SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

An Analysis of Curiens Explicit Syntax for Dependent Type Theory

av

Philip Stassen

2020 - No M3

An Analysis of Curiens Explicit Syntax for Dependent Type Theory

Philip Stassen

Självständigt arbete i matematik 30 högskolepoäng, avancerad nivå

Handledare: Peter Lumsdaine, Guillaume Brunerie

2020

Abstract

Categorical models of dependent type theories have been extensively studied over the years. A difficulty that arises is the coherence between syntactical and semantical substitution. The on-the-nose equalities of the syntax collide with the up-to-isomorphism equalities of categories. We say a model is "strict" if this mismatch is resolved and that it is "weak" if the mismatch persists. Now there are several solutions to that problem, the most successful one being a method developed by Martin Hofmann in which the semantical substitution is "strictified".

We present two dependent type theories, one of which has an additional term constructor, the explicit coercion. Pierre-Louis Curien has proven that locally cartesian closed categories provide strict models for the latter syntax without having to strictify the semantic substitution – thereby resolving the mismatch by destrictifying the syntax instead. The type theory with explicit coercions can thus be used as intermediate step towards weak models for standard dependent type theories.

Curien's results show that weak models of the standard syntax are strict models of the explicit syntax. To benefit from that we want to understand in what sense the syntaxes are equivalent. Therefore, we show a strong such result by constructing a lifting and proving a soundness theorem. This function can then be used to prove coherence theorems and by that could serve as a tool to construct weak models for dependent type theories.

Acknowledgements

While writing my master thesis I benefited from the knowledge and guidance of my supervisors Peter LeFanu Lumsdaine and Guillaume Brunerie, who were always very generous with their time and energy. Their advise and insight in the material was of immense value for this paper.
I am furthermore very grateful for the patient explanations and the thorough discussions on different strategic approaches.

Also, I want to thank Peter and Guillaume as well as Marcello Garibbo for their comments and remarks on my draft.

Contents

1 Introduction

In 1983 Robert A. G. Seely described a new categorical interpretation of type theory [See84]. He observed that locally cartesian closed categories naturally resemble the kind of logic Martin-Löf type theories (such as [ML85]) are known

for. The result of this paper was a rather informal account of how to interpret ML type theory in a locally cartesian closed category.

Following this paper it became apparent that there are technicalities that cannot be resolved and because of that the interpretation is not well-defined. Among others Pierre Louis Curien found out that the combination of conversion rules, type equalities and substitutions leads to a mismatch between the syntax and the semantics ([Cur93]).

Substitution is associative, that means that for all appropriate types σ and substitutions (*context morphisms*) γ, δ it is the case that

$$\sigma[\gamma][\delta] = \sigma[\gamma \circ \delta].$$

However, in general the pullback functor is not a strict 2-functor, but only a pseudo functor. A *pseudo functor* does preserve morphism concatenation only up to isomorphism. For an arbitrary morphism σ and two pseudo functors γ, λ we have

$$\delta(\gamma(\sigma)) \simeq (\delta \circ \gamma)(\sigma)$$

As a consequence semantically, consecutive substitution will in general only

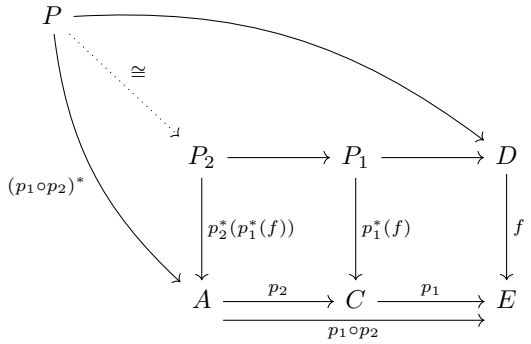

Figure 1: Pseudo functoriality of the pullback

be isomorphic to substituting the concatenated morphism. Syntactic substitution however is strict in that regard. There are two approaches to resolve this mismatch:

1. Strictify the model by imposing split fibrational hypotheses or
2. Destrictify the syntax – that means, that we interpret judgmental type equality not as equality but as isomorphism.

The first one has been a very successful approach throughout the years. Well studied strict models for type theories that satisfy split fibrational hypotheses are for example *categories with families*, *categories with attributes* or *contextual categories*. In [Hof95] Martin Hofmann proved coherence for a wide range of

models by developing a method how to turn any locally cartesian closed category into a category with families while preserving the logical structure of the category.

Following [CGH13] we distinguish three types of categories of structures

1. $\mathcal{S}\mathbb{ML}_{\mathbf{S}}$ is the category of strict structures and morphisms that strictly preserve the structure.

2. $\mathbb{ML}_{\mathbf{S}}$ is the category of non-strict structures with morphisms that strictly preserve the structure

3. $\mathbb{ML}$ is the category of non-strict structures and morphisms that preserve the structure up to isomorphism

The intuition behind this vague formulation should be that Hofmann's interpretation of the normal syntax lives in the $\mathcal{S}\mathbb{ML}_{\mathbf{S}}$ categories, while Curien's interpretation of the explicit syntax lives in the $\mathbb{ML}_{\mathbf{S}}$-category. The $\mathbb{ML}$ category is inhabited by structures such as locally cartesian closed categories or more generally *fibrations with comprehensions* (as used in [CGH13]). Note that $\mathcal{S}\mathbb{ML}_{\mathbf{S}}$ objects are also $\mathbb{ML}_{\mathbf{S}}$ objects and similarly $\mathbb{ML}_{\mathbf{S}}$ objects are also $\mathbb{ML}$ objects.

Both, the normal and the explicit syntax, have a classifying category, denoted $\mathsf{Synt} \in \mathcal{S}\mathbb{ML}_{\mathbf{S}}$, and $\mathsf{Synt}_e \in \mathbb{ML}_{\mathbf{S}}$ respectively[1]. These term models are initial in their respective category, that is, an interpretation function from the syntax into a structure also exhibits a morphism from Synt (resp. Synt_e) into that structure. Therefore, we may denote with $\mathcal{S}\mathbb{ML}_{\mathbf{S}}$ [Synt,$\mathfrak{m}$] an interpretation of the normal syntax into $\mathfrak{m}$, similarly we write $\mathbb{ML}_{\mathbf{S}}$ [Synt_e,$\mathfrak{m}_e$] for an interpretation of the explicit syntax in $\mathfrak{m}_e$.

Now let $\mathfrak{m}_e \in \mathbb{ML}$ be a locally cartesian closed category. Because of the pseudo functoriality of the pullback, a strict interpretation in $\mathbb{ML}_{\mathbf{S}}[\mathsf{Synt}, \mathfrak{m}_e]$ is not possible. Therefore, in [Cur93] Curien approached the problem differently and instead interpreted type equality as isomorphism. To achieve this, the author defines a function into a LCCC by induction on the derivations of judgments. To furthermore clarify how the raw syntax can be interpreted in such a category, he proves a coherence result that says that the interpretation of term expressions is independent from the choice of the derivation.

Theorem 1.1 (Coherence Curien)**.** *Any two proofs of a judgment* $\Gamma \vdash M : \sigma$ *receive equal interpretations.*

Curien's strategy to establish this coherence result is to use the explicit syntax (as a mediating syntax), which can be interpreted in a LCCC. This intermediate syntax reflects the usage of the conversion rule into the terms by using a new term former, the *explicit coercions*. Therefore, the author of [Cur93] defines an interpretation $[\![\ ]\!] \in \mathbb{ML}_{\mathbf{S}}[\mathsf{Synt}_e, \mathfrak{m}_e]$.

[1] As made precise in [CGH13, 5.3]

Then in a second step he uses rewriting techniques to relate derivations of this intermediate syntax with the ones of the original one and by that establishes the *coherence* for the latter one.

Following that the aim of this paper is twofold. On one hand we try to give a precise account of a rule system that has explicit coercions by formalizing it with the *Agda* proof assistant.

On the other hand, we try to establish results on how this theory relates to one that has not explicit coercions as part of its syntax. For that reason we construct both, a stripping and a lifting function on the raw syntax. While the stripping turns out to be a functor $|\ | \in \mathbb{ML}_{\mathbf{S}}[\mathsf{Synt}_e, \mathsf{Synt}]$ the situation for the lifting is less clear. Considering the coherence difficulties that cannot be circumvented, we cannot expect Synt_e to be a strict model of the original type theory. This means that a lift-functor $|\ |^{-1}$ can not be functional in a strict sense since it will not be well-defined as a function of sets (see fig. 2).

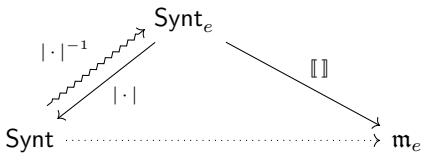

Figure 2: Relation of original and explicit theories and LCCC

Perspecitvely, the lift could be a tool to proof coherence theorems for a wide range of weak models of dependent type theory as we will briefly discuss in the last section.

Section 2 gives an exposition of a fairly standard type theory and a type theory whose syntax has explicit coercions. Suggestively we will refer to them with *original* respectively *explicit* type theory.

Eventually, in section 3 the stripping and a lifting are defined.

In contrast to the type theories discussed in [Cur93] and [CGH13] explicit substitutions are not used in this paper. Hence, in a sense we study a certain subsystem of the syntax that Curien employed. The explicit substitution can be added subsequently if this needed.

This paper is accompanied by a formalization. The code is available on my Github page:

`https://github.com/philippstassen/initiality/tree/develop1`

The files that are relevant for this paper are

common.agda, typetheory.agda, typetheorexplicit.agda, reflection.agda, reflectionexplicit.agda, syntx.agda, syntxexplicit.agda, rules.agda, rulesexplicit.agda, metatheorems.agda, translation.agda, reconstruction-approach2.agda.

The Agda code is based on Guillaume Brunerie's and Menno De Boer's code which can be found online at

`https://github.com/guillaumebrunerie/initiality`.

They are working on a formalized proof of the initiality conjecture. The type theory they use is the type theory that I refer to as original type theory. Therefore, I was fortunate to be able to use large parts of their work. The files common.agda, typetheory.agda, reflection.agda, syntx.agda and rules.agda are concerned with the original type theory and therefore did not need many modifications. The respective *explicit.agda files needed a significant makeover, but still rely strongly on the work of Brunerie and De Boer.

2 Syntax

We will refer to two different syntaxes: The syntax with explicit coercions and the one without, which I refer to as *explicit syntax* and *original* or *normal syntax* respectively.

The *meta theory* in this work will be *Agda*. I restrain of using Agda terminology in the description but formulate things in a more intuitive way. This implies that notation sometimes gets ambiguous. Whenever precision is required the correct precise notation will be used.

Throughout the paper two different notions of equality are used.

1. $=$ is the usual mathematical equality.
2. On the other hand, the *judgmental equality* $\Gamma \vdash \sigma \equiv \tau$ is a specific judgment for types, and $\Gamma \vdash M \equiv N : \sigma$ for terms respectively. Two terms or types being judgmentally equal means that there is a derivation of an respective judgment.

2.1 Variables in de Bruijn style notation

We use the de Bruijn style [Bru72] notation for types and terms. De Bruijn indices are genuinely less pleasant to read compared to named variables but behave better meta theoretically: The equality up to bound variables, also known as α-equality, coincides with the syntactic equality. Compare for instance

$$\lambda x.x =_\alpha \lambda y.y$$

Figure 3: Named variables

$$\lambda \mathbf{1} = \lambda \mathbf{1}$$

Figure 4: De Bruijn indices notation

Furthermore, de Bruijn indices are also to be preferred if one aims to formalize the theory in some programming language. They are easier and more straight forward to manipulate. This is why it is fairly standard to use de Bruijn indices in discussions on type theory.

A *variable* is a natural number, that refers back to the binding "environment" of the term, this includes binder as λ or Π, but also the types in the context. Having said this, one can think of the variable as some sort of "distance", the

larger the number, the earlier the variable must be bound (by a binder or in the context). To give an example, consider the judgments

$$\sigma \vdash \lambda.\, \mathbf{1}(\mathbf{2}) : \Pi_{\Pi_\sigma \tau} \tau$$
$$x{:}\sigma \vdash \lambda f.\, f(x) : \Pi_{\Pi_\sigma \tau} \tau$$

which is a function that applies a function to the variable x.

In the formalization de Bruijn indices are defined to be the elements of Fin_n, the *finite type with n elements*. Intuitively a finite type can be thought of as the subset of all naturals smaller then some particular natural. In our situation it is however advantageous to count backwards. Hence, we define Fin_0 to be empty and $\mathsf{Fin}_{\mathsf{suc}\, n}$ to be generated inductively by

$$\mathsf{last} \mid \mathsf{prev}\, k$$

where $k \in \mathsf{Fin}_n$. More precisely, for any $n \in \mathrm{N}$ the constructors of Fin_n are

$$\mathsf{last} : \mathsf{Fin}_{\mathsf{suc}\, n}$$
$$\mathsf{prev} \ : \mathsf{Fin}_n \to \mathsf{Fin}_{\mathsf{suc}\, n}$$

The advantage of this definition is that it anticipates the way contexts are extended. Extending the context requires that all variables are shifted to sustain their previous typing. This shifting of the variables will simply be the prev function.

This notation can be made a bit more appealing by defining

$$\boldsymbol{n+1} := \underbrace{\mathsf{prev} \ldots \mathsf{prev}}_{n\, times}\, \mathsf{last}$$

and thus in particular $\mathbf{1} = \mathsf{last}$. With this notation, the prev function becomes the successor function.

2.2 Standard Type Theory

The syntax presented in this chapter is fairly standard, but might throughout look a bit peculiar since we use de Bruijn indices instead of named variables. A more intuitive presentation of a comparable syntax can be found in [Hof97b].

The outline of this chapter is very straight forward and basically follows the formalization. First, Terms and Types are defined followed by raw contexts and context morphism. Thereafter, the notions of a judgment and a derivation are introduced such that the full theory can be presented.

2.2.1 Terms and Types

All type and term expressions are indexed by a natural number, the *scope*. Usually the scope of a term denotes the set of judgments that bind its free variables. This naming is analogous, the scope gives an upper bound to the

de Bruijn indices and thus ceils the number of variables that may occur in the term.

Therefore, it is useful to distinguish type and term expressions of a specific scope. We denote type expressions of scope n by $\mathtt{TyExpr}_n$ and term expressions respectively by $\mathtt{TmExpr}_n$. This distinction already comes into play when defining the type and term constructors, since it allows us to express elegantly that a term has more free variables than another. In the formalization this n is handled as an implicit argument that Agda can always solve by unification. Hence, we are free to omit the scope for brevity and to think of it as quantified over the naturals, whenever it shows up unbounded.

The raw *type expressions* are built up inductively for all n by the constructors

$$\begin{aligned}
&\mathcal{U} : \mathtt{TyExpr}_n \\
&\mathsf{el} : \mathtt{TmExpr}_n \to \mathtt{TyExpr}_n \\
&\Pi : \mathtt{TyExpr}_n \to \mathtt{TyExpr}_{\mathsf{suc}\, n} \to \mathtt{TyExpr}_n.
\end{aligned}$$

Similarly, the raw *term expressions* are generated inductively by

$$\begin{aligned}
&\boldsymbol{n} : \mathtt{TmExpr}_n \quad \text{with } \boldsymbol{n} \in \mathsf{Fin}_n \\
&\mathsf{lam} : \mathtt{TyExpr}_n \to \mathtt{TyExpr}_{\mathsf{suc}\, n} \to \mathtt{TmExpr}_{\mathsf{suc}\, n} \to \mathtt{TmExpr}_n \\
&\mathsf{app} : \mathtt{TyExpr}_n \to \mathtt{TyExpr}_{\mathsf{suc}\, n} \to \mathtt{TmExpr}_n \to \mathtt{TmExpr}_n \to \mathtt{TmExpr}_n.
\end{aligned}$$

Observe how this constructors contain more information then the usual recursive definitions of raw syntax

$$\begin{aligned}
&\sigma = \mathcal{U} \mid \mathsf{el}\, M \mid \Pi_\sigma \tau \quad \text{and} \\
&M = \boldsymbol{n} \mid \mathsf{lam}\, \sigma\tau.\, M \mid \mathsf{app}_{\sigma\tau} M N
\end{aligned}$$

due to the requirements we impose on the free variables via the scope.

Having clarified this we now switch to a more familiar syntax which omits some information. This is only for convenience and we will turn back to this syntax at some point (in section 3.2).

$$\begin{aligned}
&\lambda\sigma.\, M := \mathsf{lam}\, \sigma\tau.\, M \\
&fN := \mathsf{app}_{\sigma\tau} f N
\end{aligned}$$

2.2.2 Raw Contexts and Raw Context Morphisms

A raw context is list of types, a raw context morphism is a list of terms. Recall the standard definition of a raw context

$$\Gamma = (x_1{:}\sigma_1, x_2{:}\sigma_2, \ldots, x_n{:}\sigma_n) \tag{1}$$

which is list of typed variables. In our situation, we do not name the variables, but simply list the types.

Definition 2.1. A *raw-context* is a list of types

$$\sigma_0, \sigma_1, \ldots, \sigma_n \tag{2}$$

such that the scope of σ_i is equal to i for all $0 \leq i \leq n$.

Now by construction every σ_i may have variables in its preceding types. The set of raw contexts of length n is denoted by $\mathtt{Ctx}_n$. Analogously we define a raw context morphism.

Definition 2.2. A *raw (n,m)-context morphism* is a list of terms of scope n that has length m.

$$M_0, M_1, \ldots, M_m \tag{3}$$

such that the scope of M_i is equal to n for all $0 \leq i \leq m$.

$\mathtt{Mor}_{n,m}$ is the set of all raw (n, m)-context morphisms

Remark 2.3. It is conventional to write $(\Gamma_1, \ldots, \Gamma_m)$ instead of Γ. Conversely, if Γ is a context, then Γ_i denotes the respective type at the ith position. Similarly for a context morphism δ and $(\delta_1, \ldots, \delta_m)$.

Throughout the paper contexts will always mean raw contexts and similarly context morphisms will always be raw (n, m)-context morphisms.

2.2.3 Judgments

There are four kinds of judgments:

$$\Gamma \vdash \sigma \tag{4}$$
$$\Gamma \vdash M : \sigma \tag{5}$$
$$\Gamma \vdash \sigma \equiv \tau \tag{6}$$
$$\Gamma \vdash M \equiv N : \sigma. \tag{7}$$

We require context types and terms to have the same scopes. Judgments should be considered hypothetical as this is standard for intuitionistic type theory and not be confused with the derivation of a judgment.

Definition 2.4. A *derivation* of a judgment is inductively generated by the rules of the type theory. If we have derivations $\mathcal{D}_1, \ldots, \mathcal{D}_n$ of judgments $\mathcal{J}_1, \ldots, \mathcal{J}_n$ and a rule $\mathcal{R}$ such that

$$\frac{\mathcal{J}_1 \qquad \ldots \qquad \mathcal{J}_n}{\mathcal{J}}\,\mathcal{R}$$

Then we write that $\mathcal{R}\{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ is a derivation of $\mathcal{J}$.

The set of all derivations for $\mathcal{J}$ is denoted by $\mathtt{Derivation}(\mathcal{J})$. For brevity, we may omit the judgment when discussing derivations.

2.2.4 De Bruijn Index Shifting

In type theory all expressions are depending on their *environment*, that is the context plus the variable binders. Since we are using a name free presentation of type theory, already enlarging the context requires modifications in the expressions. For intuition, compare an instance of the `Weakening` rule for a theory with and without names

$$\frac{\Gamma \vdash x : \sigma \qquad \Gamma \vdash \tau}{\Gamma, y{:}\tau \vdash x : \sigma}\ \texttt{Weak1} \qquad \text{Figure 5: Named variables}$$

$$\frac{\Gamma \vdash \mathbf{1} : \sigma \qquad \Gamma \vdash \tau}{\Gamma, \tau \vdash \mathbf{2} : \sigma}\ \texttt{Weak2} \qquad \text{Figure 6: Name free variables}$$

The first rule uses names to denote variables, and thus we may extend our context while preserving the derivability of all judgments. This does not hold true if we use a name free presentation of the theory. In this case, we refer to the context by a de Bruijn index. If we extend the context, this de Bruijn index refers to a different type. Indeed, observe in the situation of fig. 6, the judgment $\Gamma, \tau \vdash \mathbf{1} : \sigma$ would have been a malformed judgment, since it is clearly the case that $\Gamma, \tau \vdash \mathbf{1} : \tau$.

To formulate such a rule, we need an operation to shift the de Bruijn indices in the raw syntax. Therefore we define $\uparrow$. Formally, this operation should be mutually defined for *types* and *terms* and for all n.

$$\begin{aligned}
&\uparrow\colon \texttt{TmExpr}_n \to \texttt{TmExpr}_{n+1} \\
&\quad \boldsymbol{n}\uparrow = \boldsymbol{n+1} \\
&\quad (\lambda\sigma.\, M)\uparrow = \lambda\sigma\uparrow.\, M\uparrow_2 \\
&\quad (MN)\uparrow = M\uparrow\, N\uparrow
\end{aligned}
\qquad
\begin{aligned}
&\uparrow\colon \texttt{TyExpr}_n \to \texttt{TyExpr}_{n+1} \\
&\quad \mathcal{U}\uparrow = \mathcal{U} \\
&\quad (\mathsf{el}\, M)\uparrow = \mathsf{el}\, (M\uparrow) \\
&\quad (\Pi_\sigma \tau)\uparrow = \Pi_{\sigma\uparrow}(\tau\uparrow_2)
\end{aligned}$$

Similarly as for the proof of weakening in the standard syntax, during the recursive calls of $\lambda\sigma.\, M$ and $\Pi_\sigma\tau$ we do not want to change the variable that is bound by the constructor. Hence, we have to shift every de Bruijn index larger or equal than $\mathbf{2}$, but not the first one – we denote this operation by $\uparrow_2$. This definition is therefore not sufficient, we need to define a more general version of the shifting that allows to shift the indices after an arbitrary position.

2.2.5 Generalized Index Shifting

In the previous chapter we used a special case of the weakening rule to motivate the de Bruijn index shifting. In full generality, the weakening rule is formulated as

$$\frac{\Gamma, \Delta \vdash M : \sigma \qquad \Gamma \vdash \tau}{\Gamma, x{:}\tau, \Delta \vdash M : \sigma}\ \texttt{Weak}$$

where Γ and Δ are of arbitrary length. In this situation we only need to weaken the variables that previously referred to Γ. However, we need to weaken them everywhere, including in Δ. Therefore, we need a third operation that shifts indices in raw contexts. In variable free syntax a similar general weakening rule should be phrased as follows: If Γ, Δ is a context – where Δ has length k, the weakening before before the $\boldsymbol{k}$th type (counting backwards) is

$$\frac{\Gamma, \Delta \vdash M : \sigma \qquad \Gamma \vdash \tau}{\Gamma, \tau, \Delta{\uparrow_k} \ \vdash M{\uparrow_k} : \sigma{\uparrow_k}} \ \mathtt{Weak}$$

The *generalized de Bruijn index* shifting is a mutual inductive definition for types, terms and contexts:

$$\begin{aligned}
\uparrow : \mathsf{Fin}_n \to \mathtt{TmExpr}_n \to \mathtt{TmExpr}_{n+1} \qquad & \uparrow : \mathsf{Fin}_n \to \mathtt{TyExpr}_n \to \mathtt{TyExpr}_{n+1} \\
\boldsymbol{i}{\uparrow_k} &:= \boldsymbol{i} + \mathbf{1} \quad \text{if } \boldsymbol{i} > k & \mathcal{U}{\uparrow_k} &:= \mathcal{U} \\
\boldsymbol{i}{\uparrow_k} &:= \boldsymbol{i} \quad \text{if } \boldsymbol{i} \leq k & (\mathsf{el}\, M){\uparrow_k} &:= \mathsf{el}\, (M{\uparrow_k}) \\
(\lambda\sigma.\, M){\uparrow_k} &:= \lambda\sigma{\uparrow_k}\, .\, M{\uparrow_{k+1}} & (\Pi_\sigma \tau){\uparrow_k} &:= \Pi_{\sigma{\uparrow_k}}\, \tau{\uparrow_{k+1}} \\
(MN){\uparrow_k} &:= M{\uparrow_k}\ N{\uparrow_k}
\end{aligned}$$

$$\begin{aligned}
\uparrow : \mathsf{Fin}_n \to \mathtt{TyExpr} \to \mathtt{Ctx}_n \to \mathtt{Ctx}_{n+1} \\
\Gamma{\uparrow_1^\tau} &:= (\Gamma, \tau) \\
(\Gamma, \sigma){\uparrow_{k+1}^\tau} &:= (\Gamma{\uparrow_k^\tau}, \tau{\uparrow_k})
\end{aligned}$$

The weakening of a contexts needs an additional type expression as argument which is meant to be analogous to the notation Γ, τ, Δ. It is noteworthy that usually the only requirements on τ is that it has the correct scope to fit the place it sits in. Therefore, it is included in the notation but barely mentioned in the hypotheses of a theorem. A more precise account of these operations is formalized in syntx.agda with the respective declarations weakenTy, weakenTm, weakenCtx.

2.2.6 Generalized Substitution

In general we have a rule saying that substituting a *well-typed* term for a free variable does preserve the validity of the judgment

$$\frac{\Gamma, x : \sigma \vdash y : \tau \qquad \Gamma \vdash M : \sigma}{\Gamma \vdash y[M/x] : \tau[M/x]} \ \mathtt{Subst}$$

Therefore, we need to define an operation $\cdot[\cdot]$ that substitutes terms for free variables in the raw syntax in such a way that the `Subst`-rule will be admissible. This is better to phrase working in a variable free syntax, since we do not have to worry what happens if we substitute a variable that is already listed in the context.

As already the index shifting, substitution will be defined by recursion on the raw syntax, and therefore we should expect to run into the same subtleties for the cases λ and Π as before. To cope with that, we need an auxiliary context morphism whose duty is to protect the first variable from being substituted (as it is bound by one of the binders).

For this purpose we define the *morphism extension* such that if $\Gamma \vdash \delta : \Delta$ and $\Delta \vdash \sigma$ then also $\Gamma, \sigma \vdash \delta^+ : \Delta, \sigma$.

$$\begin{aligned} {}^+ &: \mathtt{Mor}_{n,m} \to \mathtt{Mor}_{n+1,m+1} \\ \delta^+ &:= \delta\uparrow, \mathbf{1} \end{aligned}$$

Having addressed this issue we are now in a position to define the *generalized substitution* using the context morphisms. If δ denotes such a morphism, then we define

$$\begin{aligned} [\,] &: \mathtt{TmExpr}_n \to \mathtt{Mor}_{n,m} \to \mathtt{TmExpr}_m \\ \boldsymbol{k}[\delta] &:= \delta_{m+1-k} \\ (\lambda\sigma.\, M)[\delta] &:= \lambda\sigma[\delta].\, M[\delta^+] \\ (MM_2)[\delta] &:= M[\delta]M_1[\delta] \end{aligned}$$

$$\begin{aligned} [\,] &: \mathtt{TyExpr}_n \to \mathtt{Mor}_{n,m} \to \mathtt{TyExpr}_m \\ \mathcal{U}[\delta] &:= \mathcal{U} \\ (\mathsf{el}\, M)[\delta] &:= \mathsf{el}\,(M[\delta]) \\ (\Pi_\sigma \tau)[\delta] &:= \Pi_{\sigma[\delta]}(\tau[\delta^+]) \end{aligned}$$

Note that if $\lambda\sigma.\, M$ is in $\mathtt{TmExpr}_n$, then by construction M is in $\mathtt{TmExpr}_{n+1}$. Therefore, $M[\delta]$ would not have been well defined.

Remark 2.5. The generalized substitution subsumes *substitution* if we choose $\delta = (\boldsymbol{n}, \boldsymbol{n-1}, \ldots, \mathbf{2}, M)$ to be the morphism. Therefore, we will use the same notation for either substitution. Due to the different choice of variables it should be unambiguous which substitution is used.

$$\begin{aligned} [\,] &: \mathtt{TmExpr}_{n+1} \to \mathtt{TmExpr}_n \to \mathtt{TmExpr}_n \\ M[N] &= M[\mathsf{id}_n, N] \end{aligned} \qquad \begin{aligned} [\,] &: \mathtt{TyExpr}_{n+1} \to \mathtt{TmExpr}_n \to \mathtt{TyExpr}_n \\ \sigma[N] &= \sigma[\mathsf{id}_n, N] \end{aligned}$$

2.2.7 Structural rules

As already mentioned, we are particularly interested in derivations of judgments. We begin with presenting the structural rules that are not type or term specific.

$$\frac{\Gamma \vdash \sigma}{\Gamma, \sigma \vdash \mathbf{1} : \sigma}\ \mathtt{Var} \qquad \frac{\Gamma \vdash \sigma \qquad \Gamma \vdash \boldsymbol{k} : \sigma}{\Gamma, \tau \vdash \boldsymbol{k}\uparrow : \sigma}\ \mathtt{Var}\uparrow \qquad \frac{\Gamma \vdash \sigma}{\Gamma \vdash_e \sigma \cong \sigma}\ \mathtt{TyRefl}$$

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash_e M \equiv M : \sigma}\ \texttt{TmRefl} \qquad \frac{\Gamma \vdash \sigma \equiv \tau}{\Gamma \vdash \tau \equiv \sigma}\ \texttt{TySymm} \qquad \frac{\Gamma \vdash M \equiv N : \sigma}{\Gamma \vdash N \equiv M : \sigma}\ \texttt{TmSymm}$$

$$\frac{\Gamma \vdash \sigma_1 \qquad \Gamma \vdash \sigma_0 \equiv \sigma_1 \qquad \Gamma \vdash \sigma_1 \equiv \sigma_2}{\Gamma \vdash \sigma_0 \equiv \sigma_2}\ \texttt{TyTran}$$

$$\frac{\Gamma \vdash M_1 : \sigma \qquad \Gamma \vdash M_0 \equiv M_1 : \sigma \qquad \Gamma \vdash M_1 \equiv M_2 : \sigma}{\Gamma \vdash M_0 \equiv M_2 : \sigma}\ \texttt{TmTrans}$$

$$\frac{\Gamma \vdash \sigma \qquad \Gamma \vdash M : \sigma \qquad \Gamma \vdash \sigma \equiv \tau}{\Gamma \vdash M : \tau}\ \texttt{Conv}$$

$$\frac{\Gamma \vdash \sigma \qquad \Gamma \vdash M \equiv N : \sigma \qquad \Gamma \vdash \sigma \equiv \tau}{\Gamma \vdash M \equiv N : \tau}\ \texttt{Conv}\equiv$$

Remark 2.6. Something that might draw attention is the fact that `Var` does not require a derivation $\vdash \Gamma$ as presupposition. Consequently, we will not be able to prove that from $\Gamma \vdash \sigma$ also $\vdash \Gamma$ is derivable. The merit of this variation is that the definition is less recursive, which is beneficial in many situations. In the formalization the rules `TyRefl` and `TmRefl` are not included but instead reflexivity rules only for variables. This turns out to be sufficient to derive the full reflexivity rules.

2.2.8 Universe

The type theory gets immediately more complicated when there are non trivial type families. This requires a base type and a family of types over this base type. The simplest instance of a combination like this is a universe $\mathcal{U}$ together with an eliminator el.

$$\frac{}{\Gamma \vdash \mathcal{U}}\ \mathcal{U}\texttt{-Form} \qquad \frac{}{\Gamma \vdash \mathcal{U} \equiv \mathcal{U}}\ \mathcal{U}\equiv \qquad \frac{\Gamma \vdash M : \mathcal{U}}{\Gamma \vdash \mathsf{el}\, M}\ \texttt{el}$$

$$\frac{\Gamma \vdash M \equiv N : \mathcal{U}}{\Gamma \vdash \mathsf{el}\,(M) \equiv \mathsf{el}\,(M)}\ \texttt{el}\equiv$$

2.2.9 Pi-Types

The generalized function types $\Pi_\sigma \tau$ map σ to the type family τ, which ranges over σ.

$$\frac{\Gamma \vdash \sigma \qquad \Gamma, \sigma \vdash \tau}{\Gamma \vdash \Pi_\sigma \tau}\ \Pi\texttt{-Form} \qquad \frac{\Gamma \vdash \sigma \qquad \Gamma \vdash \sigma \equiv \sigma' \qquad (\Gamma, \sigma) \vdash \tau \equiv \tau'}{\Gamma \vdash \Pi_\sigma \tau \equiv \Pi_{\sigma'} \tau'}\ \Pi\equiv$$

$$\frac{\Gamma \vdash \sigma \qquad \Gamma, \sigma \vdash \tau \qquad \Gamma, \sigma \vdash M : \tau}{\Gamma \vdash \lambda\sigma.\, M : \Pi_\sigma \tau}\ \Pi\text{-}\mathtt{Intro}$$

$$\frac{\Gamma \vdash \sigma \qquad \Gamma, \sigma \vdash \tau \qquad \Gamma \vdash f : \Pi_\sigma \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash fN : \tau[N]}\ \Pi\mathtt{Elim}$$

$$\frac{\Gamma \vdash \sigma \qquad \Gamma \vdash \sigma \equiv \sigma' \qquad \Gamma, \sigma \vdash \tau \equiv \tau' \qquad \Gamma, \sigma \vdash M \equiv M :' \tau}{\Gamma \vdash \lambda\sigma.\, M \equiv \lambda\sigma'.\, M' : \Pi_\sigma \tau}\ \lambda\equiv$$

$$\frac{\begin{array}{c}\Gamma \vdash \sigma \qquad \Gamma \vdash \sigma \equiv \sigma' \qquad \Gamma, \sigma \vdash \tau \equiv \tau' \\ \Gamma \vdash f \equiv f' : \Pi_\sigma \tau \qquad \Gamma \vdash N \equiv N' : \sigma\end{array}}{\Gamma \vdash fN \equiv f'N' : \tau[N]}\ \mathtt{App}\equiv$$

$$\frac{\Gamma \vdash \sigma \qquad \Gamma, \sigma \vdash \tau \qquad \Gamma, \sigma \vdash M : \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash (\lambda\sigma.\, M)N \equiv M[N] : \tau[N]}\ \beta\equiv$$

$$\frac{\Gamma \vdash \sigma \qquad \Gamma, \sigma \vdash \tau \qquad \Gamma \vdash f : \Pi_\sigma \tau}{\Gamma \vdash f \equiv \lambda\sigma.\, ((f\uparrow)\mathbf{1}) : \Pi_\sigma \tau}\ \eta\equiv$$

2.2.10 Derivability of Contexts and Context Morphisms

A context is derivable if

$$\frac{}{\vdash \diamond}\ \mathtt{Ctx} \qquad\qquad \frac{\vdash \Gamma \qquad \Gamma \vdash \sigma}{\vdash (\Gamma, \sigma)}\ \mathtt{Ctx}.$$

On the other hand, recall the standard definition of a context morphism. Let Γ and $\Delta = (x_1{:}\sigma_1, \ldots, x_m{:}\sigma_m)$ be contexts. If $\delta = (M_1, \ldots, M_m)$ we write $\Gamma \vdash \delta : \Delta$ and say that δ derivable if the following judgments are valid:

$$\Gamma \vdash M_1 : \sigma_1 \tag{8}$$
$$\Gamma \vdash M_2 : \sigma_2[M_1/x] \tag{9}$$
$$\ldots \tag{10}$$
$$\Gamma \vdash M_m : \sigma_m[M_1/x_1, \ldots, M_m/x_m] \tag{11}$$

We define the variable free version of a context morphism analogously. Let $\Gamma = (\tau_1, \ldots, \tau_n)$ and $\Delta = (\sigma_1, \ldots, \sigma_m)$ be contexts. Furthermore, let $\delta = (M_1, \ldots, M_m)$ be a context morphism. Then $\Gamma \vdash \delta : \Delta$ is derivable if the following judgments hold:

$$\begin{array}{l}\Gamma \vdash M_1 : \sigma_1 \\ \Gamma \vdash M_2 : \sigma_2[M_1] \\ \qquad \ldots \\ \Gamma \vdash M_m : \sigma_m[M_1, \ldots, M_m]\end{array} \tag{12}$$

or more formally we may say that derivability of context morphism is inductively generated by the rules

$$\frac{}{\Gamma \vdash () : \diamond}\ \texttt{Mor} \qquad\qquad \frac{\Gamma \vdash \delta : \Delta \qquad \Gamma \vdash M : \sigma[\delta]}{\Gamma \vdash (\delta, M) : (\Delta, \sigma)}\ \texttt{Mor}$$

The reader may convince herself that for context morphism derived by these rules the judgments of (12) hold.

In a similar fashion we can make precise what it means for two contexts to be judgmentally equal. Derivations of such judgments are generated by the rules

$$\frac{}{\vdash \diamond \equiv \diamond}\ \texttt{Ctx=} \qquad\qquad \frac{\vdash \Gamma \equiv \Gamma' \qquad \Gamma \vdash \sigma \qquad \Gamma' \vdash \tau \qquad \Gamma \vdash \sigma \equiv \tau}{\vdash (\Gamma, \sigma) \equiv (\Gamma', \tau)}\ \texttt{Ctx=}$$

Intuitively this is exactly what we would expect. Two contexts are judgmentally equal if all the types listed in them are. Similarly, we define what it means for two context morphisms to be judgmentally equal. Again, our intuition suggests that a context morphism should be called judgmentally equal if they are as lists of terms. A more precise account of that is to define the derivation of context morphisms inductively by the rules

$$\frac{}{\Gamma \vdash () \equiv () : \diamond}\ \texttt{Mor=} \qquad\qquad \frac{\Gamma \vdash \delta \equiv \delta' : \Delta \qquad \Gamma \vdash M \equiv M' : \sigma[\delta]}{\Gamma \vdash (\delta, M) \equiv (\delta', M') : (\Delta, \sigma)}\ \texttt{Mor=}$$

2.2.11 Admissible rules

There are many important meta theoretic results that can be derived in this theory. A small subset which is needed for the main theorems of this paper are formalized in the file rules.agda. I would like to emphasize the following, very essential results. On one hand, these results are needed throughout the paper. On the other hand, it is insightful to see how these admissible rules compare to similar ones for the explicit rule system. First to mention are the weakening rules

$$\frac{\Gamma \vdash \sigma}{\Gamma\!\uparrow^{\tau}_{k} \vdash \sigma\!\uparrow_{k}}\ \texttt{WeakTy}\,. \qquad\qquad \frac{\Gamma \vdash M : \sigma}{\Gamma\!\uparrow^{\tau}_{k} \vdash M\!\uparrow_{k} : \sigma\!\uparrow_{k}}\ \texttt{WeakTm}\,.$$

$$\frac{\Gamma \vdash \sigma_1 \equiv \sigma_2}{\Gamma\!\uparrow^{\tau}_{k} \vdash \sigma_1\!\uparrow_{k} \equiv \sigma_2\!\uparrow_{k}}\ \texttt{WeakTyEq}$$

$$\frac{\Gamma \vdash M \equiv N : \sigma}{\Gamma\!\uparrow^{\tau}_{k} \vdash M\!\uparrow_{k} \equiv N\!\uparrow_{k} : \sigma\!\uparrow_{k}}\ \texttt{WeakTyEq}$$

Of similar relevance are the rules that are concerned with substitution. The most basic ones are:

$$\frac{\Delta \vdash \sigma \qquad \Gamma \vdash \delta : \Delta}{\Gamma \vdash \sigma[\delta]}\ \texttt{SubstTy} \qquad \frac{\Delta \vdash M : \sigma \qquad \Gamma \vdash \delta : \Delta}{\Gamma \vdash M[\delta] : \sigma[\delta]}\ \texttt{SubstTm}$$

$$\frac{\Delta \vdash \sigma \equiv \tau \qquad \Gamma \vdash \delta : \Delta}{\Gamma \vdash \sigma[\delta] \equiv \tau[\delta]}\ \texttt{SubstTyEq}$$

$$\frac{\Delta \vdash M \equiv N : \sigma \qquad \Gamma \vdash \delta : \Delta}{\Gamma \vdash M[\delta] \equiv N[\delta] : \sigma[\delta]}\ \texttt{SubstTmEq}$$

$$\frac{\Delta \vdash \sigma \qquad \Gamma \vdash \delta : \Delta \qquad \Gamma \vdash \delta \equiv \theta : \Delta}{\Gamma \vdash \sigma[\delta] \equiv \sigma[\theta]}\ \texttt{SubstTyMorEq}$$

$$\frac{\Delta \vdash M : \sigma \qquad \Gamma \vdash \delta : \Delta \qquad \Gamma \vdash \delta \equiv \theta : \Delta}{\Gamma \vdash M[\delta] \equiv M[\theta] : \sigma[\delta]}\ \texttt{SubstTmMorEq}$$

Another collection of useful functions are the context conversion lemmas ConvTy, ConvTm, ConvTyEq and ConvTmEq.

$$\frac{\Gamma \vdash \sigma \qquad \vdash \Gamma \equiv \Gamma'}{\Gamma' \vdash \sigma}\ \texttt{ConvTy} \qquad \frac{\Gamma \vdash M : \sigma \qquad \vdash \Gamma \equiv \Gamma'}{\Gamma' \vdash M : \sigma}\ \texttt{ConvTm}$$

$$\frac{\Gamma \vdash \sigma \equiv \tau \qquad \vdash \Gamma \equiv \Gamma'}{\Gamma' \vdash \sigma \equiv \tau}\ \texttt{ConvTyEq}$$

$$\frac{\Gamma \vdash M \equiv N : \sigma \qquad \vdash \Gamma \equiv \Gamma'}{\Gamma' \vdash M \equiv N : \sigma}\ \texttt{ConvTmEq}$$

A desired feature of the theory is that all presuppositions of a judgment are derivable. The *presuppositions* of a judgment $\mathcal{J}$ are all judgments that should be satisfied for $\mathcal{J}$ to be derivable. The intuition behind that is clear, for example if $\Gamma \vdash \sigma \equiv \tau$ is derivable, then we expect that also $\Gamma \vdash \sigma$ as well as $\Gamma \vdash \tau$ are valid.

Notice, that the rules themselves do not directly require all presuppositions. Take $\Pi\equiv$ for example, to derive $\Gamma \vdash \Pi_\sigma \tau \equiv \Pi_{\sigma_1} \tau_1$ we do not have to explicitly include derivations of $\Gamma \vdash \Pi_\sigma \tau$ or $\Gamma \vdash \Pi_{\sigma_1} \tau_1$.

This practice is justified by the fact that we are able to proof several lemmas – TmTy, TyEqTy1, TyEqTy2, TmEqTm1, TmEqTm2 – which witness the fact that for any derivable judgment all presuppositions are derivable.

2.3 Explicit Syntax

The dependent type theory with explicit coercions has been mentioned in [Cur93] for the first time. During the process of defining the interpretation of dependent type theory in a locally cartesian closed category Curien encountered difficulties to prove certain coherence theorems. The explicit coercion is an ad hoc modification of the syntax with regard to the goal of establishing coherence.

The crucial observation about this syntax is the following. We replace the `Conv` rule by some other rules that reflect the equality information into the term via an *explicit coercion*. We achieve that by introducing another term former, the coercion $\mathsf{c}_{\sigma,\tau}M$ of a term. Moreover, modify the rules appropriately.

Similar as in [CGH13] we use $\vdash_e$ instead of $\vdash$ to distinguish the judgments of the explicit syntax from their normal counterparts. Also it is suggestive to use $\cong$ instead of $\equiv$ in $\Gamma \vdash_e \sigma \cong \tau$. This anticipates the way type equality will be interpreted, namely as isomorphism. I restrain to further indicate the explicit syntax as such since it would overload the notation. Hence, the reader has to read off from the context whether an expression belongs to the original or the explicit syntax.

2.3.1 Raw Syntax

In this chapter we will examine this modified syntax. To avoid being too repetitive, the description will be brief and the focus more on the differences that matter.

The constructors of the raw type expressions remain the same as in the original syntax

$$\sigma = \mathcal{U} \mid \mathsf{el}\, M \mid \Pi_\sigma \tau. \tag{13}$$

However, we add another term constructor to the raw term expressions

$$M = \boldsymbol{n} \mid \lambda\sigma.\, M \mid MN \mid \mathsf{c}_{\sigma,\tau}M \tag{14}$$

More precisely, we need to specify the implicit information that we suppress: In addition to the conditions of section 2.2.1 we require another one specifically for the `coerc` case: If σ, τ and M have scope n, then so has $\mathsf{c}_{\sigma,\tau}M$.

Remark 2.7. Observe that although the constructors of the type expression have not changed, the set of type expression has. Type and Terms are mutually recursive and therefore the new termformer also influences the type expressions.

We will not restate the definitions of raw contexts and raw (n, m)-context morphisms, since they remain the same. Moreover, the de Bruijn index shifting and syntactic subsitution operation only need to be extended with a case for the new term constructor – compare the definitions weakenTy and weakenTm in the files syntx.agda respecitvely syntxexplicit.agda.

2.3.2 Structural rules

The purpose of this modified syntax is to replace the conversion rule `Conv` with the rule `Coerc` such that derivations become unique up to the derivations of judgmental equalities. Although this seems like a minor altercation of the theory, in fact the makeover is much more drastic. There are many meta theoretic subtleties to consider, which is why for some of the rules more presuppositions

are required. In addition to that the `TmRefl` and `TyRefl` rule are not admissible anymore – at least not as easily as for the original syntax.

$$\frac{\Gamma \vdash_e \sigma}{\Gamma, \sigma \vdash_e \mathbf{1} : \sigma}\ \texttt{Var} \qquad \frac{\Gamma \vdash_e \sigma \qquad \Gamma \vdash_e \boldsymbol{k} : \sigma}{\Gamma, \tau \vdash_e \boldsymbol{k}\uparrow : \sigma}\ \texttt{Var}\uparrow \qquad \frac{\Gamma \vdash_e \sigma}{\Gamma \vdash_e \sigma \cong \sigma}\ \texttt{TyRefl}$$

$$\frac{\Gamma \vdash_e M : \sigma}{\Gamma \vdash_e M \equiv M : \sigma}\ \texttt{TmRefl} \qquad \frac{\Gamma \vdash_e M \equiv N : \sigma}{\Gamma \vdash_e N \equiv M : \sigma}\ \texttt{TmSymm} \qquad \frac{\Gamma \vdash_e \sigma \cong \tau}{\Gamma \vdash_e \tau \cong \sigma}\ \texttt{TySymm}$$

$$\frac{\Gamma \vdash_e \sigma_1 \qquad \Gamma \vdash_e \sigma_0 \cong \sigma_1 \qquad \Gamma \vdash_e \sigma_1 \cong \sigma_2}{\Gamma \vdash_e \sigma_0 \cong \sigma_2}\ \texttt{TyTran}$$

$$\frac{\Gamma \vdash_e M_1 : \sigma \qquad \Gamma \vdash_e M_0 \equiv M_1 : \sigma \qquad \Gamma \vdash_e M_1 \equiv M_2 : \sigma}{\Gamma \vdash_e M_0 \equiv M_2 : \sigma}\ \texttt{TmTrans}$$

$$\frac{\Gamma \vdash_e \sigma \qquad \Gamma \vdash_e \tau \qquad \Gamma \vdash_e M : \sigma \qquad \Gamma \vdash_e \sigma \cong \tau}{\Gamma \vdash_e \mathsf{c}_{\sigma,\tau} M : \tau}\ \texttt{Coerc}$$

$$\frac{\Gamma \vdash_e \sigma \qquad \Gamma \vdash_e M \equiv N : \sigma \qquad \Gamma \vdash_e \sigma \cong \tau}{\Gamma \vdash_e \mathsf{c}_{\sigma,\tau} M \equiv \mathsf{c}_{\sigma,\tau} N : \tau}\ \texttt{Coerc}\equiv$$

$$\frac{\Gamma \vdash_e M : \sigma}{\Gamma \vdash_e \mathsf{c}_{\sigma,\sigma} M \equiv M : \sigma}\ \texttt{CoercRefl}$$

$$\frac{\Gamma \vdash_e \sigma_0 \qquad \Gamma \vdash_e \sigma_1 \qquad \Gamma \vdash_e \sigma_2 \qquad \Gamma \vdash_e M : \sigma_0 \qquad \Gamma \vdash_e \sigma_0 \cong \sigma_1 \qquad \Gamma \vdash_e \sigma_1 \cong \sigma_2}{\Gamma \vdash_e \mathsf{c}_{\sigma_1,\sigma_2}(\mathsf{c}_{\sigma_0,\sigma_1} M) \equiv \mathsf{c}_{\sigma_0,\sigma_2} M : \sigma_2}\ \texttt{CoercTrans}$$

2.3.3 Syntax Traces of Context Equalities

Beside the new conversion rules some congruence rules have also to be revised. Recall that the equality rules Π`Elim`, Π≡, λ≡ and `App`≡ rules require derivations of

$$\Gamma, \sigma' \vdash \tau' \quad \text{and}$$
$$\Gamma, \sigma \vdash \tau \equiv \tau'$$

as presuppositions. However, in the explicit syntax an equality like $\Gamma, \sigma \vdash \tau \equiv \tau'$ requires that both $\Gamma, \sigma \vdash \tau$ and $\Gamma, \sigma \vdash \tau'$ which can only be the case if $\sigma' = \sigma$ all along.
Hence in general, if $\Gamma \vdash \sigma \equiv \sigma'$ and $\Gamma, \sigma' \vdash \tau'$ then $\Gamma, \sigma \vdash \tau'$ is not well-typed. This contrasts the situation in the original type theory that has the `Conv` rule.

More generally, the admissible rules `ConvTy`, `ConvTm`, `ConvTyEq` and `ConvTmEq` from section 2.2.11 will not be provable for the explicit theory, if they are formulated like that.

Remark 2.8. In [Cur93], the author needs to further modify the syntax to cope with this issue. He first includes a new type former, the *context coercion*

$$\mathsf{cCtx}_{\Gamma,\Gamma'}\sigma,$$

and then adds a modification of the `ConvTy` rule to the theory.

$$\frac{\vdash_e \Gamma \equiv \Gamma' \qquad \Gamma \vdash \sigma}{\Gamma' \vdash \mathsf{cCtx}_{\Gamma,\Gamma'}\sigma}\ \texttt{ConvTy}$$

Here it is helpful that the rules are formulated in terms of the meta theoretic substitution and not the explicit substitution. This allows us to define the context coercion in terms of the generalized substitution and to derive `ConvTy` as an admissible rule of the theory. We proceed by first defining a morphism $\mathsf{convMor}_{\Gamma,\Gamma'}$.

Definition 2.9 (convMor in syntxexplicit)**.** Let Γ and Γ' be contexts. We define the context morphism $\mathsf{convMor}_{\Gamma,\Gamma'}$ by induction on the context

$$\begin{aligned}\mathsf{convMor}_{\diamond,\diamond} &:= ()\\ \mathsf{convMor}_{(\Gamma,\sigma),(\Gamma',\sigma')} &:= \mathsf{convMor}_{\Gamma,\Gamma'}, \mathsf{c}_{\sigma,\sigma'}\mathbf{1}\end{aligned}$$

The following lemma justifies that this is a sensible definition.

Lemma 2.10 (convMor-Derivable in rulesexplicit.agda)**.** *For any Γ, Γ' we have, if $\vdash_e \Gamma \equiv \Gamma'$ then*

$$\Gamma \vdash_e \mathsf{convMor}_{\Gamma,\Gamma'} : \Gamma'$$

Having generalized substitution, this lemma immediately implies the corollary

Corollary 2.11. *For any judgment $\Gamma \vdash_e \sigma$ with $\vdash_e \Gamma \equiv \Gamma'$ we have*

$$\Gamma' \vdash_e \sigma[\mathsf{convMor}_{\Gamma',\Gamma}]$$

Having these property, we are confident to define

$$\mathsf{cCtx}_{\Gamma,\Gamma'}\sigma := \sigma[\mathsf{convMor}_{\Gamma',\Gamma}]$$

Going back to the specific situation where $\Gamma := \Delta, \sigma$ and $\Gamma' := \Delta, \tau$, we can use a different morphism that is judgmentally equal to $\mathsf{convMor}_{\Gamma,\Gamma'}$, but easier to work with:

$$(\Delta, \sigma) \vdash_e (\mathsf{id}\uparrow, \mathsf{c}_{\sigma\uparrow,\tau\uparrow}\mathbf{1}) : (\Delta, \tau).$$

This motivates to use the following abbreviation going forward.

Definition 2.12 (coercTy)**.** Let τ, σ_1 and σ_2 be type expressions. We define

$$\mathsf{cTy}_{\sigma_1,\sigma_2}\tau := \tau[\mathsf{id}\uparrow, \mathsf{c}_{\sigma_2\uparrow,\sigma_1\uparrow}\mathbf{1}]$$

.

The idea behind this definition is to think of $\mathsf{cTy}_{\sigma_1,\sigma_2}\tau$ to be a function that maps a typing $\Gamma, \sigma_1 \vdash_e \tau$ to a typing $\Gamma, \sigma_2 \vdash_e \mathsf{cTy}_{\sigma_1,\sigma_2}\tau$.

Definition 2.13 (coercTm)**.** Let M a term and σ_1, σ_2 be types. Then

$$\mathsf{cTm}_{\sigma_1,\sigma_2}M := M[\mathsf{id}\uparrow \mathsf{c}_{\sigma_2\uparrow,\sigma_1\uparrow}\mathbf{1},]$$

Similar as before, the idea behind this definition is that knowing $\Gamma, \sigma_1 \vdash_e M : \tau$ and $\Gamma \vdash_e \sigma_1 \cong \sigma_2$ should give us that

$$\Gamma, \sigma_2 \vdash_e \mathsf{cTm}_{\sigma_1,\sigma_2}\tau : \mathsf{cTy}_{\sigma_1,\sigma_2}\tau.$$

Remark 2.14. Because of the inductive structure of derivations and the fact that we included dependent function types in our theory, the more specific definitions of $\mathsf{cTy}_{\sigma_1,\sigma_2}\tau$ and $\mathsf{cTm}_{\sigma_1,\sigma_2}M$ are sufficient for almost everything we do in this paper.

2.3.4 Explicit Pi-types

Having addressed the subtleties that we face because of the replacement of the conversion rule, the path is clear to restate the rules for the explicit Π-types. We start with the rules that remain in fact unchanged.

$$\frac{\Gamma \vdash_e \sigma \qquad \Gamma, \sigma \vdash_e \tau}{\Gamma \vdash_e \Pi_\sigma \tau}\ \Pi\text{-}\mathtt{Form} \qquad \frac{\Gamma \vdash_e \sigma \qquad \Gamma, \sigma \vdash_e \tau \qquad \Gamma, \sigma \vdash_e M : \tau}{\Gamma \vdash_e \lambda\sigma.\, M : \Pi_\sigma \tau}\ \Pi\text{-}\mathtt{Intro}$$

$$\frac{\Gamma \vdash_e \sigma \qquad \Gamma, \sigma \vdash_e \tau \qquad \Gamma \vdash_e f : \Pi_\sigma \tau \qquad \Gamma \vdash_e N : \sigma}{\Gamma \vdash_e fN : \tau[N]}\ \Pi\mathtt{Elim}$$

$$\frac{\Gamma \vdash_e \sigma \qquad \Gamma, \sigma \vdash_e \tau \qquad \Gamma, \sigma \vdash_e M : \tau \qquad \Gamma \vdash_e N : \sigma}{\Gamma \vdash_e (\lambda\sigma.\, M)N \equiv M[N] : \tau[N]}\ \beta\equiv$$

$$\frac{\Gamma \vdash_e \sigma \qquad \Gamma, \sigma \vdash_e \tau \qquad \Gamma \vdash_e f : \Pi_\sigma \tau}{\Gamma \vdash_e f \equiv \lambda\sigma.\, ((f\uparrow)\mathbf{1}) :}\ \eta\equiv$$

The following congruence rules are different and tailored to resemble the idea behind the explicit syntax.

$$\frac{\begin{array}{ccc}\Gamma \vdash_e \sigma & \Gamma \vdash_e \sigma' & (\Gamma, \sigma) \vdash_e \tau \\ (\Gamma, \sigma') \vdash_e \tau' & \Gamma \vdash_e \sigma \cong \sigma' & (\Gamma, \sigma) \vdash_e \tau \cong \mathsf{cTy}_{\sigma',\sigma}\tau'\end{array}}{\Gamma \vdash_e \Pi_\sigma \tau \cong \Pi_{\sigma'}\tau'}\ \Pi \equiv$$

$$\frac{\Gamma \vdash_e \sigma \qquad \Gamma \vdash_e \sigma' \qquad (\Gamma,\sigma) \vdash_e \tau \qquad [\dots] \qquad \Gamma \vdash_e \sigma \cong \sigma' \qquad (\Gamma,\sigma) \vdash_e \tau \cong \mathsf{cTy}_{\sigma',\sigma}\tau' \qquad (\Gamma,\sigma) \vdash_e M \equiv \mathsf{c}_{(\mathsf{cTy}_{\sigma',\sigma}\tau'),\tau}\mathsf{cTm}_{\sigma',\sigma}M' : \tau}{\Gamma \vdash_e \lambda\sigma.\, M \equiv \mathsf{c}_{\Pi_{\sigma'}\tau',\Pi_\sigma\tau}\lambda\sigma'.\, M' : \Pi_\sigma\tau}\ \lambda\equiv$$

$$\frac{\Gamma \vdash_e \sigma \qquad \Gamma \vdash_e \sigma' \qquad (\Gamma,\sigma) \vdash_e \tau \qquad [\dots] \qquad \Gamma \vdash_e \sigma \cong \sigma' \qquad (\Gamma,\sigma) \vdash_e \tau \cong \mathsf{cTy}_{\sigma',\sigma}\tau' \qquad \Gamma \vdash_e f \equiv \mathsf{c}_{\Pi_{\sigma'}\tau',\Pi_\sigma\tau}f' : \Pi_\sigma\tau \qquad \Gamma \vdash_e N \equiv \mathsf{c}_{\sigma',\sigma}N' : \sigma}{\Gamma \vdash_e fN \equiv \mathsf{c}_{\tau'[N'],\tau[N]}f'N' : \tau[N]}\ \mathtt{App}\equiv$$

Remark 2.15. It is possible, that more presuppositions are needed compared to the original syntax. For the time being, I decided to add all presuppositions for these three rules. Some of them may be left away without any inconvenience, some may cause complications and some may not be omitted at all. A simple guideline: One may omit all preasumptions that are not needed to prove that ALL presuppositions are admissible.

2.3.5 Contexts and Context Morphisms

The derivation of contexts and context morphisms remains the same, they are generated by the rules

$$\frac{}{\vdash_e \diamond}\ \mathtt{Ctx} \qquad \frac{\vdash_e \Gamma \qquad \Gamma \vdash_e \sigma}{\vdash_e (\Gamma,\sigma)}\ \mathtt{Ctx}.$$

$$\frac{}{\Gamma \vdash_e () : \diamond}\ \mathtt{Mor} \qquad \frac{\Gamma \vdash_e \delta : \Delta \qquad \Gamma \vdash_e M : \sigma[M_1,\cdots,M_m]}{\Gamma \vdash_e (\delta, M) : (\Delta,\sigma)}\ \mathtt{Mor}$$

This does not hold true anymore for context and morphism equalities. Here, we have again to take the more complicated equality rules into consideration.

$$\frac{}{\vdash_e \diamond \equiv \diamond}\ \mathtt{Ctx}= \qquad \frac{\vdash_e \Gamma \equiv \Gamma' \qquad \Gamma \vdash_e \sigma \cong \mathsf{cCtx}_{\Gamma',\Gamma\tau}}{\vdash_e (\Gamma,\sigma) \equiv (\Gamma',\tau)}\ \mathtt{Ctx}=$$

Similarly, there are subtleties to consider with regard to judgmental equalities between two context morphism.
Assume for a moment that we still defined $\Gamma \vdash_e \delta \equiv \delta' : \Delta$ to be:

$$\begin{aligned} &\Gamma \vdash_e \delta_1 \equiv \delta_1' : \Delta_1 \\ &\Gamma \vdash_e \delta_2 \equiv \delta_2' : \Delta_2[\delta_1] \\ &\qquad \dots \\ &\Gamma \vdash_e \delta_m \equiv \delta_m' : \Delta_m[\delta_1,\dots,\delta_m] \end{aligned} \tag{15}$$

The problem with this is, that already $\Gamma \vdash_e \delta_2 \equiv \delta_2' : \Delta_2[\delta_1]$ is not derivable. Indeed, δ' being a context morphism only hypothesises $\Gamma \vdash_e \delta_2' : \Delta_2[\delta_1']$ and without the `Conv` rule we cannot simply substitute judgmentally equal types for another.

Therefore, we restate the definition to

$$\begin{aligned}
&\Gamma \vdash_e \delta_1 \equiv \mathsf{c}_{\Delta_1',\Delta_1}\delta_1' : \Delta_1 \\
&\Gamma \vdash_e \delta_2 \equiv \mathsf{c}_{\Delta_2'[\delta_1'],\Delta_2[\delta_1]}\delta_2' : \Delta_2[\delta_1] \\
&\qquad \dots \\
&\Gamma \vdash_e \delta_m \equiv \mathsf{c}_{\Delta_m'[\dots],\Delta_m[\dots]}\delta_m' : \Delta_m[\delta_1, \dots, \delta_m],
\end{aligned} \tag{16}$$

which is a formulation that is sensible for the explicit syntax, as we will see. Now we can use $\Gamma \vdash_e \delta_m' : \Delta_m[\delta_1', \dots, \delta_{m-1}']$ and the `Coerc` rule to derive $\Gamma \vdash_e \mathsf{c}_{\Delta_m'[\dots],\Delta_m[\dots]}\delta_m' : \Delta_m[\delta_1, \dots, \delta_{m-1}]$.
An alternative formulation for the same judgments is to define the *judgmental equality of morphisms* inductively (as done in the formalization) by the following constructors:

$$\frac{}{\Gamma \vdash_e () \equiv () : \diamond}\ \texttt{Mor=}$$

$$\frac{\Gamma \vdash_e \delta \equiv \delta' : \Delta \qquad \Gamma \vdash_e M \equiv \mathsf{c}_{\sigma[\delta'],\sigma[\delta]}N : \sigma}{\Gamma \vdash_e (\delta, M) \equiv (\delta', N) : (\Delta, \sigma)}\ \texttt{Mor=}$$

2.3.6 Admissible rules

The selection presented in this chapter is similar to the selection for the normal syntax. Hence, it might be insightful for the reader to compare the admissible rules of the normal and the explicit syntax.
We start with the weakening lemmas:

$$\frac{\Gamma \vdash_e \sigma}{\Gamma\uparrow_k^\tau \vdash_e \sigma\uparrow_k}\ \texttt{WeakTy}. \qquad \frac{\Gamma \vdash_e M : \sigma}{\Gamma\uparrow_k^\tau \vdash_e M\uparrow_k : \sigma\uparrow_k}\ \texttt{WeakTm}.$$

$$\frac{\Gamma \vdash_e \sigma_1 \cong \sigma_2}{\Gamma\uparrow_k^\tau \vdash_e \sigma_1\uparrow_k \cong \sigma_2\uparrow_k}\ \texttt{WeakTyEq}$$

$$\frac{\Gamma \vdash_e M \equiv N : \sigma}{\Gamma\uparrow_k^\tau \vdash_e M\uparrow_k \equiv N\uparrow_k : \sigma\uparrow_k}\ \texttt{WeakTyEq}$$

Except for the different notation, the admissible weakening lemmas seem to remain the same. This is not the case for substitution, which requires a few adjustments, to be derivable.

$$\frac{\Delta \vdash_e \sigma \qquad \Gamma \vdash_e \delta : \Delta}{\Gamma \vdash_e \sigma[\delta]}\ \texttt{SubstTy} \qquad \frac{\Delta \vdash_e M : \sigma \qquad \Gamma \vdash_e \delta : \Delta}{\Gamma \vdash_e M[\delta] : \sigma[\delta]}\ \texttt{SubstTm}$$

$$\frac{\Delta \vdash_e \sigma \cong \tau \qquad \Gamma \vdash_e \delta : \Delta}{\Gamma \vdash_e \sigma[\delta] \cong \tau[\delta]}\ \texttt{SubstTyEq}$$

$$\frac{\Delta \vdash_e M \equiv N : \sigma \qquad \Gamma \vdash_e \delta : \Delta}{\Gamma \vdash_e M[\delta] \equiv N[\delta] : \sigma[\delta]}\ \texttt{SubstTmEq}$$

$$\frac{\begin{array}{c}\Delta \vdash_e \sigma \qquad \Gamma \vdash_e \delta : \Delta \qquad \Gamma \vdash_e \theta : \Delta \\ \Gamma \vdash_e \delta \equiv \theta : \Delta \qquad \Gamma \vdash_e \theta \equiv \delta : \Delta\end{array}}{\Gamma \vdash_e \sigma[\delta] \cong \sigma[\theta]}\ \texttt{SubstTyMorEq}$$

$$\frac{\begin{array}{c}\Delta \vdash_e M : \sigma \qquad \Gamma \vdash_e \delta : \Delta \qquad \Gamma \vdash_e \theta : \Delta \\ \Gamma \vdash_e \delta \equiv \theta : \Delta \qquad \Gamma \vdash_e \theta \equiv \delta : \Delta\end{array}}{\Gamma \vdash_e M[\delta] \equiv \mathsf{c}_{\sigma[\theta],\sigma[\delta]} M[\theta] : \sigma[\delta]}\ \texttt{SubstTmMorEq}$$

It should come to nobody's surprise that the presupposition lemmas are also provable. These lemmas could become more interesting if one tried to thin out the rules in section 2.3.4 a little more.

3 Translation

Now we turn to the main result of this paper. We want to make precise in what sense these two syntaxes are equivalent. Therefore, we first define a stripping function from the explicit syntax into the normal one and prove a soundness theorem. In a further step step we construct a lifting from the normal into the explicit syntax for which we then again can establish a soundness theorem.

3.1 Stripping

The explicit syntax is defined in a way such that the coercions occurring in the terms of a derivation may be stripped, leading to a derivation in the original syntax. The stripping is defined on the raw syntax and from there induces a mapping on judgments and the derivations.

$$\texttt{ex.TyExpr}_n \xrightarrow{|\cdot|} \texttt{TyExpr}_n \qquad \texttt{ex.TmExpr}_n \xrightarrow{|\cdot|} \texttt{TmExpr}_n$$

$$\texttt{Derivation}_e \xrightarrow{|\cdot|} \texttt{Derivation}$$

The formalization of this chapter can be found in translation.agda.

Stripping the coercions from the syntax is very straight forward: We erase every coercion. In a second step we prove that this operation preserves derivability and hence that it is sound to interpret the explicit syntax via the stripping.

$$
\begin{aligned}
|\cdot| &: \mathtt{ex.TyExpr}_n \to \mathtt{TyExpr}_n \\
|\mathcal{U}| &= \mathcal{U} \\
|\mathsf{el}\, v| &= \mathsf{el}\, |v| \\
|\Pi_\sigma \tau| &= \Pi_{|\sigma|} |\tau|
\end{aligned}
$$

$$
\begin{aligned}
|\cdot| &: \mathtt{ex.TmExpr}_n \to \mathtt{TmExpr}_n \\
|\boldsymbol{n}| &= \boldsymbol{n} \\
|\mathsf{lam}\, \sigma\tau.\, M| &= \mathsf{lam}\, |\sigma||\tau|.\, |M| \\
|\mathsf{app}_{\sigma\tau} f N| &= \mathsf{app}_{|\sigma||\tau|} |f||N| \\
|\mathsf{c}_{\sigma,\tau} M| &= M
\end{aligned}
$$

Figure 8: Types

Figure 9: Terms

Having defined the stripping for raw types and terms, it is straightforward to extend the stripping to contexts and context morphisms

$$
\begin{aligned}
|\cdot| &: \mathtt{ex.Ctx}_n \to \mathtt{Ctx}_n \\
|\diamond| &= \diamond \\
|\Gamma, \sigma| &= |\Gamma|, |\sigma|
\end{aligned}
$$

$$
\begin{aligned}
|\cdot| &: \mathtt{ex.Mor}_{n,m} \to \mathtt{Mor}_{n,m} \\
|()| &= () \\
|\delta, M| &= |\delta|, |M|
\end{aligned}
$$

and eventually to the stripping for judgments

$$
\begin{aligned}
|\cdot| &: \mathtt{Judgement}_e \to \mathtt{Judgement} \\
|\Gamma \vdash_e \sigma| &= |\Gamma| \vdash |\sigma| \\
|\Gamma \vdash_e M : \sigma| &= |\Gamma| \vdash |M| : |\sigma| \\
|\Gamma \vdash_e \sigma \cong \tau| &= |\Gamma| \vdash |\sigma| \equiv |\tau| \\
|\Gamma \vdash_e M \equiv N : \sigma| &= |\Gamma| \vdash |M| \equiv |N| : |\sigma|
\end{aligned}
$$

Now the statement can be be formulated more precisely.

Theorem 3.1 (Soundness theorem)**.** *Let $\mathcal{J}_e$ be a derivable judgment in the explicit syntax, then $|\mathcal{J}_e|$ is a derivable judgment in the original syntax.*

The proof is very straight forward and the only subtleties arise from substitution, which is used in the Π`Elim` and $\beta\equiv$ rule, as well as the de Bruijn index shifting, which occurs in the rules related to variables and $\eta\equiv$.

Therefore, to obtain this soundness result, we first show that these operations commute with the stripping on the nose.

Lemma 3.2. *For any type σ, term M, context morphism δ and k we have*

$$
\begin{aligned}
|\sigma\uparrow_k| &= |\sigma|\uparrow_k \\
|\sigma[\delta]| &= |\sigma|[|\delta|].
\end{aligned}
$$

$$
\begin{aligned}
|M\uparrow_k| &= |M|\uparrow_k \\
|M[\delta]| &= |M|[|\delta|].
\end{aligned}
$$

Furthermore, for context morphisms it is the case that

$$
|\delta\uparrow_k| = |\delta|\uparrow_k
$$

Figure 10: Morphisms

$$
|\delta^+| = |\delta|^+
$$

Figure 11: Morphism extension

Proof. The fact that the commutation is up to equality and the result merely syntactical makes the induction straight forward. The formalized proofs WeakenTy'CommStrip, WeakenTm'CommStrip, weakenMorCommStrip weakenMor+CommStrip, substTyCommStrip and substTmCommStrip can be found in translation.agda. □

Proof (Stripping soundness theorem). The name of the theorem in translation.agda is DerToNormal. To prove the soundness of the stripping we need to show that if $\mathcal{D}$ is a derivation of $\mathcal{J}$, then $|\mathcal{D}|$ is a derivation of $|\mathcal{J}|$. This is proven by induction on the derivation trees. To illustrate how this is accomplished let us consider the case were the last rule applied was $\Pi\texttt{Elim}$. In this situation we have

$$\mathcal{J} := \Gamma \vdash_e fN : \sigma[N] \tag{17}$$
$$\mathcal{D} := |\Pi\texttt{Elim}\{\mathcal{D}_\sigma, \mathcal{D}_\tau, \mathcal{D}_f, \mathcal{D}_N\}| \tag{18}$$

We may assume by induction hypothesis that

1. $|\mathcal{D}_\sigma|$ is a derivation of $|\Gamma| \vdash |\sigma|$
2. $|\mathcal{D}_\tau|$ is a derivation of $|\Gamma|, |\sigma| \vdash |\tau|$
3. $|\mathcal{D}_f|$ is a derivation of $|\Gamma| \vdash |f| : \Pi_{|\sigma|}|\tau|$.
4. $|\mathcal{D}_N|$ is a derivation of $|\Gamma| \vdash |N| : |\sigma|$

We need to show that $|\mathcal{D}|$ is a derivation of $|\mathcal{J}|$. Since the stripping on the judgments is defined inductively we know that in fact

$$|\mathcal{J}| := |\Gamma| \vdash |f||N| : |\sigma[N]|. \tag{19}$$

Furthermore, we may use $\Pi\texttt{Elim}$ and the induction hypothesis to get a derivation.

$$\frac{|\mathcal{D}_\sigma| \qquad |\mathcal{D}_\tau| \qquad |\mathcal{D}_f| \qquad |\mathcal{D}_N|}{|\Gamma| \vdash |f||N| : |\sigma|[|N|]}\ \Pi\texttt{Elim}$$

Figure 12

Observe that the judgment in eq. (19) and the conclusion of fig. 12 are almost identical except for the type, which is $|\sigma[N]|$ on one hand and $|\sigma|[|N|]$ on the other.

Since $|\sigma[N]| = |\sigma|[|N|]$ we may rewrite the type expressions and thus the derivation in fig. 12 is in fact a derivation of $|\mathcal{J}|$. □

For either syntaxes, the derivation of a context is an inductively defined notion that essentially is a list of derivations in disguise. Therefore, the following corollary is an immediate consequence of the previous theorem.

Corollary 3.3 (StripCtx)**.** *Let* $\mathcal{D}_\Gamma := (\mathcal{D}_{\Gamma_1}, \ldots, \mathcal{D}_{\Gamma_n})$ *be a derivation of* $\vdash_e \Gamma$*. Then*

$$|\mathcal{D}_\Gamma|_{\mathtt{Ctx}} := (|\mathcal{D}_{\Gamma_1}|, \ldots, |\mathcal{D}_{\Gamma_n}|) \tag{20}$$

is a derivation of $\vdash |\Gamma|$*.*

Proof. Given the inductive structure of contexts (that resembles a list structure) the claim follows from the previous theorem. □

3.2 Lift

The lifting shall be used to prove coherence results for the original syntax. We try to understand in what sense the original syntax could be interpreted in a locally cartesian closed category.

To achieve that, we define a function that from a mere syntactical standpoint is a right-inverse of the stripping. Hence we proceed by first defining the lift as function on the raw syntax. Then we prove that this function preserves derivability. This soundness theorem is then used to draw conclusions about the relation between the original and the explicit syntax. There are many different choices of liftings, not all of them are good for different reasons.

Example 3.4 (Trivial syntactic lifting)**.** Since the explicit syntax is an extension of the original syntax, there is a trivial lifting that just embeds the original syntax into the explicit syntax. This lifting however, will not preserve the derivability of judgments. Consider for instance the judgment $\Gamma \vdash M : \tau$ and a derivation $\mathcal{D} = \mathtt{Conv}\{\mathcal{D}_\sigma, \mathcal{D}_\tau, \mathcal{D}_M\}$ – i.e. the last rule applied was `Conv`. Assume that $\mathcal{D}_\sigma$, $\mathcal{D}_\tau$ and $\mathcal{D}_M$ were also derivations in the explicit system. In this situation we are in general not in the position to derive $\Gamma \vdash_e M : \tau$, but instead $\Gamma \vdash_e \mathsf{c}_{\sigma,\tau} M : \tau$. This is due to the fact that the explicit system replaces the `Conv` rule with the `Coerc` rule.

The most straightforward approach to construct such a better behaving right-inverse is to wrap a term and all its subterms into coercions. The coercions should be chosen in such a way that for any derivable judgment the use of the `Conv` rule can be substituted by the `Coerc` rule. Then it is very likely that a soundness result will be provable.

3.2.1 Unique Typings

We need to find a systematic way to patch the syntax with explicit coercions in a meaningful way. Therefore, observe that we can assign to any term a *canonical* type relative to a context. Indeed, this information is already encoded in grammar of the terms and the context is only needed to determine the typing

of the variables.

$$\begin{aligned}
\mathsf{Ty} : \mathtt{Ctx}_n \to \mathtt{TmExpr}_n &\to \mathtt{TyExpr}_n \\
\mathsf{Ty}_\Gamma(\boldsymbol{k}) &:= \Gamma_{n-k+1} \underbrace{\uparrow \ldots \uparrow}_{k-1\, times} \\
\mathsf{Ty}_\Gamma(\mathsf{lam}\, \sigma\tau.\, M) &:= \Pi_\sigma \tau \\
\mathsf{Ty}_\Gamma(\mathsf{app}_{\sigma\tau} M N) &:= \tau[N] \\
\mathsf{Ty}_\Gamma(\mathsf{c}_{\sigma,\tau} M) &:= \tau
\end{aligned}$$

This function plays an important role in the lifting. It is well behaved for our purposes since it commutes with substitution and weakening syntactically, as the following two lemmas show.

Lemma 3.5 (weakenTy'-getTy in syntxexplicit.agda)**.** *Let Γ be a context and M a term of the explicit syntax then for any $\boldsymbol{k}$ we have*

$$\mathsf{Ty}_\Gamma(M)\uparrow_{\boldsymbol{k}} = \mathsf{Ty}_{\Gamma\uparrow^\tau_{\boldsymbol{k}}}(M\uparrow_{\boldsymbol{k}})$$

Lemma 3.6 (getty-[ ]Ty in rulesexplicit.agda)**.** *For any explicit term u and context morphism δ such that $\Gamma \vdash_e \delta : \Delta$ it is the case that*

$$\mathsf{Ty}_\Delta(M)[\delta] = \mathsf{Ty}_\Gamma(M[\delta])$$

The proofs of these lemmas are carried out by induction over the raw term constructors.

In the explicit syntax it turns out that typings are in fact unique[2]. That is, if $\Gamma \vdash_e M : \sigma$, then it is already the case that $\mathsf{Ty}_\Gamma(M) = \sigma$. This information is very valuable in the handling of the explicit coercions.

Note that on a syntactic level, by the previous observation the following judgments are equal given that they are derivable.

$$\begin{aligned}
&\Gamma \vdash_e \mathsf{c}_{\sigma,\tau} M : \tau \\
=\ &\Gamma \vdash_e \mathsf{c}_{\mathsf{Ty}_\Gamma(M),\tau} M : \tau.
\end{aligned}$$

This makes the first index of $\mathsf{c}_{\sigma,\tau} M$ redundant which is why it is justifies to abbreviate to

$$\Gamma \vdash_e \mathsf{c}_{\mathsf{Ty},\tau} M : \tau.$$

Furthermore, the arguments for the Ty function can be omitted since they already occur in the judgment.

3.2.2 Lifting Raw Syntax

The idea of this lifting is to recursively wrap the terms into coercions. As discussed in the previous chapter we need to choose the correct coercion. Therefore,

[2] see also getTy=Ty in rules.agda

the term lifting needs a context and another type as argument. For brevity we define

$$\begin{aligned}\widehat{\sigma} &:= \mathsf{liftty}_\Gamma \sigma\\ \widehat{M} &:= \mathsf{lifttm}_{\Gamma\tau} M\\ \widehat{\Gamma} &:= \mathsf{liftctx}\,\Gamma\\ \widehat{\delta} &:= \mathsf{liftmor}_{\Gamma\Delta}\delta\\ \widehat{\mathcal{J}} &:= \mathsf{liftjudg}\mathcal{J}.\end{aligned}$$

Although this notation will be better to read, it is not ideal as it hides many details and technical issues by omitting arguments that are relevant. These technicalities are better visible in the formalization. Since this work is accompanied by a formalization I decided in favor of simplifications and abuse of notation for a better reading experience.

We begin now to define the lift on the raw syntax.

$$\begin{aligned}\mathsf{liftty} : \mathtt{ex.Ctx}_n \to \mathtt{TyExpr}_n \to \mathtt{ex.TyExpr}_n\\ \mathsf{liftty}_\Gamma \mathcal{U} &:= \mathcal{U}\\ \mathsf{liftty}_\Gamma \mathsf{el}\, M &:= \mathsf{el}\,\widehat{M}\\ \mathsf{liftty}_\Gamma \Pi_\sigma \tau &:= \Pi_{\widehat{\sigma}} \widehat{\tau}\end{aligned}$$

$$\begin{aligned}\mathsf{lifttm} : \mathtt{ex.Ctx}_n \to \mathtt{TmExpr}_n \to \mathtt{ex.TyExpr}_n \to \mathtt{ex.TmExpr}_n\\ \mathsf{lifttm}_{\Gamma\tau} \boldsymbol{k} &:= \mathsf{c}_{\Gamma_{n-k+1},\tau}\boldsymbol{k}\\ \mathsf{lifttm}_{\Gamma\tau} \mathsf{lam}\,\sigma_0\sigma_1.\, M &:= \mathsf{c}_{\Pi_{\widehat{\sigma_0}}\widehat{\sigma_1},\tau}(\mathsf{lam}\,\widehat{\sigma_0}\widehat{\sigma_1}.\,\widehat{M})\\ \mathsf{lifttm}_{\Gamma\tau} \mathsf{app}_{\sigma_0\sigma_1} f N &:= \mathsf{c}_{\widehat{\sigma_1}[\widehat{N}],\tau}(\mathsf{app}_{\widehat{\sigma_0}\widehat{\sigma_1}}\widehat{f}\widehat{N})\end{aligned}$$

The next step is to define the lifting of contexts and context morphisms.

$$\begin{aligned}\mathsf{liftctx} : \mathtt{Ctx}_n \to \mathtt{ex.Ctx}_n\\ \mathsf{liftctx}\,\diamond &:= \diamond\\ \mathsf{liftctx}\,(\Gamma,\sigma) &:= (\widehat{\Gamma},\widehat{\sigma})\end{aligned}$$

$$\begin{aligned}\mathsf{liftmor} : \mathtt{ex.Ctx}_n \to \mathtt{ex.Ctx}_m \to \mathtt{Mor}_{n,m} \to \mathtt{ex.Mor}_{n,m}\\ \mathsf{liftmor}_{\Gamma\Delta}(\,) &:= (\,)\\ \mathsf{liftmor}_{\Gamma(\Delta,\sigma)}(\delta, M) &:= (\widehat{\delta},\widehat{M})\end{aligned}$$

There are alternative ways of lifting morphisms and contexts. Most notably, we could be more economical with the use of explicit coercions. However, it turns out that it is not easy to decide which ones are superfluous. In particular we need the outer most coercion. Without this outermost coercion, the soundness

theorem fails. But there is a price to pay: The lift does not preserve identity context morphism syntactically. This means that is not the case that

$$\widehat{\mathsf{id}} = \mathsf{id}.$$

This is noteworthy, since although it will still be the case that

$$\Gamma \vdash_e \widehat{\mathsf{id}} \equiv \mathsf{id} : \Gamma$$

this will cause subtleties. The problem is that – unlike for the stripping – the following identity fails

$$\widehat{\sigma}[\widehat{M}] := \widehat{\sigma}[(\mathsf{id}_n, \widehat{M})] \neq \widehat{\sigma[(\mathsf{id}_n, M)]} = \widehat{\sigma[M]}.$$

Figure 13: Substitution and Lifting

The consequences of that and how to get around it will be examined in detail in section 3.2.4.

Eventually, the lifting of a judgment is defined by

$$\begin{aligned}
&\mathsf{liftjudg} : \mathtt{Judgement} \to \mathtt{Judgement}_e \\
&\qquad \widehat{(\Gamma \vdash \sigma)} := \widehat{\Gamma} \vdash_e \widehat{\sigma} \\
&\qquad \widehat{(\Gamma \vdash M : \sigma)} := \widehat{\Gamma} \vdash_e \widehat{M} : \widehat{\sigma} \\
&\qquad \widehat{(\Gamma \vdash \sigma \equiv \tau)} := \widehat{\Gamma} \vdash_e \widehat{\sigma} \cong \widehat{\tau} \\
&\qquad \widehat{(\Gamma \vdash M \equiv N : \sigma)} := \widehat{\Gamma} \vdash_e \widehat{M} \equiv \widehat{N} : \widehat{\sigma}.
\end{aligned}$$

Remark 3.7. The abbreviated notation $\widehat{\sigma}$ is only underdetermined if it stands on its own. When used in the context of a derivable judgment, it is unambiguous. This is for the reason that for the lift of a judgment to be derivable, there is a unique choice of auxiliary arguments. Take for instance the judgment $\Gamma \vdash M : \sigma$, whose lift is

$$\widehat{\Gamma} \vdash_e \widehat{M} : \widehat{\sigma}.$$

For this to be derivable it is necessary that $\widehat{\sigma} := \mathsf{liftty}_{\widehat{\Gamma}}\sigma$ and $\widehat{M} := \mathsf{lifttm}_{\widehat{\Gamma}\widehat{\sigma}}M$. Indeed, recall from section 3.2.1 that typings are unique, and thus every derivable term has a unique type. In addition to that, the context of the lift must agree with the context of the judgment. Otherwise, variables would be wrapped in nonsense coercions.

Having defined the lift function on the raw syntax, we turn our attention to the properties we require it to satisfy (recall the non-example example 3.4). We will prove that the following two statements are valid:

1. for any $\mathcal{J} \in \mathtt{Judgement}$ it is true that $|\widehat{\mathcal{J}}| = \mathcal{J}$ and

2. The lift is *sound*, which means that it preserves the derivability of judgements.

The first point is not difficult to establish and we can prove it right away.

Theorem 3.8. *The lift is a right inverse to the stripping on the raw syntax and therefore for all types σ, terms M, contexts Γ, morphisms δ and judgments $\mathcal{J}$ it is the case that*

$$|\widehat{\sigma}| = \sigma$$
$$|\widehat{M}| = M$$
$$|\widehat{\Gamma}| = \Gamma$$
$$|\widehat{\delta}| = \delta$$
$$|\widehat{\mathcal{J}}| = \mathcal{J}$$

Proof. The result is merely syntactic and a straight forward induction on the raw types and terms is sufficient to prove it. One proceeds by first showing

$$|\widehat{\sigma}| = \sigma \qquad \text{and} \qquad |\widehat{M}| = M,$$

which then implies that

$$|\widehat{\Gamma}| = \Gamma \qquad , \qquad |\widehat{\delta}| = \delta \qquad \text{and} \qquad |\widehat{\mathcal{J}}| = \mathcal{J}$$

for contexts, context morphisms and judgments.[3]. □

The rest of this chapter is dedicated to verify that also the second point is fulfilled.

3.2.3 The Weakening

Proving that the lifting is sound requires that it commutes with de Bruijn index shifting and substitution. For the de Bruijn index shifting we hence need a collection of lemmas:

Lemma 3.9. *Let $\Gamma \vdash_e \widehat{\sigma}$ be derivable, then so are*

$$\Gamma\!\uparrow^{\tau}_{\boldsymbol{k}} \vdash_e \widehat{\sigma\!\uparrow_{\boldsymbol{k}}} \qquad \Gamma\!\uparrow^{\tau}_{\boldsymbol{k}} \vdash_e \widehat{\sigma}\!\uparrow_{\boldsymbol{k}} \qquad \Gamma\!\uparrow^{\tau}_{\boldsymbol{k}} \vdash_e \widehat{\sigma}\!\uparrow_{\boldsymbol{k}} \cong \widehat{\sigma\!\uparrow_{\boldsymbol{k}}}$$

Lemma 3.10. *Let $\Gamma \vdash_e \widehat{M} : \widehat{\sigma}$ be derivable, then so are*

$$\Gamma\!\uparrow^{\tau}_{\boldsymbol{k}} \vdash_e \widehat{M\!\uparrow_{\boldsymbol{k}}} : \widehat{\sigma\!\uparrow_{\boldsymbol{k}}} \quad \Gamma\!\uparrow^{\tau}_{\boldsymbol{k}} \vdash_e \widehat{M}\!\uparrow_{\boldsymbol{k}} : \widehat{\sigma}\!\uparrow_{\boldsymbol{k}} \quad \Gamma\!\uparrow_{\boldsymbol{k}} \vdash_e \widehat{M}\!\uparrow_{\boldsymbol{k}} \equiv \widehat{M\!\uparrow_{\boldsymbol{k}}} : \widehat{\sigma}\!\uparrow_{\boldsymbol{k}}$$

Lemma 3.11. *Let $\Gamma \vdash_e \widehat{\delta} : \Delta$ be derivable, then so are*

$$\Gamma\!\uparrow^{\tau} \vdash_e \widehat{\delta}\!\uparrow : \Delta \qquad \Gamma\!\uparrow^{\tau} \vdash_e \widehat{\delta\!\uparrow} : \Delta$$

[3]The respective proofs of that are named strip-lift and can be found in reconstruction-approach2.agda

Proof. First, observe that both $\Gamma \uparrow^{\tau}_{\boldsymbol{k}} \vdash_e \widehat{\sigma} \uparrow_{\boldsymbol{k}}$ and $\Gamma \uparrow^{\tau}_{\boldsymbol{k}} \vdash_e \widehat{M} \uparrow_{\boldsymbol{k}} : \widehat{\sigma} \uparrow_{\boldsymbol{k}}$ are the conclusions of the admissible weakening rules. The other judgments now follow from the fact that we can prove a much stronger result: The de Bruijn index shifting and the lift commute up to syntactic equality. □

Lemma 3.12 (weakenTy'-liftTy and weakenTm'-liftTm)**.** *For any type σ and natural number $\boldsymbol{k}$ it is the case that*

$$(\widehat{\sigma}) \uparrow_k = \widehat{\sigma \uparrow_k}$$

Furthermore, for any term M and natural number $\boldsymbol{k}$ we have

$$(\widehat{M}) \uparrow_k = \widehat{M \uparrow_k}$$

Proof. We proceed by induction on the raw types on one hand and on raw terms on the other hand. By doing that, proving this lemma basically boils down to the fact that the de Bruijn Index Shifting commutes with Ty (section 3.2.1) syntactically. Observe that certain subtleties are swept under the rug since we abuse notation by using the comprised notation of the lifting. □

Immediate corollaries of these lemmas are similar results for contexts and context morphisms.

Corollary 3.13. *For any context Γ we have*

$$(\widehat{\Gamma}) \uparrow^{\tau}_k = \widehat{(\Gamma \uparrow^{\tau}_k)}$$

Corollary 3.14. *For any context morphism δ it is the case that*

$$(\widehat{\delta}) \uparrow_k = \widehat{\delta \uparrow_k}$$

To summarize, we have established commutation results for the lift and the de Bruijn index shifting up to syntactic equality.

$$\widehat{\sigma} \uparrow_k = \widehat{\sigma \uparrow_k} \qquad \widehat{M} \uparrow_k = \widehat{M \uparrow_k} \qquad \widehat{\Gamma} \uparrow^{\tau}_k = \widehat{\Gamma \uparrow^{\tau}_k} \qquad \widehat{\delta} \uparrow_k = \widehat{\delta \uparrow_k}, \tag{21}$$

Thereby the lemmas that are required to prove the soundness theorem for the lifting are immediate consequences. Practically, this also means that we are free to commute $\uparrow$ with substitution and the lift while not having to include additional proof steps. Hence, we treat this result as a freebie to not be bothered by any subtleties regarding the de Bruijn index shifting anymore.

3.2.4 Substitution

Unlike for the weakening, a merely syntactic result is not feasible for substitution. In fact to reject the syntactic commutation as

$$\widehat{\sigma[\delta]} = \widehat{\sigma}[\widehat{\delta}] \qquad \widehat{M[\delta]} = \widehat{M}[\widehat{\delta}]$$

is not difficult. There are two problems that make commutation up to syntactic equality between substitution and lifting fail:

1. The morphism extension δ^+ does not commute with the lift.

2. Redundancy of explicit coercions when substituting a variable

The first issue arises from the fact that substitution utilizes the morphism extension, which does not commute with the lifting syntactically.

$$\widehat{\delta^+} = \widehat{(\delta\uparrow, \mathbf{1})} = \left(\widehat{\delta\uparrow}, \widehat{\mathbf{1}}\right) = \left(\widehat{\delta\uparrow}, \mathsf{c}_{\mathsf{Ty}\sigma}\mathbf{1}\right) \neq \left(\widehat{\delta\uparrow}, \mathbf{1}\right) = \hat{\delta}^+ .$$

That these expressions are not syntactically equal is obvious since $\mathsf{c}_{\mathsf{Ty}\mathbf{1}\sigma}\mathbf{1} \neq \mathbf{1}$.

The second problem is a bit more delicate: When substituting for a variable, there is a redundancy of explicit coercions occurring. Let us construct a counterexample for this point.
Let (Γ, σ) and (Δ, τ) be explicit contexts. Assume (δ, M) that is a context morphism such that

$$(\Gamma, \sigma) \vdash_e (\hat{\delta}, \widehat{M}) : (\Delta, \tau)$$

Hence, the lift of (δ, M) has to use (Γ, σ) and (Δ, τ) as additional arguments (that are suppressed in our notation).

Now we see that the equations

$$\begin{aligned}
&\widehat{\mathbf{1}}[(\hat{\delta}, \widehat{M})] && (22)\\
&\quad = (\mathsf{c}_{\mathsf{Ty},\tau\uparrow}\mathbf{1})[(\hat{\delta}, \widehat{M})] && (23)\\
&\quad = \mathsf{c}_{\mathsf{Ty}[(\hat{\delta},\widehat{M})],\tau\uparrow[(\hat{\delta},\widehat{M})]}\widehat{M} && (24)\\
&\quad \neq \widehat{M} && (25)\\
&\quad = \widehat{\mathbf{1}[(\delta, M)]} && (26)
\end{aligned}$$

do not hold, since in eq. (24) the additional outermost explicit coercion makes the syntactic equality fail.

Having said this, a strict equality is a stronger result than needed for the soundness theorem of the lifting. In fact the following two lemmas are sufficient for our purposes.

Lemma 3.15 ([ ]-liftTy1 and [ ]-liftTy1$^=$)**.** *Let* $\vdash_e \Gamma$, $\vdash_e \Delta$, $\Delta \vdash_e \widehat{\sigma}$ *and* $\Gamma \vdash_e \hat{\delta} : \Delta$ *be derivable, then*

$$\Gamma \vdash_e \widehat{\sigma[\delta]}$$

as well as

$$\Gamma \vdash_e \widehat{\sigma}[\hat{\delta}] \cong \widehat{\sigma[\delta]}$$

are derivable.

Lemma 3.16 ([ ]-liftTm2 and [ ]-liftTm2$^{=}$)**.** *Let* $\vdash_e \Gamma$, $\vdash_e \Delta$, $\Delta \vdash_e \widehat{\sigma}$, $\Delta \vdash_e \widehat{M} : \widehat{\sigma}$ *and* $\Gamma \vdash_e \widehat{\delta} : \Delta$ *be derivable, then*

$$\Gamma \vdash_e \widehat{M[\delta]} : \widehat{\sigma[\delta]}$$

and also

$$\Gamma \vdash_e \widehat{M[\delta]} \equiv \mathsf{c}_{\widehat{\sigma[\delta]},\widehat{\sigma}[\widehat{\delta}]} \widehat{M[\delta]} : \widehat{\sigma}[\widehat{\delta}]$$

are derivable

These two lemmas have to be proven by simultaneous induction on the raw type expressions and raw term expressions of the original syntax respectively. The induction is very involving so that it makes sense to promote certain proof steps to be a lemma on their own. Since we are not replicating the formalized proofs here only the most relevant ones shall be mentioned. I restrain from calling these auxiliary lemmas "corollaries" since they are part of a simultaneous induction and some of them are actually quite complicated to prove.

First, recall that morphism extension does not commute syntactically with our choice of lifting. Therefore, the following lemma is crucial not only for the substitution lemmas.

Lemma 3.17 (weaken+-lift^{d} and weaken+-lift$^{=}$)**.** *Let* $\vdash_e \Gamma$, $\vdash_e \Delta$, $\Delta \vdash_e \widehat{\sigma}$ *and* $\Gamma \vdash_e \widehat{\delta} : \Delta$ *be derivable. Then*

$$\left(\Gamma, \widehat{\sigma[\delta]}\right) \vdash_e \widehat{\delta^{+}} : (\Delta, \widehat{\sigma})$$

and furthermore

$$\left(\Gamma, \widehat{\sigma[\delta]}\right) \vdash_e \widehat{\delta^{+}} \equiv \widehat{\delta}^{+} : (\Delta, \widehat{\sigma})$$

are derivable.

Proof. Recall the definition of judgmentally equal context morphisms. We need to show that

$$\Gamma, \widehat{\sigma[\delta]} \vdash_e \widehat{\delta\uparrow} \equiv \widehat{\delta}\uparrow : \Delta \quad \textit{and}$$
$$\Gamma, \widehat{\sigma[\delta]} \vdash_e \widehat{\mathbf{1}} \equiv \mathsf{c}_{\widehat{\sigma[\delta]\uparrow},\widehat{\sigma}[\widehat{\delta}]\uparrow} \mathbf{1} : \widehat{\sigma}[\widehat{\delta}]\uparrow .$$

In both cases the derivation follows by the respective reflexivity rules. On the surface it might seem like this corollary could be proven before the other commutation results. This however is not the case. To derive the second equality, we need a proof of $\Gamma \vdash_e \widehat{\sigma[\delta]} \cong \widehat{\sigma}[\widehat{\delta}]$, this is precisely lemma 3.15. □

Following that, the following lemmas will be of great value to conclude the induction step for cases that involve morphism extension, like for example Π or λ. These very technical results are not very insightful and thus we will not look

into a proof here. The interested reader may be referred to the formalized proof available on Github[4].

Lemma 3.18 (Mor+-[]-liftTy^d and Mor+-[]-$\mathsf{liftTy}^=$)**.** *Let* $\vdash_e \Gamma$ *and* $\vdash_e \Delta$ *be derivations of explicit contexts as well as* $\Gamma \vdash_e \widehat{\delta} : \Delta$ *a morphism between them. Furthermore, let* $\Delta \vdash_e \widehat{\sigma}$ *and* $\Delta, \widehat{\sigma} \vdash_e \widehat{\tau}$*. Then*

$$\left(\Gamma, \widehat{\sigma[\delta]}\right) \vdash_e \widehat{\tau[\delta^+]} \quad \textit{and}$$
$$\left(\Gamma, \widehat{\sigma}[\widehat{\delta}]\right) \vdash_e \widehat{\tau}[\widehat{\delta^+}] \cong \mathsf{cTy}_{\widehat{\sigma[\delta]}, \widehat{\sigma}[\widehat{\delta}]} \widehat{\tau[\delta^+]}$$

are derivable.

Lemma 3.19 (Mor+-[]-liftTm^d and Mor+-[]-$\mathsf{liftTm}^=$)**.** *Let* $\vdash_e \Gamma$, $\vdash_e \Delta$, $\Gamma \vdash_e \widehat{\delta} : \Delta$ *and* $\Delta \vdash_e \widehat{\sigma}$ *be derivable as in the previous claim. Furthermore, let* $(\Gamma, \widehat{\sigma}) \vdash_e \widehat{M} : \widehat{\tau}$*. Then*

$$\left(\Gamma, \widehat{\sigma[\delta]}\right) \vdash_e \widehat{M[\delta^+]} : \widehat{\tau[\delta^+]}$$
$$\left(\Gamma, \widehat{\sigma[\delta]}\right) \vdash_e \widehat{M[\delta^+]} \equiv \mathsf{c}_{\mathsf{Ty}, \widehat{\tau[\delta^+]}} \left(\mathsf{cTm}_{\widehat{\sigma}[\widehat{\delta}], \widehat{\sigma[\delta]}} \widehat{M}[\widehat{\delta^+}]\right) : \widehat{\tau[\delta^+]}$$

Proof of Lemma 3.15 and lemma 3.16. The proof proceeds by induction on the type and term expressions. Let us examine the induction step case of the type $\Pi_\sigma \tau$. In this situation we are given derivations

$$\vdash_e \Gamma, \quad \vdash_e \Delta, \quad \Gamma \vdash_e \widehat{\delta} : \Delta \quad \textit{and} \quad \Delta \vdash_e \Pi_{\widehat{\sigma}} \widehat{\tau}.$$

We furthermore know that the last derivation must in fact be $\Pi - \mathtt{Form}\{\mathcal{D}_{\widehat{\sigma}}, \mathcal{D}_{\widehat{\tau}}\}$ with derivations

$$\Delta \vdash_e \widehat{\sigma} \quad \textit{and} \quad (\Delta, \widehat{\sigma}) \vdash_e \widehat{\tau}.$$

By the induction hypothesis all the lemmas may be applied to these derivations which gives us

$$\Gamma \vdash_e \widehat{\sigma[\delta]}, \quad \Gamma \vdash_e \widehat{\sigma}[\widehat{\delta}] \cong \widehat{\sigma[\delta]}, \quad \left(\Gamma, \widehat{\sigma[\delta]}\right) \vdash_e \widehat{\tau[\delta^+]} \quad \textit{and}$$
$$\left(\Gamma, \widehat{\sigma}[\widehat{\delta}]\right) \vdash_e \widehat{\tau}[\widehat{\delta^+}] \cong \mathsf{cTy}_{\widehat{\sigma[\delta]}, \widehat{\sigma}[\widehat{\delta}]} \widehat{\tau[\delta^+]} \qquad \text{(Hypo)}$$

Our goal is to derive

$$\Gamma \vdash_e \widehat{\Pi_\sigma \tau[\delta]} \quad \textit{and} \quad \Gamma \vdash_e \widehat{\Pi_\sigma \tau}[\widehat{\delta}] \cong \widehat{\Pi_\sigma \tau[\delta]},$$

which by the definition of the lift and substitution is the same as deriving

$$\Gamma \vdash_e \Pi_{\widehat{\sigma[\delta]}} \widehat{\tau[\delta^+]} \quad \textit{and} \quad \Gamma \vdash_e \Pi_{\widehat{\sigma}[\widehat{\delta}]} \widehat{\tau}[\widehat{\delta^+}] \cong \Pi_{\widehat{\sigma[\delta]}} \widehat{\tau[\delta^+]}. \qquad \text{(Goal)}$$

[4]https://github.com/philippstassen/initiality/blob/develop1/reconstruction-approach2.agda

Having that, the proof tactic for either case becomes clear: We can use $\Pi - \texttt{Form}$ respectively $\Pi \equiv$ to conclude the claim. To avoid being repetitive we will only inspect the latter derivation.

We need to find derivations for all presuppositions of the rule:

$$\frac{\begin{array}{c}\Gamma \vdash_e \widehat{\sigma}[\widehat{\delta}] \qquad \Gamma \vdash_e \widehat{\sigma[\delta]} \qquad \left(\Gamma, \widehat{\sigma}[\widehat{\delta}]\right) \vdash_e \widehat{\tau}[\widehat{\delta^+}] \qquad \left(\Gamma, \widehat{\sigma[\delta]}\right) \vdash_e \widehat{\tau[\delta^+]} \\ \Gamma \vdash_e \widehat{\sigma}[\widehat{\delta}] \cong \widehat{\sigma[\delta]} \qquad \left(\Gamma, \widehat{\sigma}[\widehat{\delta}]\right) \vdash_e \widehat{\tau}[\widehat{\delta^+}] \cong \mathsf{cTy}_{\widehat{\sigma[\delta]}, \widehat{\sigma}[\widehat{\delta}]} \widehat{\tau[\delta^+]}\end{array}}{\Gamma \vdash_e \Pi_{\widehat{\sigma}[\widehat{\delta}]} \widehat{\tau}[\widehat{\delta^+}] \cong \Pi_{\widehat{\sigma[\delta]}} \widehat{\tau[\delta^+]}} \; \Pi \equiv$$

From here it is obvious how we need to apply the induction hypothesis and the auxiliary lemmas:

1. $\Gamma \vdash_e \widehat{\sigma}[\widehat{\delta}]$ is derivable by the admissibility of substitution and the assumptions $\Delta \vdash_e \widehat{\sigma}$ and $\Gamma \vdash_e \widehat{\delta} : \Delta$.
2. $\Gamma \vdash_e \widehat{\sigma[\delta]}$ follows from the induction hypothesis
3. $\left(\Gamma, \widehat{\sigma}[\widehat{\delta}]\right) \vdash_e \widehat{\tau}[\widehat{\delta^+}]$ can be derived from with the substitution rule. Note, that $\left(\Gamma, \widehat{\sigma}[\widehat{\delta}]\right) \vdash_e \widehat{\delta^+} : (\Delta, \widehat{\sigma})$ is derivable since $\Gamma \vdash_e \widehat{\delta} : \Delta$ is[5].
4. $\left(\Gamma, \widehat{\sigma[\delta]}\right) \vdash_e \widehat{\tau[\delta^+]}$ is derivable by lemma 3.18.
5. $\Gamma \vdash_e \widehat{\sigma}[\widehat{\delta}] \cong \widehat{\sigma[\delta]}$ is derivable by the induction hypothesis.
6. $\left(\Gamma, \widehat{\sigma}[\widehat{\delta}]\right) \vdash_e \widehat{\tau}[\widehat{\delta^+}] \cong \mathsf{cTy}_{\widehat{\sigma[\delta]}, \widehat{\sigma}[\widehat{\delta}]} \widehat{\tau[\delta^+]}$ is derivable by lemma 3.18. □

This concludes the induction step for Π.

3.2.5 Soundness of the Lift

Now we may turn our attention to the main result of this paper, the Soundness theorem for the lifting. A complete and formalized proof of the theorem can be found in reconstruction-approach2.agda under the name Lift-Der.

Theorem 3.20 (Soundness Theorem for Lifting)**.** *The following four claims are true*

1. *Let $\Gamma \vdash \sigma$ and $\vdash_e \widehat{\Gamma}$ be derivable, then $\widehat{\Gamma} \vdash_e \widehat{\sigma}$ is derivable*
2. *Let $\Gamma \vdash M : \sigma$ and $\vdash_e \widehat{\Gamma}$ be derivable, then also $\widehat{\Gamma} \vdash_e \widehat{M} : \widehat{\sigma}$ is derivable*
3. *Let $\Gamma \vdash \sigma \equiv \tau$ and $\vdash_e \widehat{\Gamma}$ be derivable, then so is $\widehat{\Gamma} \vdash_e \widehat{\sigma} \cong \widehat{\tau}$*
4. *Let $\Gamma \vdash M \equiv N : \sigma$ and $\vdash_e \widehat{\Gamma}$ be derivable, then also $\widehat{\Gamma} \vdash_e \widehat{M} \equiv \widehat{N} : \widehat{\sigma}$ is.*

[5] See WeakMor+ in rulesexplicit.agda

Proof. The proof proceeds by induction on the derivations of the original theory. As for the previous lemmas, we will not give a complete proof here. Instead we sketch the specific case of the $\texttt{App} \equiv$ rule. Admittedly, sketching the proof strategy does not capture the complexities that arise in the formalization.

By the induction hypothesis we have

1. A derivation of $\vdash_e \widehat{\Gamma}$ and
2. Derivations $\widehat{\mathcal{D}_\sigma}, \widehat{\mathcal{D}_{\sigma\equiv}}, \widehat{\mathcal{D}_{\tau\equiv}}, \widehat{\mathcal{D}_{f\equiv}}, \widehat{\mathcal{D}_{N\equiv}}$ of the liftings for all presuppositions of the derivation $\texttt{App} \equiv \{\mathcal{D}_\sigma, \mathcal{D}_{\sigma\equiv}, \mathcal{D}_{\tau\equiv}, \mathcal{D}_{f\equiv}, \mathcal{D}_{N\equiv}\}$ of the judgment $\Gamma \vdash fN \equiv f'N' : \tau[N]$. That is derivations of

$$\begin{aligned}
&\widehat{\Gamma} \vdash_e \widehat{\sigma}\\
&\widehat{\Gamma} \vdash_e \widehat{\sigma} \cong \widehat{\sigma'}\\
&\widehat{\Gamma, \sigma} \vdash_e \widehat{\tau} \cong \widehat{\tau'}\\
&\widehat{\Gamma} \vdash_e \widehat{f} \equiv \widehat{f'} : \widehat{\Pi_\sigma \tau}\\
&\widehat{\Gamma} \vdash_e \widehat{N} \equiv \widehat{N'} : \widehat{\sigma}
\end{aligned}$$

Our goal is to derive that $\widehat{\Gamma} \vdash_e \widehat{fN} \equiv \widehat{f'N'} : \widehat{\tau[N]}$ or more precisely,

$$\widehat{\Gamma} \vdash_e \mathsf{lifttm}_{\widehat{\Gamma}\widehat{\tau[N]}}(fN) \equiv \mathsf{lifttm}_{\widehat{\Gamma}\widehat{\tau[N]}}(f'N') : \mathsf{liftty}_{\widehat{\Gamma}}\tau[N].$$

Our proof strategy is to apply the $\texttt{App} \equiv$ of the explicit theory to the derivations we got from the induction hypothesis. This exhibits

$$\widehat{\Gamma} \vdash_e \widehat{\widehat{f}\widehat{N}} \equiv \mathsf{c}_{\widehat{\tau}\widehat{[N']},\widehat{\tau}\widehat{[N]}} \widehat{\widehat{f'}\widehat{N'}} : \widehat{\tau}\widehat{[N]} \tag{27}$$

To complete the proofs this derivation we need to prove additional judgmental equalities. Hence, let us first discuss what judgments we need to derive and how we may conclude the main theorem: Using the $\texttt{Coerc} \equiv$ and the `CoercTrans` rule as well as the previous lemmas, we find derivations for the following equalities:

$$\widehat{\Gamma} \vdash_e \widehat{fN} \equiv \mathsf{c}_{\widehat{\tau[N]},\widehat{\tau[N]}} \widehat{fN} : \widehat{\tau[N]} \tag{28}$$

$$\widehat{\Gamma} \vdash_e \mathsf{c}_{\widehat{\tau}\widehat{[N]},\widehat{\tau[N]}} \widehat{fN} \equiv \mathsf{c}_{\widehat{\tau}\widehat{[N]},\widehat{\tau[N]}} \mathsf{c}_{\widehat{\tau}\widehat{[N']},\widehat{\tau}\widehat{[N]}} \widehat{f'N'} : \widehat{\tau[N]} \tag{29}$$

$$\widehat{\Gamma} \vdash_e \mathsf{c}_{\widehat{\tau[N']},\widehat{\tau[N]}} \mathsf{c}_{\widehat{\tau}\widehat{[N']},\widehat{\tau[N']}} \widehat{f'}N' \equiv \mathsf{c}_{\widehat{\tau}\widehat{[N']},\widehat{\tau[N]}} \widehat{f'N'} : \widehat{\tau[N]} \tag{30}$$

$$\widehat{\Gamma} \vdash_e \mathsf{c}_{\widehat{\tau}\widehat{[N']},\widehat{\tau[N]}} \widehat{f'N'} \equiv \widehat{f'N'} : \widehat{\tau[N]} \tag{31}$$

Concatenating these equalities - by using the `TmTrans` rule successively – gives the correct result. It remains to show that each of the equalities eq. (28) - (31) is derivable.

1. Equation (28) and (31) are instances of lemma 3.16.

2. Equation (29) can be derived from eq. (27) and the `Coerc`$\equiv$ rule.

3. Equation (30) is the conclusion of the `CoercTrans` rule. □

A simple corollary of that theorem is:

Corollary 3.21 (Lift of Context Derivations)**.** *Let* $\vdash \Gamma$ *be derivable, then so is* $\vdash_e \widehat{\Gamma}$.

This concludes the definition of the stripping and the lifting function and the proofs of their respective soundness theorem. By that, we have proven that the normal and the explicit syntax are strongly related.

4 Conclusion

4.1 Further directions

Let us go back to the origins of the problem. One way to define an interpretation function is to first define it for derivations and then show that this function drops down to the syntax: For every derivable judgment the interpretation does not depend on the choice of the derivation.

The fact that locally cartesian closed categories resemble the structure of dependent type theories inspired R.A.G. Seely to try to interpret dependent type theories in a locally cartesian closed category.

Among others Curien observed that a strict interpretation is not possible due to the mismatch of syntactic and semantic substitution. The function defined on the derivations does not drop to the syntax. To cope with that issue, he defined a modified syntax that has explicit coercions for which coherence could be established. Hence, locally cartesian closed categories are strict models for the explicit syntax.

The key observation is that by relating the original and the explicit theory, certain coherence theorems can still be proven. In a sense locally cartesian closed categories provide a weaker kind of model for dependent type theories. In [CGH13] this *weak interpretation* is made precise by defining it to be a non-strict morphism in a category of non strict structures (as already mentioned in the introduction).

Using the results from the previous chapter and in particular theorem 3.20 the following coherence theorems, that relate the original syntax with the explicit syntax, are provable:

Theorem 4.1. *Let* $\vdash_e \Gamma$ *and* $\Gamma \vdash_e \sigma$ *be derivable, then*

$$\Gamma \vdash_e \sigma \cong \widehat{|\sigma|}$$

is derivable.

Theorem 4.2. *Let* $\vdash_e \Gamma$ *and* $\Gamma \vdash_e M : \sigma$ *be derivable, then*

$$\Gamma \vdash_e M \equiv \widehat{|M|} : \sigma$$

is derivable.

Theorem 4.3. *Let* $\vdash_e \Gamma$, $\Gamma \vdash_e \sigma$, $\Gamma \vdash_e \tau$ *and* $|\Gamma| \vdash |\sigma| \equiv |\tau|$ *be derivable, then also*

$$\Gamma \vdash_e \sigma \cong \tau$$

is derivable.

Theorem 4.4. *Let* $\vdash_e \Gamma$, $\Gamma \vdash_e M : \sigma$, $\Gamma \vdash_e N : \sigma$ *and* $|\Gamma| \vdash |M| \equiv |N| : |\sigma|$ *then it is also the case that*

$$\Gamma \vdash_e M \equiv N : \sigma$$

is derivable

Due to time constraints it was not possible for me to include the proofs of these theorems into the paper just yet.

These theorems are quite powerful. In combination with the soundness theorems and other conversion rules the following coherence results follow:

Theorem 4.5. *Let* $\vdash_e \Gamma$ *and* $|\Gamma| \vdash A$ *be derivable then there exist an* $\bar{\sigma}$ *such that*

$$\Gamma \vdash_e \bar{\sigma} \quad \textit{and} \qquad\qquad |\Gamma| \vdash |\bar{\sigma}| \equiv \sigma$$

are derivable.

Theorem 4.6. *Let* $\Gamma \vdash_e \sigma$ *and* $|\Gamma| \vdash M : |\sigma|$ *be derivable, then there exists a* $\bar{M}$ *such that*

$$\Gamma \vdash_e \bar{M} : \sigma \quad \textit{and} \qquad\qquad |\Gamma| \vdash |\bar{M}| \equiv M : |\sigma|$$

are both derivable.

These results are worth mentioning, since they relate to [Hof97a, §3.2.3] and also to [PLL16, §3.1].

4.2 Concluding Remarks

We defined a syntax that has explicit coercions, and constructed two maps, the stripping and the lifting. These functions are defined on the raw syntax but for both cases we were able to establish the soundness theorems 3.1 and 3.20.

These results go a long way to prove certain coherence results that help characterizing locally cartesian closed categories as weak models. For these results we assumed that the interpretations defined in [CGH13] work similarly

for this "sub-syntax" (without *explicit substitutions*). Therefore, it would be interesting to adapt the interpretation of [Cur93] or [CGH13] to interpret the syntax of this paper.

Furthermore, the very slim type theory could be extended; as suggested in [CGH13] extensional type theories, intensional type theories as well as homotopy type theories could be investigated by enriching the theory with new constructors and adapting the equality rules appropriately.

The most direct way to proceed with this work is to finish proving the coherence results from section 4 and thereby also investigate how an interpretation in a weak model would appear.

Beyond the technical results, I have found it very insightful and fun to work with Agda, which did not only help organising the proofs but also enforced precision in a very error-prone field of mathematics.

References

[Bru72] Nicolas De Bruijn. Lambda calculus notation with nameless dummies: A tool for automatic formula manipulation, with application to the church-rosser theorem. *Indagationes Mathematicae*, 1972.

[CGH13] P.L Curien, R. Garner, and M. Hofmann. Revisiting the categorical interpretatino of dependent type theory, 2013.

[Cur93] P.L. Curien. Substitution up to isomorphism. *Fundamenta Informaticae 19*, pages 51–85, 1993.

[Hof95] Martin Hofmann. On the interpretation of type theory in locally cartesian closed categories. In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic*, pages 427–441, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[Hof97a] Martin Hofmann. *Extensional Constructs in Intensional Type Theory.* Springer Publishing Company, Incorporated, 1st edition, 1997.

[Hof97b] Martin Hofmann. Syntax and semantics of dependent types. In *Semantics and Logics of Computation*, pages 79–130. Cambridge University Press, 1997.

[ML85] Per Martin-Löf. Intuitionistic Type Theory: predicative part. *Studies in logic and foundations of mathematics*, 80:73–118, 1985.

[PLL16] Chris Kapulkin Peter LeFanu Lumsdaine. The homotopy theory of type theories. *arxiv*, 2016.

[See84] R. Seely. Locally cartesian closed categories and type theory. *Math. Proc. Camb. Phil. Soc. 95, 33-48*, 1984.