][[\![A]\!]] \rangle \cdot \underbrace{\mathsf{Id}[B_{n-1}] \cdot \cdots \cdot \mathsf{Id}[B_0]}_{n-\text{times}}$$

More precisely, if we define $\mathsf{Su}_n([\![s]\!])(\Gamma)$ *as an abbreviation for*

$$\begin{array}{rcll} \mathsf{Su}_n([\![s]\!])[\,] & = & [\,] & \\ \mathsf{Su}_n([\![s]\!])(\Gamma \cdot E) & = & \mathsf{Su}_{n-1}(\Gamma) \cdot \mathsf{Su}_n([\![s]\!]) * E & (n > 0) \\ \mathsf{Su}_n([\![s]\!])(\Gamma) & = & \Gamma & (n = 0) \end{array}$$

then for a given context $\vdash (\Gamma, A, \Gamma')$ ctxt *with* $\Gamma' = B_{n-1} \cdot \cdots \cdot B_0$*,* $[\![(\Gamma, A, \Gamma')]\!] \in$ Obj *and a term s satisfying* $\Gamma \vdash s{:}\, A$ *and* $[\![\Gamma]\!] \rhd [\![s]\!]{:}\, 1 \rightarrow [\![A]\!]$*, we have with* $\Delta{:}= (\Gamma, \Gamma'[n \backslash s])$

(i) $[\![\Delta]\!] = [\![\Gamma]\!] \cdot \mathsf{Su}_n([\![s]\!])([\![\Gamma']\!])$

(ii) $(\Gamma, A, \Gamma') \vdash B$ type and $[\![(\Gamma, A, \Gamma')]\!] \rhd [\![B]\!]$ implies
$[\![\Delta]\!] \rhd \mathsf{Su}_n([\![s]\!]) * [\![B]\!] = [\![B[n \backslash s]]\!]$

(iii) $(\Gamma, A, \Gamma') \vdash t{:}\, B$ and $[\![(\Gamma, A, \Gamma')]\!] \rhd [\![t]\!]{:}\, 1 \rightarrow [\![B]\!]$ implies
$[\![\Delta]\!] \rhd \mathsf{Su}_n([\![s]\!]) * [\![t]\!] = [\![t[n \backslash s]]\!]{:}\, 1 \rightarrow [\![B[n \backslash s]]\!]$

(iv) $\vdash (\Gamma, A, \Gamma') = E$ and $[\![(\Gamma, \Gamma')]\!] = [\![E]\!]$ implies
$[\![\Delta]\!] = \mathsf{Su}_n([\![s]\!])([\![E]\!])$

(v) $(\Gamma, A, \Gamma') \vdash B = C$ and $\llbracket(\Gamma, A, \Gamma')\rrbracket \triangleright \llbracket B\rrbracket = \llbracket C\rrbracket$ implies
$\llbracket\Delta\rrbracket \vdash \mathsf{Su}_n(\llbracket s\rrbracket) * \llbracket B\rrbracket = \mathsf{Su}_n(\llbracket s\rrbracket) * \llbracket C\rrbracket$

(vi) $(\Gamma, A, \Gamma') \vdash t = u{:}\, B$ and $\llbracket(\Gamma, A, \Gamma')\rrbracket \triangleright \llbracket t\rrbracket = \llbracket u\rrbracket{:}\, 1{\rightarrow}\llbracket B\rrbracket$ implies
$\llbracket\Delta\rrbracket \triangleright \mathsf{Su}_n(\llbracket s\rrbracket) * \llbracket t\rrbracket = \mathsf{Su}_n(\llbracket s\rrbracket) * \llbracket u\rrbracket$

The key parts of the proof of the theorem are the steps concerning β, η and the coherence rule. These parts are considered below, whereas all other parts are omitted here for brevity. The rules in braces like {} are rules of the equational theory, whereas rules of the type theory are denoted by braces like ().

($\beta - rule$) The induction hypothesis yields:

$$\begin{array}{rcl} \llbracket\Gamma\rrbracket \cdot \llbracket A\rrbracket & \triangleright & \llbracket t\rrbracket{:}\, 1{\rightarrow}\llbracket B\rrbracket \\ \llbracket\Gamma\rrbracket & \triangleright & \llbracket s\rrbracket{:}\, 1{\rightarrow}\llbracket A\rrbracket \end{array}$$

Rule $\{\mathsf{Cur}\}$ yields:

$$\llbracket\Gamma\rrbracket \triangleright \mathsf{Cur}(\llbracket A\rrbracket, \llbracket t\rrbracket) : 1{\rightarrow}\Pi(\llbracket A\rrbracket, \llbracket B\rrbracket)$$

Using rules $\{\mathsf{App}\}$, $\{\langle\rangle\}$, one obtains

$$\llbracket\Gamma\rrbracket \triangleright \mathsf{Cur}(\llbracket A\rrbracket, \llbracket t\rrbracket); \langle \mathsf{Id}, \llbracket s\rrbracket[\llbracket A\rrbracket]\rangle * \mathsf{App}(\llbracket A\rrbracket, \llbracket B\rrbracket)) : 1{\rightarrow}\langle \mathsf{Id}, \llbracket s\rrbracket[\llbracket A\rrbracket]\rangle * \llbracket B\rrbracket$$

Using rule $\{\beta\}$ and letting

$$u := \llbracket\mathsf{App}(A, B, \lambda A.t, s)\rrbracket = \mathsf{Cur}(\llbracket A\rrbracket, \llbracket t\rrbracket); \langle \mathsf{Id}, \llbracket s\rrbracket[\llbracket A\rrbracket]\rangle * \mathsf{App}(\llbracket A\rrbracket, \llbracket B\rrbracket))$$

one gets

$$\llbracket\Gamma\rrbracket \triangleright u = \langle \mathsf{Id}, \llbracket s\rrbracket[\llbracket A\rrbracket]\rangle * \llbracket t\rrbracket{:}\, 1{\rightarrow}\langle \mathsf{Id}, \llbracket s\rrbracket[\llbracket A\rrbracket]\rangle * \llbracket B\rrbracket$$

Lemma 2.12 yields now the claim.

($\eta - rule$) The induction hypothesis yields:

$$\begin{array}{rcl} \llbracket\Gamma\rrbracket & \triangleright & \llbracket t\rrbracket{:}\, 1{\rightarrow}\Pi(\llbracket A\rrbracket, \llbracket B\rrbracket) \\ \llbracket\Gamma\rrbracket \cdot \llbracket A\rrbracket & \triangleright & \llbracket B\rrbracket \end{array}$$

Lemma 2.11 and rule $\{\mathsf{App}\}$ and $\{\mathsf{Cur}\}$ then imply:

$$\begin{array}{rcl} \llbracket\Gamma\rrbracket & \triangleright & \mathsf{Cur}(\llbracket A\rrbracket, \mathsf{Fst} * \llbracket t\rrbracket; \langle \mathsf{Id}, \mathsf{Snd}[\mathsf{Fst} * \llbracket A\rrbracket]\rangle * \\ & & \mathsf{App}(\mathsf{Fst} * \llbracket A\rrbracket, \langle \mathsf{Fst}; \mathsf{Fst}, \mathsf{Snd}[\llbracket A\rrbracket]\rangle * \llbracket B\rrbracket)){:}\, 1{\rightarrow}\Pi(\llbracket A\rrbracket, \llbracket B\rrbracket) \end{array}$$

Applying the rule $\{\mathsf{App} - nat\}$ and letting

$$\begin{array}{rcl} u & \overset{\text{def}}{=} & \llbracket\lambda A.\mathsf{App}(\mathsf{U}^1_0(A), \mathsf{U}^1_1(B), \mathsf{U}^1_0(t), 0)\rrbracket \\ & = & \mathsf{Cur}(\llbracket A\rrbracket, \mathsf{Fst} * \llbracket t\rrbracket; \langle \mathsf{Id}, \mathsf{Snd}[\mathsf{Fst} * \llbracket A\rrbracket]\rangle * \\ & & \mathsf{App}(\mathsf{Fst} * \llbracket A\rrbracket, \langle \mathsf{Fst}; \mathsf{Fst}, \mathsf{Snd}[\llbracket A\rrbracket]\rangle * \llbracket B\rrbracket)) \end{array}$$

one obtains

$$\llbracket\Gamma\rrbracket \triangleright u = \mathsf{Cur}(\llbracket A\rrbracket, \mathsf{Fst} * \llbracket t\rrbracket; \mathsf{App}(\llbracket A\rrbracket, \llbracket B\rrbracket)){:}\, 1{\rightarrow}\Pi(\llbracket A\rrbracket, \llbracket B\rrbracket)$$

and after an application of the $\{\eta\}$-rule

$$\llbracket\Gamma\rrbracket \triangleright u = \llbracket t\rrbracket{:}\, 1{\rightarrow}\Pi(\llbracket A\rrbracket, \llbracket B\rrbracket)$$

(σ_2) By the induction hypothesis we have

$$[\![\Gamma]\!] \rhd [\![\mathsf{Pair}(A,B,t,s)]\!] = \langle \mathsf{Id}, [\![t]\!][[\![A]\!]], [\![s]\!][[\![B]\!]]\rangle * \mathsf{Pair}([\![A]\!],[\![B]\!])$$

Rule $\{\sigma_2\}$ implies then that

$$[\![\Gamma]\!] \rhd [\![\pi_2(\mathsf{Pair}(A,B,t,s))]\!] = \langle \mathsf{Id}, [\![t]\!][[\![A]\!]], [\![s]\!][[\![B]\!]]\rangle * \mathsf{Pair}([\![A]\!],[\![B]\!]); \pi_2 = [\![s]\!]$$

($\forall - elim$) The induction hypothesis yields:

$$[\![\Gamma]\!] \cdot [\![A]\!] \rhd [\![p]\!]\colon 1 \rightarrow \langle\rangle * \Omega$$

Rule $\{\forall\}$ yields then

$$[\![\Gamma]\!] \rhd \forall([\![A]\!],[\![p]\!])\colon 1 \rightarrow \langle\rangle * \Omega$$

Rule $\{coh\}$ shows the claim.

□

2.3.4 Equivalence between the Calculus and the Combinators

The equivalence between categorical combinators and the Calculus of Constructions is based on a translation from the combinators to CC-expressions, or to lists $\{t_{m-1},\ldots,t_0\}$ of CC-expressions in case of context morphisms. However, it is impossible to give a translation that respects the judgements and is the inverse of the translation of CC-expressions into combinators. The reason is that for any CC-term t the combinator $[\![t]\!]$ is a morphism with domain 1. But if we restrict ourselves to those morphisms, then such an inverse translation can be given. The fullness condition ensures that it can be uniquely extended to a translation of all combinators such that if s is the translation of any combinator t with $\Gamma \rhd t\colon A \rightarrow B$ and $A \neq 1$, the combinator $[\![s]\!]^f$ is equal to t.

The translation of the combinators Fst and Id poses another problem. The length of the list of CC-terms corresponding to these combinators depends on the length of the context in which their well-formedness is derived. This implies that the translation cannot be defined on raw expressions alone but it needs a parameter indicating the length of that context.

Definition 2.13 (Inverse translation) *The translation function $()^c$ is defined by induction over the structure of those raw combinators that may appear in a morphism $t\colon 1 \rightarrow B$ as follows, where n denotes a natural number:*

1. *On contexts*

$$\begin{array}{llcl} ([\,]) & (n,[\,])^c & = & [\,] \\ (\cdot) & (n+1,\Gamma\cdot A)^c & = & ((n,\Gamma)^c,(n,A)^c) \end{array}$$

2. On context morphisms

$$\begin{array}{llcl}
(\langle\rangle) & (n,\langle\rangle)^c & = & \{\} \\
(\mathsf{Id}) & (n,\mathsf{Id})^c & = & \{n-1,\ldots,0\} \\
(\mathsf{Fst}) & (n,\mathsf{Fst})^c & = & \{n-1,\ldots,1\}
\end{array}$$

$$(;)\qquad \frac{(n,f)^c=\{t_{m-1},\ldots,t_0\}}{(n,f;g)^c=(m,g)^c[i\backslash t_i]}$$

$$(\langle -,-\rangle)\quad (n,\langle f,t[A]\rangle)^c \;=\; \{(n,f)^c,(n,t)^c\}$$

3. On types

$$(*)\qquad \frac{(n,f)^c=\{t_{m-1},\ldots,t_0\}}{(n,f*A)^c=(m,A)^c[i\backslash t_i]}$$

$$\begin{array}{lrcl}
(\Pi) & (n,\Pi(A,B))^c & = & \Pi(n,A)^c.(n+1,B)^c \\
(\Sigma) & (n,\Sigma(A,B))^c & = & \Sigma(n,A)^c.(n+1,B)^c \\
(\Omega) & (n,\Omega)^c & = & \mathsf{Prop} \\
(T) & (n,T)^c & = & \mathsf{Proof}(0)
\end{array}$$

4. On morphisms:

$$\begin{array}{lrcl}
(!) & (n,t;f*!;s)^c & = & (n,s)^c \\
 & (n,t;!;s)^c & = & (n,s)^c \\
 & (n,f*!;s)^c & = & (n,s)^c \\
 & (n,!;s)^c & = & (n,s)^c \\
(\mathsf{Id}) & (n,f*\mathsf{Id};s)^c & = & (n,s)^c \\
 & (n,\mathsf{Id};s)^c & = & (n,s)^c \\
 & (n,t;f*\mathsf{Id})^c & = & (n,t)^c \\
 & (n,t;\mathsf{Id})^c & = & (n,t)^c \\
(;) & (n,t;(s;u))^c & = & (n,(t;s);u)^c \\
 & (n,t;f*(s;u))^c & = & (n,(t;f*s);f*u)^c
\end{array}$$

$$(*)\qquad \frac{(n,f)^c=\{t_{m-1},\ldots,t_0\}}{(n,f*t)^c=(m,t)^c[i\backslash t_i]}$$

$$\begin{array}{lrcl}
(\mathsf{Snd}) & (n,\mathsf{Snd})^c & = & 0 \\
(\mathsf{Cur}) & (n,\mathsf{Cur}(A,t))^c & = & \lambda(n,A)^c.(n+1,t)^c \\
(\mathsf{App}) & (n,t;f*\mathsf{App}(A,B))^c & = & \mathsf{App}((n,f;\mathsf{Fst}*A)^c,(n+1,f;\mathsf{Fst}\cdot \\
 & & & \mathsf{Id}[A]*B)^c,(n,t)^c,(n,f*\mathsf{Snd})^c) \\
 & (n,t;\mathsf{App}(A,B))^c & = & (n,t;\mathsf{Id}*\mathsf{App}(A,B))^c \\
(\forall) & (n,\forall(A,t))^c & = & \forall(n,A)^c.(n+1,t)^c \\
(\mathsf{Pair}) & (n+2,\mathsf{Pair}(A,B))^c & = & \mathsf{Pair}((n,A)^c,(n+1,B)^c,1,0) \\
(\pi_i) & (n,t;f*\pi_i)^c & = & \pi_i((n,t)^c) \\
({}^f) & (n,t;s^f)^c & = & (n+1,s)^c[0\backslash(n,t)^c]
\end{array}$$

$$\frac{(n,f)^c=\{t_{m-1},\ldots,t_0\}}{(n,t;f*s^f)^c=(m+1,s)^c[i+1\backslash t_i,0\backslash t]}$$

Although the definition of the translation $()^c$ is more complex than that of $[\![-]\!]$, the usual technique of showing the preservation of judgements by $()^c$ applies here as well, namely induction over the derivation using appropriate lemmata for weakening and substitution. All operations on lists of CC-terms are applied componentwise.

Theorem 2.14 *The inverse translation respects the judgements. More precisely for any context* Γ *with* $|\Gamma| = n$ *we have:*

(Contexts) $\Gamma \in \mathsf{Obj}$ *implies* $\vdash (n,\Gamma)^c$ ctxt

(Context Morphisms) $\Gamma \rhd f{:}\,\Delta = [\,] \cdot B_{m-1} \cdots B_0$ *implies*
$(n,\Gamma)^c \vdash (n,f)^c = \{t_{m-1},\ldots,t_0\}$ *and*
$(n,\Gamma)^c \vdash (n,t_i)^c{:}$
$(m-i-1,B_i)^c[m-2-i\backslash t_{m-1},\ldots,0\backslash t_{i+1}]$

(Types) $\Gamma \rhd A$ *implies* $(n,\Gamma)^c \vdash (n,A)^c$ type

(Terms) $\Gamma \rhd t{:}\,1{\rightarrow}A$ *implies* $(n,\Gamma)^c \vdash (n,t)^c{:}\,(n,A)^c$

(Eq-Contexts) $\Gamma = \Gamma'$ *implies* $(n,\Gamma)^c = (n,\Gamma')^c$

(Eq-Context Morphisms) $\Gamma \rhd f = f'{:}\,\Delta$ *implies* $(n,\Gamma)^c \vdash (n,f)^c = (n,f')^c$

(Eq-Types) $\Gamma \rhd A = A'$ *implies* $(n,\Gamma)^c \vdash (n,A)^c = (n,A')^c$

(Eq-Terms) $\Gamma \rhd t = t'{:}\,1{\rightarrow}A$ *implies*
$(n,\Gamma)^c \vdash (n,t)^c = (n,t')^c{:}\,(n,A)^c$

Proof The theorem is shown by an induction over the derivation of judgements. As in the case of the translation $[\![-]\!]$, we need two lemmata concerning weakening and substitution. Their proofs are omitted because they are a routine induction over the structure of the derivation.

Lemma 2.15 (Weakening) *The translation* $()^c$ *respects weakening, i.e. for every object* $\Gamma \cdot \Gamma' \in \mathsf{Obj}$ *satisfying* $\vdash (|\Gamma \cdot \Gamma'|, \Gamma \cdot \Gamma')^c$ ctxt *and types* $A_{m-1},\ldots,A_0$ *s.t.* $\Gamma \cdot A_{m-1} \cdot \cdots \cdot A_{j+1} \rhd A_j$ *and* $(\Gamma \cdot A_{m-1} \cdot \cdots \cdot A_{j+1})^c \vdash (\Gamma \cdot A_{m-1} \cdot \cdots \cdot A_{j+1}, A_j)^c$ type, *the following equations hold with* $i = |\Gamma'|$, $k = |\Gamma \cdot \Gamma'|$ *and* $\Delta = \Gamma \cdot A_{m-1} \cdot \cdots \cdot A_0 \cdot \mathsf{S}_i^m \Gamma'$ *for any combinator* f, A *and* t *that is well-formed in context* $\Gamma \cdot \Gamma'$ *and whose translation under* $(-)^c$ *is well-formed:*

1. *on contexts*
$\vdash (k+m,\Delta)^c = ((|\Gamma|,\Gamma)^c, (|\Gamma|,A_m)^c, \ldots, (|\Gamma|+m,A_0)^c, \mathsf{U}_i^m((|\Gamma|+m,\Gamma')^c))$

2. *on context morphisms*
$(k+m,\Delta)^c \vdash (k+m, \mathsf{S}_i^m * f)^c = \mathsf{U}_i^m((k,f)^c)$

3. *on types*
$(k+m,\Delta)^c \vdash (k+m, \mathsf{S}_i^m * A)^c = \mathsf{U}_i^m((k,A)^c)$

4. *on terms*
$(k+m,\Delta)^c \vdash (k+m, \mathsf{S}_i^m * t)^c = \mathsf{U}_i^m((k,t)^c)$

Lemma 2.16 (Substitution) *The translation* $(-)^c$ *respects substitution, i.e. given an object* $\Gamma \cdot A \cdot \Gamma' \in \mathsf{Obj}$ *such that with* $n = |\Gamma'|$ *and* $k = |\Gamma \cdot A \cdot \Gamma'|$ *we have* $\vdash (k, \Gamma \cdot A \cdot \Gamma')^c$ ctxt, *and given a morphism* t *such that* $\Gamma \rhd t{:}\,1{\rightarrow}A$ *satisfying*

$$(|\Gamma|,\Gamma)^c \vdash (|\Gamma|,t)^c{:}\,(|\Gamma|,A)^c$$

we have the following judgements with $\Delta = \Gamma \cdot \mathsf{Su}_n(t)(\Gamma')$*,* $s = (|\Gamma|, t)^c$ *for every combinator* f*,* A *and* t *well-formed in context* $\Gamma \cdot A \cdot \Gamma'$ *such that the translation under* $(-)^c$ *is also well-formed:*

(i) $\vdash (k-1,\Delta)^c = ((|\Gamma|,\Gamma)^c, (|\Gamma|,\Gamma')^c[n\backslash s])$

(ii) $(k-1,\Delta)^c \vdash (k-1, \mathsf{Su}_n(t); f)^c = (k,f)^c[n\backslash s]$

(iii) $(k-1,\Delta)^c \vdash (k-1, \mathsf{Su}_n(t) * B)^c = (k,B)^c[n\backslash s]$

(iv) $(k-1,\Delta)^c \vdash (k-1, \mathsf{Su}_n(t) * u)^c = (k,u)^c[n\backslash s]$

We show only the more difficult cases of the proof of the theorem here.

($\beta - rule$) The premise for this rule is

$$\Gamma \cdot A \rhd t\colon 1 \rightarrow B$$

Therefore, the induction hypothesis and the weakening lemma 2.15 yield

$$(n+1, \Gamma \cdot A)^c \vdash (n+1, \mathsf{Fst} * \mathsf{Cur}(A,t))^c = (\lambda(n,A)^c.(n+1,t)^c) \uparrow$$

The definition of $()^c$ yields

$$\begin{array}{rcl}
(n+1,\Gamma\cdot A)^c & \vdash & (n+1, \mathsf{Fst} * \mathsf{Cur}(A,t); \mathsf{App}(A,B))^c = \\
 & = & \mathsf{App}(((n,A)^c)\uparrow, \mathsf{U}_1^1((n+1,B)^c), (\lambda(n,A)^c.(n+1,t)^c)\uparrow, 0) \\
 & = & \mathsf{App}(((n,A)^c)\uparrow, \mathsf{U}_1^1((n+1,B)^c), \lambda(((n,A)^c)\uparrow . \\
 & & \mathsf{U}_1^1((n+1,t)^c)), 0) \\
 & \overset{(\beta)}{=} & (n+1,t)^c
\end{array}$$

($\eta - rule$) The induction hypothesis and the weakening lemma 2.15 imply

$$\begin{array}{rcl}
(n,\Gamma)^c & \vdash & (n, \mathsf{Cur}(A, \mathsf{Fst} * t; \mathsf{App}(A,B)))^c \\
 & = & \lambda(n,A)^c, \mathsf{App}(((n,A)^c)\uparrow, \mathsf{U}_1^1((n+1,B)^c), ((n,t)^c)\uparrow, 0) \\
 & \overset{(\eta)}{=} & (n,t)^c
\end{array}$$

($t;\pi_2$) The induction hypothesis yields

$$(n,\Gamma)^c \vdash (n,t)^c\colon \Sigma(n,A)^c.(n+1,B)^c$$

This implies

$$(n,\Gamma)^c \vdash (n,t;\pi_2)^c = \pi_2((n,t)^c)\colon (n+1,B)^c[0\backslash\pi_1((n,t)^c)]$$

An application of the induction hypothesis for $t;\pi_1$ and of Lemma 2.12 regarding substitution proves the claim.

$(coh - rule)$ The induction hypothesis applied to the premises establishes:

$$(n+1, \Gamma \cdot A)^c \vdash (n+1, t)^c \colon \mathsf{Prop}$$

Therefore, $(n, \langle\langle\rangle, \forall(A,t)[\Omega]\rangle * T)^c$ is defined. This implies

$$\begin{array}{rcl} (n,\Gamma)^c & \vdash & (n, \langle\langle\rangle, \forall(A,t)[\Omega]\rangle * T)^c = \\ & = & \mathsf{Proof}(0)[0\backslash\forall(n,A)^c.(n+1,t)^c] \\ & = & \mathsf{Proof}(\forall(n,A)^c, (n+1,t)^c) \\ & = & \Pi(n,A)^c.\mathsf{Proof}((n+1,t)^c) \\ & = & (\Pi(A, \langle\langle\rangle, t[\Omega]\rangle * T))^c \end{array}$$

$(t; f * s^f)$ Let us assume that $\Gamma \rhd t \colon 1 \to f * A$, $f \colon \Gamma \to \Delta$ and $\Delta \cdot A \rhd s \colon 1 \to \mathsf{Fst} * B$. Then the induction hypothesis yields

$$(|\Delta| + 1, \Delta \cdot A)^c \vdash (|\Delta| + 1, s)^c = ((|\Delta|, B)^c) \uparrow$$

and

$$(n, \Gamma)^c \vdash (n,t)^c \colon (n, f * A)^c$$

The substitution lemma 2.16 implies then that

$$(n, \Gamma)^c \vdash (|\Delta| + 1, s)^c[i+1\backslash t_i, 0\backslash t] \colon (|\Delta|, B)^c[i\backslash t_i]$$

where $(n, f)^c = \{t_{m-1}, \ldots, t_0\}$.

□

Finally we can prove that the translations $()^c$ and $[\![]\!]$ are inverse to each other in the sense discussed above.

Theorem 2.17 *The translations $()^c$ and $[\![]\!]$ are inverse to each other, where n denotes the length of the context Γ:*

(i) $([\![-]\!])^c$ *is the identity:*

1. *on contexts:* $\vdash (n, [\![\Gamma]\!])^c = \Gamma$ *if* $\vdash \Gamma$ ctxt
2. *on types:* $\Gamma \vdash (n, [\![A]\!])^c = A$ *if* $\Gamma \vdash A$ type
3. *on terms:* $\Gamma \vdash (n, [\![t]\!])^c = t : A$ *if* $\Gamma \vdash t \colon A$

(ii) *The map* $[\![(-)^c]\!]$ *is the identity:*

1. *on contexts:* $[\![(n,\Gamma)^c]\!] = \Gamma$ *if* $\Gamma \in \mathrm{Obj}$
2. *on context morphisms:* $\Gamma \rhd \langle\langle\rangle, [\![t_{m-1}]\!][B_{m-1}], \ldots, [\![t_0]\!][B_0]\rangle = f$ *if* $\Gamma \vdash f \colon \Gamma \to \Delta = [\,] \cdot B_{m-1} \cdots B_0$ *and* $(n,f)^c = \{t_{m-1}, \ldots, t_0\}$
3. *on types:* $\Gamma \rhd [\![(n,A)^c]\!] = A$ *if* $\Gamma \rhd A$
4. *on terms:* $\Gamma \rhd [\![(n,t)^c]\!] = t \colon 1 \to B$ *if* $\Gamma \rhd t \colon 1 \to B$

Proof The proof proceeds by induction over the derivation of judgements.

(i) The verification of most cases involves only routine calculations and is therefore omitted. The more interesting cases are now verified:

(Var) The weakening lemmata for both the calculus and the combinators yield the claim:

$$(|\Gamma| + 1 + |\Gamma'|, \llbracket(\Gamma, A, \Gamma')\rrbracket)^c \vdash \mathsf{U}_0^{|\Gamma'|}(0) = |\Gamma'|$$

($\Pi - Elim$) The soundness theorem 2.10 yields

$$\begin{array}{rcl} \llbracket\Gamma\rrbracket & \rhd & \llbracket\mathsf{App}(A, B, t, s)\rrbracket = \\ & = & \llbracket t\rrbracket; \langle\mathsf{Id}, \llbracket s\rrbracket[\llbracket A\rrbracket]\rangle * \mathsf{App}(\llbracket A\rrbracket, \llbracket B\rrbracket) \end{array}$$

Now letting u be the right-hand side of this equation, we obtain by applying the induction hypothesis:

$$\begin{array}{rcl} (n, \llbracket\Gamma\rrbracket)^c & \vdash & (n, u)^c = \mathsf{App}(A, B, t, (n, \langle\mathsf{Id}, \llbracket s\rrbracket[\llbracket A\rrbracket]\rangle * \mathsf{Snd})^c) \\ & \overset{Lemma\ 2.16}{=} & \mathsf{App}(A, B, t, 0[n-1\backslash n-1, \ldots, 1\backslash 1, 0\backslash s]) \\ & = & \mathsf{App}(A, B, t, s) \end{array}$$

(ii) Similarly we verify only the cases $\mathsf{Cur}(A, t)$, $t; f * \mathsf{App}(A, B)$ and $t; f * s^f$.

($\mathsf{Cur}(A, t)$) The definition of $(-)^c$ and the induction hypothesis yields $\Gamma \cdot A \rhd t\colon 1 \rightarrow B$ and

$$\llbracket(n+1, t)^c\rrbracket = t$$

Hence we get

$$\begin{array}{rcl} \Gamma & \rhd & \llbracket(n, \mathsf{Cur}(A, t))^c\rrbracket = \llbracket(n, \mathsf{Cur}(A, t))^c\rrbracket \\ & \overset{I.H.}{=} & \llbracket\lambda(n, A)^c.(n+1, t)^c\rrbracket \\ & \overset{I.H.}{=} & \mathsf{Cur}(A, t) \end{array}$$

(App) The calculation is as follows:

$$\begin{array}{rcl} \Gamma & \rhd & \llbracket(n, t; f * \mathsf{App}(A, B))^c\rrbracket \\ & = & \llbracket\mathsf{App}((n, f; \mathsf{Fst} * A)^c, (n+1, \langle\mathsf{Fst}; f; \mathsf{Fst}, \mathsf{Snd}[A]\rangle * B)^c, \\ & & (n, t)^c, (n, f * \mathsf{Snd})^c)\rrbracket \\ & \overset{I.H.}{=} & t; \langle\mathsf{Id}, f * \mathsf{Snd}[f; \mathsf{Fst} * A]\rangle * \mathsf{App}(f; \mathsf{Fst} * A, \langle\mathsf{Fst}; f; \mathsf{Fst}, \\ & & \mathsf{Snd}[A]\rangle * B) \\ & \overset{\{\mathsf{App}-nat\}}{=} & t; f * \mathsf{App}(A, B) \end{array}$$

($t; f * s^f$) Let $(n, f)^c = \{t_{m-1}, \ldots, t_0\}$ and $\Gamma \rhd t\colon 1 \rightarrow A$. Then the induction hypothesis and the substitution lemmata imply

$$\begin{array}{rcl} \Gamma \rhd \llbracket(n, t; f * s^f)^c\rrbracket & = & \llbracket(m, s)^c[i+1\backslash t_i, 0\backslash(n, t)^c]\rrbracket = \langle f, t[A]\rangle * s \\ & = & \langle f, t[A\rangle * (\mathsf{Snd}; \mathsf{Fst} * s^f) = t; f * s^f \end{array}$$

□

The translation $(-)^c$ can be extended to a translation from all combinators to CC-expressions by defining $(n,t)^c$ to be $(n+1, \mathsf{Snd}; \mathsf{Fst} * t)^c$ for any combinator t such that $\Gamma \rhd t\colon A \rightarrow B$ with $A \neq 1$. Theorem 2.14 implies with $n = |\Gamma|$ that

$$(n+1, \Gamma \cdot A)^c \vdash (n+1, \mathsf{Snd}; \mathsf{Fst} * t)^c\colon ((n,B)^c) \uparrow$$

and Theorem 2.17 yields

$$\Gamma \rhd t = [\![(n,t)^c]\!]^f\colon A \rightarrow B$$

Furthermore, any extension of $(-)^c$ that satisfies the last equation has to be defined in this way:

$$\begin{array}{rcl} \Gamma & \rhd & t = [\![(n,t)^c]\!]^f \;\Rightarrow \\ \Gamma \cdot A & \rhd & \mathsf{Snd}; \mathsf{Fst} * t = \mathsf{Snd}; \mathsf{Fst} * [\![(n,t)^c]\!]^f = [\![(n,t)^c]\!] \;\Rightarrow \\ (n+1, \Gamma \cdot A)^c & \vdash & (n+1, \mathsf{Snd}; \mathsf{Fst} * t)^c = (n+1, [\![(n,t)^c]\!])^c \overset{Thm\ 2.17}{=} (n,t)^c \end{array}$$

The combinators are constructed in such a way that the initial CC-category (i.e. the initial object in the category **CCcat** of CC-categories and structure-preserving functors) can be easily described in terms of them:

Theorem 2.18 *Let $\mathcal{C}$ be the indexed category $E\colon \mathcal{B}^{op} \rightarrow \mathbf{Cat}$ defined as follows:*

- *Objects of the base category $\mathcal{B}$ are equivalence classes of combinators Γ satisfying $\Gamma \in \mathrm{Obj}$ modulo the derivable equality.*
- *Morphisms from Γ to Δ in $\mathcal{B}$ are equivalence classes of combinators f s.t. $\Gamma \rhd f\colon \Delta$ is a valid judgement in the equational theory modulo derivable equality.*
- *The identity is the identity combinator, and composition in $\mathcal{B}$ is given by the composition operator on the combinators.*
- *Objects of the fibre $E(\Gamma)$ are equivalence classes of combinators satisfying $\Gamma \rhd D$ modulo derivable equality.*
- *Morphisms from D to D' in $E(\Gamma)$ are equivalence classes of combinators satisfying $\Gamma \rhd t\colon D \rightarrow D'$ modulo derivable equality .*
- *Identities and composition in the fibres are given by the corresponding combinators.*
- *The functor $E(f)$, with $f\colon \Gamma \rightarrow \Delta$ a morphism in $\mathcal{B}$ is given by the operation on the combinators, i.e.*

$$\begin{array}{rcl} E(f)(A) & := & f * A \\ E(f)(1) & := & 1 \\ E(f)(t) & := & f * t \end{array}$$

Then $\mathcal{C}$ is the initial CC-category.

Proof The lemmata 2.7 and 2.8 are the key to showing that $\mathcal{C}$ is a CC-category. The initiality is obvious. □

2.3.5 Comparison with Ehrhard's Combinators

Ehrhard defines combinators for the Calculus of Constructions in the appendix of his thesis [Ehr88b]. They are an intermediate step between the calculus in de Bruijn-form and the categorical combinators presented here. He takes that calculus and replaces the de Bruijn-numbers by combinators for explicit substitution. These are derived with the split D-categories in mind although he only presents the equations and does not discuss their relation to the categorical structure. This implies that he has only the judgement $\Gamma \rhd t\colon A$ and not $\Gamma \rhd t\colon A \to B$, as in our approach.

The important difference for the design of abstract machines is that Ehrhard has no basic combinator that corresponds to an environment $\langle f, t\rangle$. He uses instead a combinator $(\leftarrow t); f^{\rightarrow}$, where $(\leftarrow t)$ corresponds to $\langle \mathsf{Id}, t\rangle$ and $f^{\rightarrow}$ to $\langle \mathsf{Fst}; f, \mathsf{Snd}\rangle$ in this setting. His representation seems to lead to a more complicated treatment of environments in the machine; see chapter 4.

Chapter 3

Reduction Strategies

Until now we have considered only an equational theory. We turn in this chapter to the definition of suitable reduction rules and examine possible strategies for performing the reductions. The next chapter shows how these strategies lead directly to abstract machines.

3.1 Reduction Rules

Before we can define reduction rules, we have to streamline the equational theory slightly.

Many applications of the Calculus of Constructions for theorem proving and theory abstraction do not need the η-rule or the surjective dependent sums [Tay89] [Luo90]. The combinatorial counterparts of these rules cannot be reformulated in such a way that an inspection of the structure of a combinator suffices to check their applicability. The η—rule requires a test whether a combinator t is equal to $\mathsf{Fst} * t'$ for some t', and the η-rule for dependent sums needs an equality test. This would make the reduction machine a lot more complicated. Therefore we restrict ourselves to a Calculus of Constructions without η-reductions or surjective dependent sums. As a consequence we omit the corresponding equations $\mathsf{Cur}(\mathsf{Fst} * t; \mathsf{App}(A, B)) = t$ and

$$\langle \mathsf{Fst}, \mathsf{Snd}; \mathsf{Fst} * \pi_1[A], \mathsf{Snd}; \pi_2[B] \rangle * \mathsf{Pair}(A, B) = \mathsf{Snd}$$

in the equational theory of categorical combinators. It is however possible to extend the machine to compute long $\beta\eta$-normal forms, as needed for example in [Pfe91].

We make two further changes to the equational theory. The $(-)^f$-combinator and the morphism $!\colon 1 \to 1$ were introduced in the previous chapter to turn the combinators into a full D-category. However, they are not necessary for the construction of abstract machines: they are not used in the translation of CC-expressions into combinators, and the correspondence theorems 2.10, 2.14 and 2.17 remain valid if we drop the combinators $(-)^f$ and ! from the equational theory. We will therefore consider in the sequel this theory, whose derivability relation is denoted by $\rhd_e$ [1].

[1] $\rhd_e$ stands for explicitly typed application; we will later consider a combinator App without any typing information.

3.1.1 Towards a Convergent Reduction

We aim for a notion of reduction $\leadsto_e$ for the combinators with the following properties:

- It is confluent and strongly normalizing.
- Its normal forms for types and morphisms are the translations of the normal forms of CC-expressions via β-reduction, which will be denoted by $\leadsto_e$ as well.
- The equivalence relation generated by it is the equality of combinators discussed in the previous chapter.
- It is defined in such a way that only the structure of a well-formed combinator determines whether a reduction rule applies or not.

Such a notion ensures that any reduction strategy leads on one hand to a unique normal form and on the other hand can be used to decide the equality of combinators. Furthermore the last property implies that it is not necessary to handle any information about the type of a combinator during reduction, which simplifies the machines considerably.

The normalization and the last property are the hardest to achieve. The former causes a problem because the extension of the reducibility proof to the combinators fails. Instead we will describe a strongly normalizing restriction of the reduction relation that reduces a combinator first to one corresponding to a closure and then to the normal form. The problem with the last property rests with the typed application. It occurs also in the Calculus of Constructions, where a well-formed term of the form $\mathsf{App}(A, B, \lambda A.t, s)$ may be not a β-redex because $(\Gamma, A') \vdash t{:}B'$ and $\Gamma \vdash \Pi A'.B' = \Pi A.B$ is true but not $\Gamma \vdash A = A'$ and $(\Gamma, A) \vdash B = B'$. So the definition of the reduction $\leadsto$ in the calculus uses the derivable equality in the way that $t[0\backslash s]$ is a contractum of $\mathsf{App}(A, B, \lambda A'.t, s)$ only if $\Gamma \vdash A = A'$, $\Gamma \vdash s{:}A$, $(\Gamma, A) \vdash t{:}B'$ and $(\Gamma, A) \vdash B = B'$. Hence a separate proof is required that the smallest equivalence relation is indeed the derivable equality.

Only the confluence makes it possible to eliminate this check because it implies that whenever $\Gamma \vdash \Pi A.B = \Pi A'.B'$ then also $\Gamma \vdash A = A'$ and $(\Gamma, A) \vdash B = B'$. As a consequence we can omit the typing information in the application and replace the term $\mathsf{App}(A, B, t, s)$ by ts. This approach works also in the case of the combinators if we replace the combinator $\mathsf{App}(A, B)$ by App. In contrast to the strong normalization, we can derive the confluence for combinators from the confluence of the calculus. We will define a reduction relation with the first three properties in this subsection and discuss the last property in the following one.

The obvious first approach to obtain such a reduction relation $\leadsto_e$ is to orient the equations. In most cases there is only one sensible way of doing this. For example if we orient the equation $\langle f, t[A]\rangle * \mathsf{Snd} = t$ from right to left, the reduction will certainly not be strongly normalizing. The exceptions are the η-like equations as $\langle \mathsf{Fst}, \mathsf{Snd}[A]\rangle = \mathsf{Id}$ and $\langle\rangle = \mathsf{Id}{:}[\,]{\rightarrow}[\,]$. If they are treated as rules for simplification of expressions, we have to add some more rules to achieve confluence [CD91]. As already stated, the latter rules are difficult to implement, and so we will follow the approach of Jay [Jay92] on checking the equality of combinators. He regards these rules as expansion rules, applied after reduction to β-normal form.

Remark In general, we will use the following terminology for reduction according to [Klo90]:

- The one-step reduction is denoted by $t \rightsquigarrow t'$, pronounced "t reduces to t'".
- The equivalence relation $\leftrightarrow^*$ generated by $\rightsquigarrow$ is called convertibility, and we say "t is convertible to t'" for $t \leftrightarrow^* t'$.
- $t \equiv t'$ denotes the syntactical identity of raw combinators.

On checking that the relation $\rightsquigarrow_e$ satisfies the properties we want, we see that the $\rightsquigarrow_e$-normal forms are not the intended ones. For example, the combinator

$$\llbracket \mathsf{App}(A, B, \lambda A.t, s) \rrbracket \equiv \mathsf{Cur}(\llbracket A \rrbracket, \llbracket t \rrbracket); \langle \mathsf{Id}, \llbracket s \rrbracket[\llbracket A \rrbracket] \rangle * \mathsf{App}(\llbracket A \rrbracket, \llbracket B \rrbracket)$$

can only reduce to $\mathsf{Cur}(A', t'); g' * \mathsf{App}(A'', B'')$ with $\llbracket A \rrbracket \rightsquigarrow_e^* A'$, $\llbracket t \rrbracket \rightsquigarrow_e^* t'$ and $\langle \mathsf{Id}, \llbracket s \rrbracket[\llbracket A \rrbracket] \rangle * \mathsf{App}(\llbracket A \rrbracket, \llbracket B \rrbracket) \rightsquigarrow_e g' * \mathsf{App}(A'', B'')$, i.e. there is no reduction corresponding to the β-reduction $\mathsf{App}(A, B, \lambda A.t, s) \rightsquigarrow_e t[0\backslash s]$ in the calculus. The remedy is to replace the too restrictive rule $\mathsf{Fst} * \mathsf{Cur}(A, t); \mathsf{App}(A, B) \rightsquigarrow_e t$ by the rule [2]

$$(\beta) \quad f * \mathsf{Cur}(A, t); g * \mathsf{App}(A_1, B_1) \rightsquigarrow_e \langle f, g * \mathsf{Snd}[A] \rangle * t$$

This rule has side conditions analogous to that of the β-rule in the calculus:

- $\Gamma \rhd_e f : \Delta$,
- $\Delta \cdot A \rhd_e t : 1 \rightarrow B$,
- $f * A = g; \mathsf{Fst} * A_1$ and
- $\langle \mathsf{Fst}; f, \mathsf{Snd}[A] \rangle * B = \langle \mathsf{Fst}; g; \mathsf{Fst}, \mathsf{Snd}[A_1] \rangle * B_1$

They arise when we derive the equality between redex and contractum:

$$\begin{aligned}
f * \mathsf{Cur}(A, t); g * \mathsf{App}(A_1, B_1) &= \mathsf{Cur}(f * A, f \cdot \mathsf{Id}[A] * t); \\
&\quad \langle g; \mathsf{Fst}, g * \mathsf{Snd}[A_1] \rangle * \mathsf{App}(A_1, B_1) \\
&= \mathsf{Cur}(f * A, f \cdot \mathsf{Id}[A] * t); \langle \mathsf{Id}, g * \mathsf{Snd}[g; \mathsf{Fst} \\
&\quad * A_1] \rangle * \mathsf{App}(g; \mathsf{Fst} * A_1, (g; \mathsf{Fst}) \cdot \mathsf{Id}[A_1] * B_1) \\
&= \langle \mathsf{Id}, g * \mathsf{Snd}[g; \mathsf{Fst} * A_1] \rangle * (\mathsf{Fst} * \mathsf{Cur}(f * A, f \\
&\quad \cdot \mathsf{Id}[A] * t); \mathsf{App}(g; \mathsf{Fst} * A_1, (g; \mathsf{Fst}) \cdot \mathsf{Id}[A_1] * B_1)) \\
&= \langle \mathsf{Id}, g * \mathsf{Snd}[g; \mathsf{Fst} * A_1] \rangle * (f \cdot \mathsf{Id}[A] * t) \\
&= \langle f, g * \mathsf{Snd}[A] \rangle * t
\end{aligned}$$

[2] This rule includes cases like $\mathsf{Cur}(A, t); g * \mathsf{App}(A_1, B_1) \rightsquigarrow_e \langle \mathsf{Id}, g * \mathsf{Snd}[A] \rangle * t$, in which there is no combinator f or g. If this happens we will simply take $f \equiv \mathsf{Id}$ or $g \equiv \mathsf{Id}$ as appropriate. Such modifications are done in the sequel without being mentioned explicitly.

The equation $\mathsf{Fst} * \mathsf{Cur}(A', s); \mathsf{App}(A', B') = s$ holds only if both sides have the same type, which implies $\Gamma \cdot A' \rhd s : 1 \to B'$. If we apply this condition to the combinator $\mathsf{Fst} * \mathsf{Cur}(f * A, f \cdot \mathsf{Id}[A] * t)$, we get the conditions listed above.

The remaining equations do not create any problems when turned into reduction rules. Table 3.1 contains the definition of the reduction relation $\rightsquigarrow_e$. The relation $\rightsquigarrow_e$ holds between well-formed combinators of the same type, much like the equations. Again, the typing conditions are omitted if they can be derived from the context.

Now we try to check that $\rightsquigarrow_e$ has the properties mentioned at the beginning. It is easy to see that the equivalence relation generated by $\rightsquigarrow_e$ is indeed the equality of combinators:

Lemma 3.1 *For any two combinators e and e' well-formed in context Γ*

$$\Gamma \rhd_e e = e' \textit{ iff } \Gamma \rhd_e e \leftrightsquigarrow_e^* e'$$

Proof

From left to right, an induction over the derivation of $\Gamma \rhd_e e = e'$ shows the claim, and from right to left an induction over the derivation of $\Gamma \rhd_e e \leftrightsquigarrow_e^* e'$ is used. □

The explicit substitution causes a serious problem when we try to prove strong normalization of the reduction $\rightsquigarrow_e$. The problem arises from the rules like $f * \mathsf{Cur}(A, t) \rightsquigarrow_e \mathsf{Cur}(f * A, \langle \mathsf{Fst}; f, \mathsf{Snd}[A] \rangle * t)$, which intuitively correspond in the calculus to pushing substitution inside binding operations like λ and $\forall$. They cause the usual reducibility approach to fail because in all its variants the final induction showing that for every reducible combinator f also $f * t$ is reducible requires for the case $t \equiv \mathsf{Cur}(A, t')$ a lemma similar to

Conjecture 3.2 *If s, A, B and $f * A$ are strongly normalizing and $g * t$ is strongly normalizing for any strongly normalizing g, then*

$$u := f * \mathsf{Cur}(A, t); \langle \mathsf{Id}, s[f * A] \rangle * \mathsf{App}(f * A, B)$$

is also strongly normalizing.

To see where the problem lies, consider the following reduction sequence:

$$\begin{array}{rl} u & \rightsquigarrow_e \; \mathsf{Cur}(f * A, \langle \mathsf{Fst}; f, \mathsf{Snd}[A] \rangle * t); \langle \mathsf{Id}, s[f * A] \rangle * \mathsf{App}(f * A, B) \\ & \rightsquigarrow_e \; \langle \mathsf{Id}, s[f * A] \rangle; \langle \mathsf{Fst}; f, \mathsf{Snd}[A] \rangle * t \end{array}$$

There seems to be no way of deducing from the hypotheses of the lemma that the context morphism $\langle \mathsf{Id}, s[f * A] \rangle; \langle \mathsf{Fst}; f, \mathsf{Snd}[A] \rangle$ is strongly normalizing, so the proof of the conjecture breaks down. On the other hand I have not found any non-terminating reduction sequence. This problem already occurs if we restrict ourselves to categorical combinators for the simply typed λ-calculus, and even in this case there is no proof of strong normalization as yet. It is only known that the reduction corresponding to substitution alone is strongly normalizing [HL86] [CHR92].

The reduction relation $\rightsquigarrow_e$ is the smallest relation that is compatible with the combinators (i.e. it satisfies for example $A \rightsquigarrow_e A'$ implies $\mathsf{Cur}(A,t) \rightsquigarrow_e \mathsf{Cur}(A',t)$). There is one exception: the rule $A \rightsquigarrow_e A'$ implies $f * A \rightsquigarrow_e f * A'$ applies only if $A \not\equiv T$.

1. Indexed Category

$$\begin{array}{rcl}
\langle\rangle & \rightsquigarrow_e & \mathsf{Id}\colon[\,]{\to}[\,] \\
f;\mathsf{Id} & \rightsquigarrow_e & f \\
f;(g;h) & \rightsquigarrow_e & (f;g);h \\
\mathsf{Id} * t & \rightsquigarrow_e & t \\
f * (t;s) & \rightsquigarrow_e & (f * t);(f * s) \\
f * (g * t) & \rightsquigarrow_e & (f;g) * t \\
\mathsf{Id};t & \rightsquigarrow_e & t
\end{array}
\qquad
\begin{array}{rcl}
f;\langle\rangle & \rightsquigarrow_e & \langle\rangle \\
\mathsf{Id};f & \rightsquigarrow_e & f \\
\mathsf{Id} * A & \rightsquigarrow_e & A \\
f * \mathsf{Id} & \rightsquigarrow_e & \mathsf{Id} \\
f * (g * A) & \rightsquigarrow_e & (f;g) * A \\
t;\mathsf{Id} & \rightsquigarrow_e & t \\
u;(t;s) & \rightsquigarrow_e & (u;t);s
\end{array}$$

2. Adjunction $I \dashv G$

$$\begin{array}{rcl}
\langle f, t[A']\rangle;\mathsf{Fst} & \rightsquigarrow_e & f \\
f;\langle g, t[A']\rangle & \rightsquigarrow_e & \langle f;g, f * t[A']\rangle \\
\langle \mathsf{Fst}, \mathsf{Snd}[A']\rangle & \rightsquigarrow_e & \mathsf{Id}
\end{array}
\qquad
\begin{array}{rcl}
\langle f, t[A']\rangle * \mathsf{Snd} & \rightsquigarrow_e & t \\
\langle f;\mathsf{Fst}, f * \mathsf{Snd}[A]\rangle & \rightsquigarrow_e & f
\end{array}$$

3. Dependent Products
   If

   - $\Gamma \rhd_e f\colon \Delta$,
   - $\Delta \cdot A \rhd_e t\colon 1{\to}B$,
   - $f * A = g;\mathsf{Fst} * A_1$ and
   - $\langle \mathsf{Fst};f, \mathsf{Snd}[A]\rangle * B = \langle \mathsf{Fst};g;\mathsf{Fst}, \mathsf{Snd}[A_1]\rangle * B_1$

   then

$$f * \mathsf{Cur}(A,t); g * \mathsf{App}(A_1,B_1) \rightsquigarrow_e \langle f, g * \mathsf{Snd}[A]\rangle * t$$

   Furthermore we have

$$\begin{array}{rcl}
f * \mathsf{Cur}(A,t) & \rightsquigarrow_e & \mathsf{Cur}(f * A, \langle \mathsf{Fst};f, \mathsf{Snd}[A]\rangle * t) \\
f * \Pi(A,B) & \rightsquigarrow_e & \Pi(f * A, \langle \mathsf{Fst};f, \mathsf{Snd}[A]\rangle * B) \\
f * \mathsf{App}(A,B) & \rightsquigarrow_e & \langle \mathsf{Id}, f * \mathsf{Snd}[f;\mathsf{Fst} * A]\rangle * \mathsf{App}(f;\mathsf{Fst} * A, \\
 & & \langle \mathsf{Fst};f;\mathsf{Fst}, \mathsf{Snd}[f;\mathsf{Fst} * A]\rangle * B)
\end{array}$$

Table 3.1: Reduction of Combinators with Explicit Typing

4. Dependent Sums

$$\begin{array}{rcl}
\langle f, t[A], s[B]\rangle * \mathsf{Pair}(A', B'); \pi_1 & \leadsto_e & t \\
\langle f, t[A], s[B]\rangle * \mathsf{Pair}(A', B'); \pi_2 & \leadsto_e & s \\
\langle f, t[A], s[B]\rangle * \mathsf{Pair}(A', B') & \leadsto_e & \langle \mathsf{Id}, f * t[f * A], f \cdot \mathsf{Id}[A'] * s[f \cdot \mathsf{Id}[A'] \\
 & & *B']\rangle * \mathsf{Pair}(f * A', f \cdot \mathsf{Id}[A'] * B') \\
f * \pi_1 & \leadsto_e & \pi_1 \\
f * (t; \pi_2) & \leadsto_e & f * t; \pi_2
\end{array}$$

5. Universal Quantification

$$\begin{array}{rcl}
h * \forall(A, t) & \leadsto_e & \forall(h * A, (h \cdot \mathsf{Id}[A]) * t) \\
\langle\langle\rangle, \forall(A, t)[\Omega]\rangle * T & \leadsto_e & \Pi(A, \langle\langle\rangle, t[\Omega]\rangle * T) \\
\mathsf{Fst}^k * T & \leadsto_e & \langle\langle\rangle, \mathsf{Fst}^k * \mathsf{Snd}[\Omega]\rangle * T \\
\mathsf{Id} * T & \leadsto_e & \langle\langle\rangle, \mathsf{Snd}[\Omega]\rangle * T \\
T & \leadsto_e & \langle\langle\rangle, \mathsf{Snd}[\Omega]\rangle * T
\end{array}$$

Table 3.1 (continued): Reduction of Combinators with Explicit Typing

The solution proposed here is to restrict the reduction in such a way that a combinator is reduced to a form which has no outer β-redexes before the reductions that correspond to pushing the substitution inside the binding operations are applied. Then Conjecture 3.2 becomes true because the problematic reduction sequence is no longer permitted; we have only

$$u \leadsto \langle f, s[A]\rangle * t$$

and the latter is strongly normalizing by assumption.

A similar problem occurs in connection with the reduction rule $g * \mathsf{App}(A, B) \leadsto_e \langle \mathsf{Id}, g * \mathsf{Snd}[g; \mathsf{Fst} * A]\rangle * \mathsf{App}(g; \mathsf{Fst} * A, \langle \mathsf{Fst}; g; \mathsf{Fst}, \mathsf{Snd}[A]\rangle * B)$. We have to show that $f \cdot \mathsf{Id}[g; \mathsf{Fst} * A]; (g; \mathsf{Id}) \cdot \mathsf{Id}[A]$ is strongly normalizing given that $f; g; \mathsf{Fst} * A$ is strongly normalizing, as the following reduction sequence demonstrates:

$$\begin{array}{rcl}
f; g * \mathsf{App}(A, B) & \leadsto_e & f; \langle \mathsf{Id}, g * \mathsf{Snd}[g; \mathsf{Fst} * A]\rangle * \mathsf{App}(g; \mathsf{Fst} * A, \\
 & & (g; \mathsf{Fst}) \cdot \mathsf{Id}[A] * B) \\
 & \leadsto_e^* & \langle f, f; g * \mathsf{Snd}[g; \mathsf{Fst} * A]\rangle * \mathsf{App}(g; \mathsf{Fst} * A, \\
 & & (g; \mathsf{Fst}) \cdot \mathsf{Id}[A] * B) \\
 & \leadsto_e^* & \langle \mathsf{Id}, f; g * \mathsf{Snd}[g; \mathsf{Fst} * A]\rangle * \mathsf{App}(f; g; \mathsf{Fst} * A, \\
 & & f \cdot \mathsf{Id}[g; \mathsf{Fst} * A]; (g; \mathsf{Fst}) \cdot \mathsf{Id}[A] * B)
\end{array}$$

As a remedy, we postpone the reduction inside the App-combinator until we reach a combinator $t; \langle \mathsf{Id}, s[A']\rangle * \mathsf{App}(A, B)$, and introduce a special rule

$$\begin{array}{rcl}
\langle f, t[A'']\rangle * \mathsf{App}(A, \langle \mathsf{Fst}; g, \mathsf{Snd}[A']\rangle * B) & \leadsto_e & \langle \mathsf{Id}, t[f * A]\rangle * \mathsf{App}(f * A, \\
 & & \langle \mathsf{Fst}; f; g, \mathsf{Snd}[A']\rangle * B)
\end{array}$$

to avoid the combinator $f \cdot \mathsf{Id}[g; \mathsf{Fst} * A]; (g; \mathsf{Id}) \cdot \mathsf{Id}[A]$ as a contractum.

These intuitions are captured by two reductions $\overset{W}{\leadsto}$ and $\overset{N}{\leadsto}$. The first describes the reduction to a combinator without any outer β-redex or any reduction inside an App-combinator, and the latter the reduction of those combinators to normal form. The precise definitions are given in Tables 3.2 and 3.3. The crucial part in the definition of $\overset{N}{\leadsto}$ is to choose the right conditions for the compatibility of $\overset{N}{\leadsto}$ with the combinators; these conditions are therefore explicitly stated. Note also that $\overset{W}{\leadsto}$ is contained in $\overset{N}{\leadsto}$.

It is easy to see that any reduction via $\overset{N}{\leadsto}$ is also a reduction via $\leadsto_e$. Before we can examine the relation between $\overset{N}{\leadsto}$ and $\leadsto_e$ more closely, we need the strong normalization of $\overset{N}{\leadsto}$.

Theorem 3.3 *The reduction $\overset{N}{\leadsto}$ is strongly normalizing.*

Proof An adaptation of the reducibility method shows the claim; see appendix A for details. □

Now we turn to the relation between reduction in the calculus and reduction of combinators. First, a reduction in the latter corresponds to several reductions of the former and vice versa. More precisely, we have the following theorems:

Theorem 3.4 *The translation from the calculus into combinators respects reduction, more precisely:*

(Red-Types)	$\Gamma \vdash A \leadsto_e B$ *implies* $[\![\Gamma]\!] \rhd_e [\![A]\!] \leadsto_e^* [\![B]\!]$
(Red-Terms)	$\Gamma \vdash t \leadsto_e s\colon A$ *implies* $[\![\Gamma]\!] \rhd_e [\![t]\!] \leadsto_e^* [\![s]\!]\colon 1 \to [\![A]\!]$

Theorem 3.5 *The inverse translation respects reduction, more precisely for any context Γ with $|\Gamma| = n$ we have:*

(Red-Context Morphisms)	$\Gamma \rhd_e f \leadsto_e f'\colon \Delta$ *implies* $(n,\Gamma)^c \vdash (n,f)^c \leadsto_e^* (n,f')^c$
(Red-Types)	$\Gamma \rhd_e A \leadsto_e A'$ *implies* $(n,\Gamma)^c \vdash (n,A)^c \leadsto_e^* (n,A')^c$
(Red-Terms)	$\Gamma \rhd_e t \leadsto_e t'\colon 1 \to A$ *implies* $(n,\Gamma)^c \vdash (n,t)^c \leadsto_e^* (n,t')^c\colon (n,A)^c$

Proof Both theorems are shown by an induction over the definition of $e \leadsto_e e'$ similar to the proof of theorems 2.10, 2.14 and 2.17. □

Second, not only the reduction steps but also the normal forms via $\overset{N}{\leadsto}$ and $\leadsto_e$ and those in the calculus correspond to each other:

Lemma 3.6 *Both $\leadsto_e$ and $\overset{N}{\leadsto}$ have the same normal forms of types and morphisms $\Gamma \rhd_e t\colon 1 \to A$, namely the translation of the normal forms of types and terms of the Calculus of Constructions. Furthermore, any type A and any morphism $\Gamma \rhd t\colon 1 \to B$ in $\leadsto_e$-normal form satisfies $[\![(n,A)^c]\!] \equiv A$ and $[\![(n,t)^c]\!] \equiv t$ respectively.*

The relation $\overset{W}{\leadsto}$ satisfies the same compatibility condition as $\leadsto_e$ except that no reduction takes place inside the combinator App. Furthermore in the rules (App_1) and (App_2) we have $f \not\equiv f'; \mathsf{Id}$.

Context Morphisms

$$
\begin{array}{rclrclrcl}
f;\mathsf{Id} & \overset{W}{\leadsto} & f & \mathsf{Id};f & \overset{W}{\leadsto} & f & f;(g;h) & \overset{W}{\leadsto} & (f;g);h \\
f;\langle\rangle & \overset{W}{\leadsto} & \langle\rangle & \langle f, t[A]\rangle;\mathsf{Fst} & \overset{W}{\leadsto} & f & f;\langle g, t[A]\rangle & \overset{W}{\leadsto} & \langle f;g, f * t[A]\rangle
\end{array}
$$

Types

$$
\begin{array}{rcl}
f * \Omega & \overset{W}{\leadsto} & \Omega \\
\mathsf{Id} * A & \overset{W}{\leadsto} & A \\
\langle\langle\rangle, h * \forall(A,t)[B]\rangle * T & \overset{W}{\leadsto} & h * \Pi(A, \langle\langle\rangle, t[B]\rangle * T) \\
\langle f, t[B]\rangle * T & \overset{W}{\leadsto} & \langle\langle\rangle, t[B]\rangle * T \\
\mathsf{Fst}^k * T & \overset{W}{\leadsto} & \langle\langle\rangle, \mathsf{Fst}^k * \mathsf{Snd}[\Omega]\rangle * T \\
\mathsf{Id} * T & \overset{W}{\leadsto} & \langle\langle\rangle, \mathsf{Snd}[\Omega]\rangle * T \\
T & \overset{W}{\leadsto} & \langle\langle\rangle, \mathsf{Snd}[\Omega]\rangle * T \\
f * (g * A) & \overset{W}{\leadsto} & (f;g) * A
\end{array}
$$

Morphisms

$$
\begin{array}{rclrclrcl}
f * \mathsf{Id} & \overset{W}{\leadsto} & \mathsf{Id} & t;\mathsf{Id} & \overset{W}{\leadsto} & t & \mathsf{Id};t & \overset{W}{\leadsto} & t \\
f * (t;s) & \overset{W}{\leadsto} & f * t; f * s & f * (g * t) & \overset{W}{\leadsto} & (f;g) * t & t;(s;u) & \overset{W}{\leadsto} & (t;s);u
\end{array}
$$

$$
\begin{array}{rcll}
f * \mathsf{Cur}(A,t); g * \mathsf{App}(A_1,B_1) & \overset{W}{\leadsto} & \langle f, g * \mathsf{Snd}[A]\rangle * t & (\beta) \\
\langle f, t[A'']\rangle * \mathsf{App}(A, \langle \mathsf{Fst}; g, \mathsf{Snd}[A']\rangle * B) & \overset{W}{\leadsto} & \langle \mathsf{Id}, t[f * A]\rangle * \mathsf{App}(f * A, & \\
 & & \langle \mathsf{Fst}; f; g, \mathsf{Snd}[A']\rangle * B) & (\mathsf{App}_1) \\
\langle f, t[A'']\rangle * \mathsf{App}(A,B) & \overset{W}{\leadsto} & \langle \mathsf{Id}, t[f * A]\rangle * \mathsf{App}(f * A, & \\
 & & \langle \mathsf{Fst}; f, \mathsf{Snd}[A]\rangle * B) & (\mathsf{App}_2) \\
\mathsf{Fst}^k * \mathsf{App}(A,B) & \overset{W}{\leadsto} & \langle \mathsf{Id}, \mathsf{Fst}^k * \mathsf{Snd}[A]\rangle * \mathsf{App}(\mathsf{Fst}^{k+1} * A, & \\
 & & \langle \mathsf{Fst}^{k+1}, \mathsf{Snd}[A]\rangle * B) & \\
\langle f, t[A]\rangle * \mathsf{Snd} & \overset{W}{\leadsto} & t & \\
\langle f, t_1[A_1], t_2[A_2]\rangle * \mathsf{Pair}(A,B); \pi_i & \overset{W}{\leadsto} & t_i & \\
f * \pi_1 & \overset{W}{\leadsto} & \pi_1 & \\
f * (t; \pi_2) & \overset{W}{\leadsto} & f * t; \pi_2 &
\end{array}
$$

The β-rule applies only if

- $\Gamma \rhd \langle \mathsf{Fst}; f, \mathsf{Snd}[A]\rangle * t \colon 1 \longrightarrow B$
- $\Gamma \rhd g; \mathsf{Fst} * A_1 \leftrightarrow_e^* f * A$
- $\Gamma \cdot f * A \rhd B \leftrightarrow_e^* \langle \mathsf{Fst}; g; \mathsf{Fst}, \mathsf{Snd}[A_1]\rangle * B_1$

Table 3.2: Reduction to Weak Head-Normal Form

In general, $e \overset{W}{\rightsquigarrow} e'$ implies $e \overset{N}{\rightsquigarrow} e'$.

Contexts

$$\frac{\Gamma \overset{N}{\rightsquigarrow} \Gamma'}{\Gamma \cdot A \overset{N}{\rightsquigarrow} \Gamma' \cdot A} \qquad \frac{A \overset{N}{\rightsquigarrow} A'}{\Gamma \cdot A \overset{N}{\rightsquigarrow} \Gamma \cdot A'}$$

Context Morphisms

$$\frac{f \overset{N}{\rightsquigarrow} f'}{\langle f, t[A] \rangle \overset{N}{\rightsquigarrow} \langle f', t[A] \rangle} \qquad \frac{t \overset{N}{\rightsquigarrow} t'}{\langle f, t[A] \rangle \overset{N}{\rightsquigarrow} \langle f, t'[A] \rangle}$$

$$\frac{A \overset{N}{\rightsquigarrow} A'}{\langle f, t[A] \rangle \overset{N}{\rightsquigarrow} \langle f, t[A'] \rangle} \qquad \frac{f \overset{N}{\rightsquigarrow} f'}{f; \mathsf{Fst} \overset{N}{\rightsquigarrow} f'; \mathsf{Fst}}$$

Types

$$\frac{A \overset{N}{\rightsquigarrow} A'}{\Pi(A,B) \overset{N}{\rightsquigarrow} \Pi(A',B)} \qquad \frac{B \overset{N}{\rightsquigarrow} B'}{\Pi(A,B) \overset{N}{\rightsquigarrow} \Pi(A,B')}$$

$$\frac{A \overset{N}{\rightsquigarrow} A'}{\Sigma(A,B) \overset{N}{\rightsquigarrow} \Sigma(A',B)} \qquad \frac{B \overset{N}{\rightsquigarrow} B'}{\Sigma(A,B) \overset{N}{\rightsquigarrow} \Sigma(A,B')}$$

$$\frac{}{f * \Pi(A,B) \overset{N}{\rightsquigarrow} \Pi(f * A, \langle \mathsf{Fst}; f, \mathsf{Snd}[A] \rangle * B)}$$

$$\frac{}{f * \Sigma(A,B) \overset{N}{\rightsquigarrow} \Sigma(f * A, \langle \mathsf{Fst}; f, \mathsf{Snd}[A] \rangle * B)}$$

$$\frac{f \overset{N}{\rightsquigarrow} f'}{f * T \overset{N}{\rightsquigarrow} f' * T}$$

Morphisms

$$\frac{A \overset{N}{\rightsquigarrow} A'}{\mathsf{Cur}(A,t) \overset{N}{\rightsquigarrow} \mathsf{Cur}(A',t)} \qquad \frac{t \overset{N}{\rightsquigarrow} t'}{\mathsf{Cur}(A,t) \overset{N}{\rightsquigarrow} \mathsf{Cur}(A,t')}$$

$$\frac{A \overset{N}{\rightsquigarrow} A'}{\forall(A,t) \overset{N}{\rightsquigarrow} \forall(A',t)} \qquad \frac{t \overset{N}{\rightsquigarrow} t'}{\forall(A,t) \overset{N}{\rightsquigarrow} \forall(A,t')}$$

$$\frac{f \overset{N}{\rightsquigarrow} f'}{f * \mathsf{Snd} \overset{N}{\rightsquigarrow} f' * \mathsf{Snd}} \qquad \frac{u \overset{N}{\rightsquigarrow} u'}{\mathsf{Fst}^k * \mathsf{Snd}; t; u; s \overset{N}{\rightsquigarrow} \mathsf{Fst}^k * \mathsf{Snd}; t; u'; s}$$

$$\frac{}{f * \mathsf{Cur}(A,t) \overset{N}{\rightsquigarrow} \mathsf{Cur}(f * A, \langle \mathsf{Fst}; f, \mathsf{Snd}[A] \rangle * t)}$$

$$\frac{}{f * \forall(A,t) \overset{N}{\rightsquigarrow} \forall(f * A, \langle \mathsf{Fst}; f, \mathsf{Snd}[A] \rangle * t)}$$

Table 3.3: Reduction to Normal Form

$$\begin{array}{lcl}
f * \mathsf{App}(A,B) & \overset{N}{\rightsquigarrow} & \langle \mathsf{Id}, f * \mathsf{Snd}[f;\mathsf{Fst} * A]\rangle * \mathsf{App}(f;\mathsf{Fst} * A, \\
 & & \langle \mathsf{Fst}; f; \mathsf{Fst}, \mathsf{Snd}[f;\mathsf{Fst} * A]\rangle * B) \\
g * \mathsf{Pair}(A,B) & \overset{N}{\rightsquigarrow} & \langle \mathsf{Id}, g;\mathsf{Fst} * \mathsf{Snd}[g;\mathsf{Fst};\mathsf{Fst} * A], g * \mathsf{Snd}[\langle g;\mathsf{Fst};\mathsf{Fst},\mathsf{Snd} \\
 & & [A]\rangle * B]\rangle * \mathsf{Pair}(g;\mathsf{Fst};\mathsf{Fst} * A, \langle g;\mathsf{Fst};\mathsf{Fst},\mathsf{Snd}[A]\rangle * B)
\end{array}$$

$$\frac{t \overset{N}{\rightsquigarrow} t'}{\langle \mathsf{Id}, t[A], s[B]\rangle * \mathsf{Pair}(A_1,B_1) \overset{N}{\rightsquigarrow} \langle \mathsf{Id}, t'[A], s[B]\rangle * \mathsf{Pair}(A_1,B_1)}$$ [3]

$$\frac{t \overset{N}{\rightsquigarrow} t'}{\langle \mathsf{Id}, t[C]\rangle * \mathsf{App}(A,B) \overset{N}{\rightsquigarrow} \langle \mathsf{Id}, t'[C]\rangle * \mathsf{App}(A,B)}$$ [4]

Table 3.3 (continued): Reduction to Normal Form

[3] Similar inference rules apply for the reduction of A, s, B, A_1 and B_1 via $\overset{N}{\rightsquigarrow}$.

[4] Similar inference rules apply for the reduction of C, A and B via $\overset{N}{\rightsquigarrow}$.

Proof An induction over the structure of types and morphisms shows that the set $\mathcal{N}$ of $\rightsquigarrow_e$- and $\overset{N}{\rightsquigarrow}$- normal forms of types and morphisms is given inductively as follows:

Types

$$\frac{}{\Omega \in \mathcal{N}} \qquad \frac{t \in \mathcal{N} \quad t \not\equiv f * \forall(A,t')}{\langle\langle\rangle, t[\Omega]\rangle \in \mathcal{N}} \qquad \frac{A \in \mathcal{N} \quad B \in \mathcal{N}}{\Pi(A,B) \in \mathcal{N}} \qquad \frac{A \in \mathcal{N} \quad B \in \mathcal{N}}{\Sigma(A,B) \in \mathcal{N}}$$

Morphisms

$$\frac{A \in \mathcal{N} \quad t \in \mathcal{N}}{\mathsf{Cur}(A,t) \in \mathcal{N}} \qquad \frac{A \in \mathcal{N} \quad t \in \mathcal{N}}{\forall(A,t) \in \mathcal{N}}$$

$$\frac{t \in \mathcal{N} \quad t \not\equiv \mathsf{Cur}(B',t') \quad s \in \mathcal{N} \quad A, A', B \in \mathcal{N}}{t; \langle \mathsf{Id}, s[A']\rangle * \mathsf{App}(A,B) \in \mathcal{N}}$$

$$\frac{}{\mathsf{Fst}^n * \mathsf{Snd} \in \mathcal{N}} \qquad \frac{t, s \in \mathcal{N} \quad A, A', B, B' \in \mathcal{N}}{\langle \mathsf{Id}, t[A'], s[B']\rangle * \mathsf{Pair}(A,B) \in \mathcal{N}} \qquad \frac{t \in \mathcal{N} \quad t \not\equiv f * \mathsf{Pair}}{t; \pi_i \in \mathcal{N}}$$

This shows the claim except for the cases $t; \langle \mathsf{Id}, s[A']\rangle * \mathsf{App}(A,B)$ and $\langle \mathsf{Id}, t[A'], s[B']\rangle * \mathsf{Pair}(A,B)$. In both cases it remains to show that A and A' as well as B and B' are identical. But the conditions for well-formed combinators imply that $\Gamma \rhd_e A = A'$ and $\Gamma \cdot A \rhd_e B = B'$. So by induction hypothesis $(n,A)^c$ and $(n,A')^c$ as well as $(n+1,B)^c$ and $(n+1,B')^c$ are equal normal forms in the Calculus of Constructions, hence identical because of the confluence of the calculus. So by induction hypothesis again,

$$A \equiv [\![(n,A)^c]\!] \equiv [\![(n,A')^c]\!] \equiv A'$$

and

$$B \equiv [\![(n+1, B)^c]\!] \equiv [\![(n+1, B')^c]\!] \equiv B'$$

□

The last property of $\overset{N}{\rightsquigarrow}$ and $\rightsquigarrow_e$ to be checked is the confluence. As we will see in a moment, it holds for types and morphisms but fails for context morphisms because not all η-like rules for the latter are included in $\rightsquigarrow_e$ and $\overset{N}{\rightsquigarrow}$. But we can give an easy criterion when two context morphisms are equal. The confluence of the calculus is crucial in showing the following theorem:

Theorem 3.7 (i) *The reductions* $\rightsquigarrow_e$ *and* $\overset{N}{\rightsquigarrow}$ *are confluent on types and morphisms.*

(ii) *Two context morphisms* $f\colon \Gamma \rightarrow \Delta$ *and* $f'\colon \Gamma \rightarrow \Delta$ *are equal iff for any* $\overset{N}{\rightsquigarrow}$*-normal form* g *of* f *and* g' *of* f' *at least one of the following alternatives holds:*

- $|\Delta| = 0$.
- $g \equiv \mathsf{Fst}^n$ *and* $g' \equiv \langle g'', \mathsf{Fst}^{k-1} * \mathsf{Snd}[A_{k-1}], \ldots, \mathsf{Snd}[A_0]\rangle$ *with* $g'' \equiv \mathsf{Fst}^k$ *or* $g'' \equiv \langle\rangle$*, or vice versa.*
- $g \equiv \mathsf{Id}$ *and* $g' \equiv \langle g'', \mathsf{Fst}^{k-1} * \mathsf{Snd}[A_{k-1}], \ldots, \mathsf{Snd}[A_0]\rangle$ *with* $g'' \equiv \mathsf{Fst}^k$ *or* $g'' \equiv \langle\rangle$*, or vice versa.*
- $g \equiv \langle g_1, t_1[A_1]\rangle$ *and* $g' \equiv \langle g_2, t_2[A_2]\rangle$ *where* g_1 *and* g_2 *satisfy the same conditions as* g *and* g' *and* $t_1 \equiv t_2$ *as well as* $A_1 \equiv A_2$.

Proof

(i) Given $A \rightsquigarrow_e^* A_1$ and $A \rightsquigarrow_e^* A_2$ with $\Gamma \rhd_e A$ and $|\Gamma| = n$. Let A_3 and A_4 be normal forms of A_1 and A_2 via $\overset{N}{\rightsquigarrow}$ respectively. Theorem 3.5 implies that

$$(n, A)^c \rightsquigarrow_e^* (n, A_1)^c \rightsquigarrow_e^* (n, A_3)^c \text{ and } (n, A)^c \rightsquigarrow_e^* (n, A_2)^c \rightsquigarrow_e^* (n, A_4)^c$$

By Lemma 3.6, the types $(n, A_3)^c$ and $(n, A_4)^c$ are normal forms in the Calculus of Constructions, hence identical because of the confluence. Therefore

$$A_3 \equiv [\![(n, A_3)^c]\!] \equiv [\![(n, A_4)^c]\!] \equiv A_4$$

The same argument applies for morphisms.

(ii) If one of the above alternatives holds, then f and f' are certainly equal. To show the other direction, let $\Gamma \rhd_e f\colon \Delta$, and let f and f' be equal. We can assume without loss of generality that f and f' are in $\overset{N}{\rightsquigarrow}$-normal form. Those $\overset{N}{\rightsquigarrow}$-normal forms of context morphisms are

- $\langle\rangle$
- Fst^k

- Id
- $\langle g, t[A]\rangle$ with g, t and A normal forms.

Now observe that $f = f'$ iff $f;\mathsf{Fst}^k * \mathsf{Snd} = f';\mathsf{Fst}^k * \mathsf{Snd}$ for all k such that $0 \leq k \leq |\Delta|$. This leaves only the cases cited in the theorem.

□

3.1.2 Combinators with Implicitly Typed Application

Now we look at the problem of eliminating the equality check in the reduction $f * \mathsf{Cur}(A,t); g * \mathsf{App}(A_1, B_1) \leadsto_e \langle f, g * \mathsf{Snd}[A]\rangle * t$. As already mentioned at the beginning of the last section, this problem is solved in the calculus by using the confluence and replacing the typed application $\mathsf{App}(A', B', \lambda A.t, s)$ by an untyped one $(\lambda A.t)s$. The same idea applies to the combinators: we simplify the combinator $\mathsf{App}(A, B)$ to App. For clarity, we will refer to the combinators discussed until now as "combinators with explicit typing" and to the combinators introduced in this section as "combinators with implicit typing". The reduction relation $\leadsto_i$ corresponds to the relation $\leadsto_e$ for combinators with explicit typing. However, this change alone does not remove the need for the abstract machines to store the context Δ during a reduction of a combinator $g * \mathsf{App}$ with $\Gamma \rhd_i g\colon \Delta$. This is demonstrated by the reduction rule

$$g * \mathsf{App}(A, B) \leadsto_e \langle \mathsf{Id}, g * \mathsf{Snd}[g;\mathsf{Fst} * A]\rangle * \mathsf{App}(g;\mathsf{Fst} * A, \langle \mathsf{Fst}; g; \mathsf{Fst}, \mathsf{Snd}[A]\rangle * B)$$

which corresponds to pushing substitution inside the term $\mathsf{App}(A, B, t, s)$. It becomes $g * \mathsf{App} \leadsto_i \langle \mathsf{Id}, g * \mathsf{Snd}[g;\mathsf{Fst} * A]\rangle * \mathsf{App}$ in the implicit combinators. So if $g \equiv \mathsf{Fst}^k$ this reduction can only be executed in the machine if the type A is stored in some register. But because A is unique up to derivable equality, we can simply omit it and introduce a combinator $\langle \mathsf{Id}, t\rangle * \mathsf{App}$ together with the reduction rule

$$g * \mathsf{App} \leadsto_i \langle \mathsf{Id}, g * \mathsf{Snd}\rangle * \mathsf{App}$$

We carry out one further step in eliminating superfluous type information. Consider the combinator

$$[\![\mathsf{Pair}(A, B, t, s)]\!] \equiv \langle \mathsf{Id}, [\![t]\!][[\![A]\!]], [\![s]\!][[\![B]\!]]\rangle * \mathsf{Pair}([\![A]\!], [\![B]\!])$$

The types $[\![A]\!]$ and $[\![B]\!]$ appear twice, and so the type checking algorithm in chapter 5 performs a superfluous check whether the types $[\![A]\!]$ and $[\![A]\!]$ as well $[\![B]\!]$ and $[\![B]\!]$ are equal. We avoid this by replacing the combinator $\mathsf{Pair}(A, B)$ by Pair. As in the case of the App-combinator, the reduction rule corresponding to pushing substitution inside the term $\mathsf{Pair}(A, B, t, s)$

$$\begin{aligned} g * \mathsf{Pair} \ \leadsto_i\ & \langle \mathsf{Id}, g;\mathsf{Fst} * \mathsf{Snd}[g;\mathsf{Fst};\mathsf{Fst} * A], g * \mathsf{Snd}[\langle g;\mathsf{Fst};\mathsf{Fst}, \mathsf{Snd} \\ & [A]\rangle * B]\rangle * \mathsf{Pair} \end{aligned}$$

makes it necessary to store the context Δ in the machine if a combinator $\Gamma \rhd_i g\colon \Delta$ is reduced. In contrast to the reduction rule for $g * \mathsf{App}$ it is not possible to remove the types in the contractum completely because the combinator B is not uniquely derivable from g. The solution in this case is to restrict ourselves to combinators $\langle f, t[A], s[B]\rangle * \mathsf{Pair}$ in the implicit system. The price we pay is a more complicated function $\mathtt{str}$, which assigns to every combinator in the explicit system one in the implicit one. If $f \not\equiv g; \langle f', t[A], s[B]\rangle$, we cannot simply define $\mathtt{str}(f * \mathsf{Pair}(A, B)) = \mathtt{str}(f) * \mathsf{Pair}$ but we have to use the equalities

$$h = \langle h; \mathsf{Fst}; \mathsf{Fst}, h; \mathsf{Fst} * \mathsf{Snd}[A], h * \mathsf{Snd}[B]\rangle$$

if $\Gamma \rhd_i h\colon \Delta \cdot A \cdot B$ and

$$\langle h, t[B]\rangle = \langle h; \mathsf{Fst}, h * \mathsf{Snd}[A], t[B]\rangle$$

if $\Gamma \rhd_i h\colon \Delta \cdot A$ to find a combinator $g; \langle f', t[A], s[B]\rangle$ equal to f. This leads to the following definition:

Definition 3.8 *The combinators with implicit typing are given as follows:*

(i) *The raw combinators with implicit typing are given as follows:*

$$\begin{array}{rcl}
\Gamma & ::= & [\,] \mid \Gamma \cdot A \\
f & ::= & \langle\rangle \mid \mathsf{Id} \mid f;f \mid \mathsf{Fst} \mid \langle f, t[A]\rangle \\
A & ::= & f * A \mid \Pi(A, A) \mid \Sigma(A, A) \mid \Omega \mid T \\
t & ::= & \mathsf{Id} \mid t;t \mid f * t \mid \mathsf{Snd} \mid \mathsf{Cur}(A, t) \mid \mathsf{App} \mid \langle \mathsf{Id}, t\rangle * \mathsf{App} \mid \forall(A, t) \\
 & & \mid \langle\langle f, t[A]\rangle, s[B]\rangle * \mathsf{Pair} \mid \pi_1 \mid \pi_2
\end{array}$$

(ii) *The equational theory $\rhd_i$ of combinators with implicit typing is defined by replacing in the system $\rhd_e$ the rules involving $\mathsf{App}(A, B)$ and $\mathsf{Pair}(A, B)$ by*

$$(\mathsf{App})\quad \frac{\Gamma \rhd_i A \qquad \Gamma \cdot A \rhd_i B}{\Gamma \cdot A \rhd_i \mathsf{App}\colon \mathsf{Fst} * \Pi(A, B) \to B}$$

$$(\mathsf{App})\quad \frac{\Gamma \rhd_i t\colon 1 \to A \qquad \Gamma \cdot A \rhd_i B}{\Gamma \rhd \langle \mathsf{Id}, t\rangle * \mathsf{App}\colon \Pi(A, B) \to \langle \mathsf{Id}, t[A]\rangle * B}$$

$$(\mathsf{Pair})\quad \frac{\Gamma \rhd_i \langle f, t_1[A], t_2[B]\rangle\colon \Delta \cdot A' \cdot B'}{\Gamma \rhd_i \langle f, t_1[A], t_2[B]\rangle * \mathsf{Pair}\colon 1 \to \Sigma(A, B)}$$

(iii) *The reduction rules are those for $\leadsto_e$, with the exceptions of the rules involving $\mathsf{App}(A, B)$ or $\mathsf{Pair}(A, B)$, which are replaced by*

$$\begin{array}{rcll}
f * \mathsf{Cur}(A, t); g * \mathsf{App} & \leadsto_i & \langle f, g * \mathsf{Snd}[A]\rangle * t & \textit{if } \Gamma \rhd_i g * \mathsf{Snd}\colon 1 \to A_1 \textit{ and} \\
 & & & \Gamma \rhd_i A_1 = f * A \\
g * \mathsf{App} & \leadsto_i & \langle \mathsf{Id}, g * \mathsf{Snd}\rangle * \mathsf{App} &
\end{array}$$

and

$$\begin{array}{rcl}
\langle f, t_1[A], t_2[B]\rangle * \mathsf{Pair}; \pi_i & \leadsto_i & t_i \\
\langle f, t[A], s[B]\rangle * \mathsf{Pair} & \leadsto_i & \langle \mathsf{Id}, t[f * A], s[\langle \mathsf{Fst}; f, \mathsf{Snd}[A]\rangle * B]\rangle * \mathsf{Pair}
\end{array}$$

(iv) *The map* $\mathtt{str}$ *takes a combinator with explicit typing and replaces*

- $\mathsf{App}(A,B)$ *by* App
- $g;\langle f,t[A'],s[B']\rangle * \mathsf{Pair}(A,B)$ *by* $\mathtt{str}(g);\langle\mathtt{str}(f),\mathtt{str}(t)[\mathtt{str}(A)],\mathtt{str}(s)[\mathtt{str}(B)]\rangle * \mathsf{Pair}$
- $g;\langle f,t[A']\rangle * \mathsf{Pair}(A,B)$ *by* $\mathtt{str}(g);\langle\mathtt{str}(f);\mathsf{Fst},\mathtt{str}(f);\mathsf{Fst} * \mathsf{Snd}[\mathtt{str}(A)],\mathtt{str}(t)[\mathtt{str}(B)]\rangle * \mathsf{Pair}$
- $\mathsf{Fst}^k * \mathsf{Pair}(A,B)$ *by* $\langle\mathsf{Fst}^{k+2},\mathsf{Fst} * \mathsf{Snd}[\mathtt{str}(A)],\mathsf{Snd}[\mathtt{str}(B)]\rangle * \mathsf{Pair}$
- $\mathsf{Pair}(A,B)$ *by* $\langle\mathsf{Fst};\mathsf{Fst},\mathsf{Fst} * \mathsf{Snd}[\mathtt{str}(A)],\mathsf{Snd}[\mathtt{str}(B)]\rangle * \mathsf{Pair}$

The confluence for the explicit combinators is essential in showing that the implicit combinators are merely a convenient shortcut for the explicit ones. One direction is easy:

Lemma 3.9 *For all judgements* $\Gamma \rhd_e e$ *in the theory of explicit combinators, we have*

$$\mathtt{str}(\Gamma) \rhd_i \mathtt{str}(e)$$

Proof Induction over the derivation of the judgement $\Gamma \rhd_e e$. □

The more important direction is given by the following theorem:

Theorem 3.10 (i) *Every context* Γ, Δ, *context morphism* f, *type* A *and morphism* t *in the implicit system can be uniquely extended to the explicit system in the sense that there exist a context* Γ^e *and* Δ^e, *a context morphism* f^e, *a type* A^e *and a morphism* t^e *such that*

1. $\Gamma \in \mathrm{Obj}$ *implies* $\Gamma^e \in \mathrm{Obj}$.
2. $\Gamma \rhd_i f{:}\,\Delta$ *implies* $\Gamma^e \rhd_e f^e{:}\,\Delta^e$.
3. $\Gamma \rhd_i A$ *implies* $\Gamma^e \rhd_e A^e$.
4. $\Gamma \rhd_i t{:}\,1{\to}A$ *implies* $\Gamma^e \rhd_e t^e{:}\,1{\to}B^e$.
5. $\mathtt{str}(\Gamma^e)\equiv\Gamma$, $\mathtt{str}(\Delta^e)\equiv\Delta$, $\mathtt{str}(f)\equiv f$, $\mathtt{str}(A)\equiv A$, *and* $\mathtt{str}(t)\equiv t$ [5]
6. Γ^e, Δ^e, f^e, A^e *and* t^e *are unique up to convertibility in the explicit system.*

(ii) *If* $\Gamma \rhd_i e = e'$ *then* e *and* e' *can be uniquely extended to the explicit system, and for any extension* e^e *of* e *and* e'^e *of* e', *we have* $\Gamma^e \rhd_e e^e = e'^e$.

(iii) *If* e *is a well-formed combinator in the implicit system and reduces as a raw combinator to* e', *then* e' *is well-formed as well. Furthermore for any combinator* e^e *with* $\mathtt{str}(e^e)\equiv e$, *there exists a combinator* e'^e *such that* $\mathtt{str}(e'^e)\equiv e'$ *and* $e^e \leadsto^+_e e'^e$.

[5] possibly replacing $\langle\mathsf{Id},t[A]\rangle * \mathsf{App}$ by $\langle\mathsf{Id},t\rangle * \mathsf{App}$

Proof Parts (i) and (ii) have to be shown by a simultaneous induction over the derivation of $\Gamma \rhd_i e$. We consider only the more difficult cases here.

$t; g * \mathsf{App}$ By induction hypothesis, $\Gamma \rhd_i t{:}\, 1 {\rightarrow} C$, $\Gamma \rhd_i g{:}\, \Delta \cdot A_1$ and there exists a type B_1 such that $\Delta \cdot A_1 \rhd_i B_1$ and $\Gamma \rhd_i g; \mathsf{Fst} * \Pi(A_1, B_1) = \Pi(A, B)$, and moreover $\Gamma^e \rhd_e g^e * \mathsf{Fst} * \Pi(A_1^e, B_1^e) = \Pi(A^e, B^e)$. The confluence of the explicit system also yields $\Gamma^e \rhd_e g^e; \mathsf{Fst} * A_1^e = A^e$ and $\Gamma^e \rhd_e \langle \mathsf{Fst}; g^e; \mathsf{Fst}, \mathsf{Snd}[A_1^e] \rangle * B_1^e$, and so $t^e; g^e * \mathsf{App}(A_1^e, B_1^e)$ is well-formed. Therefore we can define $(t; g * \mathsf{App})^e \overset{\text{def}}{=} t^e; g^e * \mathsf{App}(A_1^e, B_1^e)$. Now let us assume that $\Gamma^e \rhd_e t^e; g^e * \mathsf{App}(A_2^e, B_2^e)$ is another extension of $\Gamma \rhd_i t; g * \mathsf{App}$. Therefore $\Gamma^e \rhd_e g^e; \mathsf{Fst} * \Pi(A_2^e, B_2^e) = \Pi(A^e, B^e)$, and so

$$\Gamma^e \rhd_e g^e * \mathsf{App}(A_2^e, B_2^e) = \langle \mathsf{Id}, g^e * \mathsf{Snd}[A^e] \rangle * \mathsf{App}(A^e, B^e) = g^e * \mathsf{App}(A_1^e, B_1^e)$$

With respect to the equations, we demonstrate only two cases.

1. $f * \mathsf{Cur}(A, t) = \mathsf{Cur}(f * A, \langle \mathsf{Fst}; f, \mathsf{Snd}[A] \rangle * t)$:
   By induction hypothesis we know that

   $$\Gamma^e \rhd_e f^e{:}\, \Delta^e, \quad \Delta^e \rhd_e A^e \text{ and } \Delta^e \cdot A^e \rhd_e t^e{:}\, 1 {\rightarrow} B^e$$

   Therefore

   $$\Gamma^e \rhd_e f^e * \mathsf{Cur}(A^e, t^e) = \mathsf{Cur}(f^e * A^e, \langle \mathsf{Fst}; f^e, \mathsf{Snd}[A^e] \rangle * t^e)$$

   The uniqueness condition for redex and contractum is obvious.

2. $\mathsf{Fst} * \mathsf{Cur}(A, t); \mathsf{App} = t$:
   The induction hypothesis yields $\Gamma^e \cdot A^e \rhd_e t^e{:}\, 1 {\rightarrow} B^e$. Therefore

   $$\mathsf{Fst} * \mathsf{Cur}(A^e, t^e); \mathsf{App}(A^e, B^e) = t^e$$

   and a similar argument as in the case $t; g * \mathsf{App}$ shows that the combinator $\mathsf{Fst} * \mathsf{Cur}(A, t); \mathsf{App}$ can be uniquely extended to the explicit system.

Finally, consider part (iii). The critical case is again the β-rule

$$f * \mathsf{Cur}(A, t); g * \mathsf{App} \rightsquigarrow_i \langle \mathsf{Id}, g * \mathsf{Snd}[f * A] \rangle * t$$

By part 4, $\Gamma^e \rhd_e f^e * \mathsf{Cur}(A^e, t^e){:}\, 1 {\rightarrow} C^e$ with $\Gamma^e \rhd_e C^e = f^e * \Pi(A^e, B^e)$, and now an argument similar to the existence proof of $(t; g * \mathsf{App})^e$ shows that the β-rule applies. □

The restriction of $\rightsquigarrow_e$ that leads to the reductions $\overset{W}{\rightsquigarrow}$ and $\overset{N}{\rightsquigarrow}$ applies similarly to the reduction $\rhd_i$ and yields reductions $\overset{W}{\rightsquigarrow}$ and $\overset{N}{\rightsquigarrow}$ for combinators with implicit typing. Theorem 3.10 implies that Theorems 3.3 and 3.7 hold for the implicit system as well and that the normal forms in the implicit system expand to the normal forms in the explicit system.

3.2 Possible Strategies

Abstract machines require not only a notion of reduction but also a strategy for choosing which reduction to execute next. The intuition behind the strategies given below is to describe how a combinator of the form $\langle\langle\rangle, t_n[A_n], \ldots, t_0[A_0]\rangle * t$ that corresponds to the substitution of the environment $\{(t_n)^c, \ldots, (t_0)^c\}$ in the term $(t)^c$ can be reduced to a normal form. More precisely, a combinator is reduced according to them first to a so-called canonical one, which is a combinator corresponding to a substitution of an environment in a translation of a CC-expression in weak head-normal form. The second step, namely the reduction of a canonical combinator to its normal form, proceeds by pushing the substitution inside the binding operation and applying the first step again. Because $\overset{N}{\leadsto}$ is strongly normalizing, these procedures always terminate with a normal from. Therefore it is enough to describe strategies for reduction to canonical combinators.

Three factors play a key role in the selection of a reduction strategy:

- Evaluating the combinators f and t inside $\langle f, t[A]\rangle$ or not. The first choice corresponds to an eager strategy, where every canonical context morphism $\langle\langle\rangle, t_n[A_n], \ldots, t_0[A_0]\rangle$ always contains canonical morphisms t_i, and the second to a lazy one with possibly unevaluated expressions t_i.

- Evaluating the environment f independently of the morphism t in an expression $f * t$ or not. The first choice is appropriate for an eager strategy and the second for a lazy one because in the lazy case t determines if an evaluation of a component of f to a value is necessary or not.

- Evaluating t_1 or t_2 first in an expression $t_1; t_2$.

The third factor is independent of the other two, and a choice may lead to some optimizations, but not to principal differences. If the first choice is adopted for an eager strategy, we get a strategy $\Rightarrow_E$, which yields an abstract machine that generalizes the CAM. The other choice together with a lazy strategy leads to a strategy $\Rightarrow_L$ that is the basis for another abstract machine generalizing Krivine's machine.

The difference between the reduction strategies becomes apparent when we look at the way the access to an environment and the application are handled. The strategy $\Rightarrow_E$ can directly return the appropriate component of the environment in the first case whereas the lazy strategy has to schedule a reduction of the component as well. In the case of a combinator $t = f * (t_1; \langle \mathsf{Id}, t_2\rangle * \mathsf{App})$, the eager strategy reduces $f * t_1$ to a combinator s and the combinator $f * t_2$ to s_2. If $s \neq h * \mathsf{Cur}(A, s_1)$, then the result is $(h * s); \langle \mathsf{Id}, s_2\rangle * \mathsf{App}$, otherwise the combinator $\langle h, s_2[A]\rangle * s_1$ is reduced to produce the result. The lazy strategy reduces only $f * t_1$ to determine if a combinator $h * \mathsf{Cur}(A, s_1)$ results and if it does reduces the combinator $\langle h, t_2\rangle * s_1$. For a precise definition of the canonical combinators in the lazy case see Table 3.4, and for the eager case see Table 3.5. All inference rules are listed in Tables 3.6 and 3.7. All four tables specify inference rules for raw combinators with implicit typing.

Let $\pi^{(k)}$ be any composition of π_1 and π_2.

Context Morphisms

$$\frac{}{\langle\rangle \in \mathcal{C}} \qquad \frac{}{\mathsf{Fst}^k; \langle f, t[A]\rangle \in \mathcal{C}} \qquad \frac{}{\mathsf{Id} \in \mathcal{C}}$$

$$\frac{}{\mathsf{Fst}^n \in \mathcal{C}} \qquad \frac{f; \langle g, t[A]\rangle \in \mathcal{C}}{f; \langle g, t[A]\rangle; \langle h, s[B]\rangle \in \mathcal{C}}$$

Types

$$\frac{f \in \mathcal{C}}{f * \Pi(A, B) \in \mathcal{C}} \qquad \frac{f \in \mathcal{C}}{f * \Sigma(A, B) \in \mathcal{C}}$$

$$\frac{}{\Omega \in \mathcal{C}} \qquad \frac{t \in \mathcal{C}}{\langle\langle\rangle, t[A]\rangle * T \in \mathcal{C}} \; (t \not\equiv f * \forall(A', t'))$$

Morphisms

$$\frac{f \in \mathcal{C}}{f * \mathsf{Cur}(A, t) \in \mathcal{C}} \qquad \frac{f \in \mathcal{C}}{f * \forall(A, t) \in \mathcal{C}}$$

$$\frac{}{\mathsf{Fst}^k * \mathsf{Snd}; \pi^{(0)}; \langle \mathsf{Id}, t_1\rangle * \mathsf{App}; \pi^{(1)} \cdots ; \langle \mathsf{Id}, t_n\rangle * \mathsf{App}; \pi^{(n)} \in \mathcal{C}}$$

$$\frac{}{\mathsf{Fst}^n * \mathsf{Snd} \in \mathcal{C}} \qquad \frac{f \in \mathcal{C}}{f * \mathsf{Pair} \in \mathcal{C}} \qquad \frac{}{\mathsf{Fst}^k * \mathsf{Snd}; \pi^{(k)} \in \mathcal{C}}$$

Table 3.4: Canonical Combinators for the Lazy Reduction

Let $\pi^{(k)}$ be any composition of π_1 and π_2.

Context Morphisms

$$\frac{}{\langle\rangle \in \mathcal{C}} \qquad \frac{f \in \mathcal{C} \qquad t \in \mathcal{C}}{\mathsf{Fst}^k; \langle f, t[A]\rangle \in \mathcal{C}} \qquad \frac{}{\mathsf{Id} \in \mathcal{C}} \qquad \frac{}{\mathsf{Fst}^n \in \mathcal{C}}$$

Types

$$\frac{f \in \mathcal{C}}{f * \Pi(A, B) \in \mathcal{C}} \qquad \frac{f \in \mathcal{C}}{f * \Sigma(A, B) \in \mathcal{C}}$$

$$\frac{}{\Omega \in \mathcal{C}} \qquad \frac{t \in \mathcal{C}}{\langle\langle\rangle, t[A]\rangle * T \in \mathcal{C}}\ (t \not\equiv f * \forall(A', t'))$$

Morphisms

$$\frac{f \in \mathcal{C}}{f * \mathsf{Cur}(A, t) \in \mathcal{C}} \qquad \frac{f \in \mathcal{C}}{f * \forall(A, t) \in \mathcal{C}}$$

$$\frac{t_1 \in \mathcal{C}, \ldots, t_n \in \mathcal{C}}{\mathsf{Fst}^k * \mathsf{Snd}; \pi^{(0)}; \langle \mathsf{Id}, t_1\rangle * \mathsf{App}; \pi^{(1)} \cdots ; \langle \mathsf{Id}, t_n\rangle * \mathsf{App}; \pi^{(n)} \in \mathcal{C}}$$

$$\frac{}{\mathsf{Fst}^n * \mathsf{Snd} \in \mathcal{C}} \qquad \frac{f \in \mathcal{C}}{f * \mathsf{Pair} \in \mathcal{C}} \qquad \frac{}{\mathsf{Fst}^k * \mathsf{Snd}; \pi^{(k)} \in \mathcal{C}}$$

Table 3.5: Canonical Combinators for Eager Reduction

A later rule applies only if all prior rules fail. f always denotes a canoncial context morphism.

Context Morphisms

$$\frac{f;g \Rightarrow_L f' \qquad f';h \Rightarrow_L h'}{f;(g;h) \Rightarrow_L h'} \qquad \frac{}{f;\langle\rangle \Rightarrow_L \langle\rangle} \qquad \frac{}{f;\mathsf{Id} \Rightarrow_L f}$$

$$\frac{f';\langle g, t[A]\rangle \in \mathcal{C} \qquad f';g \Rightarrow_L h}{f;\mathsf{Fst} \Rightarrow_L h} \qquad \frac{}{\mathsf{Id};g \Rightarrow_L g} \qquad \frac{}{f;g \Rightarrow_L f;g}$$

Types

$$\frac{f*((h;g)*A) \Rightarrow_L B}{f*(h*(g*A)) \Rightarrow_L B} \qquad \frac{}{f*(g*\Omega) \Rightarrow_L \Omega} \qquad \frac{}{f*\Omega \Rightarrow_L \Omega}$$

$$\frac{f;g \Rightarrow_L h \qquad h*A \Rightarrow_L B}{f*(g*A) \Rightarrow_L B} \qquad \frac{f*\mathsf{Snd} \Rightarrow_L h*\forall(A,t)}{f*T \Rightarrow_L h*\Pi(A,\langle\langle\rangle, t[\Omega]\rangle * T)}$$

$$\frac{f*\mathsf{Snd} \Rightarrow_L t}{f*T \Rightarrow_L \langle\langle\rangle, t[\Omega]\rangle * T} \qquad \frac{}{f*A \Rightarrow_L f*A}$$

Morphisms

$$\frac{f*((t;s);u) \Rightarrow_L t'}{f*(t;(s;u)) \Rightarrow_L t'} \qquad \frac{f*(h;g)*t \Rightarrow_L s}{f*(h*(g*t)) \Rightarrow_L s}$$

$$\frac{f*t \Rightarrow_L s}{f*(t;g*\mathsf{Id}) \Rightarrow_L s} \qquad \frac{f*t \Rightarrow_L s}{f*(g*\mathsf{Id};t) \Rightarrow_L s}$$

$$\frac{f;\langle g, t[A]\rangle \in \mathcal{C} \qquad f*t \Rightarrow_L s}{f;\langle g, t[A]\rangle * \mathsf{Snd} \Rightarrow_L s} \qquad \frac{}{\mathsf{Id}*\mathsf{Snd} \Rightarrow_L \mathsf{Snd}}$$

$$\frac{f*t' \Rightarrow_L \langle \mathsf{Id}, s'\rangle * \mathsf{App} \qquad f*t \Rightarrow_L h*\mathsf{Cur}(A,t'') \qquad \langle h, s'[A]\rangle * t'' \Rightarrow_L s}{f*(t;t') \Rightarrow_L s}$$

$$\frac{f*t' \Rightarrow_L \pi_i \qquad f*t \Rightarrow_L f';\langle h, t_1[A], t_2[B]\rangle * \mathsf{Pair} \qquad f'*t_i \Rightarrow_L s_i}{f*(t;t') \Rightarrow_L s_i}$$

$$\frac{}{f*(g*\mathsf{App}) \Rightarrow_L \langle \mathsf{Id}, f*(g*\mathsf{Snd})\rangle * \mathsf{App}} \qquad \frac{}{f*(g*\pi_i) \Rightarrow_L \pi_i}$$

$$\frac{f*(h*t;h*t') \Rightarrow_L s}{f*(h*(t;t')) \Rightarrow_L s} \qquad \frac{f;g \Rightarrow_L h \qquad h*t \Rightarrow_L s}{f*(g*t) \Rightarrow_L s}$$

$$\frac{f*t \Rightarrow_L s \qquad f*t' \Rightarrow_L s'}{f*(t;t') \Rightarrow_L s;s'} \qquad \frac{f \in \mathcal{C}}{f*t \Rightarrow_L f*t}$$

Table 3.6: Inference Rules for Lazy Reduction

A later rule applies only if all prior rules fail. f always denotes a canonical context morphism.

Context Morphisms

$$\frac{}{f;\langle\rangle \Rightarrow_E \langle\rangle} \qquad \frac{}{f;\mathsf{Id} \Rightarrow_E f} \qquad \frac{}{\mathsf{Id};\mathsf{Fst} \Rightarrow_E \mathsf{Fst}}$$

$$\frac{\langle\langle\rangle, t[A]\rangle \in \mathcal{C}}{\mathsf{Fst}^k;\langle\langle\rangle, t[A]\rangle;\mathsf{Fst} \Rightarrow_E \langle\rangle} \qquad \frac{\langle \mathsf{Id}, t[A]\rangle \in \mathcal{C}}{\mathsf{Fst}^k;\langle \mathsf{Id}, t[A]\rangle;\mathsf{Fst} \Rightarrow_E \mathsf{Fst}^{k+1}}$$

$$\frac{}{\mathsf{Fst}^k;\mathsf{Fst} \Rightarrow_E \mathsf{Fst}^{k+1}} \qquad \frac{\langle f, s[A]\rangle \in \mathcal{C}}{\mathsf{Fst}^k;\langle f, s[A]\rangle;\mathsf{Fst} \Rightarrow_E \mathsf{Fst}^k; f}$$

$$\frac{f;g \Rightarrow_E h \qquad f * t \Rightarrow_E s}{f;\langle g, t[A]\rangle \Rightarrow_E \langle h, s[A]\rangle} \qquad \frac{f;g \Rightarrow_E f' \qquad f';h \Rightarrow_E f''}{f;(g;h) \Rightarrow_E f''}$$

Types

$$\frac{}{f * \Omega \Rightarrow_E \Omega} \qquad \frac{f' \in \mathcal{C}}{\mathsf{Fst}^k;\langle f, f' * \forall(A,t)[B]\rangle * T \Rightarrow_E \mathsf{Fst}^k; f' * \Pi(A, \langle\langle\rangle, t[B]\rangle * T)}$$

$$\frac{f;g \Rightarrow_E h \qquad h * B \Rightarrow_E A}{f * (g * B) \Rightarrow_E A} \qquad \frac{f * \mathsf{Snd} \Rightarrow_E t}{f * T \Rightarrow_E \langle\langle\rangle, t[\Omega]\rangle * T} \qquad \frac{}{f * A \Rightarrow_E f * A}$$

Morphisms

$$\frac{f * t \Rightarrow_E s \qquad f * g \Rightarrow_E h}{f * (t; g * \mathsf{Id}) \Rightarrow_E s} \qquad \frac{f * t \Rightarrow_E s \qquad f * g \Rightarrow_E h}{f * (g * \mathsf{Id}; t) \Rightarrow_E s}$$

$$\frac{\langle f, s[A]\rangle \in \mathcal{C}}{(\mathsf{Fst}^k;\langle f, s[A]\rangle) * \mathsf{Snd} \Rightarrow_E \mathsf{Fst}^k * s} \qquad \frac{}{\mathsf{Id} * \mathsf{Snd} \Rightarrow_E \mathsf{Snd}}$$

$$\frac{f * t \Rightarrow_E h * \mathsf{Cur}(A_1, s_1) \qquad f;g * \mathsf{Snd} \Rightarrow_E s_2 \qquad \langle h, s_2[A_1]\rangle * s_1 \Rightarrow_E s}{f * (t; g * \mathsf{App}) \Rightarrow_E s}$$

$$\frac{f * t \Rightarrow_E s_1 \qquad f;g * \mathsf{Snd} \Rightarrow_E s_2}{f * (t; g * \mathsf{App}) \Rightarrow_E s_1; \langle \mathsf{Id}, s_2\rangle * \mathsf{App}}$$

$$\frac{f * t \Rightarrow_E f';\langle\langle h, t_1[A]\rangle, t_2[B]\rangle * \mathsf{Pair} \qquad f;g \Rightarrow_E h'}{f * (t; g * \pi_i) \Rightarrow_E f' * t_i} \qquad \frac{f * t \Rightarrow_E s \qquad f;g \Rightarrow_E h}{f * (t; g * \pi_i) \Rightarrow_E s; \pi_i}$$

$$\frac{f;g \Rightarrow_E h \qquad h * t \Rightarrow_E s}{f * (g * t) \Rightarrow_E s} \qquad \frac{f * ((t;s);u) \Rightarrow_E t'}{f * (t;(s;u)) \Rightarrow_E t'} \qquad \frac{}{f * t \Rightarrow_E f * t}$$

Table 3.7: Inference Rules for Eager Reduction

Now we show that both strategies describe a reduction to a canonical form.

Theorem 3.11 *For both strategies $\Rightarrow_L$ and $\Rightarrow_E$, for every canonical context morphism f and every well-formed combinator g, B and t with $\Gamma \rhd_i t\colon 1 \rightarrow C$ such that $f;g$, $f * B$ and $f * t$ are well-formed, there exist unique canonical combinators h, B and s such that*

(i) $f;g \Rightarrow h$

(ii) $f * B \Rightarrow A$

(iii) $f * t \Rightarrow s$

Proof We first consider the proof for the lazy strategy $\Rightarrow_L$. By an induction over the structure of e we show that the theorem is true if it holds for any e' and f' such that $\nu(f' * e') < \nu(f * e)$, where $\nu(d)$ is the length of the longest $\overset{N}{\rightsquigarrow}$-reduction sequence of d.

($\langle\rangle$) We have $f;\langle\rangle \Rightarrow_L \langle\rangle$.

(Fst) If $f \equiv f_1;\langle g, t[A]\rangle$, then $\nu(f_1;g) < \nu(f;\mathsf{Fst})$, and so by induction hypothesis $f_1;g \Rightarrow_L h$, and therefore $f;\mathsf{Fst} \Rightarrow_L h$. Otherwise $f \equiv \mathsf{Fst}^k$, and so we have $\mathsf{Fst}^k;\mathsf{Fst} \Rightarrow_L \mathsf{Fst}^{k+1}$.

($\langle g, t[A]\rangle$) $f;\langle g, t[A]\rangle$ is canonical.

(Id) $f;\mathsf{Id} \Rightarrow_L f$.

($g;h$) By induction hypothesis, $f;g \Rightarrow_L f'$ and $f';h \Rightarrow_L h'$, so $f;(g;h) \Rightarrow_L h'$.

($f * \Omega$) $f * \Omega \Rightarrow_L \Omega$.

(T) We start with $f \equiv f';\langle g, t[A]\rangle$. Because f' is canonical, the induction hypothesis yields $f' * t \Rightarrow_L t'$, hence $f * \mathsf{Snd} \Rightarrow_L t'$. If $t' \equiv h * \forall(A, t'')$, then $f * T \Rightarrow_L h * \Pi(A, \langle\langle\rangle, t''\rangle[\Omega] * T)$, else $f * T \Rightarrow_L \langle\langle\rangle, t'[\Omega]\rangle * T$. If $f \equiv \mathsf{Fst}^n$, then $\mathsf{Fst}^n * T \Rightarrow_L \langle\langle\rangle, \mathsf{Fst}^n * \mathsf{Snd}[\Omega]\rangle * T$.

($\Pi(A,B)$) $f * \Pi(A,B)$ is already canonical.

($\Sigma(A,B)$) $f * \Sigma(A,B)$ is canonical.

($g * A$) By induction hypothesis, $f';g \Rightarrow_L h$ and $h * A \Rightarrow_L B$.

($g * \mathsf{Id};t$) The induction hypothesis yields $f * t \Rightarrow_L s$, and so $f * (g * \mathsf{Id}) \Rightarrow_L s$. The other cases involving the combinator Id follow similarly.

($g * t$) By induction hypothesis, $f;g \Rightarrow_L h$ and $h * t \Rightarrow_L s$.

($t;(s;u)$) The reduction $t;(s;u) \rightsquigarrow_i (t;s);u$ shows that $\nu(f * (t;(s;u))) > \nu(f * ((t;s);u))$. The induction hypothesis yields therefore $f * ((t;s);u) \Rightarrow_L t'$, hence $f * (t;(s;u)) \Rightarrow_L t'$.

(Snd) We have $\mathsf{Id} * \mathsf{Snd} \Rightarrow_L \mathsf{Snd}$, and $\mathsf{Fst}^n * \mathsf{Snd}$ is canonical if $n > 0$. If $f \equiv f'; \langle g, t[A]\rangle$, then by induction hypothesis $f' * t \Rightarrow_L t'$, and so $f * \mathsf{Snd} \Rightarrow_L t'$.

($\mathsf{Cur}(A, t)$) $f * \mathsf{Cur}(A, t)$ is already canonical.

($\forall(A, t)$) $f * \forall(A, t)$ is canonical.

($t; g * \mathsf{App}$) By induction hypothesis, $f * t \Rightarrow_L s$ with s canonical. First consider the case where $s \equiv h * \mathsf{Cur}(A, t')$. Because $\langle h, g * \mathsf{Snd}[A]\rangle$ is canonical and $\nu(\langle h, g * \mathsf{Snd}[A]\rangle * t') < \nu(f * (t; g * \mathsf{App}))$, the induction hypothesis yields the claim. If $s \not\equiv h * \mathsf{Cur}(A, t)$, we get immediately $f * (t; g * \mathsf{App}) \Rightarrow_L s; \langle \mathsf{Id}, f; g * \mathsf{Snd}\rangle * \mathsf{App}$.

(Pair) $f * \mathsf{Pair}$ is canonical.

($t; g * \pi_i$) By induction hypothesis, $f * t \Rightarrow_L s$ with s canonical. If $s \equiv f'; \langle h, t_1[A], t_2[B]\rangle * \mathsf{Pair}$, then again by induction hypothesis, $f' * t_i \Rightarrow_L s_i$, and so $f * (t; g * \pi_i) \Rightarrow_L s_i$. Otherwise, $s; \pi_i$ is canonical, and $f * (t; g * \pi_i) \Rightarrow_L s; \pi_i$.

The proof for the eager case is similar. Therefore we give only the details for the cases $\langle g, t[A]\rangle$ and $t; \langle \mathsf{Id}, t'\rangle * \mathsf{App}$, where the differences to the lazy case occur.

($\langle g, t[A]\rangle$) The induction hypothesis implies $f; g \Rightarrow_E h$, $f * t \Rightarrow_E s$, and so

$$f; \langle g, t[A]\rangle \Rightarrow_E \langle h, s[A]\rangle$$

.

($t; g * \mathsf{App}$) The induction hypothesis implies $f * t \Rightarrow_E s$ and $f; g * \mathsf{Snd} \Rightarrow_E s'$ for canonical s and s'. If $s \not\equiv h * \mathsf{Cur}(A, t')$ then the combinator $s; \langle \mathsf{Id}, s'\rangle * \mathsf{App}$ is canonical. Otherwise we have $\nu(\langle h, s'[A]\rangle * t') < \nu(t; g * \mathsf{App})$, and therefore the induction hypothesis yields $\langle h, s'\rangle * A \Rightarrow_E s''$ with s'' canonical.

□

Remark The proof of the eager case explains why we have to use the strong normalization for the relation $\overset{N}{\leadsto}$ and cannot on rely on the strong normalization of the calculus. If we use $\nu((e)^c)$ instead of $\nu(e)$ the above proof breaks down. Consider the case $f * t; \langle \mathsf{Id}, s\rangle * \mathsf{App}$ and suppose the induction hypothesis yields $f * t \Rightarrow_E f' * \mathsf{Cur}(A, g * t')$ and $f * s \Rightarrow_E s'$. Then it is in general false that

$$\nu((f'; g; \mathsf{Fst}^k * \mathsf{Snd})^c) \leq \nu(f * (t; \langle \mathsf{Id}, s\rangle * \mathsf{App}))$$

because the variable k may not occur in $(t')^c$.

Chapter 4

The Construction of Abstract Machines

This chapter shows how the inference rules for reduction to canonical form give rise to abstract machines. We first explain the general pattern, and then describe the lazy and eager machine in detail. The extension of these machines for reduction to normal form is discussed next. The chapter finishes with a comparison to other abstract machines, notably the CAM and Krivine's machine.

4.1 Structure of the Machines

As already mentioned in the previous chapter, the inference rules describe how a combinator $f * e$ with f canonical is reduced to a canonical one, say e'. This suggest three registers for the abstract machines: the first contains the code for the environment f, the second contains the code for e, and the third assembles the canonical combinator e' as the computation proceeds if e' is a type or a morphism. A fourth register, which operates as a stack, is necessary in the eager case to store the code for f temporarily. The code is a tree whose nodes are machine instructions. They roughly correspond to simple combinators like Fst, Snd and App together with one instruction that denotes sequential execution. Sharing can be implemented by generalizing the tree to a graph. The code will be written in a linearized way to increase readability, and the sequential execution of the code for C_1 and C_2 is denoted by C_1C_2. A state of the machines is written as

Environment	Code	Canonical Form	Stack
f	C	N	S

and a transition from such a state to another one, which is activated according to the first instruction of C, is written as

Environment	Code	Canonical Form	Stack
f	C	N	S
		$\Downarrow$	
f'	C'	N'	S'

The translation from combinators into machine code is denoted by $[\![-]\!]_m$.

The idea behind the transitions for the machines is that every inference rule

$$\frac{f_1 * e_1 \Rightarrow e_1' \cdots f_n * e_n \Rightarrow e_n'}{f * e \Rightarrow e'}$$

gives rise to machine transitions describing in an elementary way how e' can be constructed from the canonical combinators e_1' to e_n'. It is possible to divide the inference rules into several groups according to their form.

1. Every rule

   $$\frac{f' * e' \Rightarrow d}{f * e \Rightarrow d}$$

   is captured by a transition

Environment	Code	Canonical Form	Stack
$[\![f]\!]_m$	$[\![e]\!]_m$	N	S
		$\Downarrow$	
$[\![f']\!]_m$	$[\![e']\!]_m$	N	S

2. Any inference rule

   $$\frac{}{f * e \Rightarrow e'}$$

   with e a type or a morphism gives rise to a transition

Environment	Code	Canonical Form	Stack
$[\![f]\!]_m$	$[\![e]\!]_m C$	N	S
		$\Downarrow$	
$[\![f]\!]_m$	C	$N[\![e']\!]_m$	S

   and any inference rule

   $$\frac{}{f; g \Rightarrow h}$$

   leads to a transition

Environment	Code	Canonical Form	Stack
$[\![f]\!]_m$	$[\![g]\!]_m C$	N	S
	$\Downarrow$		
$[\![h]\!]_m$	C	N	S

3. The case

$$\frac{f' * e' \Rightarrow d'}{f * e \Rightarrow d}$$

which occurs only if $e \equiv T$, is covered by the introduction of a special instruction T_C transforming d' into d. This yields the two transitions

Environment	Code	Canonical Form	Stack
$[\![f]\!]_m$	$T\ C$	N	S
	$\Downarrow$		
$[\![f']\!]_m$	$[\![e']\!]_m\ T_C\ C$	N	S

and

Environment	Code	Canonical Form	Stack
E	$T_C\ C$	$N\ [\![d']\!]_m$	S
	$\Downarrow$		
E	C	$N\ [\![d]\!]_m$	S

4. The identity morphisms in the fibres and in the base are represented by the empty code, which is denoted by $-$. This code acts like an empty word, i.e. its concatenation with any other code sequence C is identical to C.

5. Composition in the base category is modelled by sequential execution of the machine instructions. Therefore the inference rule

$$\frac{f; g \Rightarrow f' \qquad f; h \Rightarrow h'}{f; (g; h) \Rightarrow h'}$$

is not modelled by a special machine transition but by the fact that the code corresponding to g is executed before the code for h.

4.2 The Lazy Machine

The differences between the lazy and the eager machine become apparent when we consider the inference rules for composition in the fibres, the $*$-operation, β- and σ-reduction and access to the environment.

The composition in the fibres poses a problem that does not arise for composition in the base. We must make sure that the environment f is still available when the second argument of a composition in the fibre is processed. This can be achieved without an extra stack register if $[\![t_1; t_2]\!]_m = [\![t_2]\!]_m [\![t_1]\!]_m$. The execution sequence of the machine for a combinator $t_1; t_2$ with $t_1 \colon 1 \rightarrow A$ and $t_2 \colon A \rightarrow B$ shows how:

- The machine starts by executing the code for t_2, which does not change the environment register.
- Then it performs the reduction of the code for t_1, which may change this register.

The second property of the lazy reduction strategy mentioned in section 3.2 on page 58 states that the reduction of the context morphism g in a combinator $g*t$ may depend on the structure of t. As a consequence we cannot translate the combinator $g*t$ simply into $[\![g]\!]_m [\![t]\!]_m$. Instead we have to keep the operation symbol $*$ as part of the machine representation of a combinator, so the translation of $g*t$ is $[\![t]\!]_m * [\![g]\!]_m$.

Now we turn to the implementation of the β–rule

$$\frac{f * t' \Rightarrow_L \langle \mathsf{Id}, s' \rangle * \mathsf{App} \qquad f * t \Rightarrow_L h * \mathsf{Cur}(A, t'') \qquad \langle h, s'[A] \rangle * t'' \Rightarrow_L s}{f * (t; t') \Rightarrow_L s}$$

It uses the fact that any derivation tree of $f * t' \Rightarrow_L h * \mathsf{Cur}(A, t'')$ has a branch with a leaf

$$h * \mathsf{Cur}(A, t'') \Rightarrow_L h * \mathsf{Cur}(A, t'')$$

Hence any transition sequence for the code for t passes through a state

Environment	Code	Canonical Form
$[\![h]\!]_m$	$\mathsf{Cur}([\![A]\!]_m, [\![t'']\!]_m)\ C$	N

Therefore it suffices to introduce the transition

Environment	Code	Canonical Form
f	$\mathsf{Cur}(A, t)\ C$	$N\mathsf{App} * \langle -, t' \rangle$
	$\Downarrow_L$	
$\langle f, t' \rangle ?(A)$	$t\ C$	N

where $[\![\langle f, t'[A] \rangle]\!]_m = \langle [\![f]\!]_m, [\![t']\!]_m \rangle ?([\![A]\!]_m)$. A similar argument yields also the transition for the σ-rule:

Environment	Code	Canonical Form
$f; \langle\langle g, t_1\rangle?(A), t_2\rangle?(B)$	$\mathsf{Pair}\ C$	$N\pi_i$
	$\Downarrow_L$	
f	$t_i\ C$	N

The inference rules for access to environments are treated by two transitions, which schedule the evaluation of the component that is selected by these inference rules:

Environment	Code	Canonical Form
$f; \langle g, t\rangle?(A)$	$\mathsf{Fst}\ C$	N
	$\Downarrow_L$	
f	$g\ C$	N

and

Environment	Code	Canonical Form
$f\langle g, t\rangle?(A)$	$\mathsf{Snd}\ C$	N
	$\Downarrow_L$	
f	$t\ C$	N

The access to the environment admits an important optimization. Consider the combinator $f * \mathsf{Cur}(A, \mathsf{Snd}); \langle \mathsf{Id}, t\rangle * \mathsf{App}$. According to the strategy $\Rightarrow_L$, its reduction amounts to reducing the combinator $\langle f, t[A]\rangle * \mathsf{Snd}$, which in turn leads to the reduction of the combinator $\mathsf{Id} * (f * t)$. Because f is canonical, the derivation tree for the latter combinator contains the judgement $f * t \Rightarrow_L s$. So we obtain an admissible inference rule

$$\frac{f \in \mathcal{C} \qquad \mathsf{Fst}^k; f * t \Rightarrow_L s}{\mathsf{Fst}^k; \langle g, f * t[A]\rangle * \mathsf{Snd} \Rightarrow_L s}$$

In the same way we get another admissible inference rule

$$\frac{f \in \mathcal{C}}{\mathsf{Fst}^k; \langle f, t[A]\rangle; \mathsf{Fst} \Rightarrow_L \mathsf{Fst}^k; f}$$

The implementation of these inference rules depends on recognizing that certain context morphisms are canonical. This is easily achieved by introducing a flag that indicates whether in a combinator $\mathsf{Fst}^k; \langle f, t[A]\rangle$ and $f * t$ respectively f is canonical or not.

Context Morphisms

$$\begin{array}{rcl@{\qquad}rcl}
\llbracket\langle\rangle\rrbracket_m & \stackrel{\text{def}}{=} & \langle\rangle & \llbracket\mathsf{Id}\rrbracket_m & \stackrel{\text{def}}{=} & -\\
\llbracket f;g\rrbracket_m & \stackrel{\text{def}}{=} & \llbracket f\rrbracket_m\llbracket g\rrbracket_m & \llbracket\mathsf{Fst}\rrbracket_m & \stackrel{\text{def}}{=} & \mathsf{Fst}\\
\llbracket\langle f,t[A]\rangle\rrbracket_m & \stackrel{\text{def}}{=} & \langle\llbracket f\rrbracket_m,\llbracket t\rrbracket_m\rangle?(\llbracket A\rrbracket_m) & & &
\end{array}$$

Types

$$\begin{array}{rcl@{\qquad}rcl}
\llbracket\Omega\rrbracket_m & \stackrel{\text{def}}{=} & \Omega & \llbracket T\rrbracket_m & \stackrel{\text{def}}{=} & T\\
\llbracket f*A\rrbracket_m & \stackrel{\text{def}}{=} & \llbracket A\rrbracket_m*\llbracket f\rrbracket_m & \llbracket\Pi(A,B)\rrbracket_m & \stackrel{\text{def}}{=} & \Pi(\llbracket A\rrbracket_m,\llbracket B\rrbracket_m)\\
\llbracket\Sigma(A,B)\rrbracket_m & \stackrel{\text{def}}{=} & \Sigma(\llbracket A\rrbracket_m,\llbracket B\rrbracket_m) & & &
\end{array}$$

Morphisms

$$\begin{array}{rcl@{\qquad}rcl}
\llbracket t;t'\rrbracket_m & \stackrel{\text{def}}{=} & \llbracket t'\rrbracket_m\llbracket t\rrbracket_m & \llbracket\mathsf{Id}\rrbracket_m & \stackrel{\text{def}}{=} & -\\
\llbracket\mathsf{Snd}\rrbracket_m & \stackrel{\text{def}}{=} & \mathsf{Snd} & \llbracket f*t\rrbracket_m & \stackrel{\text{def}}{=} & \llbracket t\rrbracket_m*\llbracket f\rrbracket_m\\
\llbracket\mathsf{App}\rrbracket_m & \stackrel{\text{def}}{=} & \mathsf{App} & \llbracket\mathsf{Cur}(A,t)\rrbracket_m & \stackrel{\text{def}}{=} & \mathsf{Cur}(\llbracket A\rrbracket_m,\llbracket t\rrbracket_m)\\
\llbracket\mathsf{Pair}\rrbracket_m & \stackrel{\text{def}}{=} & \mathsf{Pair} & \llbracket\forall(A,t)\rrbracket_m & \stackrel{\text{def}}{=} & \forall(\llbracket A\rrbracket_m,\llbracket t\rrbracket_m)\\
\llbracket\pi_i\rrbracket_m & \stackrel{\text{def}}{=} & \pi_i & & &
\end{array}$$

Table 4.1: Translation of Combinators into Lazy Machine Code

Now we formalize the previous discussion. The machine instructions are given by the following BNF-expressions:

$$\begin{array}{rcl}
f & ::= & \langle\rangle \mid - \mid \mathsf{Fst} \mid \langle f,t\rangle \mid ?(A)\\
A & ::= & A*f \mid \Omega \mid \Pi(A,A) \mid \Sigma(A,A) \mid T \mid T_C\\
t & ::= & t*f \mid \mathsf{Snd} \mid \mathsf{Cur}(A,t) \mid \mathsf{App} \mid \mathsf{Pair} \mid \pi_1 \mid \pi_2 \mid \forall(A,t)
\end{array}$$

Table 4.1 contains the translation of the combinators into machine code, and Table 4.2 contains the transitions of the machine itself. A final state is a state for which no transition rule applies. The symbol ? denotes an arbitrary code sequence.

Theorem 4.1 *For every canonical context morphism f and all combinators g, A and t such that there exist combinators h, B and s satisfying $f;g \Rightarrow_L h$, $f*A \Rightarrow_L B$ and $f*t \Rightarrow_L s$ respectively, the machine performs the following actions:*

Environment	Code	Canonical Form
$\llbracket f\rrbracket_m$	$\llbracket g\rrbracket_m\ C$	N
	$\Downarrow_L^*$	
$\llbracket h\rrbracket_m$	C	N

A later rule applies only if all earlier ones fail. If we write $e\ C$ for the content of the code register, we always assume that e is not a sequence of instructions. The rules marked with (+) apply only if the flag for $t * h$ and $\langle h, t\rangle?(A)$ respectively indicate that h is the code for a canonical combinator.

	Environment	Code	Canonical Form
$\langle\rangle$	f	$\langle\rangle\ C$	N
	$\langle\rangle$	C	N
(+)Fst	$\mathsf{Fst}^k\langle\ h, t\rangle?(A)$	$\mathsf{Fst}\ C$	N
	$\mathsf{Fst}^k h$	C	N
Fst	$f\langle\ g, t\ \rangle?(A)$	$\mathsf{Fst}\ C$	N
	f	$g\ C$	N
$f; g$	f	$g\ C$	N
	$f\ g$	C	N
$*$	f	$(A * g) * h\ C$	N
	f	$A * (hg)\ C$	N
Ω	f	$\Omega * g\ C$	N
	f	C	Ω
Ω	f	$\Omega\ C$	N
	f	C	Ω
$A * g$	f	$A\ * g\ C$	N
	f	$g\ A\ C$	N
T	f	$T\ C$	N
	f	$\mathsf{Snd}\ T_C\ C$	–
T_C	f	$T_C\ C$	$\forall(A, t) * h$
	f	C	$\Pi(A, T * \langle\langle\rangle, t\rangle?(\Omega)) * h$
T_C	f	$T_C\ C$	t
	f	C	$T * \langle\langle\rangle, t\rangle?(\Omega)$
A	f	$A\ C$	N
	f	C	$A * f$
$*$	f	$(t * g) * h\ C$	N
	f	$t * (h\ g)\ C$	N
$- * g$	f	$- * g\ C$	N
	f	C	N
(+)Snd	f	C	N
	$\mathsf{Fst}^k h$	$t\ C$	N
Snd	$f\langle g, t\rangle?(A)$	$\mathsf{Snd}\ C$	N
	f	$t\ C$	N
Snd	–	$\mathsf{Snd}\ C$	N
	–	C	$N\mathsf{Snd}$

Table 4.2: Transitions of the Lazy Machine

	Context	Code	Canonical Form
(β)	f	$\mathsf{Cur}(A,t)\ C$	$N\mathsf{App} * \langle -,s\rangle$
	$\langle f,s\rangle?(A)$[1]	t	N
App	f	$\mathsf{App} * \langle -,t\rangle\ C$	N
	f	C	$N\mathsf{App} * \langle -,t * f\rangle$[1]
App	f	$\mathsf{App} * g\ C$	N
	f	C	$N\mathsf{App} * \langle -,(\mathsf{Snd} * g) * f\rangle$[1]
Pair	$f\langle\langle g,t_1\rangle?(A),t_2\rangle?(B)$	$\mathsf{Pair}\ C$	$N\pi_i$
	f	$t_i\ C$	N
$h * (t\ t')$	f	$(t\ t') * h\ C$	N
	f	$t * h\ t' * h\ C$	N
π_i	f	$\pi_i * g\ C$	N
	f	C	$N\pi_i$
$t * h$	f	$t * h\ C$	N
	f	$ht\ C$	N

Table 4.2 (continued): Transitions of the Lazy Reduction Machine

[1] The flag indicating that f is canonical is set as well.

Environment	Code	Canonical Form
$[\![f]\!]_m$	$[\![A]\!]_m\ C$	N
	$\Downarrow^*_L$	
?	C	$[\![B]\!]_m$

Environment	Code	Canonical Form
$[\![f]\!]_m$	$[\![t]\!]_m\ C$	–
	$\Downarrow^*_L$	
?	C	$[\![s]\!]_m$

If C is the empty code sequence, then the resulting states are final.

Proof As already mentioned, the inference rules for the β- and σ-reduction make it necessary to consider not only the transitions to a final state but also to previous ones if the code for a morphism is executed. Therefore we modify the statement of the theorem for morphisms as follows:

Theorem 4.1 (modified) *If $f{*}t \Rightarrow_L s$, then we have the following cases, according to the structure of s:*

1. *$s \equiv \pi^{(0)}; \langle \mathsf{Id}, s_1\rangle * \mathsf{App}; \pi^{(1)}; \langle \mathsf{Id}, s_2\rangle * \mathsf{App}; \cdots$, where $\pi^{(k)}$ is any combination of π_1 and π_2, and*

Environment	Code	Canonical Form
$\llbracket f\rrbracket_m$	$\llbracket t\rrbracket_m\ C$	N
	$\Downarrow^*_L$	
$\llbracket f\rrbracket_m$	C	$N\llbracket s\rrbracket_m$

2. *$s \equiv d * u; v_1; \cdots v_n$ and*

Environment	Code	Canonical Form
$\llbracket f\rrbracket_m$	$\llbracket t\rrbracket_m\ C$	N
	$\Downarrow^*_L$	
$\llbracket d\rrbracket_m$	$\llbracket u\rrbracket_m\ C$	$N\llbracket v_n\rrbracket_m \cdots \llbracket v_1\rrbracket_m$

3. *$s \equiv \mathsf{Snd}; v_1; \cdots v_n$ and*

Environment	Code	Canonical Form
$\llbracket f\rrbracket_m$	$\llbracket t\rrbracket_m\ C$	N
	$\Downarrow^*_L$	
—	$\mathsf{Snd}\ C$	$N\llbracket v_n\rrbracket_m \cdots \llbracket v_1\rrbracket_m$

To simplify the notation, we write all three cases as

Environment	Code	Canonical Form
$\llbracket f\rrbracket_m$	$\llbracket t\rrbracket_m\ C$	N
	$\Downarrow^*_L$	
d'	$u'\ C$	Nv'

The original version of the theorem follows from this by an easy analysis of the canonical combinator u'. Theorem 3.11 implies that it is enough to show that the theorem holds for the conclusion of any inference rule for the strategy $\Rightarrow_L$ if it holds for all of its premises. Hence we consider all inference rules in turn. The symbol **I.H.** denotes an application of the induction hypothesis.

We start with context morphisms.

$f; (g; h) \Rightarrow_L h'$ The transition sequence is

	Environment	Code	Canonical Form
	$[\![f]\!]_m$	$[\![g]\!]_m[\![h]\!]_m\ C$	N
I.H.		$\Downarrow_L^*$	
	$[\![f']\!]_m$	$[\![h]\!]_m\ C$	N
I.H.		$\Downarrow_L^*$	
	$[\![h']\!]_m$	C	N

$f;\langle\rangle \Rightarrow_L \langle\rangle$ This corresponds directly to the transition rule

Environment	Code	Canonical Form
$[\![f]\!]_m$	$\langle\rangle\ C$	N
	$\Downarrow_L$	
$\langle\rangle$	C	N

The rules $f;\mathsf{Id} \Rightarrow_L f$, $\mathsf{Id};g \Rightarrow_L g$ and $f;g \Rightarrow_L f;g$ are similar.

$f;\mathsf{Fst} \Rightarrow_L h$ The machine sequence is

	Environment	Code	Canonical Form
	$[\![f]\!]_m\langle[\![g]\!]_m,[\![t]\!]_m\rangle?([\![A]\!]_m)$	$\mathsf{Fst}\ C$	N
		$\Downarrow_L$	
	$[\![f]\!]_m$	$[\![g]\!]_m\ C$	N
I.H.		$\Downarrow_L^*$	
	$[\![h]\!]_m$	C	N

Next we consider the types.

$f*(g*(h*A))$ The transition sequence

	Environment	Code	Canonical Form
	$[\![f]\!]_m$	$(([\![A]\!]_m * [\![g]\!]_m) * [\![h]\!]_m)\ C$	N
		$\Downarrow_L$	
	$[\![f]\!]_m$	$[\![A]\!]_m * ([\![h]\!]_m[\![g]\!]_m)\ C$	N
I.H.		$\Downarrow_L^*$	
	?	C	$[\![B]\!]_m$

shows the claim.

$f * (g * \Omega) \Rightarrow_L \Omega$ This is covered by the transition

Environment	Code	Canonical Form
$\llbracket f \rrbracket_m$	$\Omega * \llbracket g \rrbracket_m\ C$	N
	$\Downarrow_L$	
$\llbracket f \rrbracket_m$	C	Ω

The case $f * \Omega \Rightarrow_L \Omega$ is similar.

$f * (g * A) \Rightarrow_L B$ The transition sequence is

	Environment	Code	Canonical Form
	$\llbracket f \rrbracket_m$	$\llbracket A \rrbracket_m * \llbracket g \rrbracket_m\ C$	N
		$\Downarrow_L$	
	$\llbracket f \rrbracket_m$	$\llbracket g \rrbracket_m \llbracket A \rrbracket_m\ C$	N
I.H.		$\Downarrow_L^*$	
	?	C	$\llbracket B \rrbracket_m$

$f * T \Rightarrow_L h * \Pi(A, \langle\langle\rangle, t[\Omega]\rangle * T)$ Consider the following sequence:

Environment	Code	Canonical Form
$\llbracket f \rrbracket_m$	$T\ C$	N
	$\Downarrow_L$	
$\llbracket f \rrbracket_m$	$\mathsf{Snd}\ T_C\ C$	–
	$\Downarrow_E$	
	$\Downarrow_E$	
$\llbracket h \rrbracket_m$	$\forall(\llbracket A \rrbracket_m, \llbracket t \rrbracket_m)\ T_C\ C$	–
	$\Downarrow_L$	
$\llbracket h \rrbracket_m$	$T_C\ C$	$\forall(\llbracket A \rrbracket_m, \llbracket t \rrbracket_m) * \llbracket h \rrbracket_m$
	$\Downarrow_L$	
$\llbracket h \rrbracket_m$	C	$\Pi(\llbracket A \rrbracket_m, T * \langle\langle\rangle, \llbracket t \rrbracket_m\rangle?(\Omega)) * \llbracket h \rrbracket_m$

The case $f * T \Rightarrow_L \langle\langle\rangle, t[\Omega]\rangle * T$ is similar.

$f * A \Rightarrow_L f * A$ Obvious.

Finally we show the theorem for the morphisms.

$f * (t; (s; u)) \Rightarrow_L t'$ $[\![t; (s; u)]\!]_m = [\![(t; s); u]\!]_m$.

$f * (h * (g * t)) \Rightarrow_L s$ Consider the sequence

	Environment	Code	Canonical Form
	$[\![f]\!]_m$	$([\![t]\!]_m * [\![g]\!]_m) * [\![h]\!]_m\ C$	N
		$\Downarrow_L$	
	$[\![f]\!]_m$	$[\![t]\!]_m * ([\![h]\!]_m\ [\![g]\!]_m)\ C$	N
I.H.		$\Downarrow_L^*$	
	d'	$u'\ C$	Nv'

$f * (t; g * \mathsf{Id}) \Rightarrow_L s$ We have the following sequence

	Environment	Code	Canonical Form
	$[\![f]\!]_m$	$- * [\![g]\!]_m[\![t]\!]_m\ C$	N
		$\Downarrow_L$	
	$[\![f]\!]_m$	$[\![t]\!]_m\ C$	N
I.H.		$\Downarrow_L^*$	
	d'	$u'\ C$	Nv'

$f * (g * \mathsf{Id}; t) \Rightarrow_L s$ Similar to the previous case.

$f; \langle g, t[A]\rangle * \mathsf{Snd} \Rightarrow_L s$ The transition sequence is

	Environment	Code	Canonical Form
	$[\![f]\!]_m\langle [\![g]\!]_m, [\![t]\!]_m\rangle ?([\![A]\!]_m)$	$\mathsf{Snd}\ C$	N
		$\Downarrow_L$	
	$[\![f]\!]_m$	$[\![t]\!]_m\ C$	N
I.H.		$\Downarrow_L^*$	
	d'	$u'C$	Nv'

$\mathsf{Fst}^k; \langle g, f * t[A]\rangle * \mathsf{Snd} \Rightarrow_L s$ Similar to the previous case.

$\mathsf{Id} * \mathsf{Snd} \Rightarrow_L \mathsf{Snd}$ Obvious.

β-rule The machine sequence is

	Environment	Code	Canonical Form
	$[\![f]\!]_m$	$[\![t']\!]_m[\![t]\!]_m\ C$	N
I.H.		$\Downarrow_L^*$	
	$[\![f]\!]_m$	$[\![t]\!]_m$	$N\mathsf{App} * \langle -, [\![s']\!]_m\rangle$
I.H.		$\Downarrow_L^*$	
	$[\![h]\!]_m$	$\mathsf{Cur}([\![A]\!]_m, [\![t'']\!]_m)\ C$	$N\mathsf{App} * \langle -, [\![s']\!]_m\rangle$
		$\Downarrow_L$	
	$\langle h, \mathsf{Snd} * [\![f]\!]_m[\![g]\!]_m\rangle$	$[\![t'']\!]_m\ C$	N
I.H.		$\Downarrow_L^*$	
	d'	$u'\ C$	Nv'

The rule for Pair is treated similarly.

$f * (g * \mathsf{App}) \Rightarrow_L \langle \mathsf{Id}, f * (g * \mathsf{Snd})\rangle * \mathsf{App}$ The sequence is

Environment	Code	Canonical Form
$[\![f]\!]_m$	$\mathsf{App} * [\![g]\!]_m\ C$	N
	$\Downarrow_L$	
$[\![f]\!]_m$	C	$N\mathsf{App} * \langle \mathsf{Id}, (\mathsf{Snd} * [\![g]\!]_m) * [\![f]\!]_m)\rangle$

The corresponding rule for π_i is similar.

$f * (g * t) \Rightarrow_L s$ The sequence is

	Environment	Code	Canonical Form
	$[\![f]\!]_m$	$[\![t]\!]_m * [\![g]\!]_m\ C$	N
		$\Downarrow_L$	
	$[\![f]\!]_m$	$[\![g]\!]_m\ [\![t]\!]_m\ C$	N
I.H.		$\Downarrow_L^*$	
	$[\![h]\!]_m$	$[\![t]\!]_m\ C$	N
I.H.		$\Downarrow_L^*$	
	d'	$u'C$	Nv'

$f * h * (t_1; t_2) \Rightarrow_L s$ The machine sequence is

	Environment	Code	Canonical Form
	$[\![f]\!]_m$	$([\![t_2]\!]_m[\![t_1]\!]_m) * [\![h]\!]_m\ C$	N
		$\Downarrow_L$	
	$[\![f]\!]_m$	$[\![t_2]\!]_m * [\![h]\!]_m[\![t_1]\!]_m * [\![h]\!]_m\ C$	N
I.H.		$\Downarrow_L^*$	
	d'	$u'\ C$	Nv'

$f * (t; t') \Rightarrow_L s; s'$ The morphism t' is either $g * \mathsf{App}$ or $g * \pi_i$. In both cases we have

	Environment	Code	Canonical Form
	$[\![f]\!]_m$	$[\![t']\!]_m[\![t]\!]_m\ C$	N
		$\Downarrow_L$	
	$[\![f]\!]_m$	$[\![t]\!]_m\ C$	$N[\![s']\!]_m$
I.H.		$\Downarrow_L^*$	
	d'	$u'C$	$N[\![s']\!]_m v'$

□

4.3 The Eager Machine

The main difference between the eager and the lazy machine is that the former needs a stack for the execution of the code for a combinator $t_1; t_2$. As an example, consider the reduction of a combinator $f*(g_1*\mathsf{Cur}(A,t); g_2*\mathsf{App})$, which requires a reduction of both $f*g_1$ and $f*g_2$. So we store the content of the environment register on the stack before we execute the code for $g_1 * \mathsf{Cur}(A,t)$ and restore it afterwards. It does not matter for this argument whether the code for g_1 or for g_2 is executed first. The order chosen here will make it easy to recognize the CAM as a special case of the eager machine. The stack is also used when the code for a combinator $f; \langle g, t[A]\rangle$ is reduced because the canonical combinator f is needed for the code of both g and t. The other inference rules that characterize the difference between the eager and lazy reduction cause no special problems when the corresponding transitions are defined:

1. Because the combinator $g * t$ is reduced to a canonical form by first reducing g, there is no need to retain a special operation $*$ in the machine, as in the lazy case. We can simply translate $g * t$ as $[\![g]\!]_m[\![t]\!]_m$, in accordance with the inference rule

$$\frac{f; g \Rightarrow_E h \qquad h * t \Rightarrow_E s}{f * (g * t) \Rightarrow_E s}$$

2. Because the combinators f and t in a canonical combinator $\langle f, t[A]\rangle$ are already canonical, the inference rules for the access to an environment have the form

$$\frac{}{f * e \Rightarrow e'}$$

The transition rules follow therefore from the discussion at the beginning of this chapter.

3. The β-reduction is captured as follows: according to the strategy for composition, we have to execute first the code for the combinator t and then the code for t' if we consider a combinator $t; \langle \mathsf{Id}, t'\rangle * \mathsf{App}$. Finally we can schedule the reduction of the contractum of the β-rule. If the machine handled the code for the combinator $\langle \mathsf{Id}, t'\rangle$ in the way the code for the combinator $\langle g, t[A]\rangle$ is executed, we would waste effort in storing the environment on the stack and throwing it away after the computation of the canonical form of t'. The reduction $f; \langle \mathsf{Id}, t\rangle * \mathsf{App} \leadsto_i \langle \mathsf{Id}, f * t\rangle * \mathsf{App}$ suggests the introduction of an extra machine instruction $\langle\_$ with the transition rule

Environment	Code	Canonical Form	Stack
f	$\langle\_\, C$	N	S
		$\Downarrow_E$	
f	C	$-$	$S, \langle\_, N$

and add a special rule for $\rangle$

Environment	Code	Canonical Form	Stack
f	$\rangle\, C$	t	$S, \langle\_, N$
		$\Downarrow$	
$\langle\_ t\rangle$	C	N	S

The transition for the β-rule is then

Environment	Code	Canonical Form	Stack
$\langle\_\, s\rangle$	App	$N\ f\ \mathsf{Cur}(A', t')$	S
		$\Downarrow_E$	
$\langle f, s[A']\rangle$	t'	N	S

The σ-reduction is captured similarly.

Context Morphisms

$$\begin{array}{rcl@{\quad}rcl}
[\![\langle\rangle]\!]_m & \overset{\mathrm{def}}{=} & \langle\rangle & [\![f,t[A]]\!]_m & \overset{\mathrm{def}}{=} & \langle[\![f]\!]_m,[\![t]\!]_m\rangle?([\![A]\!]_m) \\
[\![\mathsf{Fst}]\!]_m & \overset{\mathrm{def}}{=} & \mathsf{Fst} & [\![f;g]\!]_m & \overset{\mathrm{def}}{=} & [\![f]\!]_m[\![g]\!]_m \\
[\![\mathsf{Id}]\!]_m & \overset{\mathrm{def}}{=} & - & [\![\langle\mathsf{Id},t\rangle]\!]_m & \overset{\mathrm{def}}{=} & \langle\text{-}[\![t]\!]_m\rangle
\end{array}$$

Types

$$\begin{array}{rcl@{\quad}rcl}
[\![\Omega]\!]_m & \overset{\mathrm{def}}{=} & \Omega & [\![\Pi(A,B)]\!]_m & \overset{\mathrm{def}}{=} & \Pi([\![A]\!]_m,[\![B]\!]_m) \\
[\![T]\!]_m & \overset{\mathrm{def}}{=} & T & [\![\Sigma(A,B)]\!]_m & \overset{\mathrm{def}}{=} & \Sigma([\![A]\!]_m,[\![B]\!]_m) \\
[\![f*A]\!]_m & \overset{\mathrm{def}}{=} & [\![f]\!]_m[\![A]\!]_m & & &
\end{array}$$

Morphisms

$$\begin{array}{rcl@{\quad}rcl}
[\![\mathsf{Snd}]\!]_m & \overset{\mathrm{def}}{=} & \mathsf{Snd} & [\![t_1;t_2]\!]_m & \overset{\mathrm{def}}{=} & \langle[\![t_1]\!]_m\mathsf{pop}[\![t_2]\!]_m \\
[\![\mathsf{Id}]\!]_m & \overset{\mathrm{def}}{=} & - & [\![\mathsf{App}]\!]_m & \overset{\mathrm{def}}{=} & \mathsf{App} \\
[\![f*t]\!]_m & \overset{\mathrm{def}}{=} & [\![f]\!]_m[\![t]\!]_m & [\![\mathsf{Cur}(A,t)]\!]_m & \overset{\mathrm{def}}{=} & \mathsf{Cur}([\![A]\!]_m,[\![t]\!]_m) \\
[\![\mathsf{Pair}]\!]_m & \overset{\mathrm{def}}{=} & \mathsf{Pair} & [\![\forall(A,t)]\!]_m & \overset{\mathrm{def}}{=} & \forall([\![A]\!]_m,[\![t]\!]_m) \\
[\![\pi_i]\!]_m & \overset{\mathrm{def}}{=} & \pi_i & & &
\end{array}$$

Table 4.3: Translation of Combinators into Eager Machine Code

A later rule applies only if all earlier ones fail.

	Context	Code	Canonical Form	Stack
$\langle\rangle$	f	$\langle\rangle\ C$	N	S
	$\langle\rangle$	C	N	S
Fst	$\mathsf{Fst}^k\langle\langle\rangle\ ,t\rangle?(A)$	$\mathsf{Fst}\ C$	N	S
	$\langle\rangle$	C	N	S
Fst	$\mathsf{Fst}^k\langle-\ ,t\rangle?(A)$	$\mathsf{Fst}\ C$	N	S
	Fst^{k+1}	C	N	S
Fst	$\mathsf{Fst}^k\langle f\ ,t\rangle?(A)$	$\mathsf{Fst}\ C$	N	S
	$\mathsf{Fst}^k f$	C	N	S
Fst	f	$\mathsf{Fst}\ C$	N	S
	$f\mathsf{Fst}$	C	N	S
$\langle$	f	$\langle\ C$	N	S
	f	C	N	S, f
$\langle\_$	f	$\langle\_\ C$	N	S
	f	C	N	$S, \langle\_, N$
$,$	g	$,\ C$	N	S, f
	f	C	$-$	S, g, N
$\rangle$	f	$\rangle\ C$	t	$S, \langle\_, N$
	$\langle\_\,t\rangle$	C	N	S
$\rangle$	f	$\rangle\ C$	t	S, g, N
	$\langle\ g,\ t\rangle$	C	N	S
$?$	f	$?(A)\ C$	N	S
	$f?(A)$	C	N	S
Ω	f	$\Omega\ C$	N	S
	f	C	Ω	S
T	$\mathsf{Fst}^k\langle\ f,\ h\forall(A,t)\rangle?(B)$	$T\ C$	N	S
	$\mathsf{Fst}^k\langle\ f,\ h\forall(A,t)\rangle?(B)$	C	$\mathsf{Fst}^k h\Pi(A,\langle\ \langle\rangle,t\rangle?(B)T)$	S
T	$\mathsf{Fst}^k\langle f,t\rangle?(A)$	$T\ C$	N	S
	$\mathsf{Fst}^k\langle f,t\rangle?(A)$	C	$\mathsf{Fst}^k\langle\langle\rangle,t\rangle?(\Omega)T$	S
T	Fst^k	$T\ C$	N	S
	Fst^k	C	$\langle\langle\rangle,\mathsf{Fst}^k\mathsf{Snd}\rangle?(\Omega)T$	S
A	f	$A\ C$	N	S
	f	C	fA	S
Snd	$\mathsf{Fst}^k\langle f,t\rangle?(A)$	$\mathsf{Snd}\ C$	$-$	S
	$\mathsf{Fst}^k\langle f,t\rangle?(A)$	C	$\mathsf{Fst}^k t$	S

Table 4.4: Transitions of the Eager Machine

	Context	Code	Canonical Form	Stack
Snd	–	$\mathsf{Snd}\ C$	–	S
	–	C	Snd	S
App	$\langle f,s\rangle?(B)$	$\mathsf{App}\ C$	$g\mathsf{Cur}(A,t)$	S
	$\langle g,s\rangle?(A)$	$t\ C$	–	S
App	Fst^k	$\mathsf{App}\ C$	$g\mathsf{Cur}(A,t)$	S
	$\langle g,\mathsf{Fst}^k\mathsf{Snd}\rangle?(A)$	$t\ C$	–	S
App	$\langle f,s\rangle?(B)$	$\mathsf{App}\ C$	N	S
	$\langle f,s\rangle?(B)$	C	$N\langle\_s\rangle\mathsf{App}$	S
App	Fst^k	$\mathsf{App}\ C$	N	S
	Fst^k	C	$N\langle\_\mathsf{Fst}^k\mathsf{Snd}\rangle\mathsf{App}$	S
pop	g	$\mathsf{pop}\ C$	N	S,f
	f	C	N	S
π_i	f	$\pi_i\ C$	$f\langle\langle g,t_1\rangle?(A),t_2\rangle?(B)\mathsf{Pair}$	S
	f	C	t_i	S
π_i	f	$\pi_i\ C$	N	S
	f	C	$N\pi_i$	S
t	f	$t\ C$	N	S
	f	C	ft	S

Table 4.4 (continued): Transitions of the Eager Machine

The grammar for the machine instructions is as follows:

$$\begin{array}{rcl} f & ::= & \langle\rangle \mid - \mid \mathsf{Fst} \mid \langle \mid \langle\_ \mid , \mid \rangle \mid ?(A) \\ A & ::= & \Omega \mid \Pi(A,A) \mid \Sigma(A,A) \mid T \mid T_C \\ t & ::= & \mathsf{Snd} \mid \mathsf{Cur}(A,t) \mid \mathsf{pop} \mid \mathsf{App} \mid \mathsf{Pair} \mid \pi_1 \mid \pi_2 \end{array}$$

The definition of the translation into machine code and the definition of the eager machine can be found in Tables 4.3 and 4.4. The correctness theorem follows exactly the same line as the corresponding theorem in the lazy case:

Theorem 4.2 *For every canonical context morphism f and all combinators g, A and t such that there exist combinators h, B and s satisfying $f; g \Rightarrow_L h$, $f * A \Rightarrow_L B$ and $f * t \Rightarrow_L s$ respectively, the machine performs the following actions:*

Environment	Code	Canonical Form	Stack
$[\![f]\!]_m$	$[\![g]\!]_m\ C$	N	S
	$\Downarrow^*_E$		
$[\![h]\!]_m$	C	N	S

Environment	Code	Canonical Form	Stack
$[\![f]\!]_m$	$[\![A]\!]_m\ C$	N	S
	$\Downarrow^*_E$		
?	C	$[\![B]\!]_m$	S

Environment	Code	Canonical Form	Stack
$[\![f]\!]_m$	$[\![t]\!]_m\ C$	$-$	S
	$\Downarrow^*_E$		
$[\![f]\!]_m$	C	$[\![s]\!]_m$	S

Proof By Theorem 3.11, it is enough to show that the theorem holds for the conclusion of any inference rule for the strategy $\Rightarrow_E$ if it holds for all of its premises.

$f; \langle\rangle \Rightarrow_E \langle\rangle$ The sequence is

Environment	Code	Canonical Form	Stack
$[\![f]\!]_m$	$\langle\rangle\ C$	N	S
	$\Downarrow_E$		
$\langle\rangle$	C	N	S

The rules for f; Id, Id; Fst, Fst^k; $\langle g, t[A]\rangle$; Fst and Fst^k; Fst are also directly translated into machine transitions.

$f; \langle g, t[A]\rangle \Rightarrow_E \langle h, s[A]\rangle$ The transition sequence is as follows:

	Environment	Code	Canonical Form	Stack
	$\llbracket f\rrbracket_m$	$\langle\llbracket g\rrbracket_m, \llbracket t\rrbracket_m\rangle?(\llbracket A\rrbracket_m)\ C$	N	S
		$\Downarrow_E$		
	$\llbracket f\rrbracket_m$	$\llbracket g\rrbracket_m, \llbracket t\rrbracket_m\rangle?(\llbracket A\rrbracket_m)\ C$	N	$S, \llbracket f\rrbracket_m$
I.H.		$\Downarrow_E^*$		
	$\llbracket h\rrbracket_m$	$, \llbracket t\rrbracket_m\rangle?(\llbracket A\rrbracket_m)\ C$	N	$S, \llbracket f\rrbracket_m$
		$\Downarrow_E$		
	$\llbracket f\rrbracket_m$	$\llbracket t\rrbracket_m\rangle?(\llbracket A\rrbracket_m)\ C$	–	$S, \llbracket h\rrbracket_m, N$
I.H.		$\Downarrow_E^*$		
	?	$\rangle?(\llbracket A\rrbracket_m)\ C$	$\llbracket s\rrbracket_m$	$S, \llbracket h\rrbracket_m, N$
		$\Downarrow_E$		
	$\langle\llbracket h\rrbracket_m, \llbracket s\rrbracket_m\rangle$	$?(\llbracket A\rrbracket_m)\ C$	N	S
		$\Downarrow_E$		
	$\langle\llbracket h\rrbracket_m, \llbracket s\rrbracket_m\rangle?(\llbracket A\rrbracket_m)$	C	N	S

$f; (g; h) \Rightarrow_E f'$ Construction.

Types

All cases for types but one correspond directly to machine transitions. The exception is the inference rule with the conclusion $f * T \Rightarrow_E \langle\langle\rangle, t[\Omega]\rangle * T$. The typing rules imply that either $f \equiv \mathsf{Fst}^k$ or $f \equiv \mathsf{Fst}^k; \langle f, t[A]\rangle$. Hence the last two transition rules for T are enough to capture this inference rule.

$f * (t; g * \mathsf{Id})$ The transition sequence is

	Environment	Code	Canonical Form	Stack
	$\llbracket f\rrbracket_m$	$\langle\llbracket t\rrbracket_m\mathsf{pop}\llbracket g\rrbracket_m -\ C$	–	S
		$\Downarrow_E$		
	$\llbracket f\rrbracket_m$	$\llbracket t\rrbracket_m\mathsf{pop}\llbracket g\rrbracket_m -\ C$	–	$S, \llbracket f\rrbracket_m$
I.H.		$\Downarrow_E$		
	?	$\mathsf{pop}\llbracket g\rrbracket_m -\ C$	$\llbracket s\rrbracket_m$	$S, \llbracket f\rrbracket_m$
		$\Downarrow_E$		

		$\Downarrow_E$		
	$[\![f]\!]_m$	$[\![g]\!]_m - C$	$[\![s]\!]_m$	S
I.H.		$\Downarrow_E^*$		
	$[\![h]\!]_m$	$- C$	$[\![s]\!]_m$	S
		$\Downarrow_E$		
	$[\![h]\!]_m$	C	$[\![s]\!]_m$	S

The cases $f * (t; g * \mathsf{Id})$ and $f * (g * \mathsf{Id}; t)$ are similar.

Snd Similar to Fst.

β-rule Because of typing constraints, we have either $f; g \Rightarrow_E \mathsf{Fst}^k; \langle f', t[A] \rangle$ or $f; g \Rightarrow_E \mathsf{Fst}^k$. We deal here only with the first case, the second is analogous.

	Environment	Code	Canonical Form	Stack
	$[\![f]\!]_m$	$\langle\ [\![t]\!]_m\mathsf{pop}\ [\![g]\!]_m\mathsf{App}\ C$	$-$	S
		$\Downarrow_E$		
	$[\![f]\!]_m$	$[\![t]\!]_m\mathsf{pop}[\![g]\!]_m\mathsf{App}\ C$	$-$	$S, [\![f]\!]_m$
I.H.		$\Downarrow_E^*$		
	?	$\mathsf{pop}[\![g]\!]_m\mathsf{App}\ C$	$[\![h]\!]_m\mathsf{Cur}([\![A]\!]_m[\![s_1]\!]_m)$	$S, [\![f]\!]_m$
		$\Downarrow_E$		
	$[\![f]\!]_m$	$[\![g]\!]_m\mathsf{App}\ C$	$[\![h]\!]_m\mathsf{Cur}([\![A]\!]_m[\![s_1]\!]_m)$	S
		$\Downarrow_E$		
	$\mathsf{Fst}^k; \langle [\![g']\!]_m, [\![t]\!]_m \rangle ?(A)$	$\mathsf{App}\ C$	$[\![h]\!]_m\mathsf{Cur}([\![A]\!]_m[\![s_1]\!]_m)$	S
		$\Downarrow_E^*$		
	$\langle [\![h]\!]_m, \mathsf{Fst}^k[\![t]\!]_m \rangle ?(A)$	$[\![s_1]\!]_m\ C$	N	S
I.H.		$\Downarrow_E^*$		
	?	C	$[\![s]\!]_m$	S

The remaining cases are similar to the β-rule.

□

4.4 Machines for Reduction to Normal Form

As the relation $\overset{N}{\leadsto}$ is defined on top of the relation $\overset{W}{\leadsto}$, the two machines for reduction to normal form are constructed on top of the two machines for reduction to canonical combinators. The former machines accept therefore the code for a canonical combinator as input and produce the code for the normal form as output by pushing the substitution inside the binding operations like Cur and $\forall$ and calling recursively the machines for reduction to canonical combinators and normal form. We will write

Input	Output
e	e'

for the behaviour of these machines, or $\mathrm{NF}(e) = e'$ whenever this is appropriate. The details are given in Tables 4.5 and 4.6. These tables use the abbreviation $\mathrm{CF}(f, d) = d'$, which means that the abstract machines for $\Rightarrow_L$ and $\Rightarrow_E$ respectively stop with the code d' in the canonical-form or environment register when started with the code f in the environment register and d in the code register.

The strong normalization of the reduction $\overset{N}{\leadsto}$ yields the correctness theorem:

Theorem 4.3 *If f, A and t are canonical combinators and f', A' and t' are their $\overset{N}{\leadsto}$-normal forms, then both machines perform the following action:*

Input	Output
$[\![f]\!]_m$	$[\![f']\!]_m$
$[\![A]\!]_m$	$[\![A']\!]_m$
$[\![t]\!]_m$	$[\![t']\!]_m$

Proof An induction over $\nu(f)$, $\nu(A)$ and $\nu(t)$ respectively does not suffice because in some cases we have to reduce the code for $\mathsf{Id} * A$ and $\mathsf{Id} * f$ to determine the normal form of A and f. Instead we use an operation $\mathsf{rem}(-)$ in the definition of the well-ordering for the induction. The combinator $\mathsf{rem}(e)$ is the result of replacing all subcombinators $\mathsf{Id}; f$, $f; \mathsf{Id}$, $\mathsf{Id} * A$ and $\mathsf{Id} * t$ in the combinator e by f, f, A and t respectively. The crucial property of this operation is that $e \Rightarrow e'$ implies $\mathsf{rem}(e) \overset{W}{\leadsto}^{*} \mathsf{rem}(e')$. An induction over the pair $(\mathsf{deg}(e), \mathsf{compl}(e))$ ordered lexicographically, where $\mathsf{deg}(e) \overset{\mathrm{def}}{=} \nu(\mathsf{rem}(e))$ and $\mathsf{compl}(e)$ denotes the complexity of $\mathsf{rem}(e)$, will prove the claim. We elaborate only on the cases $\mathsf{Id}*\Pi(A, B)$ and $f*\Pi(A, B)$ of the machine for lazy combinators because all other cases are similar.

$\mathsf{Id} * \Pi(A, B)$: Let $\Pi(A', B')$ be the normal form of $\Pi(A, B)$. By definition,

$$\mathsf{deg}(\mathsf{Id} * A) = \mathsf{deg}(A) \leq \mathsf{deg}(\Pi(A, B)) = \mathsf{deg}(\mathsf{Id} * \Pi(A, B))$$

If $\mathsf{Id} * A \Rightarrow_L A''$, then $\mathsf{rem}(A) \overset{W}{\leadsto}^{*} \mathsf{rem}(A'')$, and so $\mathsf{deg}(A'') \leq \mathsf{deg}(A)$. So $\mathsf{deg}(A'') = \mathsf{deg}(A)$ implies $\mathsf{rem}(A) \equiv \mathsf{rem}(A'')$. Hence

$$\begin{aligned}(\mathsf{deg}(A''), \mathsf{compl}(A'')) &\leq (\mathsf{deg}(A), \mathsf{compl}(A)) \\ &< (\mathsf{deg}(\Pi(A, B)), \mathsf{compl}(\Pi(A, B)))\end{aligned}$$

Input	Output
$\langle\rangle$	$\langle\rangle$
$\langle g, t\rangle?(A)$	$\langle \mathrm{NF}(\mathrm{CF}(-, g)), \mathrm{NF}(\mathrm{CF}(-, t))\rangle?(\mathrm{NF}(\mathrm{CF}(-, A)))$
$f\langle g, t\rangle?(A)$	$\langle \mathrm{NF}(\mathrm{CF}(f, g)), \mathrm{NF}(\mathrm{CF}(f, t))\rangle?(\mathrm{NF}(\mathrm{CF}(-, A)))$
$-$	$-$
Fst^n	Fst^n
$\Pi(A, B) * -$	$\Pi(\mathrm{NF}(\mathrm{CF}(-, A)), \mathrm{NF}(\mathrm{CF}(-, B)))$
$\Pi(A, B) * f$	$\Pi(\mathrm{NF}(\mathrm{CF}(f, A)), \mathrm{NF}(\mathrm{CF}(\langle \mathsf{Fst} f, \mathsf{Snd}\rangle?(A), B)))$
$\Sigma(A, B) * -$	$\Sigma(\mathrm{NF}(\mathrm{CF}(-, A)), \mathrm{NF}(\mathrm{CF}(-, B)))$
$\Sigma(A, B) * f$	$\Sigma(\mathrm{NF}(\mathrm{CF}(f, A)), \mathrm{NF}(\mathrm{CF}(\langle \mathsf{Fst} f, \mathsf{Snd}\rangle?(A), B)))$
Ω	Ω
$T * \langle\langle\rangle, t\rangle?(A)$	$T * \langle\langle\rangle, \mathrm{NF}(t)\rangle?(\Omega)$
$\mathsf{Cur}(A, t) * -$	$\mathsf{Cur}(\mathrm{NF}(\mathrm{CF}(-, A)), \mathrm{NF}(\mathrm{CF}(-, t)))$
$\mathsf{Cur}(A, t) * f$	$\mathsf{Cur}(\mathrm{NF}(\mathrm{CF}(f, A)), \mathrm{NF}(\mathrm{CF}(\langle \mathsf{Fst} f, \mathsf{Snd}\rangle?(A), t)))$
$\forall(A, t) * -$	$\forall(\mathrm{NF}(\mathrm{CF}(-, A)), \mathrm{NF}(\mathrm{CF}(-, t)))$
$\forall(A, t) * f$	$\forall(\mathrm{NF}(\mathrm{CF}(f, A)), \mathrm{NF}(\mathrm{CF}(\langle \mathsf{Fst} f, \mathsf{Snd}\rangle?(A), t)))$
$\mathsf{Snd} * \mathsf{Fst}^k$	$\mathsf{Snd} * \mathsf{Fst}^k$
$\mathsf{Pair} * \langle\langle g, t\rangle?(A), t'\rangle?(B)$	$\mathsf{Pair} * \langle\langle -, \mathrm{NF}(\mathrm{CF}(-, t))\rangle?(\mathrm{NF}(\mathrm{CF}(-, A * g))), \mathrm{NF}(\mathrm{CF}(-, t'))\rangle?(\mathrm{NF}(\mathrm{CF}(-, B * \langle \mathsf{Fst} g, \mathsf{Snd}\rangle?(A * g))))$
$\mathsf{Pair} * f\langle\langle g, t\rangle?(A), t'\rangle?(B)$	$\mathsf{Pair} * \langle\langle -, \mathrm{NF}(\mathrm{CF}(f, t))\rangle?(\mathrm{NF}(\mathrm{CF}(f, A * g))), \mathrm{NF}(\mathrm{CF}(f, t'))\rangle?(\mathrm{NF}(\mathrm{CF}(\langle \mathsf{Fst} fg, \mathsf{Snd}\rangle?(A), B)))$
π_i	π_i
$\mathsf{App} * \langle -, t * f\rangle$	$\mathsf{App} * \langle -, \mathrm{NF}(\mathrm{CF}(f, t))\rangle$
$t_1\ t_2$	$\mathrm{NF}(t_1)\ \mathrm{NF}(t_2)$

Table 4.5: Machine for Reduction of Lazy Canonical Combinators to Normal Form

Input	Output
$\langle\rangle$	$\langle\rangle$
$\mathsf{Fst}^k\langle f, t\rangle?(A)$	$\langle \mathrm{NF}(\mathsf{Fst}^k f), \mathrm{NF}(\mathsf{Fst}^k t)\rangle?(\mathrm{NF}(\mathrm{CF}(-, A)))$
$-$	$-$
Fst^k	Fst^k
$-\Pi(A, B)$	$\Pi(\mathrm{NF}(\mathrm{CF}(-, A)), \mathrm{NF}(\mathrm{CF}(-, B)))$
$f\Pi(A, B)$	$\Pi(\mathrm{NF}(\mathrm{CF}(f, A)), \mathrm{NF}(\mathrm{CF}(\langle \mathsf{Fst} f, \mathsf{Snd}\rangle?(A), B)))$
$-\Sigma(A, B)$	$\Sigma(\mathrm{NF}(\mathrm{CF}(f, A)), \mathrm{NF}(\mathrm{CF}(-, B)))$
$f\Sigma(A, B)$	$\Sigma(\mathrm{NF}(\mathrm{CF}(f, A)), \mathrm{NF}(\mathrm{CF}(\langle \mathsf{Fst} f, \mathsf{Snd}\rangle?(A), B)))$
Ω	Ω
$\langle\langle\rangle, t\rangle?(A)T$	$\langle\langle\rangle, \mathrm{NF}(t)\rangle?(\Omega)T$
$-\mathsf{Cur}(A, t)$	$\mathsf{Cur}(\mathrm{NF}(\mathrm{CF}(-, A)), \mathrm{NF}(\mathrm{CF}(-, t)))$
$f\mathsf{Cur}(A, t)$	$\mathsf{Cur}(\mathrm{NF}(\mathrm{CF}(f, A)), \mathrm{NF}(\mathrm{CF}(\langle \mathsf{Fst} f, \mathsf{Snd}\rangle?(A), t)))$
$-\forall(A, t)$	$\forall(\mathrm{NF}(\mathrm{CF}(-, A)), \mathrm{NF}(\mathrm{CF}(-, t)))$
$f\forall(A, t)$	$\forall(\mathrm{NF}(\mathrm{CF}(f, A)), \mathrm{NF}(\mathrm{CF}(\langle \mathsf{Fst} f, \mathsf{Snd}\rangle?(A), t)))$
$\mathsf{Fst}^k\mathsf{Snd}$	$\mathsf{Fst}^k\mathsf{Snd}$
$\langle\langle -, t\rangle?(A), t'\rangle?(B)\mathsf{Pair}$	$\langle\langle -, \mathrm{NF}(t)\rangle?(\mathrm{NF}(\mathrm{CF}(-, A))), \mathrm{NF}(t')\rangle$ $?(\mathrm{NF}(\mathrm{CF}(-, B)))$
$\mathsf{Fst}^k\langle\langle g, t\rangle?(A), t'\rangle?(B)\mathsf{Pair}$	$\langle\langle -, \mathrm{NF}(\mathsf{Fst}^k t)\rangle?(\mathrm{NF}(\mathrm{CF}(\mathsf{Fst}^k g, A))), \mathrm{NF}(\mathsf{Fst}^k t')\rangle$ $?(\mathrm{NF}(\mathrm{CF}(\langle \mathsf{Fst}^{k+1} g, \mathsf{Snd}\rangle?(A), B)))$
$\langle \_t\rangle\mathsf{App}$	$\langle \_\mathrm{NF}(t)\rangle\mathsf{App}$
π_i	π_i
$t_1\ t_2$	$\mathrm{NF}(t_1)\ \mathrm{NF}(t_2)$

Table 4.6: Machine for Reduction of Eager Canonical Combinators to Normal Form

Therefore the induction hypothesis yields $\mathrm{NF}(\mathrm{CF}(-, [\![A]\!]_m)) = [\![A']\!]_m$, and in the same way $\mathrm{NF}(\mathrm{CF}(-, [\![B]\!]_m)) = [\![B']\!]_m$. So we obtain

$$\mathrm{NF}([\![\mathsf{Id} * \Pi(A, B)]\!]_m) = [\![\Pi(A', B')]\!]_m$$

$f * \Pi(A, B)$: Let again be $\Pi(A', B')$ be the normal form of $f * \Pi(A, B)$. Because f is canonical and $f \not\equiv \mathsf{Id}$, we have also $\mathsf{rem}(f) \not\equiv \mathsf{Id}$. So $\mathsf{rem}(f * A) \equiv \mathsf{rem}(f) * \mathsf{rem}(A)$, and

$$\mathsf{rem}(\langle \mathsf{Fst}; f, \mathsf{Snd}[A]\rangle * B) = \langle \mathsf{Fst}; \mathsf{rem}(f), \mathsf{Snd}[\mathsf{rem}(A)]\rangle * \mathsf{rem}(B)$$

Hence $\mathsf{deg}(f * A) < \mathsf{deg}(f * \Pi(A, B))$ and

$$\mathsf{deg}(\langle \mathsf{Fst}; f, \mathsf{Snd}[A]\rangle * B) < \mathsf{deg}(f * \Pi(A, B))$$

Therefore the induction hypothesis implies that

$$\mathrm{NF}(\mathrm{CF}([\![f * A]\!]_m)) = [\![A']\!]_m$$

and

$$\mathrm{NF}(\mathrm{CF}([\![\mathsf{Fst}; f, \mathsf{Snd}[A]]\!]_m * B)) = [\![B']\!]_m$$

and now the claim follows directly.

□

One aspect of the definition of the canonical combinators improves the efficiency of the machines considerably, namely that if f is canonical, then $\langle \mathsf{Fst}; f, \mathsf{Snd}[A]\rangle$ is canonical as well. To see why, consider a canonical combinator $f * \mathsf{Cur}(A, t)$: the execution of the code for the canonical form of $\langle \mathsf{Fst}; f, \mathsf{Snd}[A]\rangle * t$ starts with $[\![\langle \mathsf{Fst}; f, \mathsf{Snd}[A]\rangle]\!]_m$ in the environment register rather than with the computation of the canonical combinator for it. Asperti [Asp92] also noticed the importance of postponing weakening during the reduction inside a binding operation.

4.5 Comparison with Other Abstract Machines

This section shows that the lazy and eager machines of section 4.2 and 4.3 are generalizations of the version of Krivine's machine and the CAM for the simply typed λ-calculus respectively. The only difference between the latter two machines and those for the untyped λ-calculus is that the term λt is replaced by $\lambda A.t$ and the combinator $\mathsf{Cur}(t)$ by $\mathsf{Cur}(A, t)$ respectively.

4.5.1 The Lazy Machine

Krivine [Kri85] describes an abstract machine for the reduction of untyped λ-terms to weak head-normal form. A more easily available account of this machine is given also in [Cré90] [Cur91]. We will review the version for the simply typed λ-calculus

Environment	Code	Stack
f	$(C_1 C_2)$	S
f	C_1	$S, (f, C_2)$
f	λt	$S, (h, C)$
$f :: (h, C)$	t	S
$f :: (h, C)$	0	S
h	C	S
$f :: (h, C)$	$n + 1$	S
f	n	S

Table 4.7: Krivine's machine

here briefly. It has three registers, called environment, code and stack. The first contains a list of closures (f, t), where f is an environment and t a term, and the elements of the stack are closures (f, t). An environment is in turn a list of closures, so we obtain a mutually recursive definition. The base case of this definition is the empty environment, which is the empty list. The transitions are given in Table 4.7. The machine starts with an empty stack, an empty environment register and with the term t in the code register and stops in a state

Environment	Code	Stack
$-$	n	$(h_1, t_1) \cdots (t_n, t_n)$

or

Environment	Code	Stack
f	$\lambda A.t$	$-$

corresponding to the weak head-normal form $nt'_1 \cdots t'_n$ and the closure $(f, \lambda A.t)$ respectively, where the t'_i are the result of the substitutiton of the environment h_i in t_i.

The aim is to recognize this machine as a special case of the lazy machine introduced in section 4.2. Because we consider only the simply typed λ-calculus, we can restrict the raw combinators to those corresponding to constant D-categories, namely

$$\begin{array}{lcl}
\Gamma & ::= & [\,] \mid \Gamma \cdot A \\
f & ::= & \langle\rangle \mid \mathsf{Id} \mid f;f \mid \mathsf{Fst} \mid \langle f, t\rangle \\
A & ::= & I \mid \Pi(A, A) \\
t & ::= & \mathsf{Id} \mid t;t \mid f * t \mid \mathsf{Snd} \mid \mathsf{Cur}(A, t) \mid \mathsf{App}
\end{array}$$

together with the combinator 1. The combinator I denotes an arbitrary base type. The reduction relations $\overset{W}{\leadsto}$ and $\overset{N}{\leadsto}$ can be adapted easily by removing the combinators

Environment	Code	Canonical Form
$\langle g, t * h\rangle$	$\mathsf{Snd}\ C$	N
h	$t\ C$	N
$\langle g, t\rangle$	$\mathsf{Snd} * \mathsf{Fst}^{n+1}\ C$	N
g	$\mathsf{Snd} * \mathsf{Fst}^{n}\ C$	N
f	$\mathsf{Cur}(A, t)\ C$	$N \mathsf{App} * \langle \mathsf{Id}, s\rangle$
$\langle f, s\rangle$	$t\ C$	N
f	$\mathsf{App} * \langle \mathsf{Id}, s\rangle\ C$	N
f	C	$N \mathsf{App} * \langle \mathsf{Id}, s * f\rangle$
Id	$\mathsf{Snd}\ C$	N
Id	C	$N \mathsf{Snd}$
f	$t\ C$	N
f	C	$N\ t * f$

Table 4.8: Restricted Lazy Machine

that do not occur in the above rules and replacing the combinators $\langle f, t[A]\rangle$ by $\langle f, t\rangle$ and $f * A$ by A. Because there are no dependent types in the simply-typed λ-calculus, the adaptation removes all reduction rules on types. So there are no transition rules for types in the machine either.

To obtain Krivine's machine, we restrict the combinators even further. During the reduction of combinators that are translations of λ-expressions only the code for the combinators

$$
\begin{array}{rcl}
f & ::= & \langle\rangle \mid \langle f, t\rangle \\
u & ::= & \mathsf{Fst}^k * \mathsf{Snd} \mid \mathsf{Cur}(A, u) \mid \langle \mathsf{Id}, u\rangle * \mathsf{App} \mid u; u \\
t & ::= & u \mid f * u \mid u; \langle \mathsf{Id}, f * u\rangle * \mathsf{App}
\end{array}
$$

occurs in the environment register, the code register and in the canonical-form register respectively. If we consider only the transitions of the lazy machine concerning the restricted combinators, we obtain the machine given in Table 4.8. The marking of the code of certain context morphisms as canonical is not necessary because the restricted machine handles only code for canonical context morphisms anyway. Krivine's machine and this machine become identical if

- we translate every term t into the code $\llbracket\llbracket t\rrbracket\rrbracket_m$, and replace every closure (f, t) by $f' * \llbracket\llbracket t\rrbracket\rrbracket_m$ and every environment $((f_1, t_1), \ldots (f_n, t_n))$ by $\langle f_1' * \llbracket\llbracket t_1\rrbracket\rrbracket_m, \ldots,$ $f_n' * \llbracket\llbracket t_n\rrbracket\rrbracket_m\rangle$, where f', $f_1', \ldots, f_n'$ are the combinators corresponding to the environments $f, f_1, \ldots, f_n$,

- we remove the last two transitions of the latter machine, so that the final states correspond to each other, and

- we replace every stack $(h_1, t_1), \ldots, (h_n, t_n)$ of closures by the code $\mathsf{App} * \langle -, [\![[\![t_1]\!]]\!]_m * h'_1\rangle \cdots \mathsf{App} * \langle -, [\![[\![t_n]\!]]\!]_m * h'_n\rangle$ in the canonical-form register, where $h'_1, \ldots, h'_n$ are the combinators corresponding to $h_1, \ldots, h_n$.

The $\lambda\sigma$-calculus has been used to derive extensions of Krivine's machine for reduction to normal form [ACCL90] [Cré90]. The calculus adds explicit substitution on top of the λ-calculus. In our notation, the typed version has the raw expressions

$$\begin{array}{rcl}
\Gamma & ::= & [\,] \mid \Gamma \cdot A \\
f & ::= & \langle\rangle \mid \mathsf{Id} \mid f;f \mid \mathsf{Fst} \mid \langle f, t\rangle \\
A & ::= & I \mid \Pi(A, A) \\
t & ::= & \mathsf{Snd} \mid \lambda A.t \mid tt \mid f * t
\end{array}$$

An expression f is usually called a *substitution*. The typing rules of the raw expressions are those of simply-typed λ-calculus with the following rules for the extra expressions:

$$\frac{}{\Gamma \vdash \langle\rangle : [\,]} \qquad \frac{}{\Gamma \vdash \mathsf{Id} : \Gamma} \qquad \frac{\Gamma \vdash f : \Delta \quad \Gamma \vdash t : A}{\Gamma \vdash \langle f, t\rangle : \Delta \cdot A}$$

$$\frac{}{\Gamma \cdot A \vdash \mathsf{Fst} : \Gamma} \qquad \frac{\Gamma \vdash f : \Gamma' \quad \Gamma' \vdash g : \Gamma''}{\Gamma \vdash f; g : \Gamma''}$$

This calculus becomes a subsystem of the combinators for constant D-categories if we replace ts by $t; \langle \mathsf{Id}, s\rangle * \mathsf{App}$ and the judgement $\Gamma \vdash t : A$ by $\Gamma \rhd t : 1 \rightarrow A$. Therefore the D-categories are an appropriate categorical framework for the $\lambda\sigma$-calculus.

An extension of Krivine's machine to a machine for the reduction of a term $f * t$ to a normal form is given in [ACCL90] as well. It does not use the notion of a canonical environment f and hence does not reduce environments to a canonical form but lists all possible cases for the reduction of $f * (g * t)$, where the code for f is stored in the environment register and the code for $g * t$ in the code register. Hence the access to an environment $\langle \mathsf{Fst}; f, t\rangle$ is more complicated than in the machine presented here.

Crégut's machine [Cré90] is based on the variant of the $\lambda\sigma$-calculus given in [Cur91] in connection with multicategories. This variant replaces the substitutions Fst and Id by the lists $(n-1, \ldots, 1)$ and $(n-1, \ldots, 0)$, where n is the number of free variables of the expression to which this substitution is applied. Hence all expressions have to be decorated with the number of free variables in order to formulate the reduction rule corresponding to $f * \mathsf{Cur}(A, t) \overset{N}{\rightsquigarrow} \mathsf{Cur}(A, \langle \mathsf{Fst}; f, \mathsf{Snd}[A]\rangle * t)$. Furthermore the substitution cannot be stored directly in the environment register because otherwise the transformation from the substitution corresponding to a combinator f to that corresponding to the combinator $\langle \mathsf{Fst}; f, \mathsf{Snd}\rangle$ becomes prohibitively expensive. Crégut's machine avoids such a complicated manipulation by handling the free variables that occur during the reduction inside a λ-abstraction differently from the other variables. For this purpose the machine maintains an index that is increased every time we start a reduction inside a λ-abstraction. The value of this index is stored as the value of such a free variable. This is a kind of reverse de

Term	Code	Stack
$\langle f, g\rangle$	$\mathsf{Fst}\,C$	S
f	C	S
$\langle f, g\rangle$	$\mathsf{Snd}\,C$	S
g	C	S
$\langle f\mathsf{Cur}(t), s\rangle$	$\mathsf{App}\,C$	S
$\langle f, s\rangle$	$t\,C$	S
f	$\langle\, C$	S
f	C	S, f
f	$,\, C$	S, h
h	C	S, f
g	$\rangle\, C$	S, f
$\langle f, g\rangle$	C	S
f	$t\,C$	S
ft	C	S

Table 4.9: The CAM

Bruijn-numbering, where the origin is the root of the term. The translation from this reverse index to the normal de Bruijn-index is done whenever the value of such a variable is computed. The special role of these free variables has no counterpart in the $\lambda\sigma$-calculus, and so the correspondence between the calculus and the machine is based on a translation of the reverse indexing into standard de Bruijn-numbers.

4.5.2 The Eager Machine

The CAM [CCM87] reduces combinators for the untyped λ-calculus according to an eager strategy to combinators corresponding to weak head-normal forms. The combinators, which are based on cartesian closed categories, are given by the BNF

$$f \;::=\; \langle\rangle \mid \mathsf{Fst} \mid \mathsf{Snd} \mid f;f \mid \mathsf{App} \mid \langle f, f\rangle \mid \mathsf{Cur}(t)$$

The machine has three registers, called *Term*, *Code* and *Stack*. The first contains the environment or the value, and the second the code that is to be executed. The transitions are given in Table 4.9. The CAM starts with a combinator $\langle f_1, \ldots, f_n\rangle$, where $f_1, \ldots, f_n$ are combinators corresponding to weak head-normal forms, in the term register, an empty stack and the code for the term t that is to be reduced in the code register. It stops with the code for the weak head-normal form of $t[x_i \backslash t_i]$, where t_i is the term corresponding to the combinator f_i, in the term register and an empty stack and code register.

In the same way as in the lazy case the connection between the CAM and the eager machine is based on a restriction of the combinators to those arizing during the reduction of translations of λ-expressions. We consider the eager machine limited to the transitions dealing with the code for the combinators

$$\begin{array}{rcl} f & ::= & \langle\rangle \mid \langle f,t\rangle \\ u & ::= & \mathsf{Fst}^k * \mathsf{Snd} \mid \mathsf{Cur}(A,u) \mid \langle \mathsf{Id},u\rangle * \mathsf{App} \mid u;u \\ t & ::= & u \mid f * \mathsf{Cur}(A,u) \end{array}$$

in the environment, code and canonical-form register respectively. This yields the machine

Environment	Code	Canonical Form	Stack
$\langle f,t\rangle$	$\mathsf{Fst}\ C$	N	S
f	C	N	S
$\langle f,t\rangle$	$\mathsf{Snd}\ C$	N	S
$\langle f,t\rangle$	C	t	S
f	$\langle\_\ C$	N	S
f	C	$-$	$S,N,\langle\_$
f	$\rangle\ C$	t	$S,\langle\_,N$
$\langle\_\ t\rangle$	C	N	S
$\langle\_s\rangle$	$\mathsf{App}\ C$	$f\mathsf{Cur}(A,t)$	S
$\langle f,s\rangle$	t	$-$	S
f	$\langle\ C$	N	S
f	C	N	S,N
f	$\mathsf{pop}\ C$	N	S,h
h	C	N	S
f	$t\ C$	N	S
f	C	Nft	S

It is no surprise that this machine and the CAM are not the same because the CAM is based on a different categorical structure, namely cartesian closed categories. However, the identification of environments and morphisms that leads from constant D-categories to cartesian closed categories turns also this machine into the CAM. If we merge the environment and the canonical-form register, the first three transitions become identical with their counterparts in the CAM. The correspondence between the combinator $t; \langle \mathsf{Id}, s\rangle * \mathsf{App}$ for D-categories and the combinator $\langle t, s\rangle; \mathsf{App}$ for cartesian closed categories transforms the transition

Environment	Code	Canonical Form	Stack
$\langle \mathsf{Id},s\rangle$	$\mathsf{App}\ C$	$f\mathsf{Cur}(A,t)$	S
		$\Downarrow$	
$\langle f,s\rangle$	$t\ C$	N	S

into

Canonical Form	Code	Stack
$\langle f\mathsf{Cur}(A,t),s\rangle$	$\mathsf{App}\ C$	S
	$\Downarrow$	
$\langle f,s\rangle$	$t\ C$	S

and the transition $\rangle$, which occurs only in the code for a combinator $\langle \mathsf{Id}, t\rangle * \mathsf{App}$, into the transition $\rangle$ of the CAM. The $\langle$_-transition occurs always after a pop-transition, and so both are amalgamated to one transition

Canonical Form	Code	Stack
f	$,C$	S, h
	$\Downarrow$	
h	C	S, f

Finally, the last transition rule of the restricted eager machine and the CAM are identified. Hence we obtain the CAM.

Chapter 5

The Inference System

Because of the dependent types the task of checking if a term t has a given type A in a given context Γ may involve reductions. This task can be reduced to that of calculating a type A' such that $\Gamma \vdash t: A'$, the so-called type synthesis, because if A_1 and A_2 are two types such that $\Gamma \vdash t: A_1$ and $\Gamma \vdash t: A_2$ then A_1 and A_2 are convertible, which is decidable. We will present in this chapter an algorithm for type synthesis based on the reduction machines of the previous chapter. Afterwards we show how to treat local variables in this framework.

5.1 An Abstract Machine for Type Synthesis

The inference rules for well-typed expressions do not specify an algorithm for type synthesis because they are not syntax-directed. The reason is the conversion rule

$$\frac{\Gamma \vdash t: A \qquad A \leftrightarrow^* B \qquad \Gamma \vdash B \text{ type}}{\Gamma \vdash t: B}$$

which may be applied at any stage during the derivation of the well-formedness. Harper and Pollack [HP91] solve this problem by observing that the conversion rule is only necessary at certain stages during type checking and moreover can be replaced by two tests. The first determines if a type A is a dependent product or a dependent sum by a reduction of A to its weak head normal form (WHNF), which yields directly the outermost constructor, and the second is a test for convertibility.

This line of thought applies also to the type checking of combinators. It can likewise be reduced to type synthesis, and the conversion rule for well-typed combinators raises exactly the same problem. A solution consists as above of restricting the application of the conversion rule and replacing it with the above tests. Because the combinators have an explicit substitution operation it is not necessary to reduce combinators until the translation of a WHNF is reached, but we can stop at a combinator $f * B$ if we know that the outermost constructor of $f * B$ is that of B. The canonical combinators are defined in such a way that on one hand f is as general as possible and on the other hand the above property of $f * B$ still holds. As we will see in a moment, the postponement of substitution captured by this definition is crucial for the efficiency of type synthesis. So both reduction machines described in the previous chapter basically implement the first test. The convertibility test is based on a modification of the machine for the reduction to normal form: A_1 and A_2 are convertible if their canonical forms B_1 and B_2 have the same outermost constructor and the components of B_1 and B_2 are convertible, otherwise not.

The combinators pose one additional problem, however. Consider the combinator $t; g * \mathsf{App}$. Suppose we have $\Gamma \rhd t{:}\,1 \to \Pi(A, B)$ and $\Gamma \rhd g{:}\,\Delta \cdot A'$. To make the combinator $t; g * \mathsf{App}$ well-formed, we need a type B such that $\langle \mathsf{Fst}; g; \mathsf{Fst}, \mathsf{Snd}[A'] \rangle * B' \leftrightarrow^* B$. But such a B' cannot be derived from Γ, t and g in general. This problem occurs because we have omitted the type information in the application but have essentially retained the rather restrictive typing rules for the application when we defined the implicit combinators. The solution is therefore to use the reduction $g * \mathsf{App} \leadsto \langle \mathsf{Id}, g * \mathsf{Snd} \rangle * \mathsf{App}$, and typecheck the latter combinator, which is the translation of the application in the Calculus of Constructions. In this case we have $\langle \mathsf{Fst}; g; \mathsf{Fst}, \mathsf{Snd}[A'] \rangle * B' \leftrightarrow^* B$, and so we can choose B' to be B. The combinators $g * \mathsf{Id}$ and $g * \pi_i$ cause similar problems. Therefore, we define a map $\mathtt{rm}$, which assigns to every combinator e the combinator e' obtained by replacing all subcombinators $t; f * (g * s)$, $t; f * (s_1; s_2)$, $g * \mathsf{App}$, $g * \mathsf{Id}$ and $g * \pi_i$ by $\mathtt{rm}(t; (f; g) * s)$, $\mathtt{rm}(t; f * s_1; f * s_2)$, $\langle \mathsf{Id}, \mathtt{rm}(g) * \mathsf{Snd} \rangle * \mathsf{App}$, Id and π_i respectively.

So the modified inference system $\vdash_{\mathrm{TS}}$, specified in Table 5.1, will use judgements like $f \Rrightarrow g$, where $\Rrightarrow$ can be either $\Rrightarrow_E$ or $\Rrightarrow_L$, and the above test for convertibility. It specifies an algorithm for type synthesis because it is completely syntax-directed: for every combinator c there is at most one possible derivation $\Gamma \vdash_{\mathrm{TS}} c{:}\,\Delta$, $\Gamma \vdash_{\mathrm{TS}} c$ or $\Gamma \vdash_{\mathrm{TS}} c{:}\,1 \to A$ respectively, which is completely determined by the structure of c.

Theorem 5.1 *For every context Γ and implicit combinator f, A and t:*

$$\begin{array}{lrll}
(i) & \Gamma \vdash_{\mathrm{TS}} f{:}\,\Delta & \text{implies} & \mathtt{rm}(\Gamma) \rhd \mathtt{rm}(f){:}\,\mathtt{rm}(\Delta) \\
(ii) & \Gamma \vdash_{\mathrm{TS}} A & \text{implies} & \mathtt{rm}(\Gamma) \rhd \mathtt{rm}(A) \\
(iii) & \Gamma \vdash_{\mathrm{TS}} t{:}\,1 \to A & \text{implies} & \mathtt{rm}(\Gamma) \rhd \mathtt{rm}(t){:}\,1 \to \mathtt{rm}(A)
\end{array}$$

Proof Induction over the definition of $\vdash_{\mathrm{TS}}$. We consider only the cases $\langle f, t[A] \rangle$ and $t; g * \mathsf{App}$.

$\langle f, t[A] \rangle$: The induction hypothesis yields $\mathtt{rm}(\Gamma) \rhd \mathtt{rm}(f){:}\,\mathtt{rm}(\Delta)$, $\mathtt{rm}(\Gamma) \rhd \mathtt{rm}(A)$ and $\mathtt{rm}(\Gamma) \rhd \mathtt{rm}(t){:}\,1 \to \mathtt{rm}(A')$. Because $\mathtt{rm}(f * A) \leftrightarrow^* \mathtt{rm}(A')$, we have also $\mathtt{rm}(\Gamma) \rhd \mathtt{rm}(\langle f, t[A] \rangle){:}\,\mathtt{rm}(\Delta) \cdot \mathtt{rm}(A)$.

$t; g * \mathsf{App}$: The induction hypothesis implies that

$$\mathtt{rm}(\Gamma) \rhd \mathtt{rm}(t){:}\,1 \to \mathtt{rm}(C) \text{ with } \mathtt{rm}(C) \leftrightarrow^* f * \Pi(A_1, B)$$

and $\mathtt{rm}(\Gamma) \rhd \mathtt{rm}(g * \mathsf{Snd}){:}\,1 \to \mathtt{rm}(A_2)$ with $\mathtt{rm}(A_2) \leftrightarrow^* f * A_1$, and therefore $f * \Pi(A_1, B) \leftrightarrow^* \Pi(\mathtt{rm}(A_2), \langle \mathsf{Fst}; f, \mathsf{Snd}[A_1], B \rangle)$. Hence

$$\mathtt{rm}(\Gamma) \rhd \mathtt{rm}(t); \mathtt{rm}(g) * \mathsf{App}{:}\,1 \to \mathtt{rm}(\langle f, g * \mathsf{Snd}[A_1] \rangle * B)$$

□

Context Morphisms

$$\frac{}{\Gamma \vdash_{\mathrm{TS}} \langle\rangle : [\,]} \qquad \frac{}{\Gamma \vdash_{\mathrm{TS}} \mathsf{Id} : \Gamma} \qquad \frac{\Gamma \vdash_{\mathrm{TS}} f : \Gamma' \quad \Gamma' \vdash_{\mathrm{TS}} g : \Gamma''}{\Gamma \vdash_{\mathrm{TS}} f ; g : \Gamma''} \qquad \frac{}{\Gamma \cdot A \vdash_{\mathrm{TS}} \mathsf{Fst} : \Gamma}$$

$$\frac{\Gamma \vdash_{\mathrm{TS}} f : \Delta \quad \Delta \vdash_{\mathrm{TS}} A \quad \Gamma \vdash_{\mathrm{TS}} t : 1 \rightarrow A' \quad \mathtt{rm}(f * A) \leftrightarrow^{*} \mathtt{rm}(A')}{\Gamma \vdash_{\mathrm{TS}} \langle f, t[A] \rangle : \Delta \cdot A}$$

Types

$$\frac{\Gamma \vdash_{\mathrm{TS}} f : \Delta \quad \Delta \vdash_{\mathrm{TS}} A}{\Gamma \vdash_{\mathrm{TS}} f * A} \qquad \frac{\Gamma \vdash_{\mathrm{TS}} A \quad \Gamma \cdot A \vdash_{\mathrm{TS}} B}{\Gamma \vdash_{\mathrm{TS}} \Pi(A, B)}$$

$$\frac{\Gamma \vdash_{\mathrm{TS}} A \quad \Gamma \cdot A \vdash_{\mathrm{TS}} B}{\Gamma \vdash_{\mathrm{TS}} \Sigma(A, B)} \qquad \frac{}{\Gamma \vdash_{\mathrm{TS}} \Omega} \qquad \frac{\mathtt{rm}(C) \Rightarrow \Omega}{[\,] \cdot C \vdash_{\mathrm{TS}} T}$$

Morphisms

$$\frac{\Gamma \vdash_{\mathrm{TS}} t : 1 \rightarrow A}{\Gamma \vdash_{\mathrm{TS}} t ; g * \mathsf{Id} : 1 \rightarrow A} \qquad \frac{}{\Gamma \cdot A \vdash_{\mathrm{TS}} \mathsf{Snd} : 1 \rightarrow \mathsf{Fst} * A}$$

$$\frac{\Gamma \vdash_{\mathrm{TS}} f : \Delta \quad \Delta \vdash_{\mathrm{TS}} t : 1 \rightarrow A}{\Gamma \vdash_{\mathrm{TS}} f * t : 1 \rightarrow f * A} \qquad \frac{\Gamma \vdash_{\mathrm{TS}} A \quad \Gamma \cdot A \vdash_{\mathrm{TS}} t : 1 \rightarrow B}{\Gamma \vdash_{\mathrm{TS}} \mathsf{Cur}(A, t) : 1 \rightarrow \Pi(A, B)}$$

$$\frac{\Gamma \vdash_{\mathrm{TS}} t : 1 \rightarrow C \quad \mathtt{rm}(C) \Rightarrow f * \Pi(A_1, B) \quad \Gamma \vdash_{\mathrm{TS}} g * \mathsf{Snd} : 1 \rightarrow A_2 \quad f * A_1 \leftrightarrow^{*} \mathtt{rm}(A_2)}{\Gamma \vdash_{\mathrm{TS}} t ; g * \mathsf{App} : 1 \rightarrow \langle f, g * \mathsf{Snd}[A_1] \rangle * B}$$

$$\frac{\Gamma \vdash_{\mathrm{TS}} \langle f, t[A], s[B] \rangle : \Delta \cdot A \cdot B}{\Gamma \vdash_{\mathrm{TS}} \langle f, t[A], s[B] \rangle * \mathsf{Pair} : 1 \rightarrow f * \Sigma(A, B)}$$

$$\frac{\Gamma \vdash_{\mathrm{TS}} t : 1 \rightarrow C \quad \mathtt{rm}(C) \Rightarrow f * \Sigma(A, B)}{\Gamma \vdash_{\mathrm{TS}} t ; g * \pi_1 : 1 \rightarrow f * A}$$

$$\frac{\Gamma \vdash_{\mathrm{TS}} t : 1 \rightarrow C \quad \mathtt{rm}(C) \Rightarrow f * \Sigma(A, B)}{\Gamma \vdash_{\mathrm{TS}} t ; g * \pi_2 : 1 \rightarrow \langle f, t ; \pi_1[A] \rangle * B}$$

$$\frac{\Gamma \vdash_{\mathrm{TS}} A \quad \Gamma \cdot A \vdash_{\mathrm{TS}} t : 1 \rightarrow B \quad \mathtt{rm}(B) \Rightarrow \Omega}{\Gamma \vdash_{\mathrm{TS}} \forall(A, t) : 1 \rightarrow \Omega}$$

$$\frac{\Gamma \vdash_{\mathrm{TS}} t ; f * s_1 ; f * s_2 : 1 \rightarrow A}{\Gamma \vdash_{\mathrm{TS}} t ; f * (s_1 ; s_2) : 1 \rightarrow A} \qquad \frac{\Gamma \vdash_{\mathrm{TS}} t ; (f ; g) * s : 1 \rightarrow A}{\Gamma \vdash_{\mathrm{TS}} \vdash_{\mathrm{TS}} t ; (f * (g * s)) : 1 \rightarrow A}$$

Table 5.1: Type Synthesis for Combinators for the Calculus of Constructions

The completeness theorem is as expected:

Theorem 5.2 *For every implicit combinator f, A, t and context Γ:*

(i) $\Gamma \rhd f\colon \Delta$ *implies for any well-formed context* $\Gamma' \leftrightarrow^* \Gamma$ *the existence of a context* $\Delta' \leftrightarrow^* \Delta$ *such that* $\Gamma' \vdash_{\mathrm{TS}} f\colon \Delta'$

(ii) $\Gamma \rhd A$ *implies that any well-formed context* $\Gamma' \leftrightarrow^* \Gamma$ *satisfies* $\Gamma' \vdash_{\mathrm{TS}} A$

(iii) $\Gamma \rhd t\colon 1 \to A$ *implies that for any well-formed context* $\Gamma' \leftrightarrow^* \Gamma$ *there exists a type* $A' \leftrightarrow^* A$ *such that* $\Gamma' \vdash_{\mathrm{TS}} t\colon 1 \to A'$.

Proof Induction over the derivation of $\Gamma \rhd e\colon A$. Again, we consider only the cases $\langle f, t[A]\rangle$ and $t; g * \mathsf{App}$.

$\langle f, t[A]\rangle$: By induction hypothesis there exists a context Δ' such that

$$\Gamma' \vdash_{\mathrm{TS}} f\colon \Delta', \quad \Delta' \vdash_{\mathrm{TS}} t\colon 1 \to A'', \quad f * A' \leftrightarrow^* A''$$

Because $f * A' \leftrightarrow^* A''$ implies $\mathtt{rm}(f * A') \leftrightarrow^* \mathtt{rm}(A'')$, we get

$$\Gamma \vdash_{\mathrm{TS}} \langle f, t[A']\rangle\colon \Delta' \cdot A'$$

$t; g * \mathsf{App}$: The confluence on types shows that there exist combinators A', B' and A_1 such that $A' \leftrightarrow^* A_1$, $\Gamma \rhd t\colon 1 \to \Pi(A', B')$ and $\Gamma \rhd g * \mathsf{Snd}\colon 1 \to A_1$. So the induction hypothesis yields for any context $\Gamma' \leftrightarrow^* \Gamma$ the existence of types C and D convertible to $\Pi(A, B)$ and A_1 respectively such that $\Gamma' \vdash_{\mathrm{TS}} t\colon 1 \to C$ and $\Gamma' \vdash_{\mathrm{TS}} g * \mathsf{Snd}\colon 1 \to D$. The confluence implies that $\mathtt{rm}(C) \Rightarrow f * \Pi(A, B)$ with $f * A \leftrightarrow^* \mathtt{rm}(A')$. Hence $\Gamma' \vdash_{\mathrm{TS}} t; g * \mathsf{App}\colon 1 \to \langle f, g * \mathsf{Snd}[A]\rangle * B$.

□

An abstract machine for type synthesis, which succeeds with result Δ, A or B for a context morphism f, a type A and morphism t such that $\Gamma \vdash_{\mathrm{TS}} f\colon \Delta$, $\Gamma \vdash_{\mathrm{TS}} A$ and $\Gamma \vdash_{\mathrm{TS}} t\colon 1 \to B$ respectively and returns an error otherwise, can be developed along the same lines as the reduction machines. It has the same architecture except that the context register now contains the machine code of the context Γ on the left side of the turnstyle $\vdash_{\mathrm{TS}}$. Some transitions can only be activated if the side conditions corresponding to the clauses $\mathtt{rm}(A) \Rightarrow f * B$ or $\mathtt{rm}(A_1) \leftrightarrow^* \mathtt{rm}(A_2)$ are fulfilled. In these cases the appropriate reduction machines can be directly activated as outlined above because they will perform the $\mathtt{rm}$-function during reduction. The machine is given in Table 5.2. The symbol $\leftrightarrow^*$ denotes the convertibility test based on the reduction machines of the previous chapter.

An induction over the definition of $\vdash_{\mathrm{TS}}$ and the structure of raw combinators shows that the machine computes the types of well-formed combinators and rejects all others.

Theorem 5.3 *For any well-formed context* Γ *and any raw combinator* f, A *and* t *we have*

	Context	Code	Type	Stack	Condition
$\langle\rangle$	Γ	$\langle\rangle\ C$	D	S	
	$[\,]$	C	D	S	–
Id	Γ	$\mathsf{Id}\ C$	D	S	
	Γ	C	D	S	–
$\langle - \rangle$	Γ	$\langle f, t\rangle\ C$	D	S	
	Γ	$f,t\ C$	D	S, f, Γ	–
,	Γ	$,\ C$	D	S, Δ	
	Δ	C	1	S, Γ	–
?	Γ	$?(A)\ C$	B	S, Δ	
	Δ	$A?_{\mathbf{c}}\ C$	B	S, Δ, B	–
$?_{\mathbf{c}}$	Γ	$?_{\mathbf{c}}\ C$	A	S, f, Δ, B	
	$\Delta \cdot A$	C	A	S	$f * A \leftrightarrow^* B$
$*$	Γ	$f * A\ C$	D	S	
	Γ	$fA *_C\ C$	D	S, f	–
$*_C$	Γ	$*_C\ C$	A	S, f	
	Γ	C	$f * A$	S	–
Π	Γ	$\Pi(A, B)\ C$	D	S	
	Γ	$A\mathtt{exc}B\Pi_C\ C$	D	S, Γ	–
exc	Γ	$\mathtt{exc}\ C$	A	S, Δ	
	$\Delta \cdot A$	C	1	S, A	–
Π_C	Γ	$\Pi_C\ C$	B	S, A	
	Γ	C	$\Pi(A, B)$	S	–
Σ	Γ	$\Sigma(A, B)\ C$	D	S	
	Γ	$A\mathtt{exc}B\Sigma_C\ C$	D	S, Γ	–
Σ_C	Γ	$\Sigma_C\ C$	B	S, A	
	Γ	C	$\Sigma(A, B)$	S	–
Ω	Γ	$\Omega\ C$	D	S	
	Γ	C	Ω	S	–
T	$[\,] \cdot A$	$T\ C$	D	S	
	$[\,] \cdot A$	C	T	S	$A \Rightarrow \Omega$
Id	Γ	$\mathsf{Id}\ C$	A	S	
	Γ	C	A	S	–
Id	Γ	$g * \mathsf{Id}\ C$	A	S	
	Γ	C	A	S	–
Snd	$\Gamma \cdot A$	$\mathsf{Snd}\ C$	1	S	
	$\Gamma \cdot A$	C	$\mathsf{Fst} * A$	S	–

Table 5.2: Abstract Machine for Type Synthesis

	Context	Code	Type	Stack
Cur	Γ	$\mathsf{Cur}(A,t)\ C$	1	S
	Γ	$A\mathsf{exc}\ t\ \mathsf{Cur}_C\ C$	1	S,Γ
Cur_C	Γ	$\mathsf{Cur}_C\ C$	B	S,A
	Γ	C	$\Pi(A,B)$	S
App_1	Γ	$g * \mathsf{App}$	D	S
	Γ	$g * \mathsf{SndApp}\ C$	1	$S,\Gamma,D',g*\mathsf{Snd}$[1]
App_1	Γ	$\langle \_s\rangle * \mathsf{App}$	D	S
	Γ	$s\mathsf{App}\ C$	1	S,Γ,D',s[1]
App_2	Γ	$\mathsf{App}\ C$	A'	S,Δ,D',s
	Δ	C	$\langle f,s\rangle?(A) * B$	S[2]
$\forall$	Γ	$\forall(A,t)\ C$	1	S
	Γ	$A\mathsf{exc}\ t\ \forall_C\ C$	1	S,Γ
$\forall_C$	Γ	$\forall_C\ C$	B	S,A
	Γ	C	Ω	S[3]
Pair_1	Γ	$\langle\langle f,t\rangle?(A),s\rangle?(B) * \mathsf{Pair}$	1	S
	Γ	$\langle\langle f,t\rangle?(A),s\rangle?(B)\mathsf{Pair}_C$	1	S,Γ,f
Pair_2	$\Gamma\cdot A\cdot B$	$\mathsf{Pair}_C\ C$	1	S,Δ,f
	Δ	C	$f*\Sigma(A,B)$	S
π_1	Γ	$\pi_1\ C$	$f*\Sigma(A,B)$	S
	Γ	C	$f*A$	S
π_1	Γ	$g*\pi_1\ C$	$f*\Sigma(A,B)$	S
	Γ	C	$f*A$	S
$t;\pi_2$	Γ	$t;\pi_2\ C$	1	S
	Γ	$t\pi_2\ C$	1	S,t
$t;\pi_2$	Γ	$t;g*\pi_2\ C$	1	S
	Γ	$t\pi_2\ C$	1	S,t
π_2	Γ	$\pi_2\ C$	$f*\Sigma(A,B)$	S,t
	Γ	C	$\langle f,t;\pi_1\rangle?(A)*B$	S
$f*t$	Γ	$f*t$	1	S
	Γ	$ft*_C$	1	S
$f*(t;s)$	Γ	$f*(t\ s)$	1	S
	Γ	$f*tf*s$	1	S
$f;g*s$	Γ	$f*(g*s)$	1	S
	Γ	$(fg)*s$	1	S

Table 5.2 (continued): Abstract Machine for Type Synthesis

[1] This rule applies only if $D \Rightarrow D' \equiv f * \Pi(A,B)$

[2] This rule applies only if $A' \leftrightarrow^* f * A$ and $D' \equiv f * \Pi(A,B)$

[3] This rule applies only if $B \Rightarrow \Omega$

(i) *1.* $\Gamma \vdash_{\mathrm{TS}} f\colon \Delta$ *implies*

Context	Code	Type	Stack	Condition
$\llbracket\Gamma\rrbracket_m$	$\llbracket f\rrbracket_m$	D	S	–
		$\Downarrow^*$		
$\llbracket\Delta\rrbracket_m$	C	D'	S	–

2. $\Gamma \vdash_{\mathrm{TS}} A$ *implies*

Context	Code	Type	Stack	Condition
$\llbracket\Gamma\rrbracket_m$	$\llbracket A\rrbracket_m\ C$	D	S	–
		$\Downarrow^*$		
Δ'	C	$\llbracket A\rrbracket_m$	S	–

3. $\Gamma \vdash_{\mathrm{TS}} t\colon 1{\to}A$ *implies*

Context	Code	Type	Stack	Condition
$\llbracket\Gamma\rrbracket_m$	$\llbracket t\rrbracket_m\ C$	1	S	–
		$\Downarrow^*$		
Δ'	C	$\llbracket A\rrbracket_m$	S	–

(ii) *1. Whenever*

Context	Code	Type	Stack	Condition
$\llbracket\Gamma\rrbracket_m$	$\llbracket f\rrbracket_m\ C$	D	S	–
		$\Downarrow^*$		
Δ'	C	D'	S	–

then there exists a well-formed context Δ *such that* $\Gamma \vdash_{\mathrm{TS}} f\colon \Delta$ *and* $\llbracket\Delta\rrbracket_m = \Delta'$.

2. Whenever

Context	Code	Type	Stack	Condition
$\llbracket\Gamma\rrbracket_m$	$\llbracket A\rrbracket_m\ C$	D	S	–
		$\Downarrow^*$		
Γ'	C	A'	S	–

then $\Gamma \vdash_{\mathrm{TS}} A$ *and* $\llbracket A\rrbracket_m = A'$.

3. Whenever

Context	Code	Type	Stack	Condition
$\llbracket\Gamma\rrbracket_m$	$\llbracket t\rrbracket_m\ C$	1	S	–
		$\Downarrow^*$		
Γ'	C	A'	S	–

then there exists a type A *such that* $\Gamma \vdash_{\mathrm{TS}} t\colon 1{\to}A$ *and* $\llbracket A\rrbracket_m = A'$.

Proof We consider only the case $\langle f, t[A]\rangle$, all other cases are similar.

(i) The transition sequence is

Context	Code	Type	Stack
$\llbracket\Gamma\rrbracket_m$	$\langle\llbracket f\rrbracket_m, \llbracket t\rrbracket_m\rangle?(\llbracket A\rrbracket_m)\ C$	D	S
	$\Downarrow$		
$\llbracket\Gamma\rrbracket_m$	$\llbracket f\rrbracket_m, \llbracket t\rrbracket_m?(\llbracket A\rrbracket_m)\ C$	D	$S, \llbracket f\rrbracket_m, \llbracket\Gamma\rrbracket_m$
	$\Downarrow^*$		
$\llbracket\Delta\rrbracket_m$	$, \llbracket t\rrbracket_m?(\llbracket A\rrbracket_m)\ C$	D'	$S, \llbracket f\rrbracket_m, \llbracket\Gamma\rrbracket_m$
	$\Downarrow$		
$\llbracket\Gamma\rrbracket_m$	$\llbracket t\rrbracket_m?(\llbracket A\rrbracket_m)\ C$	1	$S, \llbracket f\rrbracket_m, \llbracket\Delta\rrbracket_m$
	$\Downarrow^*$		
Γ'	$?(\llbracket A\rrbracket_m)\ C$	$\llbracket A'\rrbracket_m$	$S, \llbracket f\rrbracket_m, \llbracket\Delta\rrbracket_m$
	$\Downarrow$		
$\llbracket\Delta\rrbracket_m$	$\llbracket A\rrbracket_m?_{\mathbf{c}}\ C$	$\llbracket A'\rrbracket_m$	$S, \llbracket f\rrbracket_m, \llbracket\Delta\rrbracket_m, \llbracket A\rrbracket_m$
	$\Downarrow^*$		
Δ'	$?_{\mathbf{c}}\ C$	$\llbracket A'\rrbracket_m$	$S, \llbracket f\rrbracket_m, \llbracket\Delta\rrbracket_m, \llbracket A\rrbracket_m$
	$\Downarrow$		
$\llbracket\Delta\rrbracket_m \cdot \llbracket A\rrbracket_m$	C	$\llbracket A'\rrbracket_m$	S

By assumption the condition $\llbracket f\rrbracket_m * \llbracket A\rrbracket_m \leftrightarrow^* \llbracket A'\rrbracket_m$ is satisfied. The other direction is similar.

□

There are two important aspects of this algorithm for type synthesis. Firstly, it demonstrates the efficiency gains obtained by postponing substitution. Take as an example the combinator Snd corresponding to the variable x_0. It has the type $\mathsf{Fst} * A$, and so if the weakening was not postponed, it would occur at every access to a variable. The machine avoids such unwanted weakening reductions by just keeping the weakening combinator Fst as part of the context and letting the rule for variable access $\mathsf{Fst}^k; \langle f, t\rangle * \mathsf{Snd} \Rightarrow \mathsf{Fst}^k * t$ push the weakening inside subterms. Secondly, the inference rules for $\Gamma \vdash_{\mathrm{TS}} f\colon \Delta$ explain why in general the type A is included in the context morphism $\langle f, t[A]\rangle$. The reason is that it is in general impossible to derive the type A from a type B convertible to $f * A$. In the special case $f = \mathsf{Id}$, obviously $A = f * A$, and so there is no need to indicate the type A.

5.2 Local Variables

Local definitions, i.e. bindings of terms t_i to identifiers x_i require extra considerations. The first approach would be to represent a term t in this context just by $d * \llbracket t\rrbracket$, where $d = \langle\langle\rangle, \llbracket t_n\rrbracket, \ldots, \llbracket t_0\rrbracket\rangle$. This approach has one drawback, namely $d * \llbracket t\rrbracket$ is not necessarily a well-formed combinator. The reason is that only the insertion of a definition $x = s$ in t may make t well-formed, e.g. the term $\lambda\alpha\colon \mathsf{Prop}.\lambda y\colon \mathsf{Proof}(x).y\alpha\alpha$ is well-formed in the context of the binding $x = \forall\alpha\colon\mathsf{Prop}.\forall\beta\colon\mathsf{Prop}.\beta$ but not in the context of the binding $x = \forall\alpha{:}\,\mathsf{Prop}.\alpha$. Moreover, the typing rules for context morphisms cannot be applied to d. To see why, consider a combinator $d * \mathsf{Cur}(A,t)$ and assume that only $d * A$ but not A is well-formed. Then the reduction $d{*}\mathsf{Cur}(A,t) \leadsto_e \mathsf{Cur}(d{*}A, \langle \mathsf{Fst}; d, \mathsf{Snd}[A]\rangle * t)$ leads to an ill-formed combinator. This problem occurs also in the calculus of explicit substitutions [ACCL90] and leads the authors to abandon the notion of a type of a substitution, which corresponds to a context morphisms in this thesis. Because the typing of the context morphisms captures precisely what is necessary for the type checking of a dependent type and moreover can be justified categorically, it is incorporated into the type checking algorithm and the handling of bindings is added on top of that.

Therefore we add to the equational theory $\leadsto_e$ a new raw context morphism $\langle f, t\rangle$ together with the judgement

$$\Gamma \rhd_e f{:}\,(A_n, \ldots, A_0)$$

where $(A_n, \ldots, A_0)$ is a list of types. This theory is called the theory of *let-combinators*, and the derivation relation is written as $\rhd_l$. The intuition is that f is a list of terms such that $\Gamma \rhd f; \mathsf{Fst}^k * \mathsf{Snd}{:}\, 1 {\to} A_k$. The additional inference rules state when a combinator $\langle f, t\rangle * e$ is well-formed. They are the rules for $\langle f, t[A]\rangle * e$ with the combinator A removed and the precondition $\Gamma \rhd_e \langle f, t[A]\rangle{:}\, \Delta \cdot A$ replaced by $\Gamma \rhd_l \langle f, t\rangle{:}\,(A_n, \ldots, A_0)$. Furthermore these rules implicitly push substitution inside binding operations, e.g.

$$\frac{\Gamma \rhd_l f{:}\,(A_n, \ldots, A_0) \qquad \Gamma \rhd_l f * A \qquad \Gamma \cdot f * A \rhd_l \langle \mathsf{Fst}; f, \mathsf{Snd}\rangle * t{:}\, 1 {\to} B}{\Gamma \rhd_l f * \mathsf{Cur}(A,t){:}\, 1 {\to} \Pi(f * A, B)}$$

All additional rules for well-formed judgements are given in Table 5.3. The equations-in-context involving $\langle f, t\rangle$ are those of the theory $\rhd_e$ with $\langle f, t[A]\rangle$ replaced by $\langle f, t\rangle$. The function $\mathsf{sub}()$, which intuitively performs the substitution of a combinator $\Gamma \rhd_l \langle t_m, \ldots, t_0\rangle){:}\,(A_m, \ldots, A_0)$, is the key to formalize the relation between $\rhd_e$ and $\rhd_l$.

Definition 5.4 *Let f be a context morphism $\mathsf{Fst}^k; \langle t_m, \ldots, t_0\rangle$. The function $(f, e) \mapsto \mathsf{sub}(f, e)$, where e is any let-combinator, is defined as follows:*

1. *On context morphisms*

$$\begin{aligned}
\mathsf{sub}(f, \langle\rangle) &= \langle\rangle \\
\mathsf{sub}(f, \mathsf{Id}) &= f \\
\mathsf{sub}(f, \langle g, t\rangle) &= \langle \mathsf{sub}(f, g), \mathsf{sub}(f, t)\rangle \\
\mathsf{sub}(\mathsf{Fst}^k; \langle f, t\rangle, \mathsf{Fst}) &= \mathsf{Fst}^k; \langle t_m, \ldots, t_1\rangle \\
\mathsf{sub}(f, g; h) &= \mathsf{sub}(\mathsf{sub}(f, g), h) \\
\mathsf{sub}(f, \langle g, t[A]\rangle) &= \langle \mathsf{sub}(f, g), \mathsf{sub}(f, t)[A]\rangle
\end{aligned}$$

2. *On types*

$$\mathsf{sub}(f, \Omega) = \Omega$$

$$\begin{aligned}
\mathtt{sub}(f,T) &= \langle\langle\rangle, \mathtt{sub}(f,\mathsf{Snd})[\Omega]\rangle * T \\
\mathtt{sub}(f,\Pi(A,B)) &= \Pi(\mathtt{sub}(f,A), \mathtt{sub}(\langle \mathsf{Fst}; f, \mathsf{Snd}\rangle, B)) \\
\mathtt{sub}(f,g*A) &= \mathtt{sub}(\mathtt{sub}(f,g),A) \\
\mathtt{sub}(f,\Sigma(A,B)) &= \Pi(\mathtt{sub}(f,A), \mathtt{sub}(\langle \mathsf{Fst}; f, \mathsf{Snd}\rangle, B))
\end{aligned}$$

3. On morphisms

$$\begin{aligned}
\mathtt{sub}(f,\mathsf{Id}) &= \mathsf{Id} \\
\mathtt{sub}(f,t;s) &= \mathtt{sub}(f,t); \mathtt{sub}(f;s) \\
\mathtt{sub}(f,g*t) &= \mathtt{sub}(\mathtt{sub}(f,g),t) \\
\mathtt{sub}(\mathsf{Fst}^k; \langle f,t\rangle, \mathsf{Snd}) &= \mathsf{Fst}^k * t \\
\mathtt{sub}(f,\mathsf{Cur}(A,t)) &= \mathsf{Cur}(\mathtt{sub}(f,A), \mathtt{sub}(\langle \mathsf{Fst}; f, \mathsf{Snd}\rangle, t)) \\
\mathtt{sub}(f,\forall(A,t)) &= \forall(\mathtt{sub}(f,A), \mathtt{sub}(\langle \mathsf{Fst}; f, \mathsf{Snd}\rangle, t)) \\
\mathtt{sub}(f,\mathsf{App}(A,B)) &= \langle \mathsf{Id}, \mathtt{sub}(f,\mathsf{Snd})\rangle * \mathsf{App}(\mathtt{sub}(f,\mathsf{Fst}*A), \\
&\quad\ \mathtt{sub}(\langle \mathsf{Fst}; f; \mathsf{Fst}, \mathsf{Snd}\rangle, B)) \\
\mathtt{sub}(f,\mathsf{Pair}(A,B)) &= \langle \mathsf{Id}, \mathtt{sub}(f,\mathsf{Fst}*\mathsf{Snd})[\mathtt{sub}(f,A)], \mathtt{sub}(f,\mathsf{Snd})[\mathtt{sub}(\\
&\quad\ f\cdot\mathsf{Id},B)]\rangle * \mathsf{Pair}(\mathtt{sub}(f,A), \mathtt{sub}(f\cdot\mathsf{Id},B)) \\
\mathtt{sub}(f,\pi_i) &= \pi_i
\end{aligned}$$

An induction over the derivation of $\Gamma \rhd_l e\colon A$ shows

Theorem 5.5 *For any context morphism fsuch that* $\Gamma \rhd_l f\colon (A_m,\ldots,A_0)$ *and* $\Gamma \rhd_e \mathtt{sub}(f,\mathsf{Fst}^k * \mathsf{Snd})\colon 1 \longrightarrow A_k$ *and for any context morphism g, type A and morphism t, we have*

(i) $\Gamma \rhd_l f;g\colon \Delta$ *implies* $\Gamma \rhd_e \mathtt{sub}(f,g)\colon \Delta'$ *for some* $\Delta' \leftrightarrow_e^* \Delta$.

(ii) $\Gamma \rhd_l f*A$ *implies* $\Gamma \rhd_e \mathtt{sub}(f,A)$

(iii) $\Gamma \rhd_l f*t\colon A \rightarrow B$ *implies* $\Gamma \rhd_e \mathtt{sub}(f,t)\colon A' \rightarrow B'$ *with* $A \leftrightarrow_e^* A'$ *and* $B \leftrightarrow_e^* B'$.

Before the reduction machines and the type checking algorithm can be extended to the let-combinators, we have to show that they too satisfy confluence and strong normalization, which were established in Chapter 3 for the explicit combinators. We only sketch this here because otherwise large parts of Chapter 3 and of the appendix would have to be repeated. The crucial observation for the normalization proof is that the distinction between a morphism $\Gamma \rhd_l h\colon \Delta$ and a context morphism $\Gamma \rhd_l f\colon (A_m,\ldots,A_0)$ is maintained: the former are used for candidate assignments in connection with the application, and the latter has to satisfy only that $f; \mathsf{Fst}^k * \mathsf{Snd}$ is always an element of a suitable reducibility candidate. The correspondence theorems 3.4 and 3.5 between the Calculus of Constructions and the combinators remain valid

We assume throughout that $\Gamma \rhd_l f\colon (A_m,\ldots,A_0)$ except where other hypotheses for f are made.

Context Morphisms

$$\frac{\Gamma \rhd_l t\colon 1 \to A}{\Gamma \rhd_l \langle f,t\rangle\colon (A_m,\ldots,A_0,A)} \qquad \frac{}{\Gamma \rhd_l f;\langle\rangle\colon [\,]} \qquad \frac{}{\Gamma \rhd_l f;\mathsf{Id}\colon (A_m,\ldots,A_0)}$$

$$\frac{\Gamma \rhd_l f\colon [\,]\cdot A_m \cdots A_0 \qquad \Gamma \rhd_l t\colon 1\to A}{\Gamma \rhd_l \langle f,t\rangle\colon (f;\mathsf{Fst}^{m+1} * A_m,\ldots,f;\mathsf{Fst} * A_0, A)}$$

$$\frac{\Gamma \rhd_l f;g\colon \Delta \qquad \Delta \rhd_l A \qquad \Gamma \rhd_l f * t\colon 1\to f;g * A}{\Gamma \rhd_l f;\langle g,t[A]\rangle\colon \Delta\cdot A}$$

$$\frac{\Gamma \rhd_l f;g\colon (A_m,\ldots,A_0) \qquad \Gamma \rhd_l f * t\colon 1\to A}{\Gamma \rhd_l f;\langle g,t\rangle\colon (A_m,\ldots,A_0,A)} \qquad \frac{}{\Gamma \rhd_l f;\mathsf{Fst}\colon (A_m,\ldots,A_1)}$$

$$\frac{\Gamma \rhd_l f\colon \Delta \qquad \Delta \rhd_l g\colon (A_m,\ldots,A_0)}{\Gamma \rhd_l f;g\colon (A_m,\ldots,A_0)} \qquad \frac{\Gamma \rhd_l (f;g);h\colon (B_m,\ldots,B_0)}{\Gamma \rhd_l f;(g;h)\colon (B_m,\ldots,B_0)}$$

Types

$$\frac{\Gamma \rhd_l f * A \qquad \Gamma\cdot f * A \rhd_l \langle \mathsf{Fst}; f, \mathsf{Snd}\rangle * B}{\Gamma \rhd_l f * \Pi(A,B)} \qquad \frac{}{\Gamma \rhd_l f * \Omega} \qquad \frac{\Gamma \rhd_l f\colon (\Omega)}{\Gamma \rhd_l f * T}$$

$$\frac{\Gamma \rhd_l f * A \qquad \Gamma\cdot f * A \rhd_l \langle \mathsf{Fst}; f, \mathsf{Snd}\rangle * B}{\Gamma \rhd_l f * \Sigma(A,B)} \qquad \frac{\Gamma \rhd_l (f;g) * A}{\Gamma \rhd_l f * (g * A)}$$

Morphisms

$$\frac{\Gamma \rhd_l f * A}{\Gamma \rhd_l f * \mathsf{Id}\colon f * A \to f * A} \qquad \frac{}{\Gamma \rhd_l f * \mathsf{Snd}\colon 1\to A_0}$$

$$\frac{\Gamma \rhd_l f * A \qquad \Gamma\cdot f * A \rhd_l \langle \mathsf{Fst}; f, \mathsf{Snd}\rangle * t\colon 1\to B}{\Gamma \rhd_l f * \mathsf{Cur}(A,t)\colon 1\to \Pi(f * A, B)}$$

$$\frac{\Gamma \rhd_l f * A \qquad \Gamma\cdot f * A \rhd_l \langle \mathsf{Fst}; f, \mathsf{Snd}\rangle * t\colon 1\to \Omega}{\Gamma \rhd_l f * \forall(A,t)\colon 1\to\Omega}$$

$$\frac{\Gamma \rhd_l f * \mathsf{Snd}\colon 1\to f;\mathsf{Fst} * A \qquad \Gamma\cdot f;\mathsf{Fst} * A \rhd_l \langle \mathsf{Fst}; f;\mathsf{Fst}, \mathsf{Snd}\rangle * B}{\Gamma \rhd_l f * \mathsf{App}(A,B)\colon 1\to f;\mathsf{Fst} * \Pi(A,B)}$$

$$\frac{\Gamma \rhd_l f * A \qquad \Gamma\cdot f * A \rhd_l \langle \mathsf{Fst}; f, \mathsf{Snd}\rangle * B}{\Gamma \rhd_l f * \mathsf{Pair}(A,B)\colon 1\to f * \Sigma(A,B)}$$

$$\frac{\Gamma \rhd_l f * \Sigma(A,B)}{\Gamma \rhd_l \pi_1\colon f * \Sigma(A,B)\to f * A} \qquad \frac{\Gamma \rhd_l f * t\colon 1\to\Sigma(A,B)}{\Gamma \rhd_l f * (t;\pi_2)\colon 1\to\langle \mathsf{Id}, t;\pi_1\rangle * B}$$

$$\frac{\Gamma \rhd_l f * t; f * s\colon A\to B}{\Gamma \rhd_l f * (t;s)\colon A\to B} \qquad \frac{\Gamma \rhd_l f;g * t\colon A\to B}{\Gamma \rhd_l f * (g * t)\colon A\to B}$$

Table 5.3: Additional Inference Rules for Let-Combinators

for the let-combinators. Therefore Lemma 3.6, which states that the reductions $\leadsto_e$ and $\overset{N}{\leadsto}$ have the same normal forms, and Theorem 3.7 about the confluence hold as well for the reduction $\leadsto_l$. Because the translation from explicit to implicit combinators in Chapter 3 depends only on the confluence of the explicit calculus, the addition of the combinator $\langle f, t\rangle$ to the implicit combinators yields implicit combinators for $\rhd_l$. The combinator $\langle \mathsf{Id}, t\rangle$ in $\langle \mathsf{Id}, t\rangle * \mathsf{App}$, already introduced in Chapter 3, becomes then a special case of the let-combinator $\langle f, t\rangle$.

This makes it possible to generalize reduction machines at once to the $\rhd_l$-combinators: add for every transition with $\langle f, t[A]\rangle$ one involving only $\langle f, t\rangle$ (except the transition for Pair). The additional rules concerning the combinator $\langle f, t\rangle$ give rise in the same way as those for $\rhd_i$ to clauses for the type synthesis algorithm; they are listed in Table 5.4 except the clauses that arise when $\rhd_l$ is replaced by $\vdash_{\mathrm{TS}}$. The extension of the map rm to the combinator $\langle f, t\rangle$ is the trivial one, i.e. $\mathtt{rm}(\langle f, t\rangle) = \langle \mathtt{rm}(f), \mathtt{rm}(t)\rangle$. The theorems 5.1 and 5.2 are adapted as follows:

Theorem 5.6 *For every context* Γ *and implicit let-combinator* f*,* A *and* t*:*

(i)	$\Gamma \vdash_{\mathrm{TS}} f\colon (A_m, \ldots, A_0)$	implies	$\mathtt{rm}(\Gamma) \rhd_l \mathtt{rm}(f)\colon (\mathtt{rm}(A_m), \ldots, \mathtt{rm}(A_0))$
(ii)	$\Gamma \vdash_{\mathrm{TS}} f\colon \Delta$	implies	$\mathtt{rm}(\Gamma) \rhd_l \mathtt{rm}(f)\colon \mathtt{rm}(\Delta)$
(iii)	$\Gamma \vdash_{\mathrm{TS}} A$	implies	$\mathtt{rm}(\Gamma) \rhd_l \mathtt{rm}(A)$
(iv)	$\Gamma \vdash_{\mathrm{TS}} t\colon 1 \to A$	implies	$\mathtt{rm}(\Gamma) \rhd_l \mathtt{rm}(t)\colon 1 \to \mathtt{rm}(A)$

Theorem 5.7 *For every implicit let-combinator* f*,* A*,* t *and context* Γ*:*

(i) $\Gamma \rhd_l f\colon (A_m, \ldots, A_0)$ *implies for any well-formed context* $\Gamma' \leftrightarrow_l^* \Gamma$ *the existence of types* $A'_m, \ldots, A'_0$ *such that* $\Gamma \vdash_{\mathrm{TS}} f\colon (A'_m, \ldots, A'_0)$ *and* $A_i \leftrightarrow_l^* A'_i$.

(ii) $\Gamma \rhd_l f\colon \Delta$ *implies for any well-formed context* $\Gamma' \leftrightarrow_l^* \Gamma$ *the existence of a context* $\Delta' \leftrightarrow_l^* \Delta$ *such that* $\Gamma' \vdash_{\mathrm{TS}} f\colon \Delta'$

(iii) $\Gamma \rhd_l A$ *implies that any well-formed context* $\Gamma' \leftrightarrow_l^* \Gamma$ *satisfies* $\Gamma' \vdash_{\mathrm{TS}} A$.

(iv) $\Gamma \rhd_l t\colon 1 \to A$ *implies that for any well-formed context* $\Gamma' \leftrightarrow_l^* \Gamma$ *there exists a type* $A' \leftrightarrow_l^* A$ *such that* $\Gamma' \vdash_{\mathrm{TS}} t\colon 1 \to A'$.

Finally we extend the machine for type synthesis to let-combinators. The inference rule for $f * \mathsf{Cur}(A, t)$

$$\frac{\Gamma \rhd_l f\colon (A_m, \ldots, A_0) \qquad \Gamma \rhd_l f * A \qquad \Gamma \cdot f * A \rhd_l \langle \mathsf{Fst}; f, \mathsf{Snd}\rangle * t\colon 1 \to B}{\Gamma \rhd_l f * \mathsf{Cur}(A, t)\colon 1 \to \Pi(f * A, B)}$$

shows that the machine cannot process the combinator $f * \mathsf{Cur}(A, t)$ in two steps, namely first checking f and obtaining $(A_m, \ldots, A_0)$ and then examining $\mathsf{Cur}(A, t)$. It needs the combinator f for the extension from $(A_m, \ldots, A_0)$ to $(A_m, \ldots, A_0, f * A)$ when it starts testing t. Therefore the context register contains not only contexts like $\Gamma \cdot B_m \cdots B_0$ but also tuples $f :: (A_m, \ldots, A_0)$, where f is a context morphism and $A_n, \ldots, A_0$ are types. With this change in mind, we can easily derive the additional transitions corresponding to the inference rules for let-combinators; see Table 5.5.

We assume throughout that $\Gamma \vdash_{\mathrm{TS}} f\colon (A_m, \ldots, A_0)$

Context Morphisms

$$\frac{\Gamma \vdash_{\mathrm{TS}} f\colon \Delta \qquad \Delta \vdash_{\mathrm{TS}} A \qquad \Gamma \vdash_{\mathrm{TS}} t\colon 1 \to A' \qquad f * A \leftrightarrow_l^* A'}{\Gamma \vdash_{\mathrm{TS}} \langle f, t[A]\rangle\colon \Delta \cdot A}$$

Types

$$\frac{\Gamma \vdash_{\mathrm{TS}} f\colon (C) \qquad C \Rightarrow \Omega}{\Gamma \vdash_{\mathrm{TS}} f * T}$$

Morphisms

$$\frac{\Gamma \vdash_{\mathrm{TS}} f * A \qquad \Gamma \cdot f * A \vdash_{\mathrm{TS}} \langle \mathsf{Fst}; f, \mathsf{Snd}\rangle * t\colon 1 \to C \qquad C \Rightarrow \Omega}{\Gamma \vdash_{\mathrm{TS}} f * \forall(A, t)\colon 1 \to \Omega}$$

$$\frac{\Gamma \vdash_{\mathrm{TS}} f * t\colon 1 \to C \quad C \Rightarrow g * \Pi(A_1, B) \quad \Gamma \vdash_{\mathrm{TS}} f; h * \mathsf{Snd}\colon 1 \to A_2 \quad g * A_1 \leftrightarrow_l^* A_2}{\Gamma \vdash_{\mathrm{TS}} f * (t; h * \mathsf{App})\colon 1 \to \langle g, f; h * \mathsf{Snd}\rangle * B}$$ [4]

$$\frac{\Gamma \vdash_{\mathrm{TS}} f * t\colon 1 \to C \quad C \Rightarrow g * \Pi(A_1, B) \quad \Gamma \vdash_{\mathrm{TS}} f; h * \mathsf{Snd}\colon 1 \to A_2 \quad g * A_1 \leftrightarrow_l^* A_2}{\Gamma \vdash_{\mathrm{TS}} f * (t; h * \mathsf{App})\colon 1 \to \langle g, f; h * \mathsf{Snd}[A_1]\rangle * B}$$ [5]

$$\frac{\Gamma \vdash_{\mathrm{TS}} f * t\colon 1 \to C \qquad C \Rightarrow g * \Sigma(A, B)}{\Gamma \vdash_{\mathrm{TS}} f * (t; \pi_1)\colon 1 \to g * A}$$

$$\frac{\Gamma \vdash_{\mathrm{TS}} f * t\colon 1 \to C \qquad C \Rightarrow g * \Sigma(A, B)}{\Gamma \vdash_{\mathrm{TS}} f * (t; \pi_2)\colon 1 \to \langle f; g, t; \pi_1\rangle * B}$$ [4]

$$\frac{\Gamma \vdash_{\mathrm{TS}} f * t\colon 1 \to C \qquad C \Rightarrow g * \Sigma(A, B)}{\Gamma \vdash_{\mathrm{TS}} f * (t; \pi_2)\colon 1 \to \langle f; g, t; \pi_1[A]\rangle * B}$$ [5]

If

- $\Gamma \vdash_{\mathrm{TS}} \langle \mathsf{Id}, f * t, f * s\rangle\colon (A_m, \ldots, A_0, A', B')$ and
- $\Gamma \vdash_{\mathrm{TS}} f; g * \Sigma(A, B)$ and
- $\Sigma(A', B') \leftrightarrow_l^* f; g * \Sigma(A, B)$,

then

$$\Gamma \vdash_{\mathrm{TS}} f * \langle g, t[A], s[B]\rangle * \mathsf{Pair}\colon 1 \to f; g * \Sigma(A, B)$$

Table 5.4: Additional Type Synthesis Rules for Let-Combinators

[4] $g \equiv \mathsf{Fst}^k; \langle s_m, \ldots, s_0\rangle$

[5] $g \not\equiv \mathsf{Fst}^k; \langle s_m, \ldots, s_0\rangle$

Let throughout this table G be an abbreviation for $(A_m, \dots, A_0)$, and X be either a context Γ or $f{::}G$. Furthermore let $\mathtt{List}([\,] \cdot \mathtt{B_n} \cdot \mathtt{B_0}) = (\mathsf{Fst}^{n+1} * B_n, \dots, \mathsf{Fst} * B_0)$.

	Context	Code	Type	Stack	Condition
$\langle\rangle$	$f{::}G$	$\langle\rangle\ C$	D	S	
	$[\,]$	C	D	S	–
Fst	$f{::}G$	$\mathsf{Fst}\ C$	D	S	
	$f; \mathsf{Fst}{::}(A_m, \dots, A_1)$	C	D	S	–
$\langle - \rangle$	Γ	$\langle g, t\rangle\ C$	D	S	
	Γ	$g, t\rangle\ C$	D	$S, \langle g, t\rangle, \Gamma$	
$\langle - \rangle$	$f{::}G$	$\langle g, t\rangle\ C$	D	S	
	$f{::}G$	$g, t\rangle\ C$	D	$S, f\langle g, t\rangle, f{::}G$	
$?$	X	$\rangle ?(A)$	B	S, X'	
	X'	$A\ C$	B	S, X', B	
$\rangle$	X	$\rangle\ C$	A	$S, g, f{::}G$	
	$g{::}(G, A)$	C	A	S	
$\rangle$	X	$\rangle\ C$	A	S, g, Γ	
	$g{::}(\mathtt{List}(\Gamma), A)$	C	A	S	
$?_{\mathbf{c}}$	X	$?_{\mathbf{c}}\ C$	A	$S, g, f{::}G, B$	
	$g{::}(G, f * A)$	C	A	S	$f * A \leftrightarrow_l^* B$
$?_{\mathbf{c}}$	X	$?_{\mathbf{c}}\ C$	A	S, g, Γ, B	
	$\Gamma \cdot A$	C	A	S	$g; \mathsf{Fst} * A \leftrightarrow_l^* B$
$*$	$f{::}G$	$g * A\ C$	D	S	
	$f{::}G$	$gA\ C$	D	$S,$	–
$*_C$	$f{::}G$	$*_C\ C$	A	S, g	
	$f{::}G$	C	$g * A$	S	–
Π	$f{::}G$	$\Pi(A, B)\ C$	D	S	
	$f{::}G$	$A\mathrm{exc}B\Pi_C\ C$	D	$S, f{::}G$	–
exc	$f'{::}G'$	$\mathsf{exc}\ C$	A	$S, f{::}G$	
	$\langle \mathsf{Fst}; f, \mathsf{Snd}\rangle{::}(G, f * A)$	C	1	S, A	–
Π_C	$f{::}G$	$\Pi_C\ C$	B	S, A	
	$f{::}G$	C	$\Pi(A, B)$	S	–
Σ	$f{::}G$	$\Sigma(A, B)\ C$	D	S	
	$f{::}G$	$A\mathrm{exc}B\Sigma_C\ C$	D	$S, f{::}G$	–
Σ_C	$f{::}G$	$\Sigma_C\ C$	B	S, A	
	$f{::}G$	C	$\Sigma(A, B)$	S	–
Ω	$f{::}G$	$\Omega\ C$	D	S	
	$f{::}G$	C	Ω	S	–
T	(A)	$T\ C$	D	S	
	(A)	C	T	S	$A \Rightarrow \Omega$

Table 5.5: Extensions for the Type Synthesis Machine

	Context	Code	Type	Stack
Snd	$f{::}G$	$\mathsf{Snd}\ C$	1	S
	$f{::}G$	C	A_0	S
Cur	$f{::}G$	$\mathsf{Cur}(A,t)\ C$	1	S
	$f{::}G$	$A\mathsf{exc}\,t\ \mathsf{Cur}_C\ C$	1	$S, f{::}G$
Cur_C	$f{::}G$	$\mathsf{Cur}_C\ C$	B	S, A
	$f{::}G$	C	$\Pi(A,B)$	S
App_1	$f{::}G$	$h * \mathsf{App}$	D	S
	$f{::}G$	$h * \mathsf{SndApp}\ C$	1	$S, f{::}G, D', h * \mathsf{Snd}$[6]
App_1	$f{::}G$	$\langle \_s\rangle * \mathsf{App}$	D	S
	$f{::}G$	$s\mathsf{App}\ C$	1	$S, f{::}G, D', s$[6]
App_2	$f{::}G$	$\mathsf{App}\ C$	A'	$S, f'{::}G', g * \Pi(A,B)$
	$f'{::}G'$	C	$\langle g,s\rangle?(A) * B$	S[7]
App_2	$f{::}G$	$\mathsf{App}\ C$	A'	$S, f'{::}G', g * \Pi(A,B)$
	$f{::}G$	C	$\langle g,s\rangle * B$	S[8]
$\forall$	$f{::}G$	$\forall(A,t)\ C$	1	S
	$f{::}G$	$A\mathsf{exc}\,t\ \forall_C\ C$	1	$S, f{::}G$
$\forall_C$	$f{::}G$	$\forall_C\ C$	B	S, A
	$f{::}G$	C	Ω	S[9]
Pair_1	X	$\langle\langle g,t\rangle?(A), s\rangle?(B) * \mathsf{Pair}$	1	S
	X	$\langle\langle g,t\rangle?(A)\rangle\mathsf{exp}?(B)\mathsf{Pair}_C$	1	S, X, g
Pair_2	$f{::}(G,A,B)$	$\mathsf{Pair}_C\ C$	1	S, X, g
	X	C	$\Sigma(A,B)$	S
exp	Γ	$\mathsf{exp}\ C$	A	S, X
	X	C	1	S, Γ
exp	$g{::}(G', h * A)$	$\mathsf{exp}\ C$	A	$S, f{::}G$
	$f{::}G$	C	1	$S, \langle \mathsf{Fst}; h, \mathsf{Snd}\rangle{::}(G', h * A)$
π_1	$f{::}G$	$\pi_1\ C$	$g * \Sigma(A,B)$	S
	$f{::}G$	C	$g * A$	S
π_1	$f{::}G$	$h * \pi_1\ C$	$g * \Sigma(A,B)$	S
	$f{::}G$	C	$g * A$	S
$t;\pi_2$	$f{::}G$	$t;\pi_2\ C$	1	S
	$f{::}G$	$t\pi_2\ C$	1	S, t
$t;\pi_2$	$f{::}G$	$t; h * \pi_2\ C$	1	S
	$f{::}G$	$t\pi_2\ C$	1	S, t
π_2	$f{::}G$	$\pi_2\ C$	$g * \Sigma(A,B)$	S, t
	$f{::}G$	C	$\langle g, t\pi_1\rangle * B$	S
$g * t$	$f{::}G$	$g * t$	1	S
	$f{::}G$	gt	1	S

Table 5.5 (continued): Extensions for the Type Synthesis Machine

[6] This rule applies only if $D \Rightarrow D' \equiv g * \Pi(A,B)$

[7] This rule applies only if $A' \leftrightarrow^* g * A$ and $g \not\equiv \mathsf{Fst}^k; \langle s_n, \ldots, s_0\rangle$

[8] This rule applies only if $A' \leftrightarrow^* g * A$

[9] This rule applies only if $B \Rightarrow \Omega$

Chapter 6

Implementation Issues

This chapter describes an implementation of the abstract machines and of the type-checker in ML and compares them with the implementations used in the theorem provers LEGO and Coq, also written in ML. Because the only potentially time-consuming part of the type checking algorithm is the reduction to weak head-normal form, we only examine the efficiency of the reduction machines. The description is rather brief because the main thrust of this work has been to establish the theoretical underpinning of the abstract machines. Much remains to be done at the level of practical implementations. If efficiency is really critical an implementation in a language that is more closely related to the machine architecture like C should be used.

As the translation of a combinator into machine code replaces only some nodes in the graph by a list of nodes and does not change the structure of the graph, the implementation uses an ML-datatype that directly corresponds to the combinators. The transitions corresponding to the list of nodes are executed whenever the constructor corresponding to the combinator is encountered during reduction. This is easier than the introduction of a separate type of machine instructions and translation functions from them into combinators and vice versa. Efficiency considerations suggest one important difference between the combinators and the ML-datatype, however. The combinators Fst^k and $\mathsf{Fst}^k * \mathsf{Snd}$ occur throughout the transition tables for the abstract machines, and hence we introduce special constructors fstn and sndn , which take an integer parameter. The datatype for combinators is actually a sum of four datatypes, one for contexts, context morphisms, types and morphisms. In this way the typechecking algorithm of ML detects any confusion of sorts. The signature for the combinators in the eager case is

```
signature COMBT =
   sig
      datatype cm =  ecm |                                  (* <> *)
               env of cm * morphism |                   (* <f, t> *)
               fstn of int |                             (* fst^k *)
               comp_cm of cm * cm                          (* f;g *)

      and ctype = emptyt |                                   (* 1 *)
```

```
        mult_t of cm * ctype |                             (* f* A *)
        produ of ctype * ctype |                       (* Pi (A, B) *)
        sum of ctype * ctype |                         (* Si (A, B) *)
        prop |                                            (* Omega *)
        proof                                                 (* T *)

    and morphism = id_m |                                    (* id *)
        mult_m of cm * morphism |                         (* f * t *)
        comp_m of morphism * morphism|                     (* t;s  *)
        sndn of int |                               (* fst^n * snd *)
        cur of ctype * morphism|                      (* Cur (A, t *)
        app |                                               (* app *)
        forall of ctype*morphism |                 (* forall (A, t *)
        check of morphism * ctype |                         (* [A] *)
        pair |                                             (* pair *)
        pi1 |                                               (* pi1 *)
        pi2 ;                                               (* pi2 *)

    datatype context = emptyc | pairc of context * ctype

    datatype comb = conv1 of context | conv2 of cm |
                    conv3 of ctype | conv4 of morphism

end;
```

The signature for the lazy case replaces the datatype `env of cm * morphism` by `env of cm ref * morphism ref` and adds a `freeze`-constructor. This change captures sharing: the access to an environment `env (f, t)` yields `s` if `t` is a reference to an ML-expression `freeze s` and otherwise assigns the value `freeze u` to `t` if the code that `t` references evaluates to `u`.

A state of the eager machine is represented by a tuple of type

`cm * comb * NF`

where `NF` is a product of `ctype * cmorphism`, used as a sum type. In the lazy case the type `cm ref` replaces the type `cm` so that an environment can be updated after the evaluation of one of its components. The pattern matching makes it easy to formulate the machine tables in ML: every transition is captured by an alternative in a case-statement.

The theorem prover Coq represents an expression of the Calculus of Constructions directly as a datatype and uses a call-by-name strategy for their reduction to weak head-normal form or normal form. The operations for variable bindings that capture the notion of contexts are not used for the β-reduction; instead for an application $(\lambda x\colon A.t)s$ the substitution of the argument s for x in t is done as part of the β-reduction. LEGO also has a normal-order strategy for the β-reduction but implements sharing of the argument s based on the exceptions of ML during the substitution of s for x in t.

	User	Garbage collection
Eager Machine	4.20s	0.07s
Lazy Machine	2.85s	4.77s
LEGO	1102.62s	1118.13s
Coq	25.65s	66.57s

Table 6.1: Execution Times for Reduction of `test` to Normal Form

The Church-numerals provide a good example for testing the efficiency of these machines because they enable the easy construction of small terms with large normal forms. Consider the following CC-terms, where $\alpha \to \alpha$ is an abbreviation for $\forall x\colon \mathsf{Proof}(\alpha).\alpha$:

$$\begin{aligned}
\texttt{nattype} &\stackrel{\text{def}}{=} \forall\alpha\colon \mathsf{Prop}.(\alpha \to \alpha) \to \alpha \to \alpha \\
\texttt{mult} &\stackrel{\text{def}}{=} \lambda p\colon \mathsf{Proof}(\texttt{nattype}).\lambda q\colon \mathsf{Proof}(\texttt{nattype}).\lambda\alpha\colon \mathsf{Prop}.\lambda f\colon \mathsf{Proof}(\alpha \to \alpha). \\
&\quad\ \lambda x\colon \mathsf{Proof}(\alpha).q\alpha(p\alpha f)x \\
\texttt{two} &\stackrel{\text{def}}{=} \lambda\alpha\colon \mathsf{Prop}.\lambda f\colon \mathsf{Proof}(\alpha \to \alpha).\lambda x\colon \mathsf{Proof}(\alpha).f(fx) \\
\texttt{one} &\stackrel{\text{def}}{=} \lambda\alpha\colon \mathsf{Prop}.\lambda f\colon \mathsf{Proof}(\alpha \to \alpha).\lambda x\colon \mathsf{Proof}(\alpha).fx \\
\texttt{twelve} &\stackrel{\text{def}}{=} \lambda\alpha\colon \mathsf{Prop}.\lambda f\colon \mathsf{Proof}(\alpha \to \alpha).\lambda x\colon \mathsf{Proof}(\alpha). \\
&\quad\ (f(f(f(f(f(f(f(f(f(f(f(fx)))))))))))) \\
\texttt{powern} &\stackrel{\text{def}}{=} \texttt{twelve}\,\texttt{nattype}\,(\texttt{mult}\ \texttt{two})\ \texttt{one}
\end{aligned}$$

Note that the normal form of `powern` is the Church-numeral for 4096, i.e.

$$\texttt{bignum} \stackrel{\text{def}}{=} \lambda\alpha\colon \mathsf{Prop}.\lambda f\colon \mathsf{Proof}(\alpha \to \alpha).\lambda x\colon \mathsf{Proof}(\alpha).f^{4096}x$$

The normalization command in LEGO and Coq applies only to propositions and their proofs. Hence for measuring the execution times we use a proof of the proposition $\texttt{truep} \stackrel{\text{def}}{=} \forall\alpha\colon \mathsf{Prop}.\alpha \to \alpha$, namely the term

$$\texttt{test} \stackrel{\text{def}}{=} \texttt{powern}\,\texttt{truep}(\lambda x\colon \mathsf{Proof}(\texttt{truep}).x)(\lambda\alpha\colon \mathsf{Prop}.\lambda p\colon \mathsf{Proof}(\alpha).p)$$

which has the normal form

$$\lambda\alpha\colon \mathsf{Prop}.\lambda x\colon \mathsf{Proof}(\alpha).x$$

The results are given in Table 6.1. They are quite encouraging and show that these machines are an efficient alternative to previous ones.

The postponement of weakening is not only necessary for the normalization proof but also improves the efficiency of the machines significantly. As an example, consider the combinator

$$\texttt{largecomb} \stackrel{\text{def}}{=} \langle\langle\rangle, [\![\texttt{test}]\!]\rangle * [\![\lambda x_{256}\colon \mathsf{Prop}.\cdots.\lambda x_1\colon \mathsf{Prop}.x_0]\!]$$

If weakening happens at the beginning of every reduction inside a binding operation in the lazy machine, the reduction of `largecomb` requires 256 weakening operations applied to the large combinator $[\![\texttt{test}]\!]$. The execution times for the lazy machine are as follows:

	User	Garbage collection
with weakening postponed	2.13s	4.97s
with weakening not postponed	21.17s	29.87s

Chapter 7

Conclusions

The thesis has described categorical abstract machines for the Calculus of Constructions. Ehrhard's split D-categories are the only categorical structure that is suitable for the derivation of an equational theory, i.e. of categorical combinators. All other approaches yield either conditional equations or no equations at all. Furthermore, these categories separate environments from terms, whereas cartesian closed categories (CCCs) identify these two concepts. So the relation between the categorical abstract machine and the category is better behaved than that between the CAM and CCCs.

These equations can be turned easily into reductions, but there is no proof of strong normalization for this notion of reduction as yet. However, an application of the reducibility method shows that the reduction relation that reduces a combinator first to its weak head-normal form and then to its normal form is strongly normalizing. We must also ensure that the reduction can be defined on raw combinators without violating important properties like subject reduction. This is non-trivial because of the dependent types and is shown via the confluence of the Calculus of Constructions.

We then defined an eager and a lazy reduction strategy for the reduction of combinators and derived abstract machines for these strategies. These machines are generalizations of Krivine's machine and the CAM. Because of the separation of environments and terms, the correctness proof of the machine is a simple induction over the definition of the reduction strategies. Type checking is a very important application of these reduction machines. It is an essential part of theorem provers based on the Calculus of Constructions because the propositions-as-type analogy implies it is equivalent to testing whether a given term is a proof of a proposition or not. This task can be specified entirely in terms of the combinators. We constructed an abstract machine for computing the type of a given combinator. The dependent types entail that this process involves reduction, and so the reduction machines become part of the type checker. The latter can be extended to handle let-constructions, which satisfy different typing rules than an application. Finally, preliminary tests with an implementation of these machines in ML suggest that their efficiency is as good or even better than that of the reduction machines in LEGO or Coq.

There are at least two directions for further theoretical research. First, the standard categorical semantics of linear logic via symmetric monoidal closed categories identifies contexts and tensor products in the same way as the categorical semantics for the typed λ-calculus does. As already mentioned in the introduction, multicategories provide a way of separating these two issues. Contexts are modelled as lists of objects, whereas tensor products are captured by a universal construction that transforms an object that is a list (A, B) into the tensor object $A \otimes B$. Hence this structure represents the term calculus for linear logic better than the symmetric monoidal closed categories, e.g. the latter category models the term of type $A \otimes B$ in the context $(x{:}\, A, y{:}\, B)$ by the identity morphism on $A \otimes B$, whereas the former category uses a morphism arising from the universal construction defining the object $A \otimes B$. It is therefore interesting to see if multicategories can be used as a basis for categorical abstract machines for linear logic.

The second area of further research is the normalization proof. A categorical understanding of the normalization proof might be based on a suitable application of the glueing construction similar to those in [CP90] [MR92]. Given two categories $\mathcal{C}$ and $\mathcal{D}$ and a functor $H{:}\,\mathcal{C} \rightarrow \mathcal{D}$, the glueing category $\mathcal{G}l(\mathcal{C}, H)$ has as objects pairs $(\Gamma, X \longrightarrow H(\Gamma))$, where Γ is an object of $\mathcal{C}$ and $X \longrightarrow H(\Gamma)$ is a morphism in $\mathcal{D}$, and as morphisms from $(\Gamma, X \longrightarrow H(\Gamma))$ to $(\Delta, Y \longrightarrow H(\Delta))$ those pairs (ϕ, f) such that ϕ is a morphism from Γ to Δ and f is a morphism from X to Y and the diagram

$$\begin{tikzcd}
X \arrow[r] \arrow[d, "f"'] & H(\Gamma) \arrow[d, "H(\phi)"] \\
Y \arrow[r] & H(\Delta)
\end{tikzcd}$$

commutes. The important point is that often mild properties of $\mathcal{D}$ and H are enough to show that $\mathcal{G}l(\mathcal{C}, H)$ has the same categorical structure as $\mathcal{C}$ [Laf88].

Reynolds and Ma show how logical relations for models of the simply typed λ-calculus can be understood in this framework. They take $\mathcal{C}$ to be a cartesian closed category, $\mathcal{D}$ to be **Set** and $H = \mathsf{Hom}(1, -)$, and consider as objects of $\mathcal{G}l(\mathcal{C}, H)$ the monomorphims $X \hookrightarrow \mathsf{Hom}(1, \Gamma)$, i.e. subsets of global sections. The conditions imposed on a relation to be logical are exactly the conditions that are needed to turn the glueing category into a cartesian closed category. An initiality argument proves that every logical relation is satisfied for every λ-expression. But these categories identify redex and contractum, so they are not adequate to capture the normalization proofs. In the same way the logical relations for models do not permit a formulation of the normalization proof either. Statman [Sta85] introduces an additional condition to handle logical relations for applicative structures that are not models. This so-called admissibility condition ensures that logical relations are closed under λ-abstraction. Mitchell [Mit86] [Mit90] improves this condition to apply logical relations to the normalization proof for the λ-calculus. A similar modification of the definition of a cartesian closed category yields the corresponding categorical structure. We drop the equation corresponding to β-and η-reduction and retain only the naturality equation for the adjunction characterizing the exponentials. The requirement that the glueing category is such a category yields directly a categorical version of the admissibility condition.

Even these so-called combinatorial categories are not suitable for formalizing the normalization proof for the categorical combinators for the simply typed λ-calculus. They still suffer from the same defect as the original glueing categories: they identify redex and contractum of the rules for substitution. If these identities are removed as well, one obtains a category that has the objects and morphisms of a cartesian closed category but not the equalities between them. It is on one hand possible to formalize in this category the final induction in the normalization proof that proves every combinator to be reducible. Moreover, the conditions for the glueing category to be a combinatorial category are those familiar from the above induction proof in the calculus. On the other hand the closure properties of the sets of reducible combinators that are crucial to show that these conditions are fulfilled have no counterpart in this categorical framework. So the categorical treatment of the proof of strong normalization is still in an unsatisfactory state.

Appendix A

Normalization Proof for the Combinators

This appendix gives the details of the proof of strong normalization for the reduction $\overset{N}{\rightsquigarrow}$ defined in chapter 3. The proof consists of four parts. First, we give a complexity measure on types. It is used in the second part to define reducibility candidates of type e, which are sets of strongly normalizing morphisms $t\colon 1\rightarrow e$ with suitable closure properties if e is a proposition or a type, which are functions if e is a morphism $t\colon 1\rightarrow\Pi(A,B)$, or which are pairs of reducibility candidates if e is a morphism $t\colon 1\rightarrow\Sigma(A,B)$. Next, we define an interpretation function $[\![-]\!]$ that maps every type to a reducibility candidate. This function is parametrized by a context morphism $\Delta \rhd_e h\colon\Gamma$ and an assignment of a reducibility candidate of type $h;\mathsf{Fst}^i * \mathsf{Snd}$. Finally, we show that the closure properties of the reducibility candidates imply that every combinator is an element of a reducibility candidate and is therefore strongly normalizing.

The following proof does not work directly for the reduction $\overset{N}{\rightsquigarrow}$. The last part succeeds only if we do not consider reduction on raw combinators but on raw combinators modulo the equations for associativity of composition in the base and in the fibres together with the equations

$$\begin{aligned} f;g*A &= f*(g*A)\\ f;g*t &= f*(g*t)\\ f*(t;s) &= f*t;f*s \end{aligned}$$

Only the last equation causes some complications during the proof, namely whenever a combinator $t;g*\mathsf{App}(A,B)$ is considered. But it is clear that the strong normalization of $\overset{N}{\rightsquigarrow}$ on raw combinators holds if it is satisfied for raw combinators modulo these equations.

The proof presented here is an adaptation of Coquand's and Gallier's proof of strong normalization of the Calculus of Constructions [CG90]. They replace infinite contexts, as used in other proofs [Luo90] [Coq85] by the notion of a candidate assignment ρ, which is a list of terms that corresponds in the terminology used here to a context morphism together with an assignment of reducibility candidates. It is therefore no surprise that their framework is well-suited for the combinators, for it contains the notion of context morphism as part of the definition. Furthermore the proof is more uniform in the sense that the compatibility of the candidates with substitution, which has to be stated as a separate lemma in the calculus, is incorporated into the definition of the interpretation. The proof of this lemma then becomes part of the proof that the interpretation is well-defined.

The main difference between Coquand and Gallier's proof and that presented here is that the latter does not rely on any properties of the combinators that require confluence for their proofs. The reason is that several important properties like subject reduction or unicity of typing are immediate consequences of the definitions if equality judgements and explicitly typed application are used (as here). Coquand and Gallier need confluence to establish these properties as they consider a Calculus of Constructions with reduction defined on raw terms.

There are essentially two versions of the reducibility proof for the polymorphic λ-calculus and the Calculus of Constructions. One, based on Tait [Tai75] defines so-called saturated sets. They satisfy all the closure properties needed for the final induction that shows that every term is an element of a reducibility candidate. Girard [Gir71] [GTL89], who introduces the other version, identifies a common principle behind these properties: they all follow from the condition that any so-called neutral term t, which is a term that is not a λ-abstraction, is an element of a reducibility candidate C if whenever t reduces to t', t' is an element of C. This idea is especially useful for the combinators. Because there are quite a few closure properties required during the final induction (cf. Lemma A.13), the definition of reducibility candidates becomes much shorter and technically simpler if the second version is used. We define a combinator to be neutral if it is not identical to any of the combinators $f; \langle g, t[A] \rangle$, $f * \mathsf{Cur}(A, t)$ or $f * \mathsf{Pair}(A, B)$. These are the morphisms with codomain $\Gamma \cdot A$, $\Pi(A, B)$ and $\Sigma(A, B)$ respectively that arise from the adjunctions introducing these objects.

A.1 A Complexity Measure on Types

We start with a syntactic distinction between small and large types. Large types are given by

$$L ::= \Omega \mid \Pi(A, L) \mid \Sigma(A, L) \mid \Sigma(L, A) \mid f * L$$

and small types by

$$S ::= 1 \mid T \mid \Pi(A, S) \mid \Sigma(S, S) \mid f * S$$

where A is an arbitrary type. It is easy to see that large types are convertible only to large types, and small types only to small ones. Furthermore if L_1 is a large type, $\Pi(A_1, L_1)$ and $\Pi(A_2, L_2)$ are convertible iff A_1 and A_2 as well as L_1 and L_2 are convertible. Therefore we can define a complexity measure c for types by

$$\begin{array}{rclrclrcl} c(S) & = & 0 & c(\Pi(A, L)) & = & max(c(A) + c(L)) + 1 & c(f * A) & = & c(A) \\ c(\Omega) & = & 1 & c(\Sigma(A, B)) & = & c(A) + c(B) + 1 & & & \end{array}$$

where S is any small type. This measure is invariant under convertibility.

Remark If we add an impredicative existential quantification $\Gamma \cdot A \rhd_e \exists(A,p)\colon 1 \to \Omega$ for every $\Gamma \cdot A \rhd_e p\colon 1 \to \Omega$ together with the coherence rule

$$\langle\langle\rangle, \exists(A,p)[\Omega]\rangle * T = \Sigma(A, \langle\langle\rangle, p[\Omega]\rangle * T)$$

the measure c is no longer invariant under conversion. Therefore, any attempt at proving strong normalization for this system in a way similar to that described below breaks down. In fact, this system is not even normalizing because it is inconsistent [Coq86].

A.2 The Reducibility Candidates

We will use the following notation. A morphism t such $\Gamma \rhd_e t\colon 1 \to L$, where L is a large type, is called a propositional family. Furthermore we will abbreviate "strongly normalizing" to SN. If Δ is a context, Δ' denotes a context such that $\Delta' \rhd_e \mathsf{Fst}^k\colon \Delta$ for some k. A neutral context morphism is any morphism not identical to $f; \langle g, t[A]\rangle$ or to $\langle g, t[A]\rangle$. A neutral morphism is any morphism not identical to $f * \mathsf{Cur}(A,t)$, $\mathsf{Cur}(A,t)$, $f * \mathsf{Pair}(A,B)$ or $\mathsf{Pair}(A,B)$.

Definition A.1 *The family $\mathcal{C}$ of sets of reducibility candidates $\mathcal{C}_{e,\Delta}$ with Δ a well-formed context and e a type or a family of propositions well-formed in context Δ, is given by*

(i) $\mathcal{C}_{\mathrm{type},\Delta} \overset{\mathrm{def}}{=} \{C_{\mathrm{type},\Delta}\}$ *with*

$$C_{\mathrm{type},\Delta} \overset{\mathrm{def}}{=} \left\{A \mid \Delta' \rhd_e A, \mathsf{Fst}^k * A \text{ is SN}\right\}$$

We call every element of $C_{type,\Delta}$ a canonical type.

(ii) *If e is a type A, $\mathcal{C}_{A,\Delta}$ is the set of sets C such that each C is a nonempty set of morphisms $\Delta' \rhd_e t\colon 1 \to \mathsf{Fst}^m * A$ with the following properties:*

($CR1$) *If t is an element of C, then $\mathsf{Fst}^k * t$ is SN.*

($CR2$) *If $\mathsf{Fst}^k * t \overset{W}{\rightsquigarrow} t'$, then t' is an element of C.*

($CR3$) *If t is neutral, $\mathsf{Fst}^k * t$ is SN and any reduction sequence of $\mathsf{Fst}^k * t$ via $\overset{W}{\rightsquigarrow}$ leads to one in C after finitely many neutral morphisms, then t is also in C.*

($CR4$) *If t is an element of C, then also $\mathsf{Fst}^k * t$ is an element of C.*

(iii) *If e is a proposition p, then $\mathcal{C}_{p,\Delta}$ is the set $\mathcal{C}_{\langle\langle\rangle,p[\Omega]\rangle * T,\Delta}$.*

(iv) *If e is a propositional family $t\colon 1 \to \Pi(A,L)$, $\mathcal{C}_{t,\Delta}$ is the set of functions with the following properties:*

1. *If A is a large type, then $f \in \mathcal{C}_{t,\Delta}$ is a function with domain*

$$\left\{(s, C, k) \mid \Delta' \rhd_e s\colon \mathsf{Fst}^k * A, C \in \mathcal{C}_{s,\Delta'}\right\}$$

such that

- $f(s, C, k) \in \mathcal{C}_{\mathsf{Fst}^k * t; \langle \mathsf{Fst}^k, s[A] \rangle * \mathsf{App}(A,L)}$ *and*
- $f(s_1, C_1, k) = f(s_2, C_2, k)$ *whenever* $s_1 \leftrightarrow^* s_2$, $C_1 = C_2$.

2. If A is a small type, then $f \in \mathcal{C}_{A,\Delta}$ is a function with domain

$$\left\{(s, k) \mid \Delta' \rhd_e s\colon \mathsf{Fst}^k * A\right\}$$

such that

- $f(s, k) \in \mathcal{C}_{\mathsf{Fst}^k * t; \langle \mathsf{Fst}^k, s[A] \rangle * \mathsf{App}(A,L)}$ *and*
- $f(s_1, k) = f(s_2, k)$ *whenever* $s_1 \leftrightarrow^* s_2$.

(v) *If e is a morphism t such that $\Delta \rhd_e t\colon 1 \to \Sigma(A, B)$, then $\mathcal{C}_{e,\Delta}$ is the set of pairs (C_1, C_0) such that $C_1 \in \mathcal{C}_{e;\pi_1,\Delta}$ if A is a large type and $C_0 \in \mathcal{C}_{e;\pi_2,\Delta}$ if B is a large type and C_1 and C_0 are arbitrary reducibility candidates otherwise.*

An induction over the complexity $c(D)$ of the type D of the morphism $t\colon 1 \to D$ shows that this definition is proper: in clause (iv), $c(\langle \mathsf{Fst}^k, s[A] \rangle * L) < c(\Pi(A, L))$ because L is a large type and in clause (v), $c(A) < c(\Sigma(A, B))$ as well as $c(\langle \mathsf{Id}, e; \pi_1[A] \rangle * B) < c(\Sigma(A, B))$ if $\Sigma(A, B)$ is a large type.

We need two properties of the reducibility candidates for later sections. The first states that every set $\mathcal{C}_{e,\Delta}$ is nonempty. This is shown via the construction of suitable canonical elements.

Theorem A.2 *All reducibility candidates $\mathcal{C}_{e,\Delta}$ are nonempty sets.*

Proof Define the canonical elements as follows:

(i) $can_{\mathbf{type},\Delta} \overset{\text{def}}{=} C_{\mathbf{type},\Delta}$

(ii) If e is a type A, then $can_{A,\Delta} \overset{\text{def}}{=} \left\{t \mid \Delta' \rhd_e t\colon 1 \to \mathsf{Fst}^k * A, \mathsf{Fst}^m * t \text{ is SN}\right\}$.

(iii) If e is a proposition t, then $can_{t,\Delta}$ is equal to $can_{\langle\langle\rangle, t[\Omega]\rangle * T, \Delta}$.

(iv) If e is a morphism $\Gamma \rhd_e t\colon 1 \to \Pi(A, L)$ with L a large type, then $can_{t,\Delta}$ is the set containing the function f that satisfies

$$f(s, C, k) = can_{\mathsf{Fst}^k * t, \langle \mathsf{Fst}^k, s[A] \rangle * \mathsf{App}(A,L)}$$

if A is a large type. If A is a small type, then $can_{t,\Delta}$ consists of the function f with

$$f(s, k) = can_{\mathsf{Fst}^k * t; \langle \mathsf{Fst}^k, s[A] \rangle * \mathsf{App}(A,L)}$$

(v) If e is a morphism $\Gamma \rhd_e t\colon 1 \to \Sigma(A, B)$, define $can_{t,\Delta}$ to be the pair (C_1, C_0), where $C_1 = can_{t;\pi_1,\Delta}$ if A is a large type and $C_0 = can_{t;\pi_2,\Delta}$ if B is a large type and $C_1 = C_0 = can_{\mathbf{type},\Delta}$ otherwise.

Because the morphism Snd with $\Delta \cdot A \rhd_e \mathsf{Snd}\colon 1 \to \mathsf{Fst} * A$ is an element of $can_{A,\Delta}$, these sets are nonempty. The verification that the canonical elements are indeed reducibility candidates is easy and is therefore omitted. □

Second, we need a way of constructing a reducibility candidate C' of type $\mathsf{Fst}^k * e$ in context Δ' from a candidate C of type e in context Δ. This is done as follows:

(i)

$$C' \overset{\text{def}}{=} C_{\text{type},\Delta} \cap \{A \mid \Delta'' \rhd_e A, \mathsf{Fst}^m\colon \Delta'' \to \Delta' \text{ for some } m\}$$

(ii) If e is a type A, we have

$$C' \overset{\text{def}}{=} C_{A,\Delta} \cap \left\{t \mid \Delta'' \rhd_e t\colon 1 \to \mathsf{Fst}^{k+m} * A, \mathsf{Fst}^m\colon \Delta'' \to \Delta' \text{ for some } m\right\}$$

(iii) If e is a proposition p, then C' is the restriction of C regarded as a reducibility candidate of type $\langle\langle\rangle, p[\Omega]\rangle * T$.

(iv) If e is a propositional family $\Delta \rhd_e t\colon 1 \to \Pi(A, L)$, then C' is the set of functions such that $f(s, C_1, m) = g(s, C_1, k+m)$ with $g \in C$ if A is a large type and the set of functions f such that

$$f(s, m) = g(s, k+m)$$

with $g \in C$ if A is a small type.

(v) If e is a morphism $\Gamma \rhd_e t\colon 1 \to \Sigma(A, B)$ and $C_{t,\Delta} = (C_1, C_0)$, the reducibility candidate $C'_{\mathsf{Fst}^k * t, \Delta'}$ is the pair (C'_1, C'_0).

It is easy to verify that C' is also a reducibility candidate.

A.3 The Interpretation

The next step consists in the definition of an interpretation $[\![-]\!]$ that maps every type and propositional family in context $\Gamma = [\,] \cdot A_n \cdots \cdot A_0$ to a reducibility candidate. The dependent types require the interpretation to be parametrized by an assignment of a morphism of type A_i for every i. The impredicative quantification adds another parameter, namely an assignment of a reducibility candidate for every large type A_i. These two assignments are combined into a so-called *candidate assignment* .

Definition A.3 *Let $h_{cn}\colon \Delta \to \Gamma = [\,] \cdot A_n \cdots A_0$ be a context morphism and D be a list of reducibility candidates $(D_i)_{i=0 \cdots n}$ with $D_i \in can_{h;\mathsf{Fst}^i * \mathsf{Snd},\Delta}$ for every i such that A_i is a large type.*

(i) *The candidate assignment $h\colon \Delta \to \Gamma$ is the pair $(h_{cm}, (D_n, \ldots, D_0))$.*

(ii) *The candidate assignment $h; \mathsf{Fst}$ is the pair $(h_{cm}; \mathsf{Fst}, (D_n, \ldots, D_1))$*

(iii) *If D'_i denotes the restriction of D_i to the context Δ', the candidate assignment $\mathsf{Fst}^k; h$ is the pair $(\mathsf{Fst}^k; h, (D'_n, \ldots, D'_0))$.*

(iv) *Two candidate assignments h and g are called convertible if the corresponding context morphisms are convertible and the corresponding reducibility candidates are equal.*

The interpretation is a priori a partial function because some clauses in the definition only make sense if it yields a candidate assignment for certain morphisms. We will show afterwards that this function is in fact total. In the following definition of the interpretation, S denotes a small and L a large type.

Definition A.4 *Let $\Delta \rhd_e h{:}\,\Gamma$ be a candidate assignment $h = (h_{cm}, (D_n, \ldots, D_0))$, which we sometimes abbreviate to $h = (h_{cm}, D)$. We define the interpretation $[\![\Gamma \rhd_e e]\!]h\Delta$, where e is a type or a propositional family well-formed in context Γ, by induction over the structure of e as follows:*

(i) *On contexts*

$$\begin{aligned}
[\![[\,]]\!]h\Delta &\overset{\text{def}}{=} \{f{:}\,\Delta\to[\,] \mid \forall m : \mathsf{Fst}^m; f \textit{ is SN}\}\\
[\![\Gamma \cdot A]\!]h\Delta &\overset{\text{def}}{=} \{f{:}\,\Delta\to\Gamma\cdot A \mid f;\mathsf{Fst} \in [\![\Gamma]\!]h;\mathsf{Fst}\Delta \textit{ and}\\
&\qquad f * \mathsf{Snd} \in [\![\Gamma \rhd_e A]\!]h;\mathsf{Fst}\Delta\}
\end{aligned}$$

(ii) *On context morphisms*

$$\begin{aligned}
[\![\Gamma \rhd_e \langle\rangle]\!]h\Delta &\overset{\text{def}}{=} h;\langle\rangle\\
[\![\Gamma \rhd_e \mathsf{Fst}]\!]h\Delta &\overset{\text{def}}{=} (h_{cm};\mathsf{Fst}, (D_n, \ldots, D_1))\\
[\![\Gamma \rhd_e \mathsf{Id}]\!]h\Delta &\overset{\text{def}}{=} h\\
[\![\Gamma \rhd_e \langle f, t[A]\rangle]\!]h\Delta &\overset{\text{def}}{=} (h;\langle f, t[A]\rangle, C_n, \ldots, C_0, [\![\Gamma \rhd_e t]\!]h\Delta)\\
&\qquad \text{with } [\![\Gamma \rhd_e f]\!]h\Delta = (h; f, C_n, \ldots, C_0)\\
[\![\Gamma \rhd_e f; g]\!]h\Delta &\overset{\text{def}}{=} [\![\Gamma' \rhd_e g]\!]([\![\Gamma \rhd_e f]\!]h\Delta)\Delta\\
&\qquad \text{if } \Gamma \rhd_e f{:}\,\Gamma'
\end{aligned}$$

(iii) *On types*

$$\begin{aligned}
[\![\Gamma \rhd_e 1]\!]h\Delta &\overset{\text{def}}{=} \mathit{can}_{1,\Delta}\\
[\![\Gamma \rhd_e \Omega]\!]h\Delta &\overset{\text{def}}{=} \mathit{can}_{\Omega,\Delta}
\end{aligned}$$

The set $[\![\Gamma \rhd_e \Pi(L, B)]\!]h\Delta$ is defined to be the set of all morphisms $\Gamma \rhd_e t{:}\,1\to h * \Pi(L, B)$ such that

- *for all k the morphism $\mathsf{Fst}^k * t$ is SN,*
- *for all m and g such that $g * \mathsf{Snd} \in [\![\Gamma \rhd_e L]\!]\mathsf{Fst}^m; h\Delta'$ and*
- *for all $C \in \mathcal{C}_{g*\mathsf{Snd},\Delta'}$ and*
- *for all L' such that $g;\mathsf{Fst} * L' \leftrightarrow^* \mathsf{Fst}^m; h * L$ and $g;\mathsf{Fst} * L'$ is canonical and*

- *for all* B' *such that* $\langle \mathsf{Fst}; g; \mathsf{Fst}, \mathsf{Snd}[L'] \rangle * B' \leftrightarrow^* \langle \mathsf{Fst}^{m+1}; h, \mathsf{Snd}[L] \rangle * B$, *and the type* $\langle \mathsf{Fst}^{k+1}; g; \mathsf{Fst}, \mathsf{Snd}[L'] \rangle * B'$ *is canonical for any* $k \geq 0$,

we have

$$\mathsf{Fst}^m * t; g * \mathsf{App}(L', B') \in \llbracket \Gamma \cdot L \rhd_e B \rrbracket(\langle \mathsf{Fst}^m; h, g * \mathsf{Snd}[L] \rangle (D', C))\Delta'$$

The set $\llbracket \Gamma \rhd_e \Pi(S, B) \rrbracket h\Delta$ *is the set of all morphisms* $\Gamma \rhd_e t \colon 1 \rightarrow \Pi(S, B)$ *such that*

- *for all* k *the morphism* $\mathsf{Fst}^k * t$ *is SN,*
- *for all* g *such that* $g * \mathsf{Snd} \in \llbracket \Gamma \rhd_e S \rrbracket \mathsf{Fst}^m; h\Delta'$ *and*
- *for all* S' *such that* $g; \mathsf{Fst} * S' \leftrightarrow^* \mathsf{Fst}^m; h * S$ *and* $g; \mathsf{Fst} * S'$ *is canonical and*
- *for all* B' *such that* $\langle \mathsf{Fst}; g; \mathsf{Fst}, \mathsf{Snd}[S'] \rangle * B' \leftrightarrow^* \langle \mathsf{Fst}^{m+1}; h, g * \mathsf{Snd}[S] \rangle * B$ *and the type* $\langle \mathsf{Fst}^{k+1}; g; \mathsf{Fst}, \mathsf{Snd}[S'] \rangle * B'$ *is canonical for any* $k \geq 0$,

we have

$$\mathsf{Fst}^m * t; g * \mathsf{App}(S', B') \in \llbracket \Gamma \cdot S \rhd_e B \rrbracket(\langle \mathsf{Fst}^m; h, g * \mathsf{Snd}[S] \rangle (D', can_{\Omega,\Delta'}))\Delta'$$

$$\begin{aligned}
\llbracket \Gamma \rhd_e \Sigma(A, B) \rrbracket h\Delta &\stackrel{\mathrm{def}}{=} \{t \colon 1 \rightarrow h * \Sigma(A, B) \mid t; \pi_1 \in \llbracket \Gamma \rhd_e A \rrbracket h\Delta, \forall C \in \mathcal{C}_{t;\pi_1,\Delta} : \\
&\qquad t; \pi_2 \in \llbracket \Gamma \cdot A \rhd_e B \rrbracket(\langle h, t; \pi_1[A] \rangle (D, C))\Delta\} \\
\llbracket \Gamma \rhd_e f * A \rrbracket h\Delta &\stackrel{\mathrm{def}}{=} \llbracket \Gamma' \rhd_e A \rrbracket(\llbracket \Gamma \rhd_e f \rrbracket h\Delta)\Delta \\
\llbracket T \rrbracket h\Delta &\stackrel{\mathrm{def}}{=} D_0
\end{aligned}$$

(iv) *On morphisms:*
For any t *which is not a propositional family we define* $\llbracket \Gamma \rhd_e t \rrbracket h\Delta \stackrel{\mathrm{def}}{=} can_{\Omega,\Delta}$. *Otherwise we have the following inductive cases:*

$$\begin{aligned}
\llbracket \Gamma \cdot A \rhd_e \mathsf{Snd} \rrbracket h\Delta &\stackrel{\mathrm{def}}{=} D_0 \\
\llbracket \Gamma \rhd_e \mathsf{Cur}(L, t) \rrbracket h\Delta &\stackrel{\mathrm{def}}{=} \lambda k.\lambda s.\lambda C.\llbracket \Gamma \cdot L \rhd_e t \rrbracket(\langle \mathsf{Fst}^k; h, s[L] \rangle, (D', C))\Delta' \\
&\qquad \text{where } \Delta' \rhd_e s \colon 1 \rightarrow \mathsf{Fst}^k; h * L, C \in \mathcal{C}_{s,\Delta'} \\
\llbracket \Gamma \rhd_e \mathsf{Cur}(S, t) \rrbracket h\Delta &\stackrel{\mathrm{def}}{=} \lambda k.\lambda s.\llbracket \Gamma \cdot S \rhd_e t \rrbracket(\langle \mathsf{Fst}^k; h, s[S] \rangle, (D', can_{\Omega,\Delta'}))\Delta' \\
&\qquad \text{where } \Delta' \rhd_e s \colon 1 \rightarrow \mathsf{Fst}^k; h * S \\
\llbracket \Gamma \rhd_e \forall(A, t) \rrbracket h\Delta &\stackrel{\mathrm{def}}{=} \llbracket \Gamma \rhd_e \Pi(A, \langle \langle \rangle, t[\Omega] \rangle * T) \rrbracket h\Delta \\
\llbracket \Gamma \rhd_e t; f * \mathsf{App}(L, B) \rrbracket h\Delta &\stackrel{\mathrm{def}}{=} (((\llbracket \Gamma \rhd_e t \rrbracket h\Delta)0)f * \mathsf{Snd})\llbracket \Gamma \rhd_e f * \mathsf{Snd} \rrbracket h\Delta \\
\llbracket \Gamma \rhd_e t; f * \mathsf{App}(S, B) \rrbracket h\Delta &\stackrel{\mathrm{def}}{=} (((\llbracket \Gamma \rhd_e t \rrbracket h\Delta)0)f * \mathsf{Snd}) \\
\llbracket \Gamma \cdot A \cdot B \rhd_e \mathsf{Pair}(A', B') \rrbracket h\Delta &\stackrel{\mathrm{def}}{=} (D_1, D_0) \\
\llbracket \Gamma \rhd_e t; f * \pi_1 \rrbracket h\Delta &\stackrel{\mathrm{def}}{=} D_1 \textit{ if } \llbracket \Gamma \rhd_e t \rrbracket h\Delta = (D_1, D_0) \\
\llbracket \Gamma \rhd_e t; f * \pi_2 \rrbracket h\Delta &\stackrel{\mathrm{def}}{=} D_0 \textit{ if } \llbracket \Gamma \rhd_e t \rrbracket h\Delta = (D_1, D_0) \\
\llbracket \Gamma \rhd_e f * t \rrbracket h\Delta &\stackrel{\mathrm{def}}{=} \llbracket \Gamma' \rhd_e t \rrbracket(\llbracket \Gamma \rhd_e f \rrbracket h\Delta)\Delta \textit{ if } f \colon \Gamma \rightarrow \Gamma'
\end{aligned}$$

The invariance of the interpretation under conversion of combinators is the crucial point in establishing that the interpretation is a total function.

Lemma A.5 *If the candidate assignments h and h' are convertible, then $[\![\Gamma \rhd_e e]\!]h\Delta$ and $[\![\Gamma \rhd_e e]\!]h'\Delta$ are convertible if e is a context morphism and equal if e is a type or a morphism.*

Proof Induction over the definition of the interpretation. □

Lemma A.6 *If Γ and Γ' are convertible, then $[\![\Gamma \rhd_e e]\!]h\Delta$ and $[\![\Gamma' \rhd_e e]\!]\Delta$ are convertible if e is a context morphism and equal if e is a type or a morphism.*

Proof Induction over the definition of the interpretation. □

Lemma A.7 *If e and e' are convertible, then $[\![\Gamma \rhd_e e]\!]h\Delta$ and $[\![\Gamma \rhd_e e']\!]h\Delta$ are convertible if e is a context morphism and equal if e is a type or a morphism.*

Proof Induction over the definition of $\Gamma \rhd_e e = e'$. □

Lemma A.8 *If Γ and Γ' are convertible, then $[\![\Gamma]\!]h\Delta$ and $[\![\Gamma']\!]h\Delta$ are equal.*

Proof Direct consequence of Lemma A.7. □

Now we can show the promised theorem.

Theorem A.9 *For every context morphism f, $[\![\Gamma \rhd_e f]\!]h\Delta$ is a candidate assignment, and for every object in the fibre or propositional family e, the set*

$$C = \bigcup_{k \geq 0} [\![\Gamma \rhd_e e]\!]\mathsf{Fst}^k; h\Delta'$$

is a reducibility candidate of type $h * e$, i.e. $C \in \mathcal{C}_{h*e,\Delta}$.

Proof Induction over the definition of $[\![\Gamma \rhd_e e]\!]$. We will only consider the most difficult cases $\Pi(L, B)$ and $\Sigma(A, B)$; the proof of the other cases follows a similar pattern.

The case $\Pi(A, B)$ requires three lemmata to ensure that if $g * \mathsf{Snd}$ is an element of some reducibility candidate C and $g; \mathsf{Fst} * A$ and $\langle \mathsf{Fst}^k; g; \mathsf{Fst}, \mathsf{Snd}[A]\rangle * B$ are canonical and $g * \mathsf{App}(A, B) \overset{W}{\rightsquigarrow} g'' * \mathsf{App}(A', B')$, then $g'' * \mathsf{Snd}$ is an element of C as well. First we analyze all possible $\overset{W}{\rightsquigarrow}$-contracta of the combinator $g * \mathsf{App}(A, B)$.

Lemma A.10 *If $g * \mathsf{App}(A, B) \overset{W^*}{\rightsquigarrow} g'' * \mathsf{App}(A', B')$ then either $g \overset{W}{\rightsquigarrow} g''$ or $g'' \equiv g'; \langle f, t[A'']\rangle$ and there exist combinators f' and A_1 such that*

- *If $f \not\equiv \mathsf{Id}$, we have two possible cases: either $f \equiv f''; \mathsf{Id}$ and $g \overset{W^*}{\rightsquigarrow} g'; \langle f''; f', t'[A_1]\rangle$ or $g \overset{W^*}{\rightsquigarrow} g'; \langle f; f', t'[A_1]\rangle$.*

- *If $f \equiv \mathsf{Id}$ then $g \overset{W^*}{\rightsquigarrow} g'; \langle f', t'[A_1]\rangle$.*

with $g; \mathsf{Fst} * A \overset{W^*}{\rightsquigarrow} g'; f'' * A''$ *or* $g; \mathsf{Fst} * A \overset{W^*}{\rightsquigarrow} g'; f * A''$ *respectively and* $\langle \mathsf{Fst}; g; \mathsf{Fst}, \mathsf{Snd}[A]\rangle * B \overset{W^*}{\rightsquigarrow} B'$ *as well as* $g * \mathsf{Snd} \overset{W^*}{\rightsquigarrow} g' * t$.

Proof Induction over the number of reduction steps. □

Next we show that the combinator $g * \mathsf{App}(A, B)$ is strongly normalizing.

Lemma A.11 *For any context morphism* g *and types* A *and* B *and any reducibility candidate* C *such that* $g * \mathsf{Snd} \in C$ *and* $g; \mathsf{Fst} * A$ *as well as* $\langle \mathsf{Fst}^{k+1}; g; \mathsf{Fst}, \mathsf{Snd}[A]\rangle * B$ *are canonical, the combinator* $g * \mathsf{App}(A, B)$ *is strongly normalizing.*

Proof Consider any infinite reduction sequence of $g * \mathsf{App}(A, B)$. Either it consists only of $\overset{W}{\rightsquigarrow}$-reductions or it looks like

$$g * \mathsf{App}(A, B) \overset{W^*}{\rightsquigarrow} \langle \mathsf{Id}, t[A'']\rangle * \mathsf{App}(A', B') \overset{N}{\rightsquigarrow} \cdots$$

with t, A'', A' and B' as in the previous lemma. In the latter case the hypotheses imply the claim. If the infinite sequence consists only of $\overset{W}{\rightsquigarrow}$-reductions, the previous lemma shows that

$$g * \mathsf{App}(A, B) \overset{W}{\rightsquigarrow} g'; \langle f, t[A'']\rangle * \mathsf{App}(A', B')$$

The restrictions in the $\overset{W}{\rightsquigarrow}$-rules App_1 and App_2 imply that we can assume without loss of generality $g \overset{W^*}{\rightsquigarrow} g'; \langle f; f', t'[A_1]\rangle$. Now an induction over the tuple

$$(\nu(g'; f), \nu(g' * t'), \mathsf{compl}(g'), \mathsf{compl}(f), \nu(g'; f * A''))$$

ordered lexicographically, shows that this sequence is actually finite. The number $\nu(e)$ stands for the length of the longest $\overset{W}{\rightsquigarrow}$-reduction sequence of any strongly normalizing combinator e, and $\mathsf{compl}(e)$ denotes the structural complexity of e. □

Finally we strengthen this lemma to show that whenever $g * \mathsf{App}(A, B)$ reduces to $g'' * \mathsf{App}(A', B')$, the morphism $g'' * \mathsf{Snd}$ is an element of C.

Lemma A.12 *Let* g, A, B *and* C *as in the previous lemma. Then* $g * \mathsf{App}(A, B) \overset{W^*}{\rightsquigarrow} g'' * \mathsf{App}(A', B')$ *implies* $g'' * \mathsf{Snd} \in C$.

Proof By condition ($CR3$) for C, we have to show that $\mathsf{Fst}^k; g'' * \mathsf{Snd}$ is strongly normalizing and that $\mathsf{Fst}^k; g'' * \mathsf{Snd}$ reduces via $\overset{W}{\rightsquigarrow}$ only to neutral morphisms or elements of C. We can restrict ourselves to the case $k = 0$ because with g, A and B also $\mathsf{Fst}^k; g$, A and B satisfy the hypotheses of the previous lemma. Because g'' admits no infinite $\overset{W}{\rightsquigarrow}$-reduction sequence by the previous lemma, any reduction sequence of $g'' * \mathsf{Snd}$ is a prefix of

$$g'' * \mathsf{Snd} \overset{W^*}{\rightsquigarrow} h; \langle h', s[A_1]\rangle * \mathsf{Snd} \overset{W}{\rightsquigarrow} h * s \overset{N}{\rightsquigarrow} \cdots$$

with $g'' \overset{W}{\rightsquigarrow} h; \langle h', s[A_1]\rangle$. The previous lemma implies that $g'' * \mathsf{Snd} \overset{W^*}{\rightsquigarrow} h * s$, so $h * s$ is also in C. □

Now we can start proving the theorem. According to definition A.1, we have to verify that C satisfies the conditions $(CR1)$ to $(CR4)$ and is nonempty.

$(CR1)$ Follows directly from the definition.

$(CR2)$ Fix a morphism $t \in \llbracket \Gamma \rhd_e \Pi(L,B) \rrbracket \mathsf{Fst}^l; h\Delta'$ for some l and a morphism t' with $\mathsf{Fst}^k * t \overset{W}{\rightsquigarrow} t'$. For all n, $\mathsf{Fst}^n * t'$ is SN because $\mathsf{Fst}^{k+n} * t$ is. The morphism $s \stackrel{\text{def}}{=} \mathsf{Fst}^{m+k} * t; g * \mathsf{App}(L', B')$ with m, g, L', B' and C_1 as in the definition of $\llbracket \Gamma \rhd_e \Pi(L,B) \rrbracket \mathsf{Fst}^l; h\Delta'$ is an element of $\llbracket \Gamma \cdot L \rhd_e B \rrbracket (\langle \mathsf{Fst}^{l+m+k}; h, g * \mathsf{Snd}[L] \rangle (D'', C_1))\Delta''$. Because s reduces via $\overset{W}{\rightsquigarrow}$ to $\mathsf{Fst}^m * t'; g * \mathsf{App}(L', B')$, condition $(CR2)$ for $\bigcup_{j \geq 0} \llbracket \Gamma \cdot L \rhd_e B \rrbracket (\mathsf{Fst}^j; \langle \mathsf{Fst}^{l+m+k}; h, g * \mathsf{Snd}[L] \rangle (D''', C_1))\Delta'''$ implies that t' is an element of $\llbracket \Gamma \cdot \Pi(L,B) \rrbracket \mathsf{Fst}^l; h; \Delta'$ as well.

$(CR3)$ Consider any t satisfying the hypotheses of $(CR3)$. It is enough to show that $s \stackrel{\text{def}}{=} \mathsf{Fst}^m * t; g * \mathsf{App}(L', B')$ is an element of $E_0 \stackrel{\text{def}}{=} \llbracket \Gamma \cdot L \rhd_e B \rrbracket (\mathsf{Fst}^0; \langle \mathsf{Fst}^{m+l}; h, g * \mathsf{Snd}[L] \rangle (D'', C))\Delta''$. Because s is neutral, by condition $(CR3)$ for $\bigcup_{k \geq 0} E_k$ it is enough to show that $\mathsf{Fst}^k * s$ is SN and that every reduction of $\mathsf{Fst}^k * s$ via $\overset{W}{\rightsquigarrow}$ leads after finitely many neutral morphisms to one in E_k. The latter is done by an induction over the degree of $\mathsf{Fst}^k * s$, which is the tuple

$$(k, \nu(\mathsf{Fst}^{k+m} * t) + \nu(\mathsf{Fst}^k; g * \mathsf{App}(L', B')))$$

ordered lexicographically. If $\nu(\mathsf{Fst}^k * s) = 0$ and reduces via $\overset{N}{\rightsquigarrow}$ to s', then $s' = t'; g * \mathsf{App}(L'', B'')$ with $k = 0$, $\mathsf{Fst}^m * t \overset{N}{\rightsquigarrow} t'$ and $g * \mathsf{App}(L', B') \overset{N}{\rightsquigarrow} g' * \mathsf{App}(L'', B'')$. Because $\mathsf{Fst}^m * t$ and $g * \mathsf{App}(L', B')$ are both SN, $\mathsf{Fst}^0 * s \equiv s$ is SN as well. Therefore s is in E_0 in this case. If $\mathsf{Fst}^k * s \overset{W}{\rightsquigarrow} s'$ for some s', then consider any

$$u \equiv f * (t_1; g_1 * \mathsf{App}(L', B'))$$

such that $f * t_1 \equiv \mathsf{Fst}^{k+m} * t$ and $f; g_1 \equiv \mathsf{Fst}^k; g$. Then u can reduce to

- $f' * (t_1'; g_1' * \mathsf{App}(L'', B''))$ with $f * t_1 \overset{W}{\rightsquigarrow} f' * t_1'$, t_1' neutral and $f; g_1 * \mathsf{App}(L', B') \overset{W}{\rightsquigarrow} f'; g_1' * \mathsf{App}(L'', B'')$. This eventually reduces to a morphism in E_k by the induction hypothesis and Lemma A.12.
- $f' * (t_1'; g_1 * \mathsf{App}(L', B'))$ with $t_1 \overset{W}{\rightsquigarrow} t_1'$ and t_1' not neutral. Then $f_1' * t_1''$ is an element of $\llbracket \Gamma \rhd_e \Pi(L,B) \rrbracket \mathsf{Fst}^{l+m+k}; h\Delta''$, so this combinator is an element of E_k.

Any infinite reduction sequence of $\mathsf{Fst}^k * s$ has the form

$$s \overset{N}{\rightsquigarrow}^* f' * (t;' g' * \mathsf{App}(L', B')) \overset{N}{\rightsquigarrow}^* \cdots$$

where t' is not neutral and $\mathsf{Fst}^{k+m} * t \overset{W}{\rightsquigarrow}^* f' * t'$, $\mathsf{Fst}^k; g * \mathsf{App}(L, B) \overset{W}{\rightsquigarrow}^* f' * g' * \mathsf{App}(L', B')$. Hence $f' * t'$ is in $\llbracket \Gamma \rhd_e \Pi(L,B) \rrbracket \mathsf{Fst}^{k''+m+l}; h\Delta''$, so the above sequence is finite.

($CR4$) This follows directly from the definition.

The proof that the combinator $\Delta \cdot h * \Pi(A,B) \rhd_e \mathsf{Snd}\colon 1 \to \mathsf{Fst}; h * \Pi(A,B)$ is an element of C is similar to the proof of the condition ($CR3$) and is therefore omitted.

In the case of $[\![\Gamma \rhd_e \Sigma(A,B)]\!] h\Delta$, the argument goes as follows:

($CR1$) $\mathsf{Fst}^k * t; \pi_1$ strongly normalizing implies $\mathsf{Fst}^k * t$ strongly normalizing.

($CR2$) For any l and any $t \in [\![\Gamma \rhd_e \Sigma(A,B)]\!]\mathsf{Fst}^l; h\Delta' \overset{\mathrm{def}}{=} E$, $\mathsf{Fst}^k * t \overset{W}{\rightsquigarrow} t'$ implies $\mathsf{Fst}^k * t; \pi_1 \overset{W}{\rightsquigarrow} t'; \pi_1$ and $\mathsf{Fst}^k * t; \pi_2 \overset{W}{\rightsquigarrow} t'; \pi_2$, so by condition ($CR2$) for $\bigcup_{k \geq 0} [\![\Gamma \cdot A]\!]\mathsf{Fst}^{l+k}; h\Delta''$ and for $\bigcup_{k \geq 0} [\![\Gamma \cdot A \rhd_e B]\!](\mathsf{Fst}^k; \langle \mathsf{Fst}^l; h, t; \pi_1[A]\rangle(D'', C))\Delta''$, the morphism t' is an element of $[\![\Gamma \rhd_e \Sigma(A,B)]\!]\mathsf{Fst}^{l+k}; h\Delta''$.

($CR3$) Consider any l and any $t \in [\![\Gamma \rhd_e \Sigma(A,B)]\!]\mathsf{Fst}^l; h\Delta'$ that satisfies the hypotheses of ($CR3$). We have to show that $s_1 = t; \pi_1$ is an element of $[\![\Gamma \rhd_e A]\!]\mathsf{Fst}^l; h\Delta'$ and that $s_2 = t; \pi_2$ is an element of

$$E_0 \overset{\mathrm{def}}{=} [\![\Gamma \cdot A \rhd_e B]\!](\mathsf{Fst}^0; \langle \mathsf{Fst}^l; h, t; \pi_1[A]\rangle(D', C))\Delta'$$

We demonstrate this only for s_2; the proof for s_1 is similar. Any infinite reduction sequence of $\mathsf{Fst}^k * t; \pi_2$ looks like

$$s_2 \overset{N*}{\rightsquigarrow} \langle f, t_1[A'], t_2[B'] \rangle * \mathsf{Pair}(A'', B''); \pi_2 \overset{N}{\rightsquigarrow} t_2 \rightsquigarrow \cdots$$

Because $\mathsf{Fst}^k * t$ is SN, t_2 is SN as well, and so this sequence is finite. Furthermore, a reduction $\mathsf{Fst}^k * t; \pi_2 \overset{W}{\rightsquigarrow} u$ is only possible if $u = t'; \pi_2$, where $\mathsf{Fst}^k * t \overset{W}{\rightsquigarrow} t'$. So $\mathsf{Fst}^k * s_2$ reduces to neutral morphisms and eventually to a morphism in E_k. Condition ($CR3$) for $\bigcup_{l \geq 0} E_l$ implies that E_0 contains s_2.

($CR4$) Consider any $t \in [\![\Gamma \cdot \Sigma(A,B)]\!]\mathsf{Fst}^l; h\Delta'$. Condition ($CR4$) for $\bigcup_{j \geq 0} [\![\Gamma \cdot A]\!]\mathsf{Fst}^{j+l}; h\Delta''$ implies that $\mathsf{Fst}^k * t; \pi_1 \in [\![\Gamma \rhd_e A]\!]\mathsf{Fst}^{k+l}; h\Delta''$. Similarly, condition ($CR4$) for

$$\bigcup_{k \geq 0} [\![\Gamma \cdot A \rhd_e B]\!](\mathsf{Fst}^k; \langle \mathsf{Fst}^l; h, t; \pi_1[A]\rangle(D'', C))\Delta''$$

implies that $\mathsf{Fst}^k * t; \pi_2$ is an element of

$$\bigcup_{k \geq 0} [\![\Gamma \cdot A \rhd_e B]\!](\mathsf{Fst}^k; \langle \mathsf{Fst}^l; h, t; \pi_1[A]\rangle(D'', C))\Delta''$$

An argument similar to that given for the condition ($CR3$) shows that the combinator Snd with $\Delta \cdot h * \Sigma(A,B) \rhd_e \mathsf{Snd}\colon 1 \to \mathsf{Fst}; h * \Sigma(A,B)$ is an element of C. □

In the next section we will show by an initiality argument that every combinator is an element of some reducibility candidate. The proof requires additional properties of the interpretation, which are given in the following lemma:

Lemma A.13 *For any candidate assignment* $h\colon \Delta \to \Gamma$*, such that* $h \in \llbracket\Gamma\rrbracket h\Delta$ *and for all context morphisms* $\Delta \rhd_e g\colon \Gamma \in \llbracket\Gamma\rrbracket h\Delta$ *the following conditions are satisfied:*

1. *If* $\Gamma = [\,]$*, then* $g; \langle\rangle$ *is an element of* $\llbracket[\,]\rrbracket h; \Delta$.

2. *The morphism* $g; \mathsf{Id}$ *is an element of* $\llbracket\Gamma\rrbracket h\Delta$.

3. *For any* $\Gamma \rhd_e f\colon \Gamma'$*,* t *and* A *such that* $g; f \in \llbracket\Gamma'\rrbracket(\llbracket\Gamma \rhd_e f\rrbracket h\Delta)\Delta$*,* $\Gamma \rhd_e t\colon 1 \to f * A$ *with* $g * t \in \llbracket\Gamma \rhd_e f * A\rrbracket h\Delta$ *and* A *canonical, the morphism* $g; \langle f, t[A]\rangle$ *is an element of* $\llbracket\Gamma' \cdot A\rrbracket(\llbracket\langle f, t[A]\rangle\rrbracket h\Delta)\Delta$.

4. *If* h *is an element of* $\llbracket\Gamma\rrbracket h\Delta$*,* A *is an element of* $can_{\mathsf{type},\Gamma}$ *and* t *is a morphism satisfying* $\Delta \rhd_e t\colon 1 \to h*A$ *and* $t \in \llbracket\Gamma \rhd_e A\rrbracket h\Delta$*, then the morphism* $\langle h, t[A]\rangle$ *is an element of* $\llbracket\Gamma \cdot A\rrbracket(\langle h, t[A]\rangle(D, C))\Delta$ *for any reducibility candidate* $C \in \mathcal{C}_{t,\Delta}$.

5. *For any morphism* $\Gamma \cdot A \rhd_e t\colon 1 \to B$*, for any context morphism* g*, types* A_1 *and* B_1 *and reducibility candidates* $C \in \mathcal{C}_{g*\mathsf{Snd},\Delta'}$ *such that*

   - $g * \mathsf{Snd} \in \llbracket\Gamma \rhd_e A\rrbracket \mathsf{Fst}^m; h\Delta'$
   - $\mathsf{Fst}^m; h*A$ *and* $g; \mathsf{Fst}*A_1$ *as well as* $\langle \mathsf{Fst}^{m+1}; h, \mathsf{Snd}[A]\rangle * B$ *and* $\langle \mathsf{Fst}; g; \mathsf{Fst}, \mathsf{Snd}[A_1]\rangle * B_1$ *are convertible*
   - A*,* $g; \mathsf{Fst} * A_1$ *and* $\langle \mathsf{Fst}^{k+1}; g; \mathsf{Fst}, \mathsf{Snd}[A_1]\rangle * B_1$ *are canonical*
   - *For any candidate assignment* $\Delta' \rhd_e h'\colon \Gamma \cdot A$ *such that* $h' \in \llbracket\Gamma \cdot A\rrbracket h'\Delta$*,* $h' * t \in \llbracket\Gamma \cdot A \rhd_e B\rrbracket h'\Delta'$

   then

   $$\mathsf{Fst}^m; h*\mathsf{Cur}(A, t); g*\mathsf{App}(A_1, B_1) \in \llbracket\Gamma \cdot A \rhd_e B\rrbracket(\langle \mathsf{Fst}^m; h, g*\mathsf{Snd}[A]\rangle(D', C))\Delta'$$

6. *For any morphism* $t\colon 1 \to h * A \in \llbracket\Gamma \cdot A\rrbracket h\Delta$*, the morphism* $t; h*!$ *is an element of* $\llbracket\Gamma \rhd_e 1\rrbracket h\Delta$ *as well.*

7. *For any morphism* $t\colon 1 \to h*A \in \llbracket\Gamma \cdot A\rrbracket h\Delta$*, the morphism* $t; h*\mathsf{Id}$ *is an element of* $\llbracket\Gamma \rhd_e A\rrbracket h\Delta$ *as well.*

8. *For any type* A *in context* Γ*, the morphism* Snd *is an element of* $\llbracket A\rrbracket \mathsf{Fst}; h\Delta \cdot h * A$.

9. *For any context* Γ *and candidate assignment* $\Gamma \rhd_e \mathsf{Id}\colon \Gamma$*, the identity morphism is an element of* $\llbracket\Gamma\rrbracket \mathsf{Id}\Gamma$.

10. *For all context morphisms* $g \in \llbracket\Gamma\rrbracket h\Delta$*, the type* $g * \Omega$ *is canonical, and for all* $g' \in \llbracket[\,] \cdot \Omega\rrbracket h\Delta$*, the object* $g' * T$ *is canonical as well.*

11. *Let* h *be a candidate assignment* $\Delta \rhd_e h\colon \Gamma \cdot A \cdot B$ *and* $g; \mathsf{Fst}^2 * A$ *and* $\langle \mathsf{Fst}^{k+1}; g; \mathsf{Fst}^2, \mathsf{Snd}[A]\rangle * B$ *are canonical.*

    (i) *The morphism* $g * \mathsf{Pair}(A, B); \pi_1$ *is in* $\llbracket \mathsf{Fst}; \mathsf{Fst} * A\rrbracket h\Delta$ *if* $g; \mathsf{Fst} * \mathsf{Snd} \in \llbracket \mathsf{Fst}; \mathsf{Fst} * A\rrbracket h\Delta$.

(ii) *Similarly,* $g * \mathsf{Pair}(A, B); \pi_2 \in [\![\Gamma \cdot A \rhd_e B]\!](h; \mathsf{Fst}, (D_n, \dots, D_2, C))\Delta$ *if* $g * \mathsf{Snd} \in [\![\Gamma \cdot A \rhd_e B]\!](h; \mathsf{Fst}, (D_n, \dots, D_2, C))\Delta$.

Proof All these properties are in one way or another consequences of the condition $(CR3)$ in the definition of the reducibility candidates. Before we can go into the details, we have to show that $\bigcup_{l \geq 0} [\![\Gamma]\!]\mathsf{Fst}^l; h\Delta'$ satisfies the properties $(CR1)$ to $(CR4)$ as well. The proof of $(CR3)$ is the only nontrivial one; all others will be omitted. Consider any set $[\![\Gamma \cdot A]\!]\mathsf{Fst}^l; h\Delta'$ and any $f : \Delta' \to \Gamma \cdot A$. Assume f satisfies the hypotheses of condition $(CR3)$. Then we must show that $f; \mathsf{Fst}$ and $f * \mathsf{Snd}$ are elements of $[\![\Gamma]\!]\mathsf{Fst}^l; h; \mathsf{Fst}\Delta'$ and $C_0 \overset{\text{def}}{=} [\![\Gamma \rhd_e A]\!]\mathsf{Fst}^{l+0}; h; \mathsf{Fst}\Delta'$ respectively. We prove only the latter, by applying condition $(CR3)$ for $\bigcup_{k \geq 0} C_k$; the proof for the former is analogous. First, $s = \mathsf{Fst}^k; f * \mathsf{Snd}$ is SN because $\mathsf{Fst}^k; f$ is. Any reduction of s leads to a combinator $f' * \mathsf{Snd}$, where $\mathsf{Fst}^k; \mathsf{Fst} \rightsquigarrow f'$. Therefore any reduction sequence of s that is large enough must look like

$$\mathsf{Fst}^k; f * \mathsf{Snd} \rightsquigarrow f_1' * \mathsf{Snd} \rightsquigarrow \cdots \rightsquigarrow f_n' * \mathsf{Snd}$$

where $f_1', \dots, f_{n-1}'$ are neutral and f_n' is an element of $[\![\Gamma \cdot A]\!]\mathsf{Fst}^{l+k}; h\Delta''$. Hence $f_n' * \mathsf{Snd}$ is an element of C_k, and all the hypotheses of the conditions $(CR3)$ for $\bigcup_{k \geq 0} C_k$ are satisfied for $f * \mathsf{Snd}$.

Now we check each of the properties stated in the lemma in turn.

1. $\mathsf{Fst}^k; g; \langle\rangle$ is SN if $\mathsf{Fst}^k; g$ is, because the former reduces either to $g'; \langle\rangle$, where $\mathsf{Fst}^k; g \rightsquigarrow g'$, or to $\langle\rangle$.

2. $\mathsf{Fst}^k; g; \mathsf{Id}$ can either reduce to $g'; \mathsf{Id}$, where $\mathsf{Fst}^k; g \rightsquigarrow g'$, or to g. So $\mathsf{Fst}^k; g; \mathsf{Id}$ is SN, and an induction over $\nu(\mathsf{Fst}^k; g)$ shows that any reduction sequence yields one in $[\![\Gamma]\!]\mathsf{Fst}^k; h\Delta'$ after finitely many neutral morphisms.

3. $\mathsf{Fst}^k; g; \langle f, t[A]\rangle; \mathsf{Fst}$ reduces either to

   - $g'; \langle f', t'[A']\rangle; \mathsf{Fst}$ with $g \rightsquigarrow g'$, $f \rightsquigarrow f'$, $t \rightsquigarrow t'$ and $A \rightsquigarrow A'$.
   - $\mathsf{Fst}^k; g; f$, which is an element of $C \overset{\text{def}}{=} [\![\Gamma']\!]([\![\Gamma \rhd_e f]\!]\mathsf{Fst}^k; h\Delta')\Delta'$ by assumption.
   - $g_1; \langle g_2; f, g_2 * t[A]\rangle; \mathsf{Fst}$ with $g_1; g_2 = \mathsf{Fst}^k; g$.

   In all cases the combinator $\mathsf{Fst}^k; g; \langle f, t[A]\rangle; \mathsf{Fst}$ reduces to a neutral context morphism or to an element of C, and an induction over

   $$(\nu(\mathsf{Fst}^k; g), \nu(\mathsf{Fst}^k; g; f) + \nu(\mathsf{Fst}^k; g * t) + \nu(A))$$

   shows that any reduction eventually leads to a context morphism in C. The argument for the morphism $\mathsf{Fst}^k; g; \langle f, t[A]\rangle * \mathsf{Snd}$ is similar.

4. Similar to the previous condition.

5. Any combinator $u \equiv h_1 * (h_2 * \mathsf{Cur}(A, t); g_1 * \mathsf{App}(A_1, B_1))$ such that $h_1; h_2 \equiv \mathsf{Fst}^{k+m}; h$ and $h_1; g_1 \equiv \mathsf{Fst}^k * g$ reduces to

- $h_1; \langle h_2, g_1 * \mathsf{Snd}[A]\rangle * t$, which is an element of

  $$E \overset{\text{def}}{=} [\![\Gamma \cdot A \rhd_e B]\!](\langle \mathsf{Fst}^{k+m}; h, \mathsf{Fst}^k; g * \mathsf{Snd}[A]\rangle(D'', C'))\Delta''$$

  by assumption and condition 3.

- $h_1' * (h_2' * \mathsf{Cur}(A', t'); g' * \mathsf{App}(A_1', B_1'))$ with $h_1; h_2 \overset{W}{\rightsquigarrow} h_1'; h_2'$ and $h_1; g_1 * \mathsf{App}(A_1, B_1) \overset{W}{\rightsquigarrow} g' * \mathsf{App}(A_1', B_1')$, as well as $A \overset{W}{\rightsquigarrow} A'$, and $t \overset{W}{\rightsquigarrow} t'$.

These morphisms are either neutral or elements of E, and an induction over

$$(\nu(\mathsf{Fst}^{k+m}; h), \nu(A) + \nu(t) + \nu(\mathsf{Fst}^k; g * \mathsf{App}(A_1, B_1)))$$

shows that any reduction will yield a morphism in E. Because any infinite reduction path of s starts with a $\overset{W}{\rightsquigarrow}$-reduction as above and thus leads to an element of E, s is SN. Therefore the morphism $\mathsf{Fst}^m; h * \mathsf{Cur}(A, t); g * \mathsf{App}(A_1, B_1)$ is an element of

$$[\![\Gamma \cdot A \rhd_e B]\!](\langle \mathsf{Fst}^m; h, g * \mathsf{Snd}[A]\rangle(D', C))\Delta'$$

6. Analogous to the proof of condition 2.
7. Similar to the proof of condition 2.
8. Follows directly from condition ($CR3$).
9. Consequence of the previous condition.
10. If the combinator $\mathsf{Fst}^k; g$ is strongly normalizing, then also the combinators $\mathsf{Fst}^k; g * \Omega$ and $\mathsf{Fst}^k; g * T$.
11. The combinator $s \overset{\text{def}}{=} \mathsf{Fst}^k; g * \mathsf{Pair}(A, B); \pi_1$ can be reduced to

    - $g' * \mathsf{Pair}(A', B'); \pi_1$ with $\mathsf{Fst}^k; g \rightsquigarrow g'$, $A \rightsquigarrow A'$ and $B \rightsquigarrow B'$.
    - $\langle \mathsf{Id}, \mathsf{Fst}^k; g; \mathsf{Fst} * \mathsf{Snd}[\mathsf{Fst}^k; g; \mathsf{Fst}; \mathsf{Fst} * A], \mathsf{Fst}^k; g * \mathsf{Snd}[\langle \mathsf{Fst}^{k+1}; g; \mathsf{Fst}; \mathsf{Fst}, \mathsf{Snd}[A]\rangle * B]\rangle * \mathsf{Pair}(\mathsf{Fst}^k; g; \mathsf{Fst}; \mathsf{Fst} * A, \langle \mathsf{Fst}^{k+1}; g; \mathsf{Fst}; \mathsf{Fst}, \mathsf{Snd}[A]\rangle * B); \pi_1$
    - $\mathsf{Fst}^k; g; \mathsf{Fst} * \mathsf{Snd}$, which is an element of $E \overset{\text{def}}{=} [\![\Gamma \cdot A \cdot B \rhd_e \mathsf{Fst}; \mathsf{Fst} * A]\!]\mathsf{Fst}^k; h\Delta'$ by assumption.

    So, again $\mathsf{Fst}^k; g * \mathsf{Pair}(A, B); \pi_1$ reduces either to a neutral morphism or to an element in E, and an induction over

    $$\nu(\mathsf{Fst}^k; g) + \nu(\mathsf{Fst}^k; g; \mathsf{Fst}; \mathsf{Fst} * A) + \nu(\langle \mathsf{Fst}^{k+1}; g; \mathsf{Fst}; \mathsf{Fst}, \mathsf{Snd}[A]\rangle * B))$$

    proves that eventually the latter case is reached. So s is SN and therefore $g * \mathsf{Pair}(A, B); \pi_1$ is an element of $[\![\Gamma \cdot A \cdot B \rhd_e \mathsf{Fst}; \mathsf{Fst} * A]\!]h\Delta$. The combinator $g * \mathsf{Pair}(A, B); \pi_2$ is an element of $[\![\Gamma \cdot A \rhd_e B]\!](h; \mathsf{Fst}, (D_n, \ldots, D_2, C))\Delta$ for similar reasons.

□

Remark The proof of condition 5 uses the restriction imposed on the reduction $\overset{N}{\leadsto}$ in an essential way. Without it, a possible reduction sequence of $h * \mathsf{Cur}(A, t); g * \mathsf{App}(A_1, B_1)$ is:

$$\begin{aligned} h * \mathsf{Cur}(A,t); g * \mathsf{App}(A_1, B_1) &\leadsto \mathsf{Cur}(h * A, \langle \mathsf{Fst}; h, \mathsf{Snd}[A]\rangle * t); g * \mathsf{App}(A_1, B_1) \\ &\leadsto \langle \mathsf{Id}, g * \mathsf{Snd}[h * A]\rangle; \langle \mathsf{Fst}; h, \mathsf{Snd}[A]\rangle * t \end{aligned}$$

This means that we have to strengthen condition 5 to require that the morphism

$$\langle \mathsf{Id}, \mathsf{Fst}^k; g * \mathsf{Snd}[\mathsf{Fst}^{k+m}; h * A]\rangle; \langle \mathsf{Fst}^{k+m+1}; h, \mathsf{Snd}[A]\rangle * t$$

is an element of

$$[\![\Gamma \cdot A \rhd_e B]\!](\langle \mathsf{Fst}^{k+m}; h, \mathsf{Fst}^k; g * \mathsf{Snd}[A]\rangle(D'', C))\Delta''$$

But when we later on apply the condition, there seems to be no way of concluding that this is indeed the case (cf. page 137). This shows that condition 5 is the formulation of Lemma 3.2 in this context.

A.4 The Final Induction

The last step in the normalization proof is to show that every type is canonical and that every context morphism $\Gamma \rhd_e f{:}\,\Delta$ and morphism $\Gamma \rhd_e t{:}\,1 \rightarrow A$ is an element of a suitable interpretation of Δ and A respectively. This amounts to applying the appropriate closure condition of Lemma A.13.

Theorem A.14 *For any candidate assignment $\Delta \rhd_e h{:}\,\Gamma$ such that $h \in [\![\Gamma]\!]h\Delta$ and for any context morphism $\Gamma \rhd_e f{:}\,\Gamma'$, type $\Gamma \rhd_e A$ and morphism $\Gamma \rhd_e t{:}\,A \rightarrow B$:*

(i) *If $\Gamma \equiv [\,] \cdot A_n \cdots A_0$, then $h; \mathsf{Fst}^i * A_i \in can_{\mathrm{type},\Delta}$.*

(ii) $h; f \in [\![\Gamma']\!]([\![f]\!]h\Delta)\Delta$.

(iii) $h * A \in can_{\mathrm{type},\Delta}$.

(iv) *For all $s \in [\![\Gamma \rhd_e A]\!]h\Delta$, the morphism $s; h * t$ is an element of $[\![\Gamma \rhd_e B]\!]h\Delta$.*

Proof The properties stated in Lemma A.13 are used throughout this proof. They are referenced only by their number.

We use induction over the structure of the combinators.

(i) On contexts:

([ ]) Nothing to be shown.

($\Gamma \cdot A$) Follows directly from the definition of $[\![\Gamma \cdot A]\!]h\Delta$.

(ii) On context morphisms

($(\langle\rangle)$) Condition 1.

((Id)) Condition 2.

((Fst)) Definition of $\llbracket\Gamma\cdot A\rrbracket h\Delta$.

($(\langle f, t[A]\rangle)$) Condition 3.

$(g; f)$ Let $\Gamma\ \rhd_e\ g\colon\Gamma'$ and $\Gamma'\ \rhd_e\ f\colon\Gamma''$. By induction hypothesis, $h; g\in\llbracket\Gamma'\rrbracket(\llbracket g\rrbracket h\Delta)\Delta=\llbracket\Gamma'\rrbracket h; g\Delta$, and therefore by induction hypothesis again, $h; g; f\in\llbracket\Gamma''\rrbracket(\llbracket f\rrbracket h; g\Delta)\Delta$.

(iii) On types

(1) Obvious.

$(f * A)$ By induction hypothesis, $h; f\in\llbracket\Gamma'\rrbracket(\llbracket f\rrbracket h\Delta)\Delta=\llbracket\Gamma'\rrbracket h; f\Delta$. Another application of the induction hypothesis yields therefore $h*(f*A)=h; f*A\in can_{\mathrm{type},\Delta}$.

$(\Pi(A,B))$ Conditions 8 and 4 together with $(CR4)$ imply that $\langle\mathsf{Fst}; h,\mathsf{Snd}[A]\rangle$ is an element of $\llbracket\Gamma\cdot A\rrbracket(\langle\mathsf{Fst}; h_{cm},\mathsf{Snd}[A]\rangle(D_n,\ldots,D_1,C))\Delta\cdot h*A$, where $h=(h_{cm},D_n,\ldots,D_1)$ and $C\in\mathcal{C}_{\mathsf{Snd},\Delta\cdot h*A}$. Therefore $\langle\mathsf{Fst}; h,\mathsf{Snd}[A]\rangle*B$ is canonical, and because $h*\Pi(A,B)$ reduces either to $h'*\Pi(A',B')$ with $h\overset{W}{\rightsquigarrow}h'$, $A\overset{W}{\rightsquigarrow}A'$, $B\overset{W}{\rightsquigarrow}B'$ or to $\Pi(h*A,\langle\mathsf{Fst}; h,\mathsf{Snd}[A]\rangle*B)$, the type $h*\Pi(A,B)$ is SN. Because $\mathsf{Fst}^k; h$ is an element of $\llbracket\Gamma\rrbracket\mathsf{Fst}^k; h\Delta'$ by condition $(CR4)$, the type $h*\Pi(A,B)$ is canonical.

$(\Sigma(A,B))$ Similar to $\Pi(A,B)$.

(Ω, T) Condition 10.

(iv) On morphisms

($(!)$) Condition 6.

((Id)) Condition 7.

$(t_1; t_2)$ Let $\Gamma\ \rhd_e\ t_1\colon A\to B$ and $\Gamma\ \rhd_e\ t_2\colon B\to C$. The induction hypothesis yields $s; h*t_1\in\llbracket\Gamma\ \rhd_e\ B\rrbracket h\Delta$, and therefore $s; h*(t_1; t_2)=s; h*t_1; h*t_2\in\llbracket\Gamma\ \rhd_e\ C\rrbracket h\Delta$.

$(g*t)$ Same as $f*A$.

(Snd) By definition of $\llbracket\Gamma\rrbracket h\Delta$, $h*\mathsf{Snd}\in\llbracket\Gamma\cdot A\ \rhd_e\ \mathsf{Fst}*A\rrbracket h\Delta=\llbracket\Gamma\cdot A\rrbracket h;\mathsf{Fst}\Delta$.

$(\forall)$ Similar to $\Pi(A,B)$.

$(\mathsf{Cur}(A,t))$ As shown in the case $\Pi(A,B)$, the morphism $\langle\mathsf{Fst}; h,\mathsf{Snd}[A]\rangle$ is an element of $\llbracket\Gamma\cdot A\rrbracket(\langle\mathsf{Fst}^{k+1}; h,\mathsf{Snd}[A]\rangle(D_n,\ldots,D_1,C))\Delta\cdot\mathsf{Fst}^k; h*A$ for any $C\in\mathcal{C}_{\mathsf{Snd},\Delta\cdot\mathsf{Fst}^k; h*A}$. Therefore, $\langle\mathsf{Fst}^{k+1}; h,\mathsf{Snd}[A]\rangle*t$ is an element of the set $\llbracket\Gamma\cdot A\ \rhd_e\ B\rrbracket(\langle\mathsf{Fst}^{k+1}; h,\mathsf{Snd}[A]\rangle(D_n,\ldots,D_1,C))\Delta\cdot\mathsf{Fst}^k; h*A$ as well, so $\mathsf{Fst}^k; h*\mathsf{Cur}(A,t)$ is SN. Conditions 5 and 3 show now the claim.

($\mathsf{App}(A,B)$) Let $t{:}\,1{\to}h;\mathsf{Fst}*\Pi(A,B)$ be any morphism in $\llbracket\Gamma\cdot A\rhd_e \mathsf{Fst}*\Pi(A,B)\rrbracket h\Delta$. By definition, the combinator $h{*}\mathsf{Snd}$ is an element of $\llbracket\Gamma\rhd_e A\rrbracket h;\mathsf{Fst}\Delta$ and $h;\mathsf{Fst}$ is an element of $\llbracket\Gamma\rrbracket h;\mathsf{Fst}\Delta$. A similar argument like that above yields

$$\langle\mathsf{Fst};h;\mathsf{Fst},\mathsf{Snd}[A]\rangle\in\llbracket\Gamma\cdot A\rrbracket(\langle\mathsf{Fst};h;\mathsf{Fst},\mathsf{Snd}[A]\rangle(D_n,\ldots,D_0))\Delta\cdot h;\mathsf{Fst}{*}A$$

hence $\langle\mathsf{Fst};h;\mathsf{Fst},\mathsf{Snd}[A]\rangle{*}B$ is canonical. Therefore the definition of $\llbracket-\rrbracket$ implies that

$$t;h*\mathsf{App}(A,B)\in\llbracket\Gamma\cdot A\rhd_e B\rrbracket(\langle h;\mathsf{Fst},h*\mathsf{Snd}[A]\rangle(D_n,\ldots,D_1,D_0))\Delta$$

which is equivalent with

$$t;h*\mathsf{App}(A,B)\in\llbracket\Gamma\cdot A\rhd_e B\rrbracket h\Delta$$

($\mathsf{Pair}(A,B)$) Let us assume that $h=(h_{cm},(D_n,\ldots,D_0))$. Then we must show that $h*\mathsf{Pair}(A,B);\pi_1\in\llbracket\Gamma\cdot A\cdot B\rhd_e\mathsf{Fst};\mathsf{Fst}*A\rrbracket h\Delta$ and

$$\begin{aligned}
h*\mathsf{Pair}(A,B);\pi_2 &\in \llbracket\Gamma\cdot A\cdot B\cdot\mathsf{Fst}^2*A\rhd_e\langle\mathsf{Fst}^3,\mathsf{Snd}[A]\rangle*B\rrbracket\\
&\ (\langle h,h*\mathsf{Pair}(A,B);\pi_1[A]\rangle(D_n,\ldots,D_0,C))\Delta\\
\Leftrightarrow h*\mathsf{Pair}(A,B);\pi_2 &\in \llbracket\Gamma\cdot A\rhd_e B\rrbracket(\langle h;\mathsf{Fst};\mathsf{Fst},h*\mathsf{Fst}*\mathsf{Snd}[A]\rangle\\
&\ (D_n,\ldots,D_2,C))\Delta
\end{aligned}$$

for any $C\in C_{h*\mathsf{Pair}(A,B);\pi_1,\Delta}$. By an argument similar to the case $\Pi(A,B)$, the combinators $h;\mathsf{Fst};\mathsf{Fst}*A$ and $\langle\mathsf{Fst};h;\mathsf{Fst};\mathsf{Fst},\mathsf{Snd}[A]\rangle*B$ are canonical, so condition 11 yields the claim.

(π_i) Given $t\in\llbracket\Gamma\rhd_e\Sigma(A,B)\rrbracket h\Delta$, the definition of $\llbracket-\rrbracket$ implies that $t;h{*}\pi_1=t;\pi_1$ is an element of $\llbracket\Gamma\rhd_e A\rrbracket h\Delta$. Furthermore

$$\begin{aligned}
h*t;\pi_2 &\in \llbracket\Gamma\cdot A\rhd_e B\rrbracket\langle h,h*t;\pi_1[A]\rangle(D,\llbracket t;\pi_1\rrbracket h\Delta)\Delta\\
&= \llbracket\Gamma\rhd_e\langle\mathsf{Id},t;\pi_1[A]\rangle*B\rrbracket h\Delta
\end{aligned}$$

□

Remark The proof of the case $\mathsf{Cur}(A,t)$ fails if we strengthen condition 5 as in page 135 because there seems to be no way of deducing that $\langle\mathsf{Id},g*\mathsf{Snd}[g;\mathsf{Fst}*A_1]\rangle;\langle\mathsf{Fst};h,\mathsf{Snd}[A]\rangle$ is an element of $\llbracket\Gamma\cdot A\rrbracket(h,g*\mathsf{Snd}[A](D,C))\Delta$. This motivates the restriction imposed on the reduction relation $\rightsquigarrow$.

Because any element of any reducibility candidate is SN, we get the result we want:

Corollary A.15 *All combinators e are SN.*

Proof Apply the previous theorem for the candidate assignment (Id,D), where D is the list of the canonical reducibility candidates of the appropriate type. Therefore $\mathsf{Id}*e$ is strongly normalizing, and hence e. □

Appendix B

The Rules in de Bruijn form

This appendix contains the complete set of the rules for well-formed types, terms and contexts of the Calculus of Constructions in de Bruijn-form.

1. Formation of contexts and variables:

$$\text{Empty} \qquad \frac{}{\vdash [\,]\ \text{ctxt}}$$

$$\text{Cont} - \text{Intro} \qquad \frac{\Gamma \vdash A\ \text{type}}{\vdash (\Gamma, A)\ \text{ctxt}}$$

$$\text{Var} \qquad \frac{\vdash (\Gamma, A, \Gamma')\ \text{ctxt}}{(\Gamma, A, \Gamma') \vdash |\Gamma'| \colon \mathsf{U}_0^{|\Gamma'|+1}(A)}$$

2. Rules for Equality:

$$\text{Cequ} \qquad \frac{\vdash (\Gamma, A, \Gamma')\ \text{ctxt} \qquad \Gamma \vdash A = B}{(\Gamma, A, \Gamma') = (\Gamma, B, \Gamma')}$$

$$\text{Refl} \qquad \frac{\vdash \Gamma\ \text{ctxt}}{\Gamma = \Gamma} \quad \frac{\Gamma \vdash A\ \text{type}}{\Gamma \vdash A = A'} \quad \frac{\Gamma \vdash t \colon A}{\Gamma \vdash t = t \colon A}$$

$$\text{Symm} \qquad \frac{\Gamma = \Gamma'}{\Gamma' = \Gamma} \quad \frac{\Gamma \vdash t = s \colon A}{\Gamma \vdash s = t \colon A} \quad \frac{\Gamma \vdash A = B}{\Gamma \vdash B = A}$$

$$\text{Trans} \qquad \frac{\Gamma = \Gamma' \qquad \Gamma' = \Gamma''}{\Gamma = \Gamma''} \quad \frac{\Gamma \vdash A = B \qquad \Gamma \vdash B = C}{\Gamma \vdash A = C}$$

$$\frac{\Gamma \vdash t = t' \colon A \qquad \Gamma \vdash t' = t'' \colon A}{\Gamma \vdash t = t'' \colon A}$$

$$\text{Conv} \qquad \frac{\Gamma \vdash A = B \qquad \Gamma \vdash t \colon A}{\Gamma \vdash t \colon B} \quad \frac{\Gamma \vdash A = B \qquad \Gamma \vdash t = s \colon A}{\Gamma \vdash t = s \colon B}$$

3. Rules for the dependent product:

Π – form	$\dfrac{(\Gamma, A) \vdash B \text{ type}}{\Gamma \vdash \Pi A.B \text{ type}}$
Π – Equ	$\dfrac{\Gamma \vdash A = A' \quad (\Gamma, A) \vdash B = B'}{\Gamma \vdash \Pi A.B = \Pi A'.B'}$
Π – Intro	$\dfrac{(\Gamma, A) \vdash t\colon B}{\Gamma \vdash (\lambda A.t)\colon \Pi A.B}$
ξ – rule	$\dfrac{(\Gamma, A) \vdash t = t'\colon B \quad \Gamma \vdash A = A'}{\Gamma \vdash \lambda A.t = \lambda A'.t'\colon \Pi A.B}$
Π – elim	$\dfrac{\Gamma \vdash t\colon \Pi A.B \quad \Gamma \vdash s\colon A \quad (\Gamma, A) \vdash B \text{ type}}{\Gamma \vdash \mathsf{App}(A, B, t, s)\colon B[0\backslash s]}$
Π – elimequ1	$\dfrac{\Gamma \vdash A = A' \quad (\Gamma, A) \vdash B = B'}{\Gamma \vdash \mathsf{App}(A, B, t, s) = \mathsf{App}(A', B', t, s)\colon B'[0\backslash s]}$
Π – elimequ2	$\dfrac{\Gamma \vdash t = t'\colon \Pi A.B \quad \Gamma \vdash s = s'\colon A}{\Gamma \vdash \mathsf{App}(A, B, t, s) = \mathsf{App}(A, B, t', s')\colon B[0\backslash s]}$
β – rule	$\dfrac{(\Gamma, A) \vdash t\colon B \quad \Gamma \vdash s\colon A \quad (\Gamma, A) \vdash B \text{ type}}{\Gamma \vdash \mathsf{App}(A, B, \lambda A.t, s) = t[0\backslash s]\colon B[0\backslash s]}$
η – rule	$\dfrac{\Gamma \vdash t\colon \Pi A.B \quad (\Gamma, A) \vdash B \text{ type}}{\Gamma \vdash \lambda A.\mathsf{App}((A)\uparrow, \mathsf{U}_1^1(B), (t)\uparrow, 0) = t\colon \Pi A.B}$

4. Rules for dependent sum:

Σ – form	$\dfrac{(\Gamma, A) \vdash B \text{ type}}{\Gamma \vdash \Sigma A.B \text{ type}}$
Σ – Equ	$\dfrac{\Gamma \vdash A = A' \quad (\Gamma, A) \vdash B = B'}{\Gamma \vdash \Sigma A.B = \Sigma A'.B'}$
Σ – Intro	$\dfrac{\Gamma \vdash t_1\colon A \quad (\Gamma, A) \vdash B \text{ type} \quad \Gamma \vdash t_2\colon B[0\backslash t_1]}{\Gamma \vdash \mathsf{Pair}(A, B, t_1, t_2)\colon \Sigma A.B}$
Σ – Introequ1	$\dfrac{\Gamma \vdash A = A' \quad (\Gamma, A) \vdash B = B'}{\Gamma \vdash \mathsf{Pair}(A, B, t, s) = \mathsf{Pair}(A', B', t, s)\colon \Sigma A.B}$
Σ – Introequ2	$\dfrac{\Gamma \vdash t = t'\colon A \quad \Gamma \vdash s = s'\colon B[0\backslash t]}{\Gamma \vdash \mathsf{Pair}(A, B, t, s) = \mathsf{Pair}(A, B, t', s')\colon \Sigma A.B}$
Σ – Elim1	$\dfrac{\Gamma \vdash t\colon \Sigma A.B}{\Gamma \vdash \pi_1(t)\colon A}$
Σ – Elim2	$\dfrac{\Gamma \vdash t\colon \Sigma A.B}{\Gamma \vdash \pi_2\colon B[0\backslash \pi_1(t)]}$
Σ – Elimequ1	$\dfrac{\Gamma \vdash t_1 = t_2\colon \Sigma A.B}{\Gamma \vdash \pi_1(t_1) = \pi_1(t_2)\colon A}$

$$\Sigma - \text{Elimequ2} \qquad \frac{\Gamma \vdash t_1 = t_2 \colon \Sigma A.B}{\Gamma \vdash \pi_2(t_1) = \pi_2(t_2) \colon B[0\backslash\pi_1(t_1)]}$$

$$(\sigma_1) \qquad \frac{\Gamma \vdash \mathsf{Pair}(A, B, t, s) \colon \Sigma A.B}{\Gamma \vdash \pi_1(\mathsf{Pair}(A, B, t, s)) = t \colon A}$$

$$(\sigma_2) \qquad \frac{\Gamma \vdash \mathsf{Pair}(A, B, t, s) \colon \Sigma A.B}{\Gamma \vdash \pi_2(\mathsf{Pair}(A, B, t, s)) = s \colon B[0\backslash t]}$$

$$(surj) \qquad \frac{\Gamma \vdash t \colon \Sigma A.B}{\Gamma \vdash \mathsf{Pair}(A, B, \pi_1(t), \pi_2(t)) = t}$$

5. Rules for propositions:

$$\mathsf{Prop1} \qquad \frac{\vdash \Gamma \text{ ctxt}}{\Gamma \vdash \mathsf{Prop} \text{ type}}$$

$$\mathsf{Proof} \qquad \frac{\Gamma \vdash p \colon \mathsf{Prop}}{\Gamma \vdash \mathsf{Proof}(p) \text{ type}}$$

$$\mathsf{Prop} - \text{equ} \qquad \frac{\Gamma \vdash p = p' \colon \mathsf{Prop}}{\Gamma \vdash \mathsf{Proof}(p) = \mathsf{Proof}(p')}$$

$$\forall - \text{Intro} \qquad \frac{(\Gamma, A) \vdash p \colon \mathsf{Prop}}{\Gamma \vdash \forall A.p \colon \mathsf{Prop}}$$

$$\forall - \text{equ} \qquad \frac{(\Gamma, A) \vdash p = p' \colon \mathsf{Prop} \qquad \Gamma \vdash A = A'}{\Gamma \vdash \forall A.p = \forall A'.p' \colon \mathsf{Prop}}$$

$$\forall - \text{elim} \qquad \frac{(\Gamma, A) \vdash p \colon \mathsf{Prop}}{\Gamma \vdash \mathsf{Proof}(\forall A.p) = \Pi A.\mathsf{Proof}(p)}$$

List of Tables

Bibliography

[ACCL90] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. In *Proceedings of the 15th Symposium on Principles of Programming Languages*, pages 31–46. ACM, 1990.

[Asp92] Andrea Asperti. A categorical understanding of environment machines. *Journal of Functional Programming*, 2(1):23–39, January 1992.

[BW90] Michael Barr and Charles Wells. *Category Theory for Computing Science.* International Series in Computer Science. Prentice Hall, New York, London, Toronto, 1990.

[Car86] John Cartmell. Generalised algebraic theories and contextual categories. *Annals of Pure and Applied Logic*, 32:209–243, 1986.

[CCM87] Guy Cousineau, Pierre-Louis Curien, and Michel Mauny. The categorical abstract machine. *Science of Computer Programming*, 8:173–202, 1987.

[CD91] Pierre-Louis Curien and Roberto Di Cosmo. A confluent reduction for the λ-calculus with surjective pairing and terminal object. In J. Leach Albert, B. Monien, and M. Rodriguez Artalejo, editors, *Proceedings of the 18th International Colloquium on Automata, Languages and Programming.* Lecture Notes in Computer Science 510, Springer Verlag, 1991.

[CE87] Thierry Coquand and Thomas Ehrhard. An equational presentation of higher order logic. In D. Pitt, A. Poigné, and D. Rydeheard, editors, *Category Theory and Computer Science*, pages 40–56. LNCS 283, Springer, September 1987.

[CG90] Thierry Coquand and Jean Gallier. A proof of strong normalization for the theory of constructions using a Kripke-like interpretation. In Gérard Huet and Gordon Plotkin, editors, *Proceedings of the First Workshop on Logical Frameworks, Antibes*, pages 479–497, 1990.

[CH85] Thierry Coquand and Gérard Huet. Constructions: A higher order proof system for mechanizing mathematics. In Bruno Buchberger, editor, *EUROCAL '85: European Conference on Computer Algebra, Vol. 1: Invited Lectures*, pages 151–184. LNCS 203, Berlin, Heidelberg, New York, 1985.

[CH88] Thierry Coquand and Gérard Huet. The Calculus of Constructions. *Information and Computation*, 76:95–120, 1988.

[CHR92] P.-L. Curien, T. Hardin, and A. Ríos. Strong normalisation of substitutions. In I.M. Havel and V. Koubek, editors, *Mathematical Foundations of Computer Science 1992*, pages 209–217. Lecture Notes in Computer Science No. 629, Berlin, Heidelberg, New York, 1992.

[Coq85] Thierry Coquand. *Une Théorie des Constructions.* PhD thesis, L'Université Paris VII, January 1985.

[Coq86] Thierry Coquand. An analysis of Girard's paradox. In *First Annual Symposium on Logic in Computer Science*, pages 227–236, Boston, 1986. IEEE.

[CP90] Roy L. Crole and Andrew M. Pitts. New foundations for fixpoint computations. In *Fifth Annual Symposium on Logic in Computer Science*, pages 489–497. IEEE, 1990.

[Cré90] P. Crégut. An abstract machine for the normalization of λ-terms. In *Proceedings of the 1990 ACM Conference on LISP and Functional Programming*, pages 333–340, New York, 1990.

[Cur86] Pierre-Louis Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming.* Pitman, 1986.

[Cur89] Pierre-Louis Curien. Alpha-conversion, conditions on variables and categorical logic. *Studia Logica*, 18(3), 1989.

[Cur91] Pierre-Louis Curien. An abstract framework for environment machines (Note). *Theoretical Computer Science*, 82(2):389–402, 1991.

[DFH$^+$91] Gilles Dowek, A. Felty, Gérard Huet, H. Herbelin, Christine Paulin-Mohring, and B. Werner. The system Coq. Users Guide, 1991.

[Ehr88a] Thomas Ehrhard. A categorical semantics of constructions. In *Third Annual Symposium on Logic in Computer Science*, pages 264–273. IEEE, 1988.

[Ehr88b] Thomas Ehrhard. *Une sémantique catégorique des types dépendants: Application au Calcul des Constructions.* PhD thesis, Université Paris VII, 1988.

[Geu92] Herman Geuvers. The church-rosser property for beta-eta reduction in typed lambda calculi. In *Seventh Annual IEEE Symposium on Logic in Computer Science*, New York, 1992.

[Gir71] Jean-Yves Girard. Une extension de l'interpretation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. In J. E. Fenstad, editor, *2nd Scandinavian Logic symposium*, pages 63–92. North Holland, 1971.

[GTL89] Jean-Yves Girard, Paul Taylor, and Yves Lafonts. *Proofs and Types.* Cambridge University Press, Cambridge, 1989.

[HL86] Thérèse Hardin and Alain Laville. Proof of termination of the rewriting system Subst on CCL. *Theoretical Computer Science*, 46:305–312, 1986.

[Hoa69] C. A. R. Hoare. An axiomatic basis of computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[Hoa70] C. A. R. Hoare. Procedures and parameters: An axiomatic approach. In E. Engeler, editor, *Symposium on Semantics of Algorithmic Languages*, pages 102–116. LNM 188, Springer Verlag, Berlin, 1970.

[How80] W. A. Howard. The formulae-as-types-notion of construction. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, 1980.

[HP89] J. Martin E. Hyland and Andrew M. Pitts. The theory of constructions: Categorical semantics and topos theoretic models. *Contemporary Mathematics*, 92:137–198, 1989.

[HP91] Robert Harper and Robert Pollack. Type checking with universes. *Theoretical Computer Science*, 89(1):107–136, 1991.

[Hue89] Gérard Huet. The constructive engine. In *The Calculus of Constructions, Documentation and user's guide, Version 4.10, Projet FORMEL (Technical Report No. 110).* INRIA, Paris, August 1989.

[Jac91] Bart Jacobs. *Categorical Type Theory.* PhD thesis, University of Nijmegen, 1991.

[Jac92] Bart Jacobs. Simply typed and untyped lambda calculus revisited. In Michael Fourman, Peter Johnstone, and Andrew Pitts, editors, *Applications of Categories in Computer Science*, LMS Lecture Note Series 177, pages 119–142. Cambridge University Press, 1992.

[Jay92] C. Barry Jay. Long $\beta\eta$ normal forms and confluence (revised). Preprint, February 1992.

[Klo90] Jan Willem Klop. Term rewriting system: From church-rosser to knuth-bendix and beyond. In M. S. Paterson, editor, *Proceedings of the 17th International Colloquium on Automata, Languages and Programming, Warwick*, pages 350–369. Lecture Notes in Computer Science No. 443, Berlin, Heidelberg, New York, 1990.

[Kri85] J.-L. Krivine. Un interpréteur du λ-calcul. Unpublished, 1985.

[Laf88] Yves Lafont. *Logiques, Categories & Machines: Implantation de Langages de Programmation guidée par la Logique Catégorique.* PhD thesis, Université de Paris VII, 1988.

[Law69] F. William Lawvere. Adjointness in foundations. *Dialectica*, 23(3–4):281–296, 1969.

[Law70] F. William Lawvere. Equality in hyperdoctrines and comprehension schema as an adjoint functor. In *Proceedings of Symposia in Pure Mathematics, Vol. XVIII: Applications of Categorical Algebra*, pages 1–14. American Mathematical Society, 1970.

[LS85] Joachim Lambek and Philip J. Scott. *Introduction to Higher-Order Categorical Logic.* Cambridge University Press, 1985.

[Luo90] Zhaohui Luo. *An Extended Calculus of Constructions.* PhD thesis, Laboratory for Foundations for Computer Science, Edinburgh, July 1990.

[Mac71] Saunders Mac Lane. *Categories for the Working Mathematician.* Graduate Texts in Mathematics. Springer, 1971.

[MH88] John C. Mitchell and Robert Harper. The essence of ML. In *15th Symposium on Principles of Programming Languages*, pages 28–46. ACM, January 1988.

[Mit86] John C. Mitchell. A type-inference approach to reduction properties and semantics of polymorphic expressions (summary). In *Proceedings of the 1986 ACM Conference on Lisp and Functional Programming*, pages 308–319, New York, 1986. ACM.

[Mit90] John C. Mitchell. A type-inference approach to reduction properties and semantics of polymorphic expressions. In Gérard Huet, editor, *Logical Foundations of Functional Programming*, chapter 9, pages 195–212. Addison Wesley, 1990.

[Mog89] Eugenio Moggi. A category-theoretic account of program modules. In *Category Theory and Computer Science.* Springer, LNCS 389, 1989.

[MP88] John C. Mitchell and Gordon D. Plotkin. Abstract types have existential type. *ACM Transactions on Programming Languages and Systems*, 10(3):470–502, July 1988.

[MR92] QingMing Ma and John C. Reynolds. Types, abstraction and parametric polymorphism, part 2. In *Proceedings of the 1991 Mathematical Foundations of Programming Semantics Conference*, pages 1–40. Lecture Notes in Computer Science No. 598, Berlin, Heidelberg, New York, 1992.

[Pfe91] Frank Pfenning. Unification and anti-unification in the calculus of constructions. In *Proceedings of the Sixth Annual IEEE symposium on Logic in Computer Science*, pages 74–85, 1991.

[Pit87] Andrew M. Pitts. Polymorphism is set theoretic, constructively. In David H. Pitt, Axel Poigné, and David E. Rydeheard, editors, *Category Theory and Computer Science*, pages 12–38. Springer, LNCS 283, 1987.

[Pit89] Andrew M. Pitts. Categorical semantics of dependent types. Talk given at SRI, June 1989.

[RPar] John C. Reynolds and Gordon D. Plotkin. On functors expressible in the polymorphic typed lambda calculus. *Information and Computation*, to appear.

[See87] Robert A. G. Seely. Categorical semantics for higher-order polymorphic λ-calculus. *Journal of Symbolic Logic*, 52(4):969–989, December 1987.

[Sta85] R. Statman. Logical relations and the typed λ-calculus. *Information and Control*, 65:85–97, 1985.

[Str89] Thomas Streicher. *Correctness and Completeness of a Categorical Semantics of the Calculus of Constructions.* PhD thesis, Universität Passau, Passau, West Germany, June 1989.

[Tai75] W. W. Tait. A realizability interpretation of the theory of species. In R. Parikh, editor, *Logic Colloquium*, pages 240–251. LNM 453, Springer Verlag, 1975.

[Tay86] Paul Taylor. *Recursive domains, indexed category theory and polymorphism.* PhD thesis, University of Cambridge, 1986.

[Tay89] Paul Taylor. Playing with LEGO: Some examples of developing mathematics in the Calculus of Constructions. Technical Report ECS-LFCS-89-89, LFCS, Edinburgh, 1989.