Master's Thesis

Towards a Categorical Semantics for the Open Calculus of Constructions

Max Schäfer
xiemaisi@yahoo.de

Overseeing Professor:	Prof. Dr. rer. nat. habil. Steffen Hölldobler, TU Dresden
Supervisor:	Dr. Michael Posegga, TU Dresden
Co-Supervisor:	Dr. Tyng-Ruey Chuang, Academia Sinica (Taiwan)

International MSc Program in Computational Logic

Fakultät Informatik, TU Dresden

March 13, 2007

Declaration

I hereby declare that all the results in this thesis are my own, except where explicitly attributed to someone else. In particular, this thesis is not the result of any form of joint work, and I have not used any other sources or materials besides the indicated.

Dresden, March 13, 2007

Abstract

Stehr's Open Calculus of Constructions combines a type theory in the style of the Calculus of Constructions with concepts from rewriting logic. Stehr's original thesis gives a very simple set-theoretic semantics for the calculus, which abstracts away many of its more peculiar features, and is mainly used to obtain a semantic consistency proof. We present a more sophisticated, categorical semantics in the tradition of Streicher and Jacobs for a restricted variant of Stehr's original system. Specifically, we show that rewriting can be modelled by enriching the traditional, (1-)categorical approach with a 2-category structure, much in the spirit of Meseguer's 2-functorial semantics for rewriting logic. The semantics thus obtained is a good match for the system under consideration, as witnessed by a soundness proof, and could serve as a basis for further investigations into the meta-theoretic properties of related systems.

Contents

1 Introduction

One of the great promises that the theoretical investigation of programming languages holds for the world of computer science is the construction of provably correct software. For the achievement of this goal, two general approaches have been proposed. One is model-checking, in which software is still written in basically the same way as it is today, with similar tools and languages like they are used in current industry practice, and then automatically checked against a mathematical specification to uncover bugs and verify correctness. The other approach is more radical, putting forward new languages and new programming systems, in which programs are correct *by design*, in which programs that do not meet their specification cannot even be created. It is this latter approach with which we will be concerned here.

Again, there are two major research directions. The first one derives its basic ingredients from (constructive) type theory on the one hand and lambda calculi and functional programming on the other hand. By the celebrated Curry-Howard correspondence, which connects propositions with types and their proofs with programs of the corresponding type, we can conceive systems which can be used both as higher-order logics, in which mathematical propositions can be formalized and proved, and typed lambda calculi, in which programs can be written and typechecked. The use of dependent and polymorphic types makes it possible to express the specification of a program completely within the type system, so that the correctness proof of an implementation of a program amounts to nothing more than type checking its code.

The other direction builds on the theory of algebraic specifications and (in spirit at least) logic programming. The idea is to provide the programmer with a framework in which specifications can conveniently be expressed and then directly executed, without having to provide a lower-level implementation, thus eliminating the very problem of checking the implementation against its specification.

Both directions have their strengths and weaknesses; programs written in type theory can often be compiled to lower-level languages for efficient execution, which might not be possible for pure specification languages; but while specifications are often easy to write, implementing them in type theory can be a trying task.

Thus it is natural to ask whether one could not try to unite these two approaches. For example, a programmer might first want to prototype her code using a specification language so as to be able to concentrate on understanding, clarifying, and maybe correcting the specification without having to worry about low-level implementation details, and then set out to write a concrete implementation in type theory, never having to fear that the resulting program does not match the specification.

Different systems have been proposed to achieve this unification, many of them with working implementations. In order to explore their potential, a thorough theoretical investigation seems in order, and an important tool in such an investigation would certainly be a sophisticated semantics.

As a contribution to this, our thesis aims at paving the way for the semantic investigation of systems integrating type theory with concepts of rewriting logic, in particular the Open Calculus of Constructions [20], by showing how to extend traditional categorical semantics of type theories to cover rewriting.

We cannot give a general introduction to type theory and rewriting logic and their respective semantics here; instead we content ourselves with a very cursory and incomplete overview of these two topics. The reader who is already familiar with them should feel free to skip or skim it, while the type theory novice is urged to first consult one of the many introductory books and articles on the subject such as [14] or [9]. A comprehensive introduction to rewriting logic can be found in [15].

1.1 Type Theory and Rewriting Logic

The history of type theory is long and complicated[1]. Here we will only be concerned with its use in computer science: As mentioned above, the Curry-Howard correspondence allows us to understand specifications, expressed through logical propositions, as types, and programs, expressed in a powerful functional programming language, as proofs of such propositions, so that program verification is nothing more than type checking.

Of course, this is most useful if type checking is decidable in the underlying lambda calculus. Indeed, many type theoretic systems, such as most incarnations of Martin-Löf's type theory [14] and Coquand and Huet's Calculus of Constructions (CC) [7], do have decidable type checking[2]. To achieve this, it is necessary that their term language is strongly normalizing, i.e. every typable program terminates.

In practical programming, this restriction can be quite cumbersome: general recursion is not allowed, instead, recursive algorithms have to be expressed purely through recursors. Extensions of CC such as the Calculus of Inductive Constructions (CIC) [8] which lies at the heart of the Coq proof assistant [3] try to mitigate this by providing explicit support for inductive datatypes and structural induction, but writing recursive programs is still by no means trivial.

A very different approach is taken by algebraic specification languages such as Maude [6], which is based on Meseguer's rewriting logic [15]. Rewriting logic axiomatizes an abstract rewriting relation "$\rightarrow$" on top of an underlying equational logic (membership equational logic in the case of Maude). In Maude, the user can express specifications using this framework, and then immediately execute them. General recursion is allowed as a matter of course, leading to often very simple and elegant definitions, but also to potential non-termination. Moreover, Maude is purely first-order, and does not support dependent or polymorphic types.

Stehr's Open Calculus of Constructions, first introduced in his thesis [20], is one of several systems that aim at combining both approaches. Roughly speaking, it can be seen either as a type theory with support for rewriting, or as a rewriting logic on top of a higher-order logic. No particular attention is paid to issues of termination or confluence, but a lot of emphasis is put on the increased flexibility gained by the rewriting capabilities. A peculiar feature of OCC is its distinction between equalities of different "granularity", which overcomes the usual dichotomy between intensional and extensional handling of equality in type theory.

[1] See the introductory chapter of Streicher [21] for a partial overview.

[2] Type inference, however, is undecidable, hence the programmer has to provide more explicit typing annotations than would be necessary in a conventional functional programming language such as ML or Haskell.

In type theories with intensional equality, the equality of two terms is simply viewed as a proposition (and hence a type), for which proof objects can be constructed and passed around. Given some appropriate definition of the natural numbers and addition and two variables x, y of natural number type, it would, for example, be possible to construct some proof object p witnessing the equality $x + y = y + x$; this would be expressed by a typing judgement like $p\colon \texttt{Eq}(x+y, y+x)$. The existence of the proof object p, however, has no bearing on the treatment of the terms $x + y$ and $y + x$, respectively. In particular, a theorem prover for such a type theory will never automatically substitute one by the other (in Coq, for example, this substitution could only be effected by an explicit command to the interactive theorem prover).

Some versions of Martin-Löf's type theory opt for an extensional treatment of equality, where the type checker makes use of available equality proofs. For example, suppose we have a type `NatList` depending on a natural number such that $\texttt{NatList}(n)$ is the type of natural number lists of length n. Furthermore, suppose that we have some expression e of type $\texttt{NatList}(x+y)$; in a type theory with extensional equality, the existence of a proof object p as before would then allow us to conclude that e is also of type $\texttt{NatList}(y + x)$, and can hence be compared with other expressions of type $\texttt{NatList}(y+x)$ without sacrificing type safety. While very convenient, this feature is also quite dangerous, and can lead to non-terminating reduction sequences (and hence undecidable type checking) unless some judiciously chosen constraints are imposed.

OCC's solution builds on the approach of Maude, which distinguishes two kinds of equalities: The first, most fine-grained is the so-called structural equality; structurally equal terms are viewed as operationally indistinguishable, either can be substituted for the other. The second, coarser one, is known as computational equality; computational equalities can be viewed as reduction rules, in which the left hand side can be turned into the right hand side, but not the other way around. This distinction allows the user more control over the reduction process.

A typical example of the application of these two equalities is shown in Figure 1 (adapted from [20]). Here, computational equalities (written $!!(- = -)$) are used to express the usual reduction rules for the definition of addition, while structural equalities (written $||(- = -)$) are used to express commutativity and associativity. The reader interested in the pragmatics of these different equalities should refer to Stehr's thesis for further examples.

OCC adds a third and fourth kind of equality, assertional equality and propositional equality, which are closer to the intensional equality of Coq, but we will not treat them in detail in this thesis.

1.2 Semantics of Type Theory and Rewriting Logic

Although a classical set-theoretic semantics can be given for many flavors of type theory (see, e.g., [22] for a set-theoretic semantics of CC), such semantics tend not to be very useful for meta-theoretic investigations: As was proved by Reynolds in [17], polymorphic lambda calculi (such as CC and OCC) can only be given a so-called proof irrelevance semantics, in which all propositions are interpreted as singleton sets; their proof objects must then all be interpreted as the same semantic object. This precludes any finer analysis of such proof objects beyond the mere acknowledgement of their existence.

$$
\begin{aligned}
\mathtt{N} &: \mathtt{Type} \\
\mathtt{0} &: \mathtt{N} \\
\mathtt{suc} &: \mathtt{N} \to \mathtt{N} \\
\\
\mathtt{plus} &: \mathtt{N} \to \mathtt{N} \to \mathtt{N} \\
\\
\mathtt{pluscomm} &: \Pi i, j : \mathtt{N}.||((\mathtt{plus}\ i\ j) = (\mathtt{plus}\ j\ i)) : \mathtt{N} \\
\mathtt{plusass} &: \Pi i, j, k : \mathtt{N}.||((\mathtt{plus}\ i(\mathtt{plus}\ j\ k)) = (\mathtt{plus}\ (\mathtt{plus}\ i\ j)k)) \\
\\
\mathtt{plusz} &: \Pi i : \mathtt{N}.!!((\mathtt{plus}\ i\ \mathtt{0}) = i) : \mathtt{N} \\
\mathtt{pluss} &: \Pi i, j : \mathtt{N}.!!((\mathtt{plus}\ i(\mathtt{suc}\ j)) = (\mathtt{suc}\ (\mathtt{plus}\ i\ j))) : \mathtt{N}
\end{aligned}
$$

Figure 1: A specification of the natural numbers and addition

Category theory provides a way out of this dilemma. Using categorical language, one can, for many type theories, give a succinct definition of a very broad class of structures in which the type theory can be interpreted. To just name two of the many works on categorical semantics of type theory, Jacobs' thesis [10] treats a rather wide range of different type theories, while Streicher's thesis, later expanded into the monograph [21], contains a very thorough and detailed presentation of a semantics for the Calculus of Constructions. The class of structures used in his account contains structures which do allow for the distinction between proof objects, and even a term model, so that a completeness proof can be obtained. More interestingly, Streicher uses his semantics for further investigations, proving results such as uniqueness of typing and of product formation, as well as some independence results.

For rewriting logic there also is a quite simple categorical semantics, presented for example in [15] and [13]. The underlying equational logic used in these accounts is untyped, hence the categorical setup is quite different from the one used in the semantics of type theories.

Our aim is to extend and integrate these existing approaches to arrive at a categorical semantics for systems combining both type theory and rewriting logic, with the hope that such a semantics can then be used in the further study of the systems, probably obtaining similarly interesting results as in the case of the Calculus of Constructions.

1.3 Outline

The Open Calculus of Constructions is a very large and complex system. Introducing its semantics in one swoop would be quite a tour de force; we will not do so. Rather, we start from a very simple, one might say trivial, subsystem, which we dub OCC_0. Although not very useful as a type theory, it will allow us to introduce the basic ideas behind our categorical semantics. We start by giving its abstract syntax and formal system, then proceed to define the corresponding notion of a semantic structure and specify how to interpret expressions of the

language in such a structure. A soundness proof shows that this interpretation makes sense. All results in this section are well-known in the literature.

We then introduce an extended system, called OCC_1, which extends OCC_0 with product and equality types, yielding a system similar in power to the Calculus of Constructions. We show how product and equality types can be interpreted categorically, and give a soundness proof extending the proof for OCC_0. Again, the approach taken and the results obtained in this section are anything but new.

With the third system, OCC_2, we enter new territory: it supports rewriting. We show how to interpret rewriting (2-)categorically, and mention the connection to Meseguer's 2-functorial semantics for Rewriting Logic [13]. As before, a soundness proof is provided on the basis of the soundness proof for OCC_1.

Our final system, OCC_3, is already quite close to Stehr's original formulation of OCC. We show how the newly introduced feature of computational equality corresponds to 2-categorically lax type constructors, and give one final soundness proof.

Taken together, these two chapters, which are the heart of this thesis, present the main result of our work: They demonstrate how to enrich a categorical semantics for a traditional type theory with additional structure such that rewriting and non-structural equalities can be modelled.

As a byproduct of our investigations, we will see that the CINNI calculus of explicit substitutions, which is employed by OCC and its variants, is closely connected to the categorical semantics in that its three basic constructors (replacement, shifting, and lifting) correspond exactly to three special kinds of arrows in the semantics[3].

We conclude with a discussion of some of the remaining features of full OCC not covered by our semantics as well as other differences between Stehr's system and ours. Finally, we give some pointers to related work and possible directions of future research.

Acknowledgements

This thesis could not have reached its present form without the help of many people. I want to thank Dr. Chuang for giving me the opportunity to write my thesis at his laboratory, for assistance in matters small and large, and unwavering support throughout; Prof. Hölldobler for making it all possible; and Dr. Posegga for patient support over 8000 kilometers and seven timezones. Mark-Oliver Stehr was swift and helpful in answering all my questions about the Open Calculus of Constructions. My colleagues at Academia Sinica provided a hospitable and cheerful atmosphere. Dr. Shin-Cheng Mu and Dr. Bow-Yaw Wang gave valuable comments on different stages of my work. And finally, my most heartfelt thanks go to Shixin, who helped and supported me in more ways than she knows herself.

[3] A similar result was, for a different substitution calculus, proved by P.-L. Curien [1].

2 OCC_0: The Algebraic Substrate

We begin our investigations of the categorical semantics of the Open Calculus of Constructions with a very simple system, OCC_0, which focuses entirely on OCC's "algebraic substrate", almost without any type constructors. While certainly not a very powerful, or even very useful, system, it allows us to introduce the basic ideas on which our semantics is based. Systems like OCC_0 have been a staple of categorical semantics since its early days, and there are no novel results to be expected in this section.

Although in a rudimentary way, OCC_0 already deals with the same three levels of entities as the later systems, namely universes, types, and atoms[4]. Atoms belong to types which belong to universes. For example, the natural number 0 is an atom belonging to the type `Nat` of natural numbers, which in turn belongs to the universe `Type` of types (of course, this example is not expressible in such a simple-minded type theory as OCC_0). Note that even in OCC_0 these levels are by no means mutually disjoint: both universes and types can also be seen as atoms, and universes can also be types.

2.1 Syntax

OCC_0 is a family of type systems parametrized by a countable set $\mathcal{S}$ of universe constants and an acyclic partial function $\mathcal{A}\colon \mathcal{S} \rightsquigarrow \mathcal{S}$ describing universe containment. The set $\mathcal{S}$ has to contain a designated propositional universe Prop. These data are gathered into a triple $\Sigma = (\mathcal{S}, \mathrm{Prop}, \mathcal{A})$, called an *$OCC_0$-signature*.

Given such a signature Σ, the set of *OCC_0 pseudoterms* $\mathcal{T}_0^\Sigma$, or simply $\mathcal{T}_0$ when the signature is understood from context, is defined by the following grammar:

$$\mathcal{T}_0 ::= \mathcal{S} \mid \mathcal{V}^{\mathbb{N}} \mid \mathrm{unit} \mid *$$

where $\mathcal{V}$ is a fixed set of syntactic variable names and $\mathbb{N}$ is the set of natural numbers (including 0).

Thus, a pseudoterm is either a universe constant (such as Prop), an indexed variable, the unit type, or its inhabitant $*$. Intuitively, the index of a variable indicates how many bindings of the variable to skip; for example, the indexed variable x^0 refers to the innermost binding of x in the current context, whereas x^1 refers to the binding beyond it. This notation, a generalization of de Bruijn's index notation, comes from the CINNI calculus of explicit substitutions [19], about which we will have more to say below. Note that we write the index of a CINNI variable as superscript (as in x^0) instead of Stehr's subscript notation x_0.

A unit type does not occur in Stehr's formulation of OCC, but we will see that it has a very useful place in the categorical semantics.

The set of pseudoterms contains both terms which we perceive to stand for types and terms which stand for atoms. If we want to emphasize that a certain pseudoterm stands for a type, we will sometimes call it a *pseudotype*, but this is a purely rhetoric distinction.

[4] In most other accounts, entities of this latter level are called objects or elements. Both of these terms, however, have fixed and quite different meanings in category theory, so we will generally avoid them.

Definition 2.1. The size of a pseudoterm M, written $|M|$, is defined as usual:

1. for $s \in \mathcal{S}$: $|s| = 1$
2. for $x \in \mathcal{V}$, $n \in \mathbb{N}$: $|x^n| = 1$
3. $|\,\mathrm{unit}\,| = |\,*\,| = 1$

The following two definitions will be the same for all of the systems we describe:

Definition 2.2. A *pseudocontext* is a list of *entries* of the form $x\colon A$, with *label* $x \in \mathcal{V}$ and $A \in \mathcal{T}_0$. The empty context is written $\diamond$.

Definition 2.3. The size of a pseudocontext Γ, written $|\Gamma|$ is defined as

$$|x_1\colon A_1, \ldots, x_n\colon A_n| = |A_1| + \cdots + |A_n|$$

In particular, of course, $|\diamond| = 0$.

Syntactic equality of pseudoterms and pseudocontexts is denoted by the symbol $\equiv$.

2.2 TCINNI-OCC$_0$

An important aspect of OCC (and all its subsystems considered here) is that it "takes names seriously". Instead of the rather informal freshness conditions and implicit renaming assumptions commonly encountered in more traditional calculi, its use of CINNI (the "Calculus of Indexed Names and Named Indices") provides a rigorous account of the syntactic operations of renaming and substitution. For our purposes, the most important contribution of CINNI is that it allows to define complicated syntactic operations (such as capture avoiding substitution under a binder) in a syntax-directed way without having to assume alpha equivalence. The reader who is not familiar with CINNI or some other explicit substitution calculus may want to refer to [19] for a thorough introduction.

CINNI's approach fits very well with a categorical semantics, in which much attention has to be paid to such details[5], and allows us to obtain a clear correspondence between CINNI operations in the syntax, and categorical operations in the semantics.

We use a slightly modified version of CINNI with type annotations, instantiated for the syntax of OCC$_0$; this system is referred to as TCINNI-OCC$_0$("Typed CINNI for OCC$_0$").

The set Θ of TCINNI-substitutions over a set Ty of types, a set Tm of terms, and a set $\mathcal{V}$ of names is given by the following grammar:

$$\Theta ::= [\mathcal{V} := \mathrm{Tm}\colon \mathrm{Ty}] \mid \uparrow_{(\mathcal{V}\colon \mathrm{Ty})} \mid \Uparrow_{(\mathcal{V}\colon \mathrm{Ty})}\Theta$$

TCINNI-OCC$_0$ is simply TCINNI with $\mathrm{Ty} = \mathrm{Tm} = \mathcal{T}_0$.

[5] As Jacobs [10] remarks: "We like to see [categorical semantics of type theory] as description at the assembly level: categorical formulations require far more attention for details, like substitution and coherence conditions".

Intuitively, a substitution of the form $[x := M\colon A]$, called *replacement*, replaces every occurrence of x^0 by the pseudoterm M, while decreasing the indices of other occurrences of x. M is supposed to have type A, but this fact has no influence on the substitution behavior. Replacements are used to model beta-reduction, which can (in informal syntax) be written as $(\lambda x\colon A.M)N \to_\beta [x := N\colon A]M$.

A substitution of the form $\uparrow_{(x\colon A)}$, called *shifting*, increases the index of every occurrence of x by one. This is needed when one wants to make sure that a term cannot refer to a certain binding, as for example in η-expansion (although this particular example does not occur in OCC), which we could formulate as $M \to_\eta (\lambda x\colon A.\uparrow_{(x\colon A)} Mx)$. Again, the type annotation A is purely for bookkeeping.

The behavior of a *lifting* substitution like $\Uparrow_{(x\colon A)}\vartheta$, with ϑ being another substitution, is somewhat harder to explain intuitively (the precise definition is given below). Roughly speaking, $\Uparrow_{(x\colon A)}\vartheta$ performs capture-avoiding substitution under a binding for x. This will become more important in the next chapter when we extend our system by an abstraction mechanism.

Definition 2.4. The action of substitutions on OCC_0-pseudoterms is defined as follows:

$$
\begin{array}{lcll}
[x := M\colon A]x^0 & = & M & \\
[x := M\colon A]x^{m+1} & = & x^m & \\
[x := M\colon A]y^n & = & y^n & \text{where } x \not\equiv y \\
 & & & \\
\uparrow_{(x\colon A)} x^n & = & x^{n+1} & \\
\uparrow_{(x\colon A)} y^m & = & y^m & \text{where } x \not\equiv y \\
 & & & \\
(\Uparrow_{(x\colon A)}\vartheta)x^0 & = & x^0 & \\
(\Uparrow_{(x\colon A)}\vartheta)x^{n+1} & = & \uparrow_{(x\colon \vartheta A)}(\vartheta x^n) & \\
(\Uparrow_{(x\colon A)}\vartheta)y^m & = & \uparrow_{(x\colon \vartheta A)}(\vartheta y^m) & \text{where } x \not\equiv y \\
 & & & \\
\vartheta s & = & s & \\
\vartheta\,\mathrm{unit} & = & \mathrm{unit} & \\
\vartheta * & = & * &
\end{array}
$$

The extra type annotations play no role when applying a substitution to a term (and we will omit them when no ambiguity can arise); so we get the following theorem for free:

Theorem 2.5. *For every TCINNI-substitution ϑ and every OCC_0-pseudoterm M, the expression ϑM reduces by the above rules to a unique, well-defined pseudoterm N.*

Proof. This was shown by Stehr in [19] for CINNI (without type annotations), but since the substitution behavior is the same, the result immediately carries over to the case of TCINNI. □

While the set of TCINNI-substitutions will essentially be the same for all our systems, every extension of the pseudoterm syntax will make it necessary to extend Definition 2.4 by new clauses.

It is easy to see that shifting does not change the size of a term:

Lemma 2.6. *For any variable x, any pseudotype A and any pseudoterm M, we have $|\uparrow_{(x:\ A)} M| = |M|$*

Proof. Induction on the structure of M. □

We also define the size of a substitution.

Definition 2.7. The *size* of a substitution ϑ, written $|\vartheta|$ is inductively defined as follows:

1. for $x \in \mathcal{V}$, $N, A \in \mathcal{T}_0$: $|[x := N : A]| = |N|$
2. for $x \in \mathcal{V}$, $A \in \mathcal{T}_0$: $|\uparrow_{(x:\ A)}| = 1$
3. for $x \in \mathcal{V}$, $A \in \mathcal{T}_0$, $\vartheta' \in S$: $|\Uparrow_{(x:\ A)} \vartheta'| = |\vartheta'| + 1$

Lemma 2.8. *For a substitution ϑ, a variable $y \in \mathcal{V}$, and an index $n \in \mathcal{N}$, we have $|\vartheta y^n| \leq |\vartheta|$.*

Proof. Induction on the structure of ϑ.

1. $\vartheta \equiv [x := M : A]$:

   By definition, $|\vartheta| = |M|$; if $y^n \equiv x^0$, then $\vartheta y^n \equiv M$, otherwise it is an indexed variable. In either case, the claim is fulfilled.

2. $\vartheta \equiv \uparrow_{(x:\ A)}$:

   ϑy^n is an indexed variable, hence the claim is fulfilled.

3. $\vartheta \equiv \Uparrow_{(x:\ A)} \vartheta'$:

   If $y^n \equiv x^0$, the result is trivial. Otherwise, it follows by induction hypothesis and Lemma 2.6.

□

2.3 Formal System

Definition 2.9. There is only a single form of *judgement* in OCC_0, namely

$$A : B$$

where A is a pseudoterm, and B a pseudotype. Put into a pseudocontext Γ, the judgement-in-context $\Gamma \vdash A : B$, means that under Γ, A is of type B.

A *valid OCC_0 judgement* is any judgement that can be derived by the following rules of inference:

$$\text{(Ax)}\ \frac{}{\Gamma \vdash s_1 : s_2}\ , \mathcal{A}(s_1) = s_2$$

$$\text{(Start1)}\ \frac{}{\Gamma, x : A \vdash x^0 :\ \uparrow_{(x:\ A)} A}$$

$$\text{(Start2)}\ \frac{\Gamma \vdash x^i : A}{\Gamma, x : B \vdash x^{i+1} :\ \uparrow_{(x:\ B)} A}$$

$$\text{(Start3)}\ \frac{\Gamma \vdash y^m : A}{\Gamma, x : B \vdash y^m : \uparrow_{(x:\ B)} A}\ , x \not\equiv y$$

$$\text{(UnitForm)}\ \frac{}{\Gamma \vdash \mathrm{unit} : \mathrm{Prop}}$$

$$\text{(UnitIntro)}\ \frac{}{\Gamma \vdash * : \mathrm{unit}}$$

The following easy result (often called substitution lemma in the literature on typed lambda calculi) indicates a connection between the structure of contexts and the application of substitutions, which will become very important in the motivation of the categorical semantics below:

Lemma 2.10. *1. For any pseudocontext Γ, name $x \in \mathcal{V}$, pseudoterms M, N, and pseudotypes A, B, the following rule is admissible in OCC_0:*

$$\frac{\Gamma, x : A \vdash M : B \qquad \Gamma \vdash N : A}{\Gamma \vdash [x := N : A]M : [x := N : A]B}$$

2. For any pseudocontext Γ, name $x \in \mathcal{V}$, pseudoterm M, and pseudotypes A, B, the following rule is admissible in OCC_0:

$$\frac{\Gamma \vdash M : B}{\Gamma, x : A \vdash \uparrow_{(x:\ A)} M : \uparrow_{(x:\ A)} B}$$

Proof. The results are proved by induction on the possible derivations of $\Gamma, x : A \vdash M : B$ and $\Gamma \vdash M : B$, respectively.

1. If the last rule used was (Ax), (UnitForm), or (UnitIntro), the claim immediately follows from the induction hypothesis.

   For (Start1), observe that we must have $M \equiv x^0$ and $B \equiv \uparrow_{(x:\ A)} A$, hence the conclusion is nothing but the second premise.

   Cases (Start2) and (Start3) are very similar and follow by the definition of substitution application.

2. Again, cases (Ax), (UnitForm) and (UnitIntro) are trivial. The other cases are instances of (Start2) and (Start3).

□

2.4 OCC_0 Structures

We are now going to describe a class of structures that OCC_0 can be interpreted in. Before diving into the precise definitions, we take a bird's eye view of the semantics, and try to motivate the different categorical features comprising an OCC_0 structure.

Our underlying categorical setup is based on Jacobs' comprehension categories with unit [10], which are an alternative formulation of Ehrhard's D-categories.

The basic realization underlying this approach is that in type theory, types only ever occur "in context", i.e. to fully describe them we have to provide a context that specifies the types of all free variables occurring in the type. This is

reflected in the semantics, where we have one category $\mathcal{E}$ whose objects interpret types and universes, and another category $\mathcal{C}$ whose objects interpret contexts. A "parent" functor p assigns to every type in $\mathcal{E}$ its associated context in $\mathcal{C}$. All types of a certain context C can be assembled into a subcategory of $\mathcal{E}$, called the fiber over C.

But this rather static setting is not enough. If we have a type A in context Γ, then there should be some sort of connection between the interpretation of A (which we will denote as $[\![\Gamma \vdash\colon A]\!]$ to emphasize its dependence on Γ) and the interpretation of the extended context $\Gamma, x\colon A$ (denoted $[\![\Gamma, x\colon A]\!]$). In Streicher's semantics, for example, these two are interpreted as the same categorical object.

This is impossible in our setup, since $[\![\Gamma \vdash\colon A]\!]$ is an object of $\mathcal{E}$, whereas $[\![\Gamma, x\colon A]\!]$ is an object of $\mathcal{C}$. Instead we require the existence of a functor $\{-\}$, called comprehension functor, which maps $[\![\Gamma \vdash\colon A]\!]$ to $\{[\![\Gamma \vdash\colon A]\!]\}_{[\![x]\!]} = [\![\Gamma, x\colon A]\!]$.

In including the "semantic name" $[\![x]\!]$ in the definition, we somewhat depart from traditional approaches which ignore names and force alpha equivalent terms to be interpreted as the same semantic object. Since OCC's basic philosophy is to avoid such identifications, we feel justified to respect names in the semantics. Of course, the semantic names do not have to be the same as the syntactic ones (i.e., the set $\mathcal{V}$ introduced before), and one can, in fact, regain name irrelevance by simply mapping all syntactic names to the same semantic name.

Because types are interpreted as objects of $\mathcal{E}$ and atoms belong to types, we interpret atoms as categorical elements of the corresponding objects in $\mathcal{E}$, i.e. arrows out of terminal objects. But like types, atoms depend on a context, so there is not *one* terminal object in $\mathcal{E}$ out of which all these arrows come, but one for every fiber. Thus, if we can derive the judgement $\Gamma \vdash M\colon A$, then we would expect that Γ, M, and A are interpreted in such a way that the interpretation of M is an arrow from $\mathbf{1}([\![\Gamma]\!])$, the terminal object in the fiber above $[\![\Gamma]\!]$, to $[\![\Gamma \vdash\colon A]\!]$.

There is a slight catch having to do with the double role played by types: In the above example, the type A in turn belongs to some universe s. Thus it should also be interpreted as a categorical element of $[\![\Gamma \vdash\colon s]\!]$ – but this is impossible, since we already decided to interpret it as an *object* of $\mathcal{E}$, not a morphism. The solution is to give A two different interpretations, depending on whether we want to view it as a type or as a term. The first interpretation is denoted as $[\![\Gamma \vdash\colon A]\!]$ (which is an object of $\mathcal{E}$), the second one as $[\![\Gamma \vdash A]\!]$ (which is a morphism of $\mathcal{E}$). Obviously, these two semantic entities should be related, so we need a mapping $\mathcal{U}$ that maps the latter to the former.

A second cornerstone of the categorical interpretation of type theory are context morphisms, which are the semantic counterpart of the TCINNI substitutions introduced above. Looking at Lemma 2.10, we see that, intuitively speaking, the application of a substitution $\vartheta_1 = [x := N\colon A]$ transports a valid judgement in context $\Gamma' = (\Gamma, x\colon A)$ to a valid judgement in context Γ, and conversely the application of a substitution $\vartheta_2 = \uparrow_{(x\colon A)}$ transports a valid judgement in context Γ to a valid judgement in context Γ'.

Since in the first case the term N is a term in context Γ (and not in context Γ'), we see the substitution ϑ_1 as a context morphism from Γ to Γ' and ϑ_2 as a context morphism from Γ' to Γ. These morphisms should now be interpreted

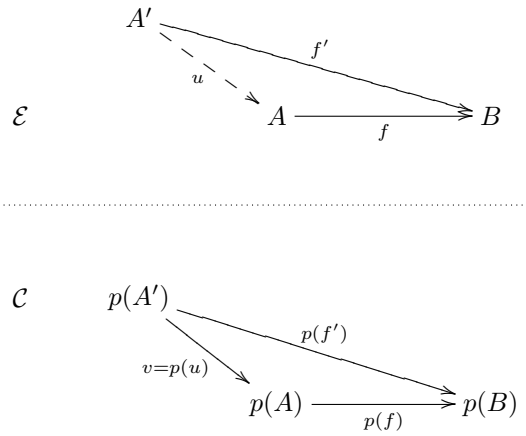


Figure 2: Cartesian Arrow

as arrows t_1, t_2 between the interpretations C and C' of Γ and Γ' in $\mathcal{C}$, in such a way that t_1 can "pull back" any object and arrow of the fiber over C' into the fiber over C and the other way around for t_2. This is achieved by the categorical construction of a fibration introduced below.

To make these considerations precise, we start with some technical definitions which are adapted from [10], but originate with Grothendieck.

Let p be a functor from a category $\mathcal{E}$ to another category $\mathcal{C}$.

Definition 2.11. We picture $\mathcal{E}$ as being located "above" $\mathcal{C}$, and hence say that an object A of $\mathcal{E}$ is *above* $C \in \mathrm{Ob}(\mathcal{C})$ if $p(A) = C$, and likewise for arrows. An arrow above an identity arrow is called *vertical*.

The *fiber* over an object $C \in \mathcal{C}$, written $\mathcal{E}_C$, is the subcategory of $\mathcal{E}$ containing all objects above C and the vertical arrows between them.

Definition 2.12. An arrow $f\colon A \to B$ in $\mathcal{E}$ is called *cartesian* if, for any arrow $f'\colon A' \to B$ in $\mathcal{E}$ and an arrow $v\colon p(A') \to p(A)$ in $\mathcal{C}$ such that $p(f') = p(f) \circ v$, there is a unique arrow $u\colon A' \to A$ with $p(u) = v$ and $f' = f \circ u$ (see Figure 2).

Definition 2.13. A functor p from $\mathcal{C}$ to D is called a *cloven fibration* (or a fibration with cleavage) if for every object A of $\mathcal{C}$ and every morphism $f\colon B \to p(A)$ in $\mathcal{D}$ there is a chosen cartesian arrow (called the *cartesian lifting* of f) $\overline{f}(A)$ above f. The domain of this arrow is written $f^*(A)$.

Definition 2.14. A cloven fibration p is called *split* if

1. $\mathrm{id}^*(A) = A$
2. $\overline{\mathrm{id}}(A) = \mathrm{id}_A$
3. $(f \circ g)^*(A) = g^*(f^*(A))$
4. $\overline{f \circ g}(A) = \overline{f}(A) \circ \overline{g}(f^*(A))$

A very basic construction in categorical semantics are the so-called reindexing functors, which will be used to model application of substitution.

Definition 2.15. Given a cloven fibration p as above, every arrow $f\colon D \to C$ in $\mathcal{C}$ induces a functor f^* from $\mathcal{E}_C$ to $\mathcal{E}_D$, called a *reindexing* or *pullback functor*. On objects, it is defined by the cleavage. For a vertical arrow $k\colon A \to B$, we can see that both $k \circ \overline{f}(A)$ and $\overline{f}(B)$ are above f and the latter is cartesian. Thus, there must be a unique vertical arrow from $f^*(A)$ to $f^*(B)$, which we will call $f^*(k)$, such that $\overline{f}(B) \circ f^*(k) = k \circ \overline{f}(A)$.

Based on these definitions, we can now describe the class of structures we can interpret OCC_0 in.

Definition 2.16. An OCC_0-structure $\mathcal{M}$ is given by

- a category $\mathcal{C}$ called the base category ("contexts and substitutions"), with a terminal object $\mathbf{1}_\mathcal{C}$
- a category $\mathcal{E}$ called the total category ("types and terms")
- a non-empty set $\mathcal{N}$ ("semantic names")
- a split fibration p from $\mathcal{E}$ to $\mathcal{C}$ ("parent functor")
- an $\mathcal{N}$-indexed family of functors $(\{-\}_n)_{n\in\mathcal{N}}$ from $\mathcal{E}$ to $\mathcal{C}$ ("comprehension")
- for every $x \in \mathcal{N}$, a natural transformation $P_x\colon \{-\}_x \overset{\bullet}{\to} p$; for $A \in \mathcal{E}$, its component $P_x(A)$ is called a *display map*
- a subcategory $\mathcal{S}_\mathcal{C}$ of $\mathcal{E}$ ("universes"); each object $s \in \mathcal{S}_\mathcal{C}$ is above $\mathbf{1}_\mathcal{C}$, hence we can form the object $C^*s := !_C^*(s)$ for any $C \in \mathcal{C}$
- for every $x \in \mathcal{N}$, a functor $\mathbf{1}_x$ that is left-adjoint to $\{-\}_x$; the counit $\varepsilon_\mathbf{1}$ is required to be above P_x and $\mathbf{1}_x$'s action on objects must not depend on x; for any $C \in \mathcal{C}$ and $f\colon D \to C$, we furthermore need to ensure that $f^*(\mathbf{1}(C)) = \mathbf{1}(D)$
- for every $C \in \mathcal{C}$, a mapping $\mathcal{U}_C\colon (\bigcup_{s\in\mathcal{S}_\mathcal{C}} \mathcal{E}_C(\mathbf{1}(C), C^*s)) \to \mathcal{E}_C$ from elements of C^*s, for any $s \in \mathcal{S}$, to objects of $\mathcal{E}$; this mapping is required to be stable under substitution, i.e. for $f\colon D \to C$ and $m \in \mathcal{E}_C(\mathbf{1}(C), C^*s)$ for some $s \in \mathcal{S}$ we have $f^*(\mathcal{U}_C(m)) = \mathcal{U}_D(f^*(m))$
- for $C \in \mathcal{C}$ an arrow $\hat{\mathbf{1}}(C)\colon \mathbf{1}(C) \to C^*\mathrm{Prop}$ such that $\mathcal{U}_C(\hat{\mathbf{1}}(C)) = \mathbf{1}(C)$, and for $f\colon C \to D$ we have $f^*(\hat{\mathbf{1}}(D)) = \hat{\mathbf{1}}(C)$

The objects of $\mathcal{S}_\mathcal{C}$ are used to interpret universes. Universes do not depend on a context and hence they are interpreted as objects above the terminal object $\mathbf{1}(\mathcal{C})$. For a universe s we define the full subcategory $\mathcal{E}(s)$ of $\mathcal{E}$ to contain all objects A such that there is a $C \in \mathcal{C}$, $m\colon \mathbf{1}(C) \to C^*s$ with $\mathcal{U}_C(m) = A$; more succinctly, $\mathcal{E}(s)$ contains all objects corresponding to types in universe s.

Note that these subcategories $\mathcal{E}(s)$ are not required to be disjoint, i.e. the same type-in-context can belong to different universes; this is inevitable if we want to model cumulative universe hierarchies.

Terms are interpreted as elements of objects in $\mathcal{E}$, i.e. as vertical arrows out of $\mathbf{1}(C)$ for some object C. In [10], it is shown that $\mathbf{1}(C)$ is a terminal object in $\mathcal{E}_C$, so this interpretation matches the intuition that elements are arrows out of the terminal object.

In [21], for example, terms are modelled as sections of display maps, instead. We reserve sections to model substitution (see below), but in a sense a term M of type A corresponds to a substitution of the form $[x := M\colon A]$. This correspondence is expressed by the following result adapted from [10].

Lemma 2.17. *For $A \in \mathcal{E}$ above $C \in \mathcal{C}$ and $x \in \mathcal{N}$, there is a bijective correspondence between sections of $P_x(A)$ and vertical morphisms from $\mathbf{1}(C)$ to A.*

Proof. Since $\mathbf{1}_x \dashv \{-\}_x$, we have $\mathcal{E}(\mathbf{1}_x(-), -) \cong \mathcal{C}(-, \{-\}_x)$; each $m\colon \mathbf{1}(C) \to A$ corresponds to $\{m\}_x \circ \eta_{\mathbf{1}}(C)\colon C \to \{A\}_x$, and each $n\colon C \to \{A\}_x$ to $\varepsilon_{\mathbf{1}}(A) \circ \mathbf{1}_x(n)\colon 1_C \to A$. It remains to show that the one's being a section implies the other's being vertical and vice versa.

So assume m is vertical, i.e. $p(m) = \mathrm{id}_C$. We need to show that $\{m\}_x \circ \eta_{\mathbf{1}}(C)$ is a section of $P_x(A)$. Indeed,

$$\begin{aligned}
&P_x(A) \circ \{m\}_x \circ \eta_{\mathbf{1}}(C)\\
&= p(m) \circ P_x(\mathbf{1}(C)) \circ \eta_{\mathbf{1}}(C)\\
&= P_x(\mathbf{1}(C)) \circ \eta_{\mathbf{1}}(C)\\
&= p(\varepsilon_{\mathbf{1}}(\mathbf{1}(C))) \circ p(\mathbf{1}_x(\eta_{\mathbf{1}}(C)))\\
&= p(\varepsilon_{\mathbf{1}}(\mathbf{1}(C)) \circ \mathbf{1}_x(\eta_{\mathbf{1}}(C)))\\
&= \mathrm{id}_C
\end{aligned}$$

For the other direction, assume n is a section of $P_x(A)$, i.e $P_x(A) \circ n = \mathrm{id}_C$. We need to show that $\varepsilon_{\mathbf{1}}(A) \circ \mathbf{1}_x(n)$ is vertical. This is also not difficult:

$$\begin{aligned}
&p(\varepsilon_{\mathbf{1}}(A) \circ \mathbf{1}_x(n))\\
&= p(\varepsilon_{\mathbf{1}}(A) \circ \overline{n}(\mathbf{1}(\{A\}_x)))\\
&= p(\varepsilon_{\mathbf{1}}(A)) \circ p(\overline{n}(\mathbf{1}(\{A\}_x)))\\
&= P_x(A) \circ n\\
&= \mathrm{id}_C
\end{aligned}$$

□

Notice that this holds for *every* $x \in \mathcal{N}$ (a term M corresponds to different substitutions, say $[x := M\colon A]$, $[y := M\colon A]$, ...), but for a fixed name, the correspondence is one-to-one.

We use a special notation to indicate the passage from vertical morphisms to sections (the other direction is much less frequently used): If m is a vertical morphism, then $\mathrm{Sect}_x(m)$ is the corresponding section.

Definition 2.18. For any object $A \in \mathcal{E}$, there is a useful *injection arrow* $\delta_{x,A}\colon \mathbf{1}(\{A\}_x) \to P_x(A)^*(A)$ which is vertical[6], i.e. it corresponds to a section of $P_y(P_x(A)^*(A))$ for any $y \in \mathcal{N}$. This arrow is obtained from the unique

[6]For this reason, we would rather avoid calling it by its common name of "diagonal arrow".

lifting property; for observe that both $\overline{P_x(A)}(A)$ and $\varepsilon_{\mathbf{1}}(A)$ are arrows from $\mathbf{1}(\{A\}_x)$ to A, both of which are above $P_x(A)$. Hence there must be such an arrow δ_A, which arises as the lifting of $\mathrm{id}_{\{A\}_x}$.

Finally, we introduce a third kind of special arrows (besides display maps and sections):

Definition 2.19. We write $q(f, x, A)$ for $\{\overline{f}(A)\}_x$. This arrow is called a *lifting* of f along $P_x(A)$ (not to be confused with the cartesian lifting mentioned above).

Adapting a proof in [10], it can be shown that for any $f\colon D \to C$ in $\mathcal{C}$ and $A \in \mathcal{E}$ above C, the following is a pullback diagram:

$$\begin{array}{ccc} \{f^*(A)\}_x & \xrightarrow{q(f,x,A)} & \{A\}_x \\ {\scriptstyle P_x(f^*(A))}\downarrow & & \downarrow{\scriptstyle P_x(A)} \\ D & \xrightarrow[f]{} & C \end{array}$$

The injection arrow has the interesting property that it acts as a right neutral element for reindexing along sections:

Lemma 2.20. *For $A \in \mathcal{E}$ above $\mathcal{C}$ and a section N of $P_x(A)$, we have*

$$N^*(\delta_{x,A}) = \mathrm{Sect}^{-1}(N)$$

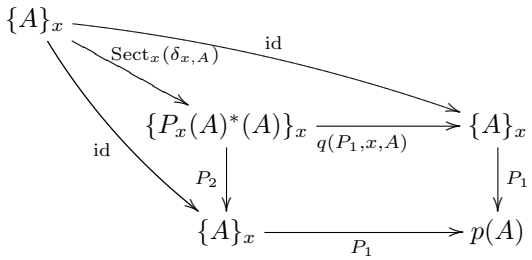
Proof.

$$\begin{aligned} N^*(\delta_{x,A}) &= \overline{P_x(A)}(A) \circ \overline{N}(P_x(A)^*(A)) \circ N^*(\delta_{x,A}) \\ &= \varepsilon_{\mathbf{1}}(A) \circ \overline{N}(\mathbf{1}(\{A\}_x)) \\ &= \mathrm{Sect}^{-1}(N) \end{aligned}$$

□

Furthermore, its section equivalent $\mathrm{Sect}_x(\delta_{x,A})$ turns out to be the mediating arrow in a special pullback situation (adapted from [10]):

Lemma 2.21. *Let $A \in \mathcal{E}$ be an object of the total category. Then $\mathrm{Sect}_x(\delta_{x,A})$ is the mediating arrow in the following pullback situation (where we write P_1 for $P_x(A)$ and P_2 for $P_x(P_x(A)^*(A))$)*



Proof. We must show that the diagram commutes. Indeed,

$$\begin{aligned} q(P_1, x, A) \circ \mathrm{Sect}_x(\delta_{x,A}) &= \{\overline{P_1}(A)\}_x \circ \{\delta_{x,A}\}_x \circ \eta_{\mathbf{1}}(\{A\}_x) \\ &= \{\overline{P_1}(A) \circ \delta_{x,A}\}_x \circ \eta_{\mathbf{1}}(\{A\}_x) \\ &= \{\varepsilon_{\mathbf{1}}(A)\}_x \circ \eta_{\mathbf{1}}(\{A\}_x) \\ &= \mathrm{id}_{\{A\}_x} \end{aligned}$$

and

$$P_2 \circ \mathrm{Sect}_x(\delta_{x,A}) = \mathrm{id}_{\{A\}_x}$$

since it is a section. □

2.5 The Interpretation Function

Let an OCC_0-signature $\Sigma = (\mathcal{S}, \mathrm{Prop}, \mathcal{A})$ be given. To model the corresponding OCC_0-instance, we need an OCC_0 structure $\mathcal{M}$ such that

- for every universe $s \in \mathcal{S}$, there is an object $\ulcorner s \urcorner \in \mathcal{S}_{\mathcal{C}}$; when there can be no ambiguity, we do not notationally distinguish between the two
- for every axiom $s_1 : s_2 \in \mathcal{A}$, there is a morphism $\hat{s_1} : \mathbf{1}(\mathbf{1}_{\mathcal{C}}) \to s_2$ such that $\mathcal{U}_{\mathbf{1}_C}(\hat{s_1}) = s_1$
- there is a mapping from syntactic names $\mathcal{V}$ to semantic names $\mathcal{N}$; often, we will neglect this difference and simply treat a syntactic name x as a semantic name

As mentioned before, we are going to interpret contexts as objects of $\mathcal{C}$, types as objects of $\mathcal{E}$, and elements of types as their (category theoretical) elements. Types themselves can be atoms of (universe) types, and the mapping $\mathcal{U}$ serves to translate between the morphism used to interpret it as an atom and the object used to interpret it as a type. The unit type is interpreted through the $\mathbf{1}_x$ functor, establishing a pattern that datatype constructors in the language are interpreted through adjoint functors in the semantics; this beautiful parallel was perhaps first discovered by Bill Lawvere, and has been extensively used in categorical semantics ever since.

It is not in general possible to give syntax-directed definition of the semantic interpretation function $[\![-]\!]$ for dependently typed calculi, since the definitions of types and terms depend on each other. Hence we will make use of a device due to Streicher [21]: We first define a partial interpretation function from pseudoterms to semantic entities, and then prove that this function is defined on all pseudoterms we might want to interpret.

More precisely, we need to define *three different* interpretation functions, one for contexts, one for types-in-context, and one for atoms-in-context, which we write in a notation suggestive of judgements:

- $[\![\Gamma]\!]$ maps a pseudocontext Γ to an object in $\mathcal{C}$
- $[\![\Gamma \vdash : M]\!]$ maps a pseudotype M in pseudocontext Γ to an object in $\mathcal{E}$
- $[\![\Gamma \vdash M]\!]$ maps a pseudoterm M in pseudocontext Γ to a morphism in $\mathcal{E}$

We now give the definitions of the three (partial) interpretation functions[7] by induction on $|\Gamma|+|M|$. The definitions do not spell out definedness conditions in detail, instead we use the convention that an expression can only be defined if its every subexpression is defined.

2.5.1 Interpretation of Contexts

The empty context is interpreted as the terminal object in $\mathcal{C}$, non-empty contexts are interpreted as the comprehension of their last entry:

- $[\![\diamond]\!] = \mathbf{1}_{\mathcal{C}}$
- $[\![\Gamma, x\colon A]\!] = \{[\![\Gamma \vdash\colon A]\!]\}_x$

2.5.2 Interpretation of Types

The interpretation of types, in turn, makes use of the interpretation of terms defined below. Observe that if $[\![\Gamma \vdash\colon A]\!]$ is defined, then it is above $[\![\Gamma]\!]$ as expected.

- for $s \in \mathcal{S}$: $[\![\Gamma \vdash\colon s]\!] = [\![\Gamma]\!]^* s$
- for $A \notin \mathcal{S}$: $[\![\Gamma \vdash\colon A]\!] = \mathcal{U}_{[\![\Gamma]\!]}([\![\Gamma \vdash A]\!])$, where $[\![\Gamma \vdash A]\!]$ is an element of $[\![\Gamma]\!]^* s$ for some $s \in \mathcal{S}_{\mathcal{C}}$; this does not always have to be defined, but if it is then it does not depend on the choice of s

2.5.3 Interpretation of Terms

If defined, $[\![\Gamma \vdash M]\!]$ is a morphism with domain $\mathbf{1}([\![\Gamma]\!])$, that is an element of some object in $\mathcal{E}_{[\![\Gamma]\!]}$.

- $[\![\Gamma \vdash s]\!] = (!_{[\![\Gamma]\!]})^*(\hat{s})$
  Note that, given this definition, the first case of the interpretation function for types is in fact a special case of the second when $\hat{s}$ is defined:

$$\begin{aligned}\mathcal{U}_{[\![\Gamma]\!]}([\![\Gamma \vdash s]\!]) &= \mathcal{U}_{[\![\Gamma]\!]}((!_{[\![\Gamma]\!]})^*(\hat{s})) \\ &= !_{[\![\Gamma]\!]}(\mathcal{U}^*_{[\![\Gamma]\!]}(\hat{s})) \\ &= !_{[\![\Gamma]\!]}(s)\end{aligned}$$

- $[\![\Gamma, x\colon A \vdash x^0]\!] = \delta_{x,[\![\Gamma \vdash\colon A]\!]}$
- $[\![\Gamma, x\colon A \vdash x^{n+1}]\!] = P_x([\![\Gamma \vdash\colon A]\!])^*([\![\Gamma \vdash x^n]\!])$
- $[\![\Gamma, y\colon A \vdash x^n]\!] = P_y([\![\Gamma \vdash\colon A]\!])^*([\![\Gamma \vdash x^n]\!])$ for $x \not\equiv y$
- $[\![\Gamma \vdash \mathrm{unit}]\!] = \hat{\mathbf{1}}([\![\Gamma]\!])$
- $[\![\Gamma \vdash *]\!] = \mathrm{id}_{\mathbf{1}([\![\Gamma]\!])}$

The interpretation of contexts and types will be the same in all systems, but the interpretation of atoms will have to be extended to cover new language constructs.

[7] By and large, these definitions follow those in Streicher's monograph.

2.6 Semantics of TCINNI-OCC$_0$

When motivating our categorical semantics, we mentioned that substitutions are interpreted as context morphisms, i.e. arrows in the category $\mathcal{C}$. In this section, we exhibit a fourth partial interpretation function, which (when defined) maps a pseudocontext Γ and a TCINNI substitution ϑ to an arrow $[\![\Gamma \vdash \vartheta]\!]$. The reindexing functor induced by this arrow corresponds to the application of the substitution to terms.

We then have to prove an important coherence result: in OCC, substitutions are not first-class objects, but a meta-notation which can be eliminated through the reduction rules given before; for any substitution ϑ and any pseudoterm M, there is a unique pseudoterm N which results from applying ϑ to M. Thus we have to make sure that $[\![\Gamma \vdash \vartheta]\!]^*([\![\Gamma \vdash M]\!]) = [\![\Gamma \vdash N]\!]$, i.e. we obtain the same result no matter whether we first interpret the substitution and then (semantically) apply it to the interpretation of M, or first apply the substitution (syntactically) to the term M (yielding N), and then interpreting the result.

It turns out that TCINNI substitutions can, in fact, be given a surprisingly neat categorical semantics: The three substitution operators of TCINNI correspond exactly to the three kinds of "special" morphisms in our semantics, namely sections, display maps, and liftings.

Definition 2.22. We define a partial interpretation function $[\![- \vdash -]\!]$, mapping a pseudoterm Γ and a TCINNI substitution ϑ to an arrow $[\![\Gamma \vdash \vartheta]\!]$ in $\mathcal{C}$:

1. $[\![\Gamma \vdash [x := N\colon A]]\!] = \mathrm{Sect}_x([\![\Gamma \vdash N]\!])$, when $[\![\Gamma \vdash N]\!]$ is an element of $[\![\Gamma \vdash\colon A]\!]$
2. $[\![\Gamma, x\colon A \vdash \uparrow_{(x\colon A)}]\!] = P_x([\![\Gamma \vdash\colon A]\!])$, when $[\![\Gamma \vdash\colon A]\!]$ is defined
3. $[\![\Gamma, x\colon \vartheta A \vdash \Uparrow_{(x\colon A)}\vartheta]\!] = q([\![\Gamma \vdash \vartheta]\!], x, [\![\mathrm{cod}(\Gamma, \vartheta) \vdash\colon A]\!])$, when $[\![\Gamma \vdash \vartheta]\!]$ and $[\![\mathrm{cod}(\Gamma, \vartheta) \vdash\colon A]\!]$ are defined

   $\mathrm{cod}(\Gamma, \vartheta)$ is defined by induction on ϑ:

   (a) $\mathrm{cod}(\Gamma, [x := N\colon A]) = (\Gamma, x\colon A)$ if $[\![\Gamma \vdash N]\!]$ is an element of $[\![\Gamma \vdash\colon A]\!]$

   (b) $\mathrm{cod}((\Gamma, x\colon A), \uparrow_{(x\colon A)}) = \Gamma$

   (c) $\mathrm{cod}((\Gamma, x\colon \vartheta A), \Uparrow_{(x\colon A)}\vartheta) = (\mathrm{cod}(\Gamma, \vartheta), x\colon A)$

Theorem 2.23 (Soundness of the semantics for TCINNI-OCC$_0$)**.** *For a pseudocontext Γ, a substitution ϑ and a pseudoterm M, we have*

$$(*) \qquad [\![\Gamma \vdash \vartheta M]\!] = [\![\Gamma \vdash \vartheta]\!]^*([\![\mathrm{cod}(\Gamma, \vartheta) \vdash M]\!])$$

and

$$(**) \qquad [\![\Gamma \vdash\colon \vartheta M]\!] = [\![\Gamma \vdash \vartheta]\!]^*([\![\mathrm{cod}(\Gamma, \vartheta) \vdash\colon M]\!])$$

whenever all the involved quantities are defined.
We then also have

$$(+) \qquad [\![\Gamma \vdash \vartheta]\!]\colon [\![\Gamma]\!] \to [\![\mathrm{cod}(\Gamma, \vartheta)]\!]$$

Proof. We prove the theorem by induction on $|\Gamma| + |\vartheta| + |M|$. In the induction step, we prove $(+)$ and $(*)$ by a case analysis on the structure of ϑ and M, and then show that $(**)$ follows.

We first treat the case that M is a variable.

1. Case $\vartheta \equiv [x := N\colon A]$:

   Since $[\![\Gamma \vdash \vartheta]\!]$ and $\mathrm{cod}(\vartheta, \Gamma)$ are assumed to be defined, we conclude that $N := [\![\Gamma \vdash N]\!]$ is an element of $A := [\![\Gamma \vdash\colon A]\!]$, and $\mathrm{cod}(\vartheta, \Gamma) = (\Gamma, x\colon A)$. Finally, we can assume that $m := [\![\Gamma, x\colon A \vdash M]\!]$ is defined.

   Obviously, $(+)$ holds, since $[\![\Gamma \vdash [x := N\colon A]]\!] = \mathrm{Sect}_x([\![\Gamma \vdash N]\!])\colon [\![\Gamma]\!] \to [\![\Gamma, x\colon A]\!]$. We show

   $$[\![\Gamma \vdash \vartheta M]\!] = \mathrm{Sect}_x([\![\Gamma \vdash N]\!])^*([\![\Gamma, x\colon A \vdash M]\!])$$

   by a case analysis of the structure of M.

   (a) Case $M \equiv x^0$.
   By Lemma 2.20 we have

   $$\begin{aligned}[\![\Gamma \vdash \vartheta M]\!] &= \mathrm{Sect}_x(N) = \mathrm{Sect}_x(N)^*(m)\\ &= \mathrm{Sect}_x([\![\Gamma \vdash N]\!])^*([\![\Gamma, x\colon A \vdash x^0]\!])\end{aligned}$$

   (b) Case $M \equiv x^{n+1}$:

   $$\begin{aligned}[\![\Gamma \vdash \vartheta M]\!] &= [\![\Gamma \vdash [x := N\colon A]x^{n+1}]\!]\\ &= [\![\Gamma \vdash x^n]\!]\\ &= (P_x([\![\Gamma \vdash\colon A]\!]) \circ \mathrm{Sect}_x([\![\Gamma \vdash N]\!]))^*([\![\Gamma \vdash x^n]\!])\\ &= \mathrm{Sect}_x([\![\Gamma \vdash N]\!])^*(P_x([\![\Gamma \vdash\colon A]\!])^*([\![\Gamma \vdash x^n]\!]))\\ &= \mathrm{Sect}_x([\![\Gamma \vdash N]\!])^*([\![\Gamma, x\colon A \vdash x^{n+1}]\!])\end{aligned}$$

   (c) Case $M \equiv y^n$: The argument is almost literally the same as in the previous case.

2. Case $\vartheta \equiv \uparrow_{(x\colon A)}$:

   Since $[\![\Gamma \vdash \vartheta]\!]$ is assumed to be defined, Γ must be of the form $\Gamma', x\colon A$. Assume $[\![\Gamma]\!]$ and $[\![\Gamma' \vdash M]\!]$ are defined.

   Obviously, $(+)$ holds, since $[\![\Gamma \vdash \uparrow_{(x\colon A)}]\!] = P_x([\![\Gamma]\!])\colon [\![\Gamma]\!] \to [\![\Gamma']\!] = \mathrm{cod}(\Gamma, \uparrow_{(x\colon A)})$.

   We show

   $$[\![\Gamma \vdash \uparrow_{(x\colon A)} M]\!] = P_x([\![\Gamma]\!])^*([\![\Gamma' \vdash M]\!])$$

   by a case analysis of the structure of M; both cases are almost trivial:

   (a) Case $M \equiv x^n$:

   $$\begin{aligned}[\![\Gamma', x\colon A \vdash \uparrow_{(x\colon A)} x^n]\!] &= [\![\Gamma', x\colon A \vdash x^{n+1}]\!]\\ &= P_x([\![\Gamma]\!])^*([\![\Gamma' \vdash x^n]\!])\end{aligned}$$

(b) Case $M \equiv y^n$, $y \not\equiv x$:

$$\begin{aligned}[\![\Gamma', x\colon A \vdash \uparrow_{(x\colon A)} y^n]\!] &= [\![\Gamma', x\colon A \vdash y^n]\!] \\ &= P_x([\![\Gamma]\!])^*([\![\Gamma' \vdash y^n]\!])\end{aligned}$$

3. Case $\vartheta \equiv \Uparrow_{(x\colon A)} \vartheta'$: Since $[\![\Gamma \vdash \vartheta]\!]$ is assumed to be defined, Γ must be of the form $\Gamma', x\colon \vartheta' A$.

   By induction hypothesis, (+) is assumed to hold for ϑ', hence it also holds for ϑ:

$$\begin{aligned}&[\![\Gamma \vdash \Uparrow_{(x\colon A)} \vartheta']\!] \\ &= q([\![\Gamma' \vdash \vartheta']\!], x, [\![\mathrm{cod}(\Gamma', \vartheta') \vdash\colon A]\!]) \\ &\in \mathcal{C}(\{[\![\Gamma' \vdash \vartheta']\!]^*([\![\mathrm{cod}(\Gamma', \vartheta'), \vdash\colon A]\!])\}_x, \{[\![\mathrm{cod}(\Gamma', \vartheta'), \vdash\colon A]\!]\}_x) \\ &= \mathcal{C}([\![\Gamma', x\colon \vartheta' A]\!], [\![\mathrm{cod}(\Gamma', \vartheta'), x\colon A]\!]) \\ &= \mathcal{C}([\![\Gamma]\!], [\![\mathrm{cod}(\Gamma, \vartheta)]\!])\end{aligned}$$

   We show

$$\begin{aligned}&[\![\Gamma \vdash (\Uparrow_{(x\colon A)} \vartheta')M]\!] = \\ &\qquad q([\![\Gamma' \vdash \vartheta']\!], x, [\![\mathrm{cod}(\Gamma', \vartheta') \vdash\colon A]\!])^*([\![\mathrm{cod}(\Gamma', \vartheta') \vdash\colon A \vdash M]\!])\end{aligned}$$

   by case analysis:

   (a) Case $M \equiv x^0$:
   Writing A for $[\![\mathrm{cod}(\Gamma', \vartheta') \vdash\colon A]\!]$, A' for $[\![\Gamma' \vdash\colon \vartheta' A]\!]$ and ϑ' for $[\![\Gamma' \vdash \vartheta']\!]$, we see that

$$\begin{aligned}&q(P_x(A'), x, A') \circ \mathrm{Sect}_x(q(\vartheta', x, A)^*(\delta_{x,A})) \\ &= q(P_x(A'), x, A') \circ \{q(\vartheta', x, A)^*(\delta_{x,A})\}_x \circ \eta_{\mathbf{1}}(\{A'\}_x) \\ &= \{\varepsilon_{\mathbf{1}}(A')\}_x \circ \eta_{\mathbf{1}}(\{A'\}_x) \\ &= \mathrm{id}_{\{A'\}_x}\end{aligned}$$

   Hence, by Lemma 2.21, $\mathrm{Sect}_x(\delta_{x,A'}) = \mathrm{Sect}_x(q(\vartheta', x, A)^*(\delta_{x,A}))$, yielding

$$\begin{aligned}&[\![\Gamma, x\colon \vartheta' A \vdash x^0]\!] = \delta_{x,A'} = q(\vartheta', x, A)^*(\delta_{x,A}) \\ &\quad = q([\![\Gamma' \vdash \vartheta']\!], x, [\![\mathrm{cod}(\Gamma', \vartheta') \vdash\colon A]\!])^*([\![\mathrm{cod}(\Gamma', \vartheta'), x\colon A \vdash x^0]\!])\end{aligned}$$

   (b) Case $M \equiv x^{n+1}$:
   Writing A for $[\![\mathrm{cod}(\Gamma', \vartheta') \vdash\colon A]\!]$ and ϑ' for $[\![\Gamma' \vdash \vartheta']\!]$, we get

$$\begin{aligned}& [\![\Gamma', x\colon \vartheta' A \vdash (\Uparrow_{(x\colon A)} \vartheta')x^{n+1}]\!] \\ =\ & [\![\Gamma', x\colon \vartheta' A \vdash \uparrow_{(x\colon \vartheta' A)}(\vartheta' x^n)]\!] \\ =\ & P_x([\![\Gamma \vdash\colon \vartheta' A]\!])^*([\![\Gamma' \vdash \vartheta' x^n]\!]) \\ =\ & P_x(\vartheta'^*(A))^*(\vartheta'^*([\![\mathrm{cod}(\Gamma', \vartheta') \vdash x^n]\!])) \\ =\ & (\vartheta' \circ P_x(\vartheta'^*(A)))^*([\![\mathrm{cod}(\Gamma', \vartheta') \vdash x^n]\!]) \\ =\ & (P_x(A) \circ q(\vartheta', x, A))^*([\![\mathrm{cod}(\Gamma', \vartheta') \vdash x^n]\!]) \\ =\ & q(\vartheta', x, A)^*(P_x(A)^*([\![\mathrm{cod}(\Gamma', \vartheta') \vdash x^n]\!])) \\ =\ & [\![\Gamma', x\colon \vartheta A \vdash \Uparrow_{(x\colon A)} \vartheta]\!]^*([\![\mathrm{cod}(\Gamma', \vartheta'), x\colon A \vdash x^{n+1}]\!])\end{aligned}$$

(c) Case $M \equiv y^m$: This case is completely analogous to the last one.

Now we deal with $M \notin \mathcal{V}$, assuming that ϑ satisfies $(+)$, which we have already shown.

1. Case $M \equiv \mathrm{unit}$:

$$\begin{aligned}
[\![\Gamma \vdash \vartheta\,\mathrm{unit}]\!] &= [\![\Gamma \vdash \mathrm{unit}]\!] \\
&= \hat{\mathbf{1}}([\![\Gamma]\!]) \\
&= [\![\Gamma \vdash \vartheta]\!]^*\hat{\mathbf{1}}([\![\mathrm{cod}(\Gamma,\vartheta)]\!]) \\
&= [\![\Gamma \vdash \vartheta]\!]^*([\![\mathrm{cod}(\Gamma,\vartheta) \vdash \mathrm{unit}]\!])
\end{aligned}$$

2. Case $M \equiv *$:

$$\begin{aligned}
[\![\Gamma \vdash \vartheta\,*]\!] &= [\![\Gamma \vdash *]\!] \\
&= \mathrm{id}_{\mathbf{1}([\![\Gamma]\!])} \\
&= \mathrm{id}_{[\![\Gamma\vdash\vartheta]\!]^*(\mathbf{1}([\![\mathrm{cod}(\Gamma,\vartheta)]\!]))} \\
&= [\![\Gamma \vdash \vartheta]\!]^*(\mathrm{id}_{\mathbf{1}([\![\mathrm{cod}(\Gamma,\vartheta)]\!])}) \\
&= [\![\Gamma \vdash \vartheta]\!]^*([\![\mathrm{cod}(\Gamma,\vartheta) \vdash *]\!])
\end{aligned}$$

3. Case $M \equiv s \in \mathcal{S}$:

$$\begin{aligned}
[\![\Gamma \vdash \vartheta s]\!] &= [\![\Gamma \vdash s]\!] \\
&= (!_{[\![\Gamma]\!]})^*(\hat{s}) \\
&= (!_{[\![\mathrm{cod}(\Gamma,\vartheta)]\!]} \circ [\![\Gamma \vdash \vartheta]\!])^*(\hat{s}) \\
&= [\![\Gamma \vdash \vartheta]\!]^*((!_{[\![\mathrm{cod}(\Gamma,\vartheta)]\!]})^*(\hat{s})) \\
&= [\![\Gamma \vdash \vartheta]\!]^*([\![\mathrm{cod}(\Gamma,\vartheta) \vdash M]\!])
\end{aligned}$$

From $(*)$ and $(+)$ we can now derive $(**)$:

1. Case $M \equiv s$:

$$\begin{aligned}
[\![\Gamma \vdash\colon \vartheta s]\!] &= [\![\Gamma \vdash s]\!] \\
&= (!_{[\![\Gamma]\!]})^*(s) \\
&= (!_{[\![\mathrm{cod}(\Gamma,\vartheta)]\!]} \circ [\![\Gamma \vdash \vartheta]\!])^*(s) \\
&= [\![\Gamma \vdash \vartheta]\!]^*((!_{[\![\mathrm{cod}(\Gamma,\vartheta)]\!]})^*(s)) \\
&= [\![\Gamma \vdash \vartheta]\!]^*([\![\mathrm{cod}(\Gamma,\vartheta) \vdash\colon s]\!])
\end{aligned}$$

2. Case $M \equiv A \notin S$:

$$\begin{aligned}
[\![\Gamma \vdash\colon \vartheta A]\!] &= \mathcal{U}_{[\![\Gamma]\!]}([\![\Gamma \vdash \vartheta A]\!]) \\
&= \mathcal{U}_{[\![\Gamma]\!]}([\![\Gamma \vdash \vartheta]\!]^*([\![\mathrm{cod}(\Gamma,\vartheta) \vdash A]\!])) \\
&= [\![\Gamma \vdash \vartheta]\!]^*(\mathcal{U}_{[\![\mathrm{cod}(\Gamma,\vartheta)]\!]}([\![\mathrm{cod}(\Gamma,\vartheta) \vdash A]\!])) \\
&= [\![\Gamma \vdash \vartheta]\!]^*([\![\mathrm{cod}(\Gamma,\vartheta) \vdash\colon A]\!])
\end{aligned}$$

□

Corollary 2.24. *We record the following four special cases of the above result for later use:*

1. $[\![\Gamma, x\colon A \vdash \uparrow_{(x\colon A)} B]\!] = P_x([\![\Gamma \vdash\colon A]\!])^*([\![\Gamma \vdash B]\!])$
2. $[\![\Gamma, x\colon A \vdash\colon \uparrow_{(x\colon A)} B]\!] = P_x([\![\Gamma \vdash\colon A]\!])^*([\![\Gamma \vdash\colon B]\!])$
3. $[\![\Gamma \vdash [x := N\colon S]M]\!] = \mathrm{Sect}_x([\![\Gamma \vdash N]\!])^*([\![\Gamma \vdash M]\!])$
4. $[\![\Gamma \vdash\colon [x := N\colon S]M]\!] = \mathrm{Sect}_x([\![\Gamma \vdash N]\!])^*([\![\Gamma \vdash\colon M]\!])$

2.7 Soundness

We now proceed to prove soundness of the inference system of OCC_0. Since OCC_0, like the original OCC, is a very flexible system, the mere derivability of a judgement $\Gamma \vdash J$ does not imply that Γ is a sensible context and can meaningfully be interpreted. Hence we give the following more careful formulation of the soundness theorem (which is close to the operational semantics given by Stehr), where we write $[\![-]\!] \downarrow$ to indicate (unique) definedness of the interpretation function on the pseudoterm or -context in question.

Theorem 2.25 (Soundness of OCC_0)**.** *If $\Gamma \vdash A\colon B$, and $C := [\![\Gamma]\!] \downarrow$, then $A := [\![\Gamma \vdash A]\!] \downarrow$ and $B := [\![\Gamma \vdash\colon B]\!] \downarrow$; A is above C and A is an element of B.*

Proof. The proof of the soundness theorem proceeds by induction on derivations, making use of the previously proved substitution soundness theorem.

1. Last rule used was

$$\text{(Ax)}\ \frac{}{\Gamma \vdash s_1\colon s_2}\ , \mathcal{A}(s_1) = s_2$$

   Assume $C := [\![\Gamma]\!] \downarrow$, then $S_1 := [\![\Gamma \vdash s_1]\!] = !^*_C(\hat{s_1})$ is also defined. Observe that $\hat{s_1}$ is an element of $\ulcorner s_2 \urcorner$ by definition, hence S_1 is an element of $S_2 := !^*_C(s_2) = [\![\Gamma \vdash\colon s_2]\!]$, and S_2 is above C.

2. Last rule used was

$$\text{(Start1)}\ \frac{}{\Gamma, x\colon A \vdash x^0\colon \uparrow_{(x\colon A)} A}$$

   Assume $[\![\Gamma, x\colon A]\!] \downarrow$, then also $[\![\Gamma \vdash\colon A]\!] \downarrow$. By Corollary 2.24 we have

$$[\![\Gamma, x\colon A \vdash\colon \uparrow_{(x\colon A)} A]\!] = P_x([\![\Gamma \vdash\colon A]\!])^*([\![\Gamma \vdash\colon A]\!]),$$

   of which $\delta_{x, [\![\Gamma \vdash\colon A]\!]} = [\![\Gamma, x\colon A \vdash x^0]\!]$ is an element.

3. Last rule used was

$$\text{(Start2)}\ \frac{\Gamma \vdash x^i\colon A}{\Gamma, x\colon B \vdash x^{i+1}\colon \uparrow_{(x\colon B)} A}$$

   Assume $[\![\Gamma, x\colon B]\!] \downarrow$. Then also $[\![\Gamma]\!] \downarrow$, hence $[\![\Gamma \vdash x^i]\!]$ is an element of $[\![\Gamma \vdash\colon A]\!]$ by IH. Then $[\![\Gamma, x\colon B \vdash x^{i+1}]\!] \downarrow$, too, and it is an element of $[\![\Gamma, x\colon B \vdash\colon \uparrow_{(x\colon B)} A]\!]$ by Corollary 2.24.

4. Last rule used was

$$\text{(Start3)}\ \frac{\Gamma \vdash y^m\colon A}{\Gamma, x\colon B \vdash y^m\colon \uparrow_{(x\colon B)} A}\ x \not\equiv y$$

   This case is almost literally the same as case 3.

5. Last rule used was

$$\text{(TruthForm)}\ \frac{}{\Gamma \vdash \text{unit} : \text{Prop}}$$

Assume $[\![\Gamma]\!]\downarrow$, then $[\![\Gamma \vdash \text{unit}]\!] = \hat{\mathbf{1}}([\![\Gamma]\!])$, which is an element of $[\![\Gamma]\!]^*\text{Prop} = [\![\Gamma \vdash : \text{Prop}]\!]$ as required.

6. Last rule used was

$$\text{(TruthIntro)}\ \frac{}{\Gamma \vdash * : \text{unit}}$$

Assume $[\![\Gamma]\!]\downarrow$, then $[\![\Gamma \vdash *]\!] = \text{id}_{\mathbf{1}([\![\Gamma]\!])}$ which is an element of $[\![\Gamma \vdash : \text{unit}]\!]$ as required.

□

Later soundness proofs will extend this proof by clauses for new language constructs.

3 OCC_1: Product Types and Structural Equality

In this section, we extend the rudimentary calculus OCC_0 with type formers, more specifically dependent product and structural equality, to yield a new system OCC_1. These two type formers are central to type theory, and indeed the expressive power of systems like the Calculus of Constructions is in no small measure due to the power and flexibility of the product forming operations they offer.

In OCC, product types are even more flexible than in other calculi, since their behavior can be customized through a partial function $\mathcal{R}$ that is part of the signature. Unbridled use of this feature is dangerous, though, since it allows the formulation of OCC instances that fall prey to Girard's paradox. This famous result, explained for example in [2], shows that by allowing certain kinds of product formation one obtains inconsistent systems in which every type is inhabited. To avoid this, OCC posits a number of restrictions on $\mathcal{R}$, which we directly adopt in OCC_1.

Following established tradition, product types are modelled categorically via a right adjoint to a particular reindexing functor. For equality types, often modelled via adjoints as well, we take a somewhat different approach, which is suggested in Streicher's work. Both product and equality types have been extensively studied in the literature, so we are still on firm ground in this chapter.

Incidentally, while OCC_1 lacks many features of full OCC, it is more or less the same as the system Stehr gives his semantics for (the most important omission being subtyping).

3.1 Syntax

An OCC_1 signature is a tuple $\Sigma = (\mathcal{S}, \mathcal{S}^i, \mathcal{S}^p, \mathrm{Prop}, \mathcal{A}, \mathcal{R}, \leq)$, where

1. $(\mathcal{S}, \mathrm{Prop}, \mathcal{A})$ is an OCC_0 signature
2. $\mathcal{S}^i$ and $\mathcal{S}^p$ form a partition of $\mathcal{S}$ (of impredicative and predicative universes)
3. $\mathrm{Prop} \in \mathcal{S}^i$
4. $\leq$ is a partial order on $\mathcal{S}$ with respect to which impredicative universes are minimal
5. $\mathcal{R} \colon \mathcal{S} \times \mathcal{S} \rightharpoonup \mathcal{S}$ is a partial function such that
   (a) if $\mathcal{R}(s_1, s_2)$ is defined and $s_1' \leq s_1$, $s_2' \leq s_2$, then $\mathcal{R}(s_1', s_2') \leq \mathcal{R}(s_1, s_2)$ and it is in particular defined
   (b) for $s_1 \in \mathcal{S}$ and $s_2 \in \mathcal{S}^i$, $\mathcal{R}(s_1, s_2) = s_2$ if defined
   (c) for $s_1 \in \mathcal{S}$ and $s_2 \in \mathcal{S}^p$, $s_1, s_2 \leq \mathcal{R}(s_1, s_2)$ if defined

Given an OCC_1 signature Σ, the set of *OCC_1 pseudoterms* $\mathcal{T}_1^\Sigma$, or simply $\mathcal{T}_1$ when the signature is understood from context, is defined by the following grammar:

$$\begin{aligned}\mathcal{T}_1 ::= {} & \mathcal{S} \mid \mathcal{V}^{\mathbb{N}} \mid \mathrm{unit} \mid * \\ & \mid \lambda\mathcal{V}\colon \mathcal{T}_1.\mathcal{T}_1 \mid \Pi\mathcal{V}\colon \mathcal{T}_1.\mathcal{T}_1 \mid \mathrm{App}([\mathcal{V}\colon \mathcal{T}_1]\,\mathcal{T}_1, \mathcal{T}_1, \mathcal{T}_1) \\ & \mid \mathrm{StrEq}_{\mathcal{T}_1}(\mathcal{T}_1, \mathcal{T}_1) \mid \sigma(\mathcal{T}_1, \mathcal{T}_1)\end{aligned}$$

where $\mathcal{V}$ is a fixed set of syntactic variable names and $\mathbb{N}$ is the set of natural numbers (including 0).

In particular, every OCC_0 pseudoterm is also an OCC_1 pseudoterm. As before, there are a number of syntactic differences from Stehr's notation, because we choose a more conventional notation to express abstraction and product types. More significantly, our system requires applications to be explicitly typed, which greatly simplifies the definition of the interpretation function; see Streicher's monograph [21] for a discussion of this issue. The same holds for the structural equality type $\mathrm{StrEq}_{-}(-,-)$, where $\mathrm{StrEq}_A(M,N)$ is understood to be the type of proofs of the proposition that the atoms denoted by M and N, both of which are of type A, are in fact equal. Finally, the term $\sigma(M,A)$ is the canonical inhabitant of the type $\mathrm{StrEq}_A(M,M)$ corresponding to a proof that the atom of type A denoted by term M equals itself.

Definition 3.1. The definition of the size of a pseudoterm is extended to include the new syntactic constructs:

1. for $x \in \mathcal{V}$, $S,T,M,N \in \mathcal{T}_1$: $|\mathrm{App}([x\colon S]\,T,M,N)| = |S|+|T|+|M|+|N|+1$
2. for $x \in \mathcal{V}$, $S,M \in \mathcal{T}_1$: $|\lambda x\colon S.M| = |S| + |M| + 1$
3. for $x \in \mathcal{V}$, $S,T \in \mathcal{T}_1$: $|\Pi x\colon S.T| = |S| + |T| + 1$
4. for $A,M,N \in \mathcal{T}_1$: $|\,\mathrm{StrEq}_A(M,N)| = |M| + |A| + |N| + 1$
5. for $A,M \in \mathcal{T}_1$: $|\sigma(M,A)| = |M| + |A| + 1$

Pseudocontexts and their sizes are defined in exactly the same way as for OCC_0, see Definitions 2.2 and 2.3.

3.2 TCINNI-OCC_1

We extend Definition 2.4 to cover the new syntactic constructs of OCC_1.

Definition 3.2. On those OCC_1 pseudoterms which are not also OCC_0 pseudoterms, substitutions behave as follows:

$$\begin{array}{lcl}\vartheta\mathrm{App}([y\colon S]\,T,A,B) & = & \mathrm{App}([y\colon \vartheta S]\,(\Uparrow_{(y\colon \vartheta A)})T, \vartheta A, \vartheta B) \\ \vartheta(\lambda y\colon A.B) & = & \lambda y\colon \vartheta A.(\Uparrow_{(y\colon \vartheta A)})B \\ \vartheta(\Pi y\colon A.B) & = & \Pi y\colon \vartheta A.(\Uparrow_{(y\colon \vartheta A)})B \\ \vartheta\,\mathrm{StrEq}_S(A,B) & = & \mathrm{StrEq}_{\vartheta S}(\vartheta A, \vartheta B) \\ \vartheta\sigma(M,A) & = & \sigma(\vartheta M, \vartheta A)\end{array}$$

Lemma 2.6 and Lemma 2.8 carry over unchanged, although the latter has to be strengthened to cover all substitutions composed entirely of shifting and lifting operations.

3.3 Formal System

Definition 3.3. An OCC_1 judgement is either an OCC_0 judgement or it is of the form

$$\Gamma \vdash ||(M = N) \colon A$$

where Γ is a pseudocontext, M and N are pseudoterms, and A is a pseudotype; this judgement expresses that in context Γ, M and N are equal atoms of type A.

A *valid OCC_1 judgement* is any judgement that can be derived by the rules of OCC_0 complemented with the following new rules:

$$\text{(StrRefl)}\ \frac{\Gamma \vdash M \colon A}{\Gamma \vdash ||(M = M) \colon A}$$

$$\text{(StrSymm)}\ \frac{\Gamma \vdash ||(M = N) \colon A}{\Gamma \vdash ||(N = M) \colon A}$$

$$\text{(StrTrans)}\ \frac{\Gamma \vdash ||(P = Q) \colon A \qquad \Gamma \vdash ||(Q = R) \colon A}{\Gamma \vdash ||(P = R) \colon A}$$

$$\text{(TypeConv)}\ \frac{\Gamma \vdash M \colon S \qquad \Gamma \vdash ||(S = T) \colon s}{\Gamma \vdash M \colon T}\ , s \in \mathcal{S}$$

$$\text{(UnitElim)}\ \frac{\Gamma \vdash Q \colon \mathrm{unit}}{\Gamma \vdash ||(Q = *) \colon \mathrm{unit}}$$

$$\text{(Pi)}\ \frac{\Gamma \vdash S \colon s_1 \qquad \Gamma, x \colon S \vdash T \colon s_2}{\Gamma \vdash \Pi x \colon S.T \colon s_3}\ , \mathcal{R}(s_1, s_2) = s_3$$

$$\text{(Lda)}\ \frac{\Gamma \vdash S \colon s \qquad \Gamma, x \colon S \vdash M \colon T}{\Gamma \vdash \lambda x \colon S.M \colon \Pi x \colon S.T}\ , s \in \mathcal{S}$$

$$\text{(App)}\ \frac{\Gamma \vdash M \colon \Pi x \colon S.T \qquad \Gamma \vdash N \colon S}{\Gamma \vdash \mathrm{App}([x \colon S]\, T, M, N) \colon [x := S \colon N]T}$$

$$\text{(StrPi)}\ \frac{\Gamma \vdash ||(S = S') \colon s_1 \qquad \Gamma, x \colon S \vdash ||(T = T') \colon s_2}{\Gamma \vdash ||(\Pi x \colon S.T = \Pi x \colon S'.T') \colon s_3}\ , \mathcal{R}(s_1, s_2) = s_3$$

$$\text{(StrLda)}\ \frac{\Gamma \vdash ||(S = S') \colon s_1 \qquad \Gamma, x \colon S \vdash ||(M = M') \colon T}{\Gamma \vdash ||(\lambda x \colon S.M = \lambda x \colon S'.M') \colon \Pi x \colon S.T}\ , s_1 \in \mathcal{S}$$

$$\text{(StrApp)}\ \frac{\Gamma \vdash ||(M = M') \colon \Pi x \colon S.T \qquad \Gamma \vdash ||(N = N') \colon S}{\Gamma \vdash ||(\mathrm{App}([x \colon S]\, T, M, N) = \mathrm{App}([x \colon S]\, T, M', N')) \colon [x := N \colon S]T}$$

$$\text{(BetaRed)}\ \frac{\Gamma \vdash N \colon S \qquad \Gamma, x \colon S \vdash M \colon T}{\Gamma \vdash ||(\mathrm{App}([x \colon S]\, T, \lambda x \colon S.M, N) = [x := N \colon S]M) \colon [x := N \colon S]T}$$

$$\text{(StrEqTyForm)}\ \frac{\Gamma \vdash M \colon S \qquad \Gamma \vdash N \colon S}{\Gamma \vdash \mathrm{StrEq}_S(M, N) \colon \mathrm{Prop}}$$

$$\text{(StrEqTyIntro)}\ \frac{\Gamma \vdash ||(M = N) \colon S}{\Gamma \vdash \sigma(M, S) \colon \mathrm{StrEq}_S(M, N)}$$

$$\text{(StrEqTyElim)}\ \frac{\Gamma \vdash P \colon \mathrm{StrEq}_A(M, N)}{\Gamma \vdash ||(M = N) \colon A}$$

3.4 OCC_1 Structures

An OCC_1 structure $\mathcal{M}$ is an OCC_0 structure with some extra conditions.

Definition 3.4. In order for an OCC_0-structure $\mathcal{M}$ to be an OCC_1-structure, there must be appropriate categorical type constructors, i.e. functors corresponding to product and equality type formation:

- For any $A \in \mathcal{E}$ above $C \in \mathcal{C}$ and any $x \in \mathcal{N}$, the reindexing functor $P_x(A)^*$ must have a right adjoint $\Pi_{x,A}$ with vertical unit η_Π and vertical counit ε_Π, which is compatible with reindexing, i.e. for $f\colon D \to C$ and B above $\{A\}_x$ we have

$$f^* \circ \Pi_{x,A} = \Pi_{x,f^*(A)} \circ q(f, x, A)^* \tag{1}$$

  and for $g\colon K \to \{A\}_x$,

$$g^*(\varepsilon_\Pi(B)) = \varepsilon_\Pi(g^*(B)) \tag{2}$$

  The requirement for compatibility with reindexing is a strong form of the so-called *Beck-Chevalley condition* [10].

  Note that the functor Π_A maps elements of B to elements of $\Pi_{x,A}(B)$. Intuitively speaking, this means that the product functor takes a term of type B depending on type A and abstracts out the dependency on A, yielding a term of type $\Pi_{x,A}(B)$; hence it behaves similar to a λ-abstraction (and is, indeed, used together with the adjunction's unit η_Π to provide a semantics for abstractions later on).

- For any object $A \in \mathcal{E}$ and name $x \in \mathcal{N}$, there must be an object $\mathrm{StrEq}_{x,A}$ above $\{P_x(A)^*(A)\}_x$ such that the two morphisms

$$P_x(P_x(A)^*(A)) \circ P_y(\mathrm{StrEq}_{x,A}) \tag{3}$$

  and

$$q(P_x(A), x, A) \circ P_y(\mathrm{StrEq}_{x,A}) \tag{4}$$

  are equal for any $y \in \mathcal{N}$. As we will see below, these two morphisms can be understood as "extraction" morphisms, which allow us to access the atoms whose equality the type expresses.

  Furthermore, we require the existence of an element $s_x(A)$ of $\delta^*_{x,A}(\mathrm{StrEq}_{x,A})$, which will be used to model the canonical reflexivity witness.

  Structural equality types are likewise required to be stable under substitution, i.e. for $f\colon D \to p(A)$ we must have

$$q(q(f, x, A), x, P_x(A)^*(A))^*(\mathrm{StrEq}_{x,A}) = \mathrm{StrEq}_{x,f^*(A)} \tag{5}$$

  and

$$q(f, x, A)^*(s_x(A)) = s_x(f^*(A)) \tag{6}$$

Recall that the injection morphism $\delta_{x,A} \colon \mathbf{1}(A) \to P_x(A)^*(A)$ corresponds to a section $\delta_{x,A} \colon \{A\}_x \to \{P_x(A)^*(A)\}_x$. This section, in turn, induces a reindexing functor $\delta^*_{x,A} \colon \mathcal{E}_{\{P_x(A)^*(A)\}_x} \to \mathcal{E}_{\{A\}_x}$. It is possible to define equality types via a left adjoint to this functor, see for example [10]. The approach we use here is derived from Streicher's account [21]; it generalizes more smoothly to computational equality and rewriting types.

Using these two constructions, we can interpret product and equality types as objects in $\mathcal{E}$. But this is not enough: since we are dealing with a type theory with universes, product and equality types can themselves be regarded as atoms belonging to an appropriate universe. But atoms are semantically interpreted as arrows, not objects, of $\mathcal{E}$. Hence we need an additional set of operations to construct the arrows corresponding to product and equality types:

- We require a partial operator $\hat{\Pi}$ such that for
  - $C \in \mathrm{Ob}(\mathcal{C})$
  - $s_1, s_2 \in \mathcal{S}_\mathcal{C}$
  - $\hat{A} \colon \mathbf{1}(C) \to C^* s_1$
  - $\hat{B} \colon \mathbf{1}(\{A\}_x) \to \{A\}_x^* s_2$, where $A := \mathcal{U}_C(\hat{A})$

  we have
  $$\hat{\Pi}_{x,\hat{A}}(\hat{B}) \colon \mathbf{1}(C) \to C^* s_3 \tag{7}$$
  for some $s_3 \in \mathcal{S}_\mathcal{C}$, and
  $$\mathcal{U}_C(\hat{\Pi}_{x,\hat{A}}(\hat{B})) = \Pi_{x,A}(\mathcal{U}_{\{A\}_x}(\hat{B})) \tag{8}$$

  The operator $\hat{\Pi}$ is required to be stable under reindexing in the same way as Π is, i.e. we have

  $$f^*(\hat{\Pi}_{x,\hat{A}}(\hat{B})) = \hat{\Pi}_{x,f^*(\hat{A})}(q(f, x, A)^*(\hat{B})) \tag{9}$$

  Since $\hat{\Pi}$ is only partial, the arrow in (7) does not have to be defined for all $\hat{A}$ and $\hat{B}$ satisfying the above conditions.

  If there is a partial function $\mathcal{R} \colon \mathcal{S}_\mathcal{C} \times \mathcal{S}_\mathcal{C} \rightsquigarrow \mathcal{S}_\mathcal{C}$ such that (7) is defined whenever $\mathcal{R}(s_1, s_2) \downarrow$, and if in that case $s_3 = \mathcal{R}(s_1, s_2)$, then $\hat{\Pi}$ is said to be *governed* by $\mathcal{R}$.

- We require a (total) operator $\hat{\mathrm{StrEq}}$, such that for
  - $C \in \mathcal{C}$
  - $s \in \mathcal{S}_\mathcal{C}$
  - $\hat{A} \colon \mathbf{1}(C) \to C^* s$

we have (writing A for $\mathcal{U}_C(\hat{A})$ and A^*A for $\{P_x(A)^*(A)\}_x$):

$$\hat{\mathrm{StrEq}}_{x,\hat{A}} \colon \mathbf{1}(A^*A) \to (A^*A)^*\mathrm{Prop} \tag{10}$$

and

$$\mathcal{U}_{A^*A}(\hat{\mathrm{StrEq}}_{x,\hat{A}}) = \mathrm{StrEq}_{x,A} \tag{11}$$

Again, $\hat{\mathrm{StrEq}}$ is required to be stable under reindexing just like StrEq:

$$q(q(f,x,A),x,P_x(A)^*(A))^*(\hat{\mathrm{StrEq}}_{x,\hat{A}}) = \hat{\mathrm{StrEq}}_{x,f^*(A)} \tag{12}$$

3.5 The Interpretation Function

Let an OCC_1-signature $\Sigma = (\mathcal{S}, \mathcal{S}^i, \mathcal{S}^p, \mathrm{Prop}, \mathcal{A}, \mathcal{R}, \leq)$ be given. To model the corresponding OCC_1-instance, we need an OCC_1-structure $\mathcal{M}$ such that the OCC_0-instance for $\Sigma' := (\mathcal{S}, \mathrm{Prop}, \mathcal{A})$ can be modelled in $\mathcal{M}$ and the operator $\hat{\Pi}$ is governed by $\mathcal{R}$.

The interpretation functions to be defined are as with OCC_0, and in fact the definitions of the interpretation functions for contexts and types need not be changed. Only the definition of the interpretation function for terms is augmented by the following cases:

3.5.1 Interpretation of Terms

1. application:

   $$[\![\Gamma \vdash \mathrm{App}([x\colon A]\,B, M, N)]\!] = \\ \mathrm{Sect}_x([\![\Gamma \vdash N]\!])^*(\varepsilon_\Pi([\![\Gamma, x\colon A \vdash\colon B]\!]) \circ P_x([\![\Gamma \vdash\colon A]\!])^*([\![\Gamma \vdash M]\!]))$$

   when

   (a) $[\![\Gamma \vdash M]\!]$ is an element of $\Pi_{x,[\![\Gamma \vdash\colon A]\!]}([\![\Gamma, x\colon A \vdash\colon B]\!])$

   (b) $[\![\Gamma \vdash N]\!]$ is an element of $[\![\Gamma \vdash\colon A]\!]$, and hence $\mathrm{Sect}_x([\![\Gamma \vdash N]\!])$ is a section of $P_x(\Gamma, x\colon A)$

2. lambda abstraction:

   $[\![\Gamma \vdash \lambda x\colon A.M]\!] = \Pi_{x,[\![\Gamma \vdash\colon A]\!]}([\![\Gamma, x\colon A \vdash M]\!]) \cdot \eta_\Pi(\mathbf{1}([\![\Gamma]\!]))$

3. dependent product formation:

   $[\![\Gamma \vdash \Pi x\colon S.T]\!] = (\hat{\Pi}_{x,[\![\Gamma \vdash S]\!]}([\![\Gamma, x\colon S \vdash T]\!]))$, when

   (a) $C := [\![\Gamma]\!] \downarrow$

   (b) $[\![\Gamma \vdash S]\!]$ is an element of C^*s_1 for some $s_1 \in \mathcal{S}_\mathcal{C}$

   (c) $[\![\Gamma, x\colon S \vdash T]\!]$ is an element of C^*s_2 for some $s_2 \in \mathcal{S}_\mathcal{C}$

   (d) $\mathcal{R}(s_1, s_2) \downarrow$

   As is to be hoped, we have $[\![\Gamma \vdash\colon \Pi x\colon S.T]\!] = \Pi_{x,[\![\Gamma \vdash\colon S]\!]}([\![\Gamma, x\colon S \vdash\colon T]\!])$

4. structural equality:

$$[\![\Gamma \vdash \mathrm{StrEq}_A(M,N)]\!] = \\ \mathrm{Sect}_x([\![\Gamma \vdash M]\!])^*(\mathrm{Sect}_x(P_x([\![\Gamma \vdash\colon A]\!])^*([\![\Gamma \vdash N]\!]))^*(\hat{\mathrm{StrEq}}_{x,[\![\Gamma\vdash A]\!]}))$$

where $[\![\Gamma \vdash M]\!]$ and $[\![\Gamma \vdash N]\!]$ are elements of $[\![\Gamma \vdash\colon A]\!]$.

This somewhat unwieldy definition can be simplified using the soundness results for substitution proved in the next section:

By definition of substitutions we know that $\mathrm{StrEq}_A(M,N)$ is the same as $[x := M][x := \uparrow_x N]\,\mathrm{StrEq}_{\uparrow_x\uparrow_x A}(x^1, x^0)$, where type annotations have been omitted to improve legibility.

The above definition can now be stated as

$$[\![\Gamma, x\colon A, x\colon \uparrow_x A \vdash \mathrm{StrEq}_{\uparrow_x\uparrow_x A}(x^1,x^0)]\!] = \hat{\mathrm{StrEq}}_{x,[\![\Gamma\vdash A]\!]}$$

The general case is then derived using Theorem 3.5 below:

$$\begin{aligned}
&[\![\Gamma \vdash \mathrm{StrEq}_A(M,N)]\!] \\
&= [\![\Gamma \vdash [x := M][x := \uparrow_x N]\,\mathrm{StrEq}_{\uparrow_x A}(x^1,x^0)]\!] \\
&= [\![\Gamma \vdash [x := M]]\!]^*[\![\Gamma, x\colon A \vdash [x := \uparrow_x N]\,\mathrm{StrEq}_{\uparrow_x\uparrow_x A}(x^1,x^0)]\!] \\
&= [\![\Gamma \vdash [x := M]]\!]^* \\
&\qquad ([\![\Gamma, x\colon A \vdash [x := \uparrow_x N]]\!]^*[\![\Gamma, x\colon A, x\colon \uparrow_x A \vdash \mathrm{StrEq}_{\uparrow_x\uparrow_x A}(x^1,x^0)]\!]) \\
&= \mathrm{Sect}_x([\![\Gamma \vdash M]\!])^*(\mathrm{Sect}_x([\![\Gamma, x\colon A \vdash \uparrow_x]\!]^*([\![\Gamma \vdash N]\!]))^*(\hat{\mathrm{StrEq}}_{x,[\![\Gamma\vdash A]\!]})) \\
&= \mathrm{Sect}_x([\![\Gamma \vdash M]\!])^*(\mathrm{Sect}_x(P_x([\![\Gamma \vdash\colon A]\!])^*([\![\Gamma \vdash N]\!]))^*(\hat{\mathrm{StrEq}}_{x,[\![\Gamma\vdash A]\!]}))
\end{aligned}$$

It should be understood that this is only an *explanation*, and not a definition, since the proof of the soundness theorem for substitutions depends on the definition of the interpretation function.

5. reflexivity of structural equality:

   $[\![\Gamma \vdash \sigma(M,A)]\!] = \mathrm{Sect}_x([\![\Gamma \vdash M]\!])^*(s_x([\![\Gamma \vdash\colon A]\!]))$, when

   (a) $A := [\![\Gamma \vdash\colon A]\!] \downarrow$

   (b) $M := [\![\Gamma \vdash M]\!] \downarrow$ and M is an element of A

3.6 Semantics of TCINNI-OCC$_1$

TCINNI-OCC$_1$ substitutions are interpreted in the same way as for TCINNI-OCC$_0$. To prove soundness of this interpretation, we have to extend the proof of Theorem 2.23 by clauses for the new constructs.

Theorem 3.5 (Soundness of the semantics for TCINNI-OCC$_1$)**.** *For a pseudocontext Γ, a substitution ϑ and an OCC$_1$ pseudoterm M, we have*

$$(*) \qquad [\![\Gamma \vdash \vartheta M]\!] = [\![\Gamma \vdash \vartheta]\!]^*([\![\mathrm{cod}(\Gamma,\vartheta) \vdash M]\!])$$

and

$$(**) \qquad [\![\Gamma \vdash\colon \vartheta M]\!] = [\![\Gamma \vdash \vartheta]\!]^*([\![\mathrm{cod}(\Gamma, \vartheta) \vdash\colon M]\!])$$

whenever all the involved quantities are defined.
We then also have

$$(+) \qquad [\![\Gamma \vdash \vartheta]\!] \colon [\![\Gamma]\!] \to [\![\mathrm{cod}(\Gamma, \vartheta)]\!]$$

Proof. Let M be an OCC_1 pseudoterm which is not an OCC_0 pseudoterm, and ϑ a substitution. Assume ϑ satisfies $(+)$.

1. Case $M \equiv \mathrm{App}([x\colon A]\, B, T_1, T_2)$:

   Writing Δ for $\mathrm{cod}(\Gamma, \vartheta)$, A for $[\![\Delta \vdash\colon A]\!]$ and B for $[\![\Delta, x\colon A \vdash\colon B]\!]$, we see that

$$\begin{aligned}
& [\![\Gamma \vdash \vartheta M]\!] \\
= \;& [\![\Gamma \vdash \mathrm{App}([x\colon \vartheta A]\, (\Uparrow_{(x\colon \vartheta A)})B, \vartheta T_1, \vartheta T_2)]\!] \\
= \;& \mathrm{Sect}_x([\![\Gamma \vdash \vartheta T_2]\!])^*(\varepsilon_\Pi([\![\Gamma, x\colon \vartheta A \vdash\colon (\Uparrow_{(x\colon \vartheta A)})B]\!]) \\
& \qquad \circ P_x([\![\Gamma \vdash\colon \vartheta A]\!])^*([\![\Gamma \vdash \vartheta T_1]\!])) \\
= \;& \mathrm{Sect}_x([\![\Gamma \vdash \vartheta]\!]^*([\![\Delta \vdash T_2]\!]))^*(\varepsilon_\Pi(q([\![\Gamma \vdash \vartheta]\!], x, A)^*(B)) \circ \\
& \qquad P_x([\![\Gamma \vdash \vartheta]\!]^*(A))^*([\![\Gamma \vdash \vartheta]\!]^*([\![\Delta \vdash T_1]\!]))) \\
= \;& \mathrm{Sect}_x([\![\Gamma \vdash \vartheta]\!]^*([\![\Delta \vdash T_2]\!]))^*(q([\![\Gamma \vdash \vartheta]\!], x, A)^*(\varepsilon_\Pi(B)) \circ \\
& \qquad q([\![\Gamma \vdash \vartheta]\!], x, A)^*(P_x(A)^*([\![\Delta \vdash T_1]\!])))) \\
= \;& (q([\![\Gamma \vdash \vartheta]\!], x, A) \circ \mathrm{Sect}_x([\![\Gamma \vdash \vartheta]\!]^*([\![\Delta \vdash T_2]\!])))^* \\
& \qquad (\varepsilon_\Pi(B) \circ P_x(A)^*([\![\Delta \vdash T_1]\!])) \\
= \;& (\mathrm{Sect}_x([\![\Delta \vdash T_2]\!]) \circ [\![\Gamma \vdash \vartheta]\!])^*(\varepsilon_\Pi(B) \circ P_x(A)^*([\![\Delta \vdash T_1]\!])) \\
= \;& [\![\Gamma \vdash \vartheta]\!]^*(\mathrm{Sect}_x([\![\Delta \vdash T_2]\!])^*(\varepsilon_\Pi(B) \circ P_x(A)^*([\![\Delta \vdash T_1]\!]))) \\
= \;& [\![\Gamma \vdash \vartheta]\!]^*([\![\Delta \vdash \mathrm{App}([x\colon A]\, B, T_1, T_2)]\!])
\end{aligned}$$

2. Case $M \equiv \lambda x\colon A.B$:

   Again, write Δ for $\mathrm{cod}(\Gamma, \vartheta)$ and A for $[\![\Delta \vdash\colon A]\!]$.

$$\begin{aligned}
& [\![\Gamma \vdash \vartheta M]\!] \\
= \;& [\![\Gamma \vdash \lambda x\colon \vartheta A.(\Uparrow_{(x\colon \vartheta A)})B]\!] \\
= \;& \Pi_{x, [\![\Gamma \vdash\colon \vartheta A]\!]}([\![\Gamma, x\colon \vartheta A \vdash (\Uparrow_{(x\colon \vartheta A)})B]\!]) \circ \eta_\Pi(\mathbf{1}([\![\Gamma]\!])) \\
= \;& \Pi_{x, [\![\Gamma \vdash \vartheta]\!]^*(A)}([\![\Gamma, x\colon \vartheta A \vdash \Uparrow_{(x\colon \vartheta A)}]\!]^*([\![\Delta, x\colon A \vdash B]\!])) \circ \\
& \qquad \eta_\Pi([\![\Gamma \vdash \vartheta]\!]^*(\mathbf{1}([\![\Delta]\!]))) \\
= \;& \Pi_{x, [\![\Gamma \vdash \vartheta]\!]^*(A)}(q([\![\Gamma \vdash \vartheta]\!], x, A)^*([\![\Delta, x\colon A \vdash B]\!])) \circ \\
& \qquad [\![\Gamma \vdash \vartheta]\!]^*(\eta_\Pi(\mathbf{1}([\![\Delta]\!]))) \\
= \;& [\![\Gamma \vdash \vartheta]\!]^*(\Pi_{x,A}([\![\Delta, x\colon A \vdash B]\!])) \circ [\![\Gamma \vdash \vartheta]\!]^*(\eta_\Pi(\mathbf{1}([\![\Delta]\!]))) \\
= \;& [\![\Gamma \vdash \vartheta]\!]^*([\![\Delta \vdash M]\!])
\end{aligned}$$

3. Case $M \equiv \Pi x\colon A.B$:

   As above, write Δ for $\mathrm{cod}(\Gamma, \vartheta)$, A for $[\![\Delta \vdash A]\!]$, and B for $[\![\Delta, x\colon A \vdash B]\!]$.

$$\begin{aligned}
& [\![\Gamma \vdash \vartheta M]\!] \\
= \;& [\![\Gamma \vdash \Pi x\colon \vartheta A.(\Uparrow_{(x\colon \vartheta A)})B]\!] \\
= \;& \hat{\Pi}_{x, [\![\Gamma \vdash \vartheta A]\!]}([\![\Gamma, x\colon \vartheta A \vdash (\Uparrow_{(x\colon \vartheta A)})B]\!]) \\
= \;& \hat{\Pi}_{x, [\![\Gamma \vdash \vartheta]\!]^*(A)}([\![\Gamma, x\colon \vartheta A \vdash \Uparrow_{(x\colon \vartheta A)}]\!]^*(B)) \\
= \;& \hat{\Pi}_{x, [\![\Gamma \vdash \vartheta]\!]^*(A)}(q([\![\Gamma \vdash \vartheta]\!], x, A)^*(B)) \\
= \;& [\![\Gamma \vdash \vartheta]\!]^*(\hat{\Pi}_{x,A}(B))) \\
= \;& [\![\Gamma \vdash \vartheta]\!]^*([\![\Delta \vdash M]\!])
\end{aligned}$$

4. Case $M \equiv \mathrm{StrEq}_S(A, B)$:

   Write $\Delta := \mathrm{cod}(\Gamma, \vartheta)$, $A := [\![\Delta \vdash A]\!]$, $B := [\![\Delta \vdash B]\!]$, $S := [\![\Delta \vdash S]\!]$, $\vartheta := [\![\Delta \vdash \vartheta]\!]$.

   Then

$$\begin{aligned}
& [\![\Gamma \vdash \vartheta\, \mathrm{StrEq}_S(A, B)]\!] \\
= \;& [\![\Gamma \vdash \mathrm{StrEq}_{\vartheta S}(\vartheta A, \vartheta B)]\!] \\
= \;& \mathrm{Sect}_x([\![\Gamma \vdash \vartheta A]\!])^*(\mathrm{Sect}_x(P_x([\![\Gamma \vdash : \vartheta S]\!])^*([\![\Gamma \vdash \vartheta B]\!]))^* \\
& \qquad (\hat{\mathrm{StrEq}}_{x,[\![\Gamma \vdash \vartheta S]\!]})) \\
= \;& \mathrm{Sect}_x([\![\Gamma \vdash \vartheta]\!]^*(A))^*(\mathrm{Sect}_x(P_x([\![\Gamma \vdash \vartheta]\!]^*(S))^*([\![\Gamma \vdash \vartheta]\!]^*(B)))^* \\
& \qquad (\hat{\mathrm{StrEq}}_{x,[\![\Gamma \vdash \vartheta]\!]^*(S)})) \\
= \;& [\![\Gamma \vdash \vartheta]\!]^*(\mathrm{Sect}_x(A)^*((P_x(S)^*(B)))^*(\hat{\mathrm{StrEq}}_{x,S})) \\
= \;& [\![\Gamma \vdash \vartheta]\!]^*[\![\Delta \vdash \mathrm{StrEq}_S(A, B)]\!]
\end{aligned}$$

5. Case $M \equiv \sigma(M, A)$:

   With the usual abbreviations, we derive

$$\begin{aligned}
& [\![\Gamma \vdash \vartheta\sigma(A, S)]\!] \\
& = [\![\Gamma \vdash \sigma(\vartheta A, \vartheta S)]\!] \\
& = \mathrm{Sect}_x([\![\Gamma \vdash \vartheta A]\!])^*(s_x([\![\Gamma \vdash \vartheta S]\!])) \\
& = \mathrm{Sect}_x([\![\Gamma \vdash \vartheta]\!]^*(A))^*(q([\![\Gamma \vdash \vartheta]\!], x, S)^*(s_x(S))) \\
& = (q([\![\Gamma \vdash \vartheta]\!], x, S) \circ \mathrm{Sect}_x([\![\Gamma \vdash \vartheta]\!]^*(A)))^*(s_x(S)) \\
& = (\mathrm{Sect}_x(A) \circ [\![\Gamma \vdash \vartheta]\!])^*(s_x(S)) \\
& = [\![\Gamma \vdash \vartheta]\!]^*([\![\Delta \vdash \sigma(A, S)]\!])
\end{aligned}$$

□

3.7 Soundness

Finally, we extend the proof of the soundness theorem to OCC_1.

Theorem 3.6. *The given interpretation of contexts, types, and terms is sound in the sense that*

- *if* $\Gamma \vdash A : B$ *and* $C := [\![\Gamma]\!]\downarrow$, *then* $A := [\![\Gamma \vdash A]\!]\downarrow$ *and* $B := [\![\Gamma \vdash : B]\!]\downarrow$; A *is above* C *and* A *is an element of* B.
- *if* $\Gamma \vdash ||(A = B) : S$ *and* $C := [\![\Gamma]\!]\downarrow$, *then* $A := [\![\Gamma \vdash A]\!]\downarrow$, $B := [\![\Gamma \vdash : B]\!]\downarrow$ *and* $S := [\![\Gamma \vdash : S]\!]\downarrow$, *and* A *and* B *are equal elements of* S, *which is above* C.

Proof. Only the new rules need to be considered.

- Soundness of the rules (StrRefl), (StrSymm), and (StrTrans) is evident.
- Last rule used was

$$(\mathrm{TypeConv})\ \frac{\Gamma \vdash M : S \qquad \Gamma \vdash ||(S = T) : s}{\Gamma \vdash M : T}$$

Assume $C := [\![\Gamma]\!] \downarrow$; then by IH $M := [\![\Gamma \vdash M]\!] \downarrow$, $S := [\![\Gamma \vdash S]\!] \downarrow$ and $T := [\![\Gamma \vdash T]\!] \downarrow$, M is an element of $\mathcal{U}_C(S)$, and S and T are equal. Thus, M is also an element of $\mathcal{U}_C(T)$, and the conclusion is sound.

- Last rule used was

$$\text{(UnitElim)}\ \frac{\Gamma \vdash Q\colon \text{unit}}{\Gamma \vdash ||(Q = *)\colon \text{unit}}$$

  Assume $C := [\![\Gamma]\!] \downarrow$, then by IH $Q := [\![\Gamma \vdash Q]\!] \downarrow$, and Q is an element of $[\![\Gamma \vdash\colon \text{unit}]\!] = \mathbf{1}([\![\Gamma]\!])$. But $\mathbf{1}([\![\Gamma]\!])$ is terminal in the fiber over Γ, hence we must have $Q = \text{id}_{\mathbf{1}([\![\Gamma]\!])} = [\![\Gamma \vdash *]\!]$, verifying the conclusion.

- Last rule used was

$$\text{(Pi)}\ \frac{\Gamma \vdash S\colon s_1 \qquad \Gamma, x\colon S \vdash T\colon s_2}{\Gamma \vdash \Pi x\colon S.T\colon s_3}\ , \mathcal{R}(s_1, s_2) = s_3$$

  Assume $[\![\Gamma]\!] \downarrow$; then by IH $S := [\![\Gamma \vdash S]\!]$ is a uniquely defined element of $[\![\Gamma \vdash\colon s_1]\!]$. This means that $[\![\Gamma, x\colon S]\!] \downarrow$, hence again by IH that $T := [\![\Gamma, x\colon S \vdash T]\!]$ is a uniquely defined element of $[\![\Gamma, x\colon S \vdash\colon s_2]\!]$. But then, $[\![\Gamma \vdash \Pi x\colon S.T]\!] = \hat{\Pi}_{x,S}(T)$ is an element of $[\![\Gamma \vdash\colon s_3]\!]$, since $\mathcal{R}(s_1, s_2)$ is assumed to be defined.

- Last rule used was

$$\text{(Lda)}\ \frac{\Gamma \vdash S\colon s \qquad \Gamma, x\colon S \vdash M\colon T}{\Gamma \vdash \lambda x\colon S.M\colon \Pi x\colon S.T}\ , s \in \mathcal{S}$$

  Assume $[\![\Gamma]\!] \downarrow$. By the same reasoning as above, we find that $S := [\![\Gamma \vdash\colon S]\!] \downarrow$, and also $M := [\![\Gamma, x\colon S \vdash M]\!] \downarrow$, which is an element of $T := [\![\Gamma, x\colon S \vdash\colon T]\!] \downarrow$. Hence, $[\![\Gamma \vdash \lambda x\colon S.M]\!] = \Pi_{x,S}(M) \circ \eta_\Pi(\mathbf{1}([\![\Gamma]\!]))$ is an element of $\Pi_{x,S}(T) = [\![\Gamma \vdash\colon \Pi x\colon S.T]\!]$ as required.

- Last rule used was

$$\text{(App)}\ \frac{\Gamma \vdash M\colon \Pi x\colon S.T \qquad \Gamma \vdash N\colon S}{\Gamma \vdash \text{App}([x\colon S]\,T, M, N)\colon [x := N\colon S]T}$$

  Assume $[\![\Gamma]\!] \downarrow$. Then by IH

  - $P := [\![\Gamma \vdash\colon \Pi x\colon S.T]\!] \downarrow$, which implies that $T := [\![\Gamma, x\colon S \vdash\colon T]\!] \downarrow$
  - $M := [\![\Gamma \vdash M]\!] \downarrow$ and it is an element of P
  - $S := [\![\Gamma \vdash\colon S]\!] \downarrow$
  - $N := [\![\Gamma \vdash N]\!] \downarrow$ and it is an element of S

  Hence $(P_x(S))^*(M)$ is an element of $(P_x(S))^*(P) = (P_x(S))^*(\Pi_S(T))$.

  By definition,

$$\varepsilon_\Pi(T)\colon (P_x(S))^*(\Pi_S(T)) \to T$$

  This allows us to conclude that $\varepsilon_\Pi(T) \circ (P_x(S))^*(M)$ is an element of T, and finally that

$$[\![\Gamma \vdash \mathrm{App}([x\colon S]\,T, M, N)]\!] = \mathrm{Sect}_x(N)^*(\varepsilon_\Pi(T) \circ (P_x(S))^*(M))$$

is an element of $\mathrm{Sect}_x(N)^*([\![\Gamma, x\colon S \vdash\colon T]\!])$, which by Corollary 2.24 is the same as $[\![\Gamma \vdash\colon [x := N\colon S]T]\!]$.

- Last rule used was (StrPi), (StrLda), or (StrApp):

  All these rules are easily proved sound since structural equality is mapped to "real" equality between morphisms in the structure.

- Last rule used was

$$\text{(BetaRed)}\ \frac{\Gamma \vdash N\colon S \qquad \Gamma, x\colon S \vdash M\colon T}{\Gamma \vdash ||(\mathrm{App}([x\colon S]\,T, \lambda x\colon S.M, N) = [x := N\colon S]M)\colon [x := N\colon S]T}$$

  Assume $C := [\![\Gamma]\!]\downarrow$, then

  - by IH $N := [\![\Gamma \vdash N]\!]\downarrow$
  - N is an element of $S := [\![\Gamma \vdash\colon S]\!]\downarrow$
  - $[\![\Gamma, x\colon S]\!]\downarrow$, and again by IH, $M := [\![\Gamma, x\colon S \vdash M]\!]\downarrow$
  - M is an element of $T := [\![\Gamma, x\colon S \vdash\colon T]\!]\downarrow$

  Hence by definition of the interpretation function

  - $[\![\lambda x\colon S.M]\!] = \Pi_{x,S}(M) \circ \eta_\Pi(\mathbf{1}(C))$ is an element of $\Pi_{x,S}(T)$
  - $[\![\mathrm{App}([x\colon S]\,T, \lambda x\colon S.M, N)]\!] = \mathrm{Sect}_x(N)^*(\varepsilon_\Pi T \circ P_x(S)^*(\Pi_{x,S}(M) \circ \eta_\Pi(\mathbf{1}(C))))$ is an element of $\mathrm{Sect}_x(N)^*T$, which by Corollary 2.24 equals $[\![\Gamma \vdash\colon [x := N\colon S]T]\!]$
  - $[\![\Gamma \vdash [x := N\colon S]M]\!] = \mathrm{Sect}_x(N)^*M$ again by Corollary 2.24, and this also is an element of $\mathrm{Sect}_x(N)^*T = [\![\Gamma \vdash\colon [x := N\colon S]T]\!]$

  Using the definition of product types, we can now infer

$$\begin{aligned}
&\varepsilon_\Pi(T) \circ P_x(S)^*(\Pi_{x,S}(M) \circ \eta_\Pi(\mathbf{1}(C))) \\
&= \varepsilon_\Pi(T) \circ P_x(S)^*(\Pi_{x,S}(M)) \circ P_x(S)^*(\eta_\Pi(\mathbf{1}(C))) \\
&= M \circ \varepsilon_\Pi(P_x(S)^*(\mathbf{1}(C))) \circ P_x(S)^*(\eta_\Pi(\mathbf{1}(C))) \\
&= M \circ ((\varepsilon_\Pi P_x(S)^* \circ P_x(S)^*\eta_\Pi)(\mathbf{1}(C))) \\
&= M
\end{aligned}$$

- Last rule used was

$$\text{(StrEqTyForm)}\ \frac{\Gamma \vdash M\colon S \qquad \Gamma \vdash N\colon S}{\Gamma \vdash \mathrm{StrEq}_S(M, N)\colon \mathrm{Prop}}$$

  Assume $[\![\Gamma]\!]\downarrow$. Then by IH

  - $S := [\![\Gamma \vdash S]\!]\downarrow$
  - $M := [\![\Gamma \vdash M]\!]\downarrow$ and it is an element of S
  - $N := [\![\Gamma \vdash N]\!]\downarrow$ and it is an element of S

Thus by definition of $\hat{\text{StrEq}}$, $[\![\Gamma \vdash \text{StrEq}_S(M,N)]\!] \downarrow$ is an element of $[\![\Gamma \vdash \colon \text{Prop}]\!]$.

- Last rule used was

$$\text{(StrEqTyIntro)}\ \frac{\Gamma \vdash ||(M = N) \colon A}{\Gamma \vdash \sigma(M,A) \colon \text{StrEq}_A(M,N)}$$

Assume $[\![\Gamma]\!] \downarrow$, then by IH $M := [\![\Gamma \vdash M]\!] \downarrow$, $A := [\![\Gamma \vdash \colon A]\!] \downarrow$, and $N := [\![\Gamma \vdash N]\!] \downarrow$, and M and N are equal elements of A.

Thus we have (again omitting obvious type annotations):

$$\begin{aligned}
&[\![\Gamma \vdash \colon \text{StrEq}_A(M,N)]\!] \\
&= [\![\Gamma \vdash \colon \text{StrEq}_A(M,M)]\!] \\
&= [\![\Gamma \vdash \colon [x := M]\,\text{StrEq}_{\uparrow_x A}(x^0, x^0)]\!] \\
&= [\![\Gamma \vdash \colon [x := M][x := x^0]\,\text{StrEq}_{\uparrow_x \uparrow_x A}(x^1, x^0)]\!] \\
&= \text{Sect}_x(M)^*(\text{Sect}_x([\![\Gamma, x \colon A \vdash x^0]\!])^*(\text{StrEq}_{x,A})) \\
&= \text{Sect}_x(M)^*(\text{Sect}_x(\delta_{x,A})^*(\text{StrEq}_{x,A})) \\
&=: S
\end{aligned}$$

Also, $R := [\![\Gamma \vdash \sigma(M,A)]\!] = \text{Sect}_x(M)^*(s_x(A))$, and because $s_x(A)$ is an element of $\text{Sect}_x(\delta_{x,A})^*(\text{StrEq}_{x,A})$, we see that R is an element of S, as required.

- Last rule used was

$$\text{(StrEqTyElim)}\ \frac{\Gamma \vdash Q \colon \text{StrEq}_A(M,N)}{\Gamma \vdash ||(M = N) \colon A}$$

Assume $[\![\Gamma]\!] \downarrow$, then by IH $Q := [\![\Gamma \vdash Q]\!] \downarrow$ and $[\![\Gamma \vdash \colon \text{StrEq}_A(M,N)]\!] \downarrow$. By definition of the interpretation function, however, we can also deduce that $M := [\![\Gamma \vdash M]\!] \downarrow$, $N := [\![\Gamma \vdash N]\!] \downarrow$, $A := [\![\Gamma \vdash \colon A]\!] \downarrow$ and the former two are sections of the latter. Setting $P := P_x(A)$, $P' := P_x(P^*(A))$, $P'' := P_x(\text{StrEq}_{x,A})$, $N' := P^*(N)$, we find that

$$\begin{aligned}
&\text{Sect}_x(M) \\
&= P' \circ P'' \circ q(\text{Sect}_x(N') \circ \text{Sect}_x(M), \text{StrEq}_{x,A}) \circ \text{Sect}_x(Q) \\
&= q(P,x,A) \circ P'' \circ q(\text{Sect}_x(N') \circ \text{Sect}_x(M), \text{StrEq}_{x,A}) \circ \text{Sect}_x(Q) \\
&= \text{Sect}_x(N)
\end{aligned}$$

and hence $M = N$.

□

4 OCC_2: Rewriting

We now turn to our next system, OCC_2, which features support for rewriting types, i.e. types of proofs that some atom M can be rewritten to another atom N. Since atoms are themselves represented as morphisms in the semantics, it seems natural to represent rewrites as morphisms between morphisms, that is 2-cells. Consequently, we will extend our notion of a structure to use 2-categories as basis. The properties of a 2-category turn out to be a remarkably good match for modelling rewriting: 2-cells can only exist between 1-cells with the same source and target, just as rewritings can only exist between terms of the same type, identity and composite 2-cells guarantee reflexivity and transitivity of rewriting, and while vertical composition of 2-cells corresponds to sequential composition of rewrites, horizontal composition is connected with rewriting inside a term. These parallels were already exploited in Meseguer's 2-functorial account of the semantics of rewrite theories [13], and one might view our definitions as a step towards generalizing his "Lawvere 2-theories" to a dependently typed setting.

In Stehr's original presentation of OCC, rewriting is only available through rewriting predicates, there is no rewriting judgement. The different kinds of equality, on the other hand, are all modelled both by a type constructor and a corresponding judgement. We feel that the system becomes more symmetric with a new rewriting judgement, introduced below.

Our handling of rewriting in OCC_2 is intensional: the existence of a rewriting between two atoms does not provide us with new ways to reason about them (roughly speaking, rewriting types provide similar capabilities like Maude's `rl` rules). A more extensional version of rewriting is, in fact, computational equality, which we will introduce in the next section.

4.1 Syntax

An OCC_2 signature is an OCC_1 signature.

The set of *OCC_2 pseudoterms* $\mathcal{T}_2^\Sigma$, or simply $\mathcal{T}_2$ when the signature is understood from context, is defined as follows:

$$\begin{aligned}\mathcal{T}_2 ::= {} & \mathcal{S} \mid \mathcal{V}^{\mathbb{N}} \mid \mathrm{unit} \mid * \\ & \mid \lambda \mathcal{V}\colon \mathcal{T}_2.\mathcal{T}_2 \mid \Pi \mathcal{V}\colon \mathcal{T}_2.\mathcal{T}_2 \mid \mathrm{App}([\mathcal{V}\colon \mathcal{T}_2]\,\mathcal{T}_2, \mathcal{T}_2, \mathcal{T}_2) \\ & \mid \mathrm{StrEq}_{\mathcal{T}_2}(\mathcal{T}_2, \mathcal{T}_2) \mid \sigma(\mathcal{T}_2, \mathcal{T}_2) \\ & \mid \mathcal{T}_2 \xrightarrow{\mathcal{T}_2} \mathcal{T}_2 \mid \rho(\mathcal{T}_2, \mathcal{T}_2)\end{aligned}$$

where $\mathcal{V}$ is a fixed set of syntactic variable names and $\mathbb{N}$ is the set of natural numbers (including 0).

Besides the OCC_2 pseudoterms encountered before, there are two new syntactic constructions, namely rewriting types of the form $A \xrightarrow{S} B$, understood as the type of witnesses for rewrites from atom A to atom B (both of type S), and the canonical atom $\rho(A, S)$ which witnesses a rewrite from atom A to itself. A more full-fledged term language for rewrites (expressing, for example, transitivity or context closure like the one in [15]) would perhaps be desirable; we omit it for simplicity's sake.

Definition 4.1. The definition of the size of a pseudoterm is extended by two new clauses:

1. for $A, M, N \in \mathcal{T}_2$: $|M \xrightarrow{A} N| = |M| + |A| + |N| + 1$
2. for $A, M \in \mathcal{T}_2$: $|\rho(M, A)| = |M| + |A| + 1$

4.2 TCINNI-OCC$_2$

Definition 3.2 is extended once more to cover rewriting types.

Definition 4.2. On those OCC$_2$ pseudoterms which are not also OCC$_1$ pseudoterms, substitutions behave as follows:

$$\begin{array}{lcl} \vartheta(A \xrightarrow{S} B) & = & \vartheta A \xrightarrow{\vartheta S} \vartheta B \\ \vartheta\rho(M, A) & = & \rho(\vartheta M, \vartheta A) \end{array}$$

Not surprisingly, Lemma 2.6 and Lemma 2.8 carry over unchanged.

4.3 Formal System

There is one new judgement form:

Definition 4.3. An OCC$_2$ judgement is either an OCC$_1$ judgement or it is of the form

$$\Gamma \vdash M \to N : A$$

where Γ is a pseudocontext, M and N are pseudoterms, and A is a pseudotype; this judgement expresses that in context Γ, M and N are atoms of type A, and atom M rewrites to atom N.

A *valid OCC$_2$ judgement* is any judgement that can be derived by the rules of OCC$_1$ complemented with these new rules:

$$\text{(RewRefl)}\ \frac{\Gamma \vdash M : A}{\Gamma \vdash M \to M : A}$$

$$\text{(RewCong)}\ \frac{\Gamma \vdash M \to N : S \qquad \Gamma, x : S \vdash A : T \qquad \Gamma \vdash ||([x := M : S]T = [x := N : S]T) : s}{\Gamma \vdash [x := M : S]A \to [x := N : S]A : [x := M : S]T}\ , s \in \mathcal{S}$$

$$\text{(RewTrans)}\ \frac{\Gamma \vdash M \to M' : A \qquad \Gamma \vdash M' \to M'' : A}{\Gamma \vdash M \to M'' : A}$$

$$\text{(RwTyForm)}\ \frac{\Gamma \vdash M : S \qquad \Gamma \vdash N : S}{\Gamma \vdash M \xrightarrow{S} N : \text{Prop}}$$

$$\text{(RwTyIntro)}\ \frac{\Gamma \vdash M : S}{\Gamma \vdash \rho(M, S) : M \xrightarrow{S} N}$$

$$\text{(RwTyElim)}\ \frac{\Gamma \vdash R : M \xrightarrow{A} N}{\Gamma \vdash M \to N : A}$$

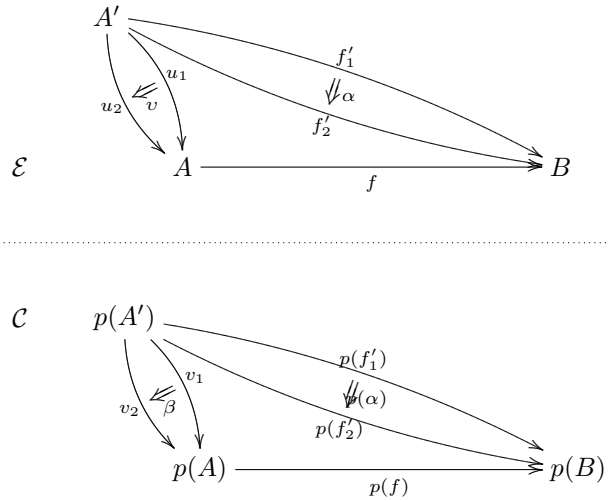


Figure 3: Deep Cartesian Arrow

Reflexivity and transitivity are natural properties of rewritings, and we will see that they arise equally naturally from our semantic model. In [20], it is emphasized that OCC does not stipulate any congruence closure properties of rewritings, and indeed our model does not a priori provide such properties; a simple extension of the semantics, though, will suffice to accommodate the above congruence rule (RewCong).

Observe that the third premise of this rule ensures that rewriting does not change the type of an object, even in the presence of dependent types. In the categorical interpretation, this will make sure that two source objects are mapped to the same target object by a functor.

The rule (RewCong) should not be confused with the following one (which is admissible in OCC_2):

$$\text{(RewSubst)}\ \frac{\Gamma, x\colon S \vdash A \to B\colon T \qquad \Gamma \vdash M\colon S}{\Gamma \vdash [x := M\colon S]A \to [x := M\colon S]B\colon [x := M\colon S]T}$$

4.4 OCC_2 Structures

Since we want to extend our structures to be based on 2-categories, we have to adapt some of our earlier definitions.

Definition 4.4. Let p be a 2-functor from $\mathcal{E}$ to $\mathcal{C}$. An arrow $f\colon A \to B$ in $\mathcal{E}$ is called *deep cartesian* if, for any arrow $f'_1\colon A' \to B$ in $\mathcal{E}$ and an arrow $v_1\colon p(A') \to p(A)$ such that $p(f'_1) = p(f) \circ v_1$, there is a unique arrow $u_1\colon A' \to A$ with $p(u_1) = v_1$ and $f'_1 = f \circ u_1$ (i.e., it is cartesian). Furthermore, if there is another arrow $f'_2\colon A' \to B$ in $\mathcal{E}$ and an arrow $v_2\colon p(A') \to p(A)$ such that, again, $p(f'_2) = p(f) \circ v_2$, and there are 2-cells $\alpha\colon f'_1 \Rightarrow f'_2$ and $\beta\colon v_1 \Rightarrow v_2$ such that $p(f) * \beta = p(\alpha)$, then there is a unique 2-cell $\upsilon\colon u_1 \Rightarrow u_2$ with $p(\upsilon) = \beta$ and $\alpha = f * \upsilon$.

The situation discussed in the definition is depicted in Figure 3.

Definition 4.5. A 2-functor p from $\mathcal{E}$ to $\mathcal{C}$ is called a *deep cloven fibration* if for every object A of $\mathcal{E}$ and every morphism $f\colon B \to p(A)$ in $\mathcal{C}$ there is a chosen deep cartesian arrow (called the *deep cartesian lifting* of f) $\overline{f}(A)$ above f. The domain of this arrow is written $f^*(A)$.

Definition 4.6. p is called a *deep split fibration* if it is a deep cloven fibration which is also split.

Deep split fibrations make it possible to extend reindexing functors to 2-functors:

Definition 4.7. Given a deep split fibration p from $\mathcal{E}$ to $\mathcal{C}$, every arrow $f\colon D \to C$ in $\mathcal{C}$ induces a 2-functor f^* from $\mathcal{E}_C$ to $\mathcal{E}_D$, called a *deep reindexing* or *deep pullback functor*.

On objects, it is defined by the cleavage. For a vertical arrow $k\colon A \to B$, we can see that both $k \circ \overline{f}(A)$ and $\overline{f}(B)$ are above f and the latter is deep cartesian. Thus, there must be a unique vertical arrow from $f^*(A)$ to $f^*(B)$, which we will call $f^*(k)$, such that $\overline{f}(B) \circ f^*(k) = k \circ \overline{f}(A)$.

Also, if there is another vertical arrow $h\colon A \to B$ and a 2-cell $r\colon h \to k$, then $r * \overline{f}(A)$ is a 2-cell from $h \circ \overline{f}(A)$ to $k \circ \overline{f}(A)$, thus there must be a unique 2-cell from $f^*(h)$ to $f^*(k)$, which we will call $f^*(r)$, such that $\overline{f}(B) * f^*(r) = r * \overline{f}(A)$.

Now we can define OCC_2-structures.

Definition 4.8. An OCC_2-structure $\mathcal{M}$ is an OCC_1-structure fulfilling the following additional requirements:

- $\mathcal{C}$ and $\mathcal{E}$ are 2-categories (since every 2-category is also a 1-category this does not prevent $\mathcal{M}$ from being an OCC_1-structure).
- p is a deep split fibration.
- For $x \in \mathcal{N}$, $\{-\}_x$ is a 2-functor and P_x is a 2-natural transformation.
- $\mathbf{1}_x$ and $\Pi_{x,A}$ are 2-functors for every $x \in \mathcal{N}$ and $A \in \mathcal{E}$; the corresponding adjunctions are 2-adjunctions.
- For any object $A \in \mathcal{E}$ and name $x \in \mathcal{N}$, there must be an object $\mathrm{Rw}_{x,A}$ above $\{P_x(A)^*(A)\}_x$ such that between the two morphisms

  $$P_x(P_x(A)^*(A)) \circ P_y(\mathrm{Rw}_{x,A})$$

  and

  $$q(P_x(A), x, A) \circ P_y(\mathrm{Rw}_{x,A})$$

  there is a 2-cell for any $y \in \mathcal{N}$[8]. Furthermore, there must be an element $r_x(A)$ of $\delta^*_{x,A}(\mathrm{Rw}_{x,A})$.

  Given this definition, we can see rewriting types as a "lax" kind of equality types, where we do not require the two extraction morphisms to be identical (i.e., connected by an identity 2-cell) but only to be connected

[8]This condition closely parallels the one for structural equality types.

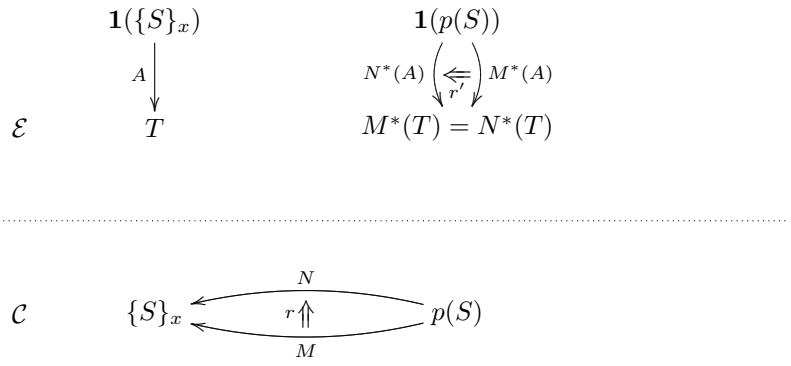


Figure 4: Congruence Closure

by *some* 2-cell; conversely, we can understand structural equality types as rewriting types where the 2-cell in question is not only guaranteed to exist, but guaranteed to be an identity 2-cell.

Like all other type constructors, rewriting types must be stable under substitution, i.e. for $f\colon D \to p(A)$ we must have

$$q(q(f, x, A), x, P_x(A)^*(A))^*(\mathrm{Rw}_A) = \mathrm{Rw}_{f^*(A)}$$

and

$$q(f, x, A)^*(r_x(A)) = r_x(f^*(A))$$

As before, we need an operator $\hat{\mathrm{Rw}}$ to mirror Rw's workings on the level of types: for $C \in \mathcal{C}$, $s \in \mathcal{S}_{\mathcal{C}}$, $\hat{A}\colon \mathbf{1}(C) \to C^*s$, $A := \mathcal{U}_C(\hat{A})$, $A^*A := \{P_x(A)^*(A)\}_x$, we have $\hat{\mathrm{Rw}}_{x,\hat{A}}\colon \mathbf{1}(A^*A) \to (A^*A)^*\mathrm{Prop}$, and $\mathcal{U}_{A^*A}(\hat{\mathrm{Rw}}_{x,A}) = \mathrm{Rw}_{x,A}$. Again, this operator needs to be stable under substitutions.

- An OCC_2-structure has to support *congruence closure*, that is, for any $S \in \mathcal{E}$ and T above $\{S\}_x$, any element A of T, two sections M, N of $\{S\}_x$ with $M^*(T) = N^*(T)$ and a 2-cell r between them we can choose a 2-cell $r^*(A)$ such that
$$r^*(A)\colon M^*(A) \Rightarrow N^*(A)$$

  The situation is illustrated in Figure 4 (where for technical reasons we abbreviate $r^*(A)$ as r').

  For comparison, Figure 5 shows the situation corresponding to the rule (RewSubst) mentioned above. Notice that the symmetry between the two rules (somewhat obscured in the syntax) is very clear here.

The correspondence between sections and elements introduced in Lemma 2.17 extends to 2-cells:

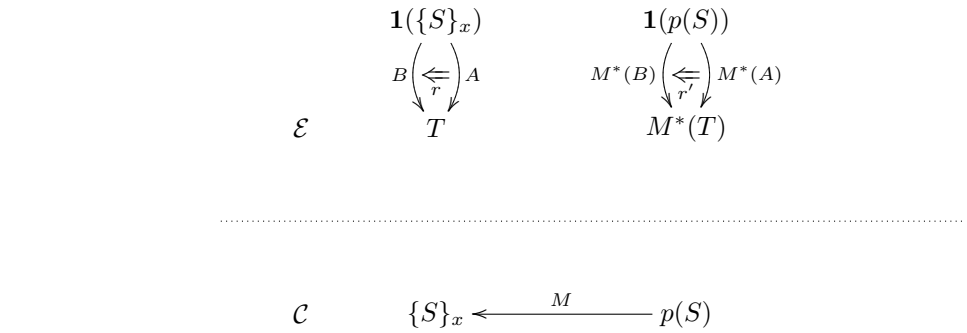


Figure 5: Rewriting under Substitution

Lemma 4.9. *For an object A of $\mathcal{E}$ and a name $x \in \mathcal{N}$, let* $\mathrm{Sections}_x(A)$ *be the full subcategory of $\mathcal{C}(p(A), \{A\}_x)$ comprising all sections of $P_x(A)$. Then there is a bijective functor* Sect_x *from $\mathcal{E}(\mathbf{1}(p(A)), A)$ to* $\mathrm{Sections}_x(A)$.

Proof. On objects (i.e., elements of A), we define the functor to simply be the mapping Sect_x seen before. For an arrow $\alpha\colon m \Rightarrow n$, where m and n are elements of A, define $\mathrm{Sect}_x(\alpha) := \{\alpha\}_x * \eta_{\mathbf{1}}(p(A))$.

This mapping preserves identities

$$\begin{aligned}\mathrm{Sect}_x(\mathrm{id}_m) &= \{\mathrm{id}_m\}_x * \eta_{\mathbf{1}}(p(A)) \\ &= \mathrm{id}_{\{m\}_x} * \mathrm{id}_{\eta_{\mathbf{1}}(p(A))} \\ &= \mathrm{id}_{\{m\}_x \circ \eta_{\mathbf{1}}(p(A))} \\ &= \mathrm{id}_{\mathrm{Sect}_x(m)}\end{aligned}$$

and composites

$$\begin{aligned}\mathrm{Sect}_x(\beta \circ \alpha) &= \{\beta \circ \alpha\}_x * \eta_{\mathbf{1}}(p(A)) \\ &= (\{\beta\}_x \circ \{\alpha\}_x) * (\mathrm{id}_{\eta_{\mathbf{1}}(p(A))} \circ \mathrm{id}_{\eta_{\mathbf{1}}(p(A))}) \\ &= (\{\beta\}_x * \mathrm{id}_{\eta_{\mathbf{1}}(p(A))}) \circ (\{\alpha\}_x * \mathrm{id}_{\eta_{\mathbf{1}}(p(A))}) \\ &= \mathrm{Sect}_x(\beta) \circ \mathrm{Sect}_x(\alpha)\end{aligned}$$

Bijectivity is established by giving an inverse functor: The mapping Sect_x^{-1} on elements was given before, for a 2-cell $\gamma\colon k \Rightarrow l$ (where $k, l \in \mathrm{Sect}_x(A)$), we define $\mathrm{Sect}_x^{-1}(\gamma) := \varepsilon_{\mathbf{1}}(A) * \mathbf{1}_x(\gamma)$. It is easily established that this, too, gives a functor; mutual inverseness follows from the triangular equations of the 2-adjunction.

□

In the same way that a 1-category can be turned into a 2-category, we can turn an OCC_1-structure into an OCC_2-structure.

Theorem 4.10. *Every OCC_1-structure can be extended to an OCC_2-structure.*

Proof. An identity 2-cell is adjoined to every arrow in $\mathcal{C}$ and $\mathcal{E}$, which makes both of them 2-categories. The functors p and $\{-\}_x$ are extended to 2-functors in

the canonical way (by mapping identity 2-cells to identity 2-cells), which makes p into a deep split fibration. Since there are only identity 2-cells, every natural transformation and adjoint becomes a 2-natural transformation resp. 2-adjoint. To complete the construction, set $\mathrm{Rw} := \mathrm{StrEq}$, $r_x := s_x$, and $\hat{\mathrm{Rw}} := \hat{\mathrm{StrEq}}$, which is easily seen to fulfill the conditions. □

4.5 The Interpretation Function

After these preparations, extending the interpretation function to cover the new syntactic constructs is easy. Let an OCC_2-signature

$$\Sigma = (\mathcal{S}, \mathcal{S}^i, \mathcal{S}^p, \mathrm{Prop}, \mathcal{A}, \mathcal{R}, \leq)$$

be given. To model the corresponding OCC_2-instance, we need an OCC_2-structure $\mathcal{M}$ such that the OCC_1-instance for Σ can be modelled in $\mathcal{M}$.

Again, only the interpretation function for terms needs to be extended:

4.5.1 Interpretation of Terms

1. rewriting types:

   $$[\![\Gamma \vdash M \xrightarrow{A} N]\!] = \\ \mathrm{Sect}_x([\![\Gamma \vdash M]\!])^*(\mathrm{Sect}_x(P_x([\![\Gamma \vdash : A]\!])^*([\![\Gamma \vdash N]\!]))^*(\hat{\mathrm{Rw}}_{x,[\![\Gamma \vdash A]\!]}))$$

   where $[\![\Gamma \vdash M]\!]$ and $[\![\Gamma \vdash N]\!]$ are elements of $[\![\Gamma \vdash : A]\!]$.

   Note that this definition exactly parallels the interpretation of structural equality types.

2. reflexivity of rewriting:
   $[\![\Gamma \vdash \rho(M, A)]\!] = \mathrm{Sect}_x([\![\Gamma \vdash M]\!])^*(r_x([\![\Gamma \vdash : A]\!]))$, when

   (a) $A := [\![\Gamma \vdash : A]\!] \downarrow$

   (b) $M := [\![\Gamma \vdash M]\!] \downarrow$ and M is an element of A

4.6 Semantics of TCINNI-OCC_2

The soundness proof for the interpretation of TCINNI-OCC_2 substitutions is a straightforward extension of the corresponding proof for OCC_1 in the same way as done for structural equality.

4.7 Soundness

Once more, we extend the proof of the soundness theorem.

Theorem 4.11. *The given interpretation of contexts, types, and terms is sound in the sense that*

- *if* $\Gamma \vdash A : B$ *and* $C := [\![\Gamma]\!] \downarrow$, *then* $A := [\![\Gamma \vdash A]\!] \downarrow$ *and* $B := [\![\Gamma \vdash : B]\!] \downarrow$; A *is above* C *and* A *is an element of* B.

- *if* $\Gamma \vdash ||(A = B)\colon S$ *and* $C := [\![\Gamma]\!]\downarrow$*, then* $A := [\![\Gamma \vdash A]\!]\downarrow$*,* $B := [\![\Gamma \vdash\colon B]\!]\downarrow$ *and* $S := [\![\Gamma \vdash\colon S]\!]\downarrow$*, and* A *and* B *are equal elements of* S*, which is above* $\mathcal{C}$*.*
- *if* $\Gamma \vdash A \to B\colon S$ *and* $C := [\![\Gamma]\!]\downarrow$*, then* $A := [\![\Gamma \vdash A]\!]\downarrow$*,* $B := [\![\Gamma \vdash\colon B]\!]\downarrow$ *and* $S := [\![\Gamma \vdash\colon S]\!]\downarrow$*, and* A *and* B *are elements of* S*, which is above* $\mathcal{C}$*, and there exists a 2-cell* $\alpha\colon A \Rightarrow B$*.*

Proof. Only the new rules need to be considered.

1. Last rule used was

$$\text{(RewRefl)}\ \frac{\Gamma \vdash M\colon A}{\Gamma \vdash M \to M\colon A}$$

   Assume $[\![\Gamma]\!]\downarrow$, then by IH $[\![\Gamma \vdash M]\!]\downarrow$ is an element of $[\![\Gamma \vdash\colon A]\!]\downarrow$. Since $\mathcal{E}(\mathbf{1}([\![\Gamma]\!]), [\![\Gamma \vdash\colon A]\!])$ is a category itself, it must have identity arrows, in particular there must be an identity 2-cell on $[\![\Gamma \vdash M]\!]$, which validates the conclusion.

2. Last rule used was

$$\text{(RewCong)}\ \frac{\begin{array}{c}\Gamma \vdash M \to N\colon S \qquad \Gamma, x\colon S \vdash A\colon T \\ \Gamma \vdash ||([x := M\colon S]T = [x := N\colon S]T)\colon s\end{array}}{\Gamma \vdash [x := M\colon S]A \to [x := N\colon S]A\colon [x := M\colon S]T}\ , s \in \mathcal{S}$$

   Assume $[\![\Gamma]\!]\downarrow$. From the premises, we obtain by induction hypothesis

   - $S := [\![\Gamma \vdash\colon S]\!]\downarrow$
   - $M := [\![\Gamma \vdash M]\!]\downarrow$, $N := [\![\Gamma \vdash N]\!]\downarrow$
   - M and N are elements of S and there is a 2-cell $r\colon M \Rightarrow N$
   - $T := [\![\Gamma, x\colon S \vdash\colon T]\!]\downarrow$
   - $A := [\![\Gamma, x\colon S \vdash A]\!]$ is an element of T
   - $\mathrm{Sect}_x(M)^*(T) = [\![\Gamma \vdash [x := T\colon M]]\!] = [\![\Gamma \vdash [x := T\colon N]]\!] = \mathrm{Sect}_x(N)^*(T)$ (see Corollary 2.24)

   To summarize, we have an object S, another object T above $\{S\}_x$, an element A of T, two sections $\mathrm{Sect}_x(M)$ and $\mathrm{Sect}_x(N)$ of $\{S\}_x$ such that $\mathrm{Sect}_x(M)^*(T) = \mathrm{Sect}_x(N)^*(T)$, and a 2-cell between them. The definition of congruence closure then assures us that there is a 2-cell

$$r^*(A)\colon\ \mathrm{Sect}_x(M)^*(A) \Rightarrow \mathrm{Sect}_x(N)^*(A)$$

   And since $\mathrm{Sect}_x(M)^*(A) = [\![\Gamma \vdash [x := M\colon S]A]\!]$ and $\mathrm{Sect}_x(N)^*(A) = [\![\Gamma \vdash [x := N\colon S]A]\!]$, the conclusion is sound.

3. Last rule used was

$$\text{(RewTrans)}\ \frac{\Gamma \vdash M \to M'\colon A \qquad \Gamma \vdash M' \to M''\colon A}{\Gamma \vdash M \to M''\colon A}$$

Assuming $[\![\Gamma]\!]\downarrow$, we get $M := [\![\Gamma \vdash M]\!]\downarrow$, $M' := [\![\Gamma \vdash M']\!]\downarrow$, $M'' := [\![\Gamma \vdash M'']\!]\downarrow$, all of them are elements of $[\![\Gamma \vdash : A]\!]\downarrow$, and there are 2-cells between M and M' and between M' and M'', respectively. Of course, this entails the existence of a 2-cell between M and M'' by category laws.

4. Last rule used was

$$\text{(RwTyForm)}\ \frac{\Gamma \vdash M : S \qquad \Gamma \vdash N : S}{\Gamma \vdash M \xrightarrow{S} N : \mathrm{Prop}}$$

Assume $[\![\Gamma]\!]\downarrow$. Then by IH

- $S := [\![\Gamma \vdash S]\!]\downarrow$
- $M := [\![\Gamma \vdash M]\!]\downarrow$ and it is an element of S
- $N := [\![\Gamma \vdash N]\!]\downarrow$ and it is an element of S

Thus by the definition of $\hat{\mathrm{Rw}}$, $[\![\Gamma \vdash M \xrightarrow{S} N]\!]$ is an element of $[\![\Gamma \vdash : \mathrm{Prop}]\!]$.

5. Last rule used was

$$\text{(RwTyIntro)}\ \frac{\Gamma \vdash M : A}{\Gamma \vdash \rho(M, A) : M \xrightarrow{A} M}$$

Assume $[\![\Gamma]\!]\downarrow$, then by IH $M := [\![\Gamma \vdash M]\!]\downarrow$, $A := [\![\Gamma \vdash : A]\!]\downarrow$, and M is an element of A.

An argument nearly identical to the case of structural equality types gives the desired result.

6. Last rule used was

$$\text{(RwTyElim)}\ \frac{\Gamma \vdash Q : M \xrightarrow{A} N}{\Gamma \vdash M \to N : A}$$

Assume $[\![\Gamma]\!]\downarrow$, then by IH $Q := [\![\Gamma \vdash Q]\!]\downarrow$ and $[\![\Gamma \vdash : M \xrightarrow{A} N]\!]\downarrow$. By definition of the interpretation function, however, we can also deduce that $M := [\![\Gamma \vdash M]\!]\downarrow$, $N := [\![\Gamma \vdash N]\!]\downarrow$, $A := [\![\Gamma \vdash : A]\!]\downarrow$ and the former two are sections of the latter.

Setting $P := P_x(A)$, $P' := P_x(P^*(A))$, $P'' := P_x(\mathrm{Rw}_{x,A})$, $N' := P^*(N)$, we find that

$$\mathrm{Sect}_x(M) = P' \circ P'' \circ q(\mathrm{Sect}_x(N') \circ \mathrm{Sect}_x(M), x, \mathrm{Rw}_{x,A}) \circ \mathrm{Sect}_x(Q)$$

and

$$\mathrm{Sect}_x(N) = q(P, x, A) \circ P'' \circ q(\mathrm{Sect}_x(N') \circ \mathrm{Sect}_x(M), x, \mathrm{Rw}_{x,A}) \circ \mathrm{Sect}_x(Q)$$

By the definition of rewriting types, there must be a 2-cell $\alpha : (P' \circ P'') \Rightarrow (q(P, x, A) \circ P'')$; but then, $\alpha * (q(\mathrm{Sect}_x(N') \circ \mathrm{Sect}_x(M), \mathrm{Rw}_{x,A}) \circ \mathrm{Sect}_x(Q))$ is a 2-cell between $\mathrm{Sect}_x(M)$ and $\mathrm{Sect}_x(N)$, which by Lemma 4.9 entails the existence of a 2-cell between M and N.

□

5 OCC_3: Computational Equality

Our last and most complicated system introduces computational equality, a rather peculiar feature of OCC. Computational equalities are reduction rules to be applied left-to-right, hence they can be understood as rewritings. Indeed, in our categorical model we will treat computational equality as a special class of 2-cells. In the formal system, some elimination rules for type constructors are relaxed from structural to computational equality. In the semantics this means relaxing an equality, which can be seen as requiring the existence of an identity 2-cell, to some (not necessarily identity) 2-cell. Category theorists have long been doing this in their investigation of 2-categories, creating "lax" versions of many categorical constructions such as functors, natural transformations, or adjoints [11].

Of course, this means that computationally equal terms are not always interpreted as equal elements in the semantics. We do, however, have to require that computationally equal types do in fact correspond to the same object: otherwise, the type conversion rule would become unsound.

5.1 Syntax

An OCC_3 signature is again nothing more than an OCC_2 signature.

The set of pseudoterms is extended by one new type constructor for computational equality and a canonical inhabitant:

$$\begin{aligned}\mathcal{T}_3 ::= {} & \mathcal{S} \mid \mathcal{V}^{\mathbb{N}} \mid \mathrm{unit} \mid * \\ & \mid \lambda \mathcal{V}\colon \mathcal{T}_3.\mathcal{T}_3 \mid \Pi \mathcal{V}\colon \mathcal{T}_3.\mathcal{T}_3 \mid \mathrm{App}([\mathcal{V}\colon \mathcal{T}_3]\,\mathcal{T}_3, \mathcal{T}_3, \mathcal{T}_3) \\ & \mid \mathrm{StrEq}_{\mathcal{T}_3}(\mathcal{T}_3, \mathcal{T}_3) \mid \sigma(\mathcal{T}_3, \mathcal{T}_3) \\ & \mid \mathcal{T}_3 \xrightarrow{\mathcal{T}_3} \mathcal{T}_3 \mid \rho(\mathcal{T}_3, \mathcal{T}_3) \\ & \mid \mathrm{CompEq}_{\mathcal{T}_3}(\mathcal{T}_3, \mathcal{T}_3) \mid \kappa(\mathcal{T}_3, \mathcal{T}_3)\end{aligned}$$

where $\mathcal{V}$ is a fixed set of syntactic variable names and $\mathbb{N}$ is the set of natural numbers (including 0).

As before, we introduce one new pseudotype former for computational equality types ($\mathrm{CompEq}_{\_}(-,-)$) and one pseudoterm former for the canonical witness of reflexivity of computational equality, both of which are used in the same way as the corresponding constructs for structural equality. Again, one might consider including more term formers here.

Definition 5.1. The new clause in the definition of the size of a pseudoterm should not be a surprise:

1. for $A, M, N \in \mathcal{T}_3$: $|\mathrm{CompEq}_A(M,N)| = |M| + |A| + |N| + 1$
2. for $A, M \in \mathcal{T}_3$: $|\kappa(M,A)| = |M| + |A| + 1$

5.2 TCINNI-OCC_3

Definition 3.2 is extended one last time to deal with computational equality types.

Definition 5.2. On those OCC_3 pseudoterms which are not also OCC_2 pseudoterms, substitutions behave as follows:

$$\begin{array}{lcl} \vartheta\,\mathrm{CompEq}_S(A,B) & = & \mathrm{CompEq}_{\vartheta S}(\vartheta A, \vartheta B) \\ \vartheta\kappa(M,A) & = & \kappa(\vartheta M, \vartheta A) \end{array}$$

Lemma 2.6 and Lemma 2.8 carry over unchanged.

5.3 Formal System

There is one new judgement form:

Definition 5.3. An OCC_3 judgement is either an OCC_2 judgement or it is of the form

$$\Gamma \vdash !!(M = N) \colon A$$

where Γ is a pseudocontext, M and N are pseudoterms, and A is a pseudotype; this judgement expresses that in context Γ, M and N are atoms of type A, and atom M is computationally equal to atom N.

A *valid OCC_3 judgement* is any judgement that can be derived by the rules of OCC_2 other than (BetaRed) and (TypeConv) or any of the following rules, which include new versions of the former two rules; observe that in the new (BetaRed) rule, the contractum is no longer structurally, but only computationally equal to the redex (as in OCC), while in (TypeRed), computational equality of types is now sufficient (again, as in OCC):

$$(\text{RedRefl})\ \frac{\Gamma \vdash M \colon A}{\Gamma \vdash !!(M = M) \colon A}$$

$$(\text{RedTrans})\ \frac{\Gamma \vdash !!(P = Q) \colon A \qquad \Gamma \vdash !!(Q = R) \colon A}{\Gamma \vdash !!(P = R) \colon A}$$

$$(\text{TypeRed})\ \frac{\Gamma \vdash M \colon S \qquad \Gamma \vdash !!(S = T) \colon s}{\Gamma \vdash M \colon T}$$

$$(\text{BetaRed})\ \frac{\Gamma \vdash N \colon S \qquad \Gamma, x \colon S \vdash M \colon T}{\Gamma \vdash !!(\mathrm{App}([x \colon S]\,T, \lambda x \colon S.M, N) = [x := N \colon S]M) \colon [x := N \colon S]T}$$

$$(\text{CompEqTyForm})\ \frac{\Gamma \vdash M \colon S \qquad \Gamma \vdash N \colon S}{\Gamma \vdash \mathrm{CompEq}_S(M,N) \colon \mathrm{Prop}}$$

$$(\text{CompEqTyIntro})\ \frac{\Gamma \vdash ||(M = N) \colon S}{\Gamma \vdash \kappa(M,S) \colon\ \mathrm{CompEq}_S(M,N)}$$

$$(\text{CompEqTyElim})\ \frac{\Gamma \vdash P \colon\ \mathrm{CompEq}_A(M,N)}{\Gamma \vdash !!(M = N) \colon A}$$

5.4 OCC_3 Structures

Definition 5.4. An OCC_3 structure $\mathcal{M}$ is an OCC_2 structure fulfilling the following extra conditions:

- There is a distinguished class Ξ of 2-cells, called *reduction 2-cells* in the total category. This class is closed under identity, vertical composition, and reindexing.
- Computationally equal types should correspond to the same object: For $C \in \mathcal{C}$, $s \in \mathcal{S}_{\mathcal{C}}$ and two elements A, B of C^*s with $\xi\colon A \Rightarrow B$, $\xi \in \Xi$, we must have $\mathcal{U}_C(A) = \mathcal{U}_C(B)$.
- For any object $A \in \mathcal{E}$ and name $x \in \mathcal{N}$, there is an object $\mathrm{CompEq}_{x,A}$ above $\{P_x(A)^*(A)\}_x$ such that between the two morphisms

  $$P_x(P_x(A)^*(A)) \circ P_y(\mathrm{CompEq}_{x,A})$$

  and

  $$q(P_x(A), x, A) \circ P_y(\mathrm{CompEq}_{x,A})$$

  there is a reduction 2-cell for any $y \in \mathcal{N}$. Furthermore, we require the existence of an element $c_x(A)$ of $\delta^*_{x,A}(\mathrm{CompEq}_{x,A})$.

  Thus, the type constructor for computational equality is located in between the very strict structural equality type constructor (which requires existence of an identity 2-cell between the two morphisms), and the very lax rewrite type constructor (where any 2-cell will do).

  Computational equality types must be stable under substitution, i.e. for $f\colon D \to p(A)$ we must have

  $$q(q(f, x, A), x, P_x(A)^*(A))^*(\mathrm{CompEq}_A) = \mathrm{CompEq}_{f^*(A)}$$

  and

  $$q(f, x, A)^*(c_x(A)) = c_x(f^*(A))$$

  Additionally, there is an operator $\hat{\mathrm{CompEq}}$ to mirror CompEq's workings on the level of types: for $C \in \mathcal{C}$, $s \in \mathcal{S}_{\mathcal{C}}$, $\hat{A}\colon \mathbf{1}(C) \to C^*s$, $A := \mathcal{U}_C(\hat{A})$, $A^*A := \{P_x(A)^*(A)\}_x$, we have $\hat{\mathrm{CompEq}}_{x,\hat{A}}\colon \mathbf{1}(A^*A) \to (A^*A)^*\mathrm{Prop}$, and $\mathcal{U}_{A^*A}(\hat{\mathrm{CompEq}}_{x,A}) = \mathrm{CompEq}_{x,A}$. This operator is required to be stable under substitutions in the same sense as $\hat{\mathrm{StrEq}}$.
- Product types in OCC_3 are no longer modelled by the strict product type former introduced before, but by a *lax product former*. Its definition is almost identical to the strict product types, and we will use the same notation for it. The only difference is that instead of requiring $\Pi_{x,A}$ to be a (strict) 2-adjoint to $P_x(A)^*$, we only need it to be a lax 2-adjoint, which

(for our purposes[9]) means that the triangular equations are only fulfilled up to reduction 2-cells:

For $A, B \in \mathcal{E}$ with B above $\{A\}_x$ and $X, Y \in \mathcal{E}$ with X above $p(A)$ and Y above $\{A\}_x$, there must be 2-cells $L(X), R(Y) \in \Xi$ such that

$$L(X)\colon \varepsilon_\Pi(P_x(A)^*(X)) \circ P_x(A)^*(\eta_\Pi(X)) \Rightarrow \mathrm{id}_{P_x(A)}$$

and

$$R(Y)\colon \mathrm{id}_{\Pi_{x,A}(Y)} \Rightarrow \Pi_{x,A}(\varepsilon_\Pi(Y)) \circ \eta_\Pi(\Pi_{x,A}(Y))$$

Theorem 5.5. *Every OCC_2-structure can be extended to an OCC_3-structure.*

Proof. We identify computational and structural equality, letting Ξ be the set of all identity 2-cells, and setting $\mathrm{CompEq} := \mathrm{StrEq}$, $c_x := s_x$, $\hat{\mathrm{CompEq}} := \hat{\mathrm{StrEq}}$. Due to the definition of Ξ, lax product types are the same as strict product types in this case. □

5.5 The Interpretation Function

Extending the interpretation function is routine. Let an OCC_3-signature $\Sigma = (\mathcal{S}, \mathcal{S}^i, \mathcal{S}^p, \mathrm{Prop}, \mathcal{A}, \mathcal{R}, \leq)$ be given. To model the corresponding OCC_3-instance, we need an OCC_3-structure $\mathcal{M}$ such that the OCC_2-instance for Σ can be modelled in $\mathcal{M}$.

Here are the interpretations of the new terms:

5.5.1 Interpretation of Terms

1. computational equality types:

   $$[\![\Gamma \vdash \mathrm{CompEq}_A(M, N)]\!] =$$
   $$\mathrm{Sect}_x([\![\Gamma \vdash M]\!])^*(\mathrm{Sect}_x(P_x([\![\Gamma \vdash\colon A]\!])^*([\![\Gamma \vdash N]\!]))^*(\hat{\mathrm{CompEq}}_{x,[\![\Gamma \vdash A]\!]}))$$

   where $[\![\Gamma \vdash M]\!]$ and $[\![\Gamma \vdash N]\!]$ are elements of $[\![\Gamma \vdash\colon A]\!]$.

   Note that this definition (again) exactly parallels the interpretation of structural equality types.

2. reflexivity of computational equality:

   $[\![\Gamma \vdash \kappa(M, A)]\!] = \mathrm{Sect}_x([\![\Gamma \vdash M]\!])^*(c_x([\![\Gamma \vdash\colon A]\!]))$, when

   (a) $A := [\![\Gamma \vdash\colon A]\!] \downarrow$

   (b) $M := [\![\Gamma \vdash M]\!] \downarrow$ and M is an element of A

5.6 Semantics of TCINNI-OCC_3

The soundness proof for the interpretation of TCINNI-OCC_3 substitutions is again nothing but an easy extension of the corresponding proof for OCC_2, which we will skip.

[9] In most definitions, lax 2-adjoints are even more flexible, requiring only lax functors and lax natural transformations; we will not need this extra flexibility.

5.7 Soundness

Here is the final version of the soundness theorem:

Theorem 5.6. *The given interpretation of contexts, types, and terms is sound in the sense that*

- *if* $\Gamma \vdash A : B$ *and* $C := [\![\Gamma]\!]\downarrow$, *then* $A := [\![\Gamma \vdash A]\!]\downarrow$ *and* $B := [\![\Gamma \vdash : B]\!]\downarrow$; A *is above* C *and* A *is an element of* B.
- *if* $\Gamma \vdash ||(A = B) : S$ *and* $C := [\![\Gamma]\!]\downarrow$, *then* $A := [\![\Gamma \vdash A]\!]\downarrow$, $B := [\![\Gamma \vdash : B]\!]\downarrow$ *and* $S := [\![\Gamma \vdash : S]\!]\downarrow$, *and* A *and* B *are equal elements of* S, *which is above* $\mathcal{C}$.
- *if* $\Gamma \vdash !!(A = B) : S$ *and* $C := [\![\Gamma]\!]\downarrow$, *then* $A := [\![\Gamma \vdash A]\!]\downarrow$, $B := [\![\Gamma \vdash : B]\!]\downarrow$ *and* $S := [\![\Gamma \vdash : S]\!]\downarrow$, *and* A *and* B *are elements of* S, *which is above* $\mathcal{C}$, *and there exists a reduction 2-cell* $\alpha : A \Rightarrow B$
- *if* $\Gamma \vdash A \to B : S$ *and* $C := [\![\Gamma]\!]\downarrow$, *then* $A := [\![\Gamma \vdash A]\!]\downarrow$, $B := [\![\Gamma \vdash : B]\!]\downarrow$ *and* $S := [\![\Gamma \vdash : S]\!]\downarrow$, *and* A *and* B *are elements of* S, *which is above* $\mathcal{C}$, *and there exists a 2-cell* $\alpha : A \Rightarrow B$.

Proof. Only the new rules need to be considered.

1. Last rule was

$$\text{(RedRefl)}\ \frac{\Gamma \vdash M : A}{\Gamma \vdash !!(M = M) : A}$$

   This case is handled like (RewRefl), noting that Ξ is closed under identity.

2. Last rule was

$$\text{(RedTrans)}\ \frac{\Gamma \vdash !!(P = Q) : A \qquad \Gamma \vdash !!(Q = R) : A}{\Gamma \vdash !!(P = R) : A}$$

   Again, Ξ's closure under transitivity allows us to handle this case like (RewTrans).

3. Last rule was

$$\text{(TypeRed)}\ \frac{\Gamma \vdash M : S \qquad \Gamma \vdash !!(S = T) : s}{\Gamma \vdash M : T}$$

   Assume $C := [\![\Gamma]\!]\downarrow$, then by IH $M := [\![\Gamma \vdash M]\!]\downarrow$ is an element of $[\![\Gamma \vdash : S]\!]\downarrow$. Also by IH, $S := [\![\Gamma \vdash S]\!]\downarrow$ and $T := [\![\Gamma \vdash T]\!]\downarrow$ are elements of $[\![\Gamma \vdash : s]\!]\downarrow$ with a reduction 2-cell $\xi \in \Xi$ between them. Thus, of course, $[\![\Gamma \vdash : S]\!] = \mathcal{U}_C(S) = \mathcal{U}_C(T) = [\![\Gamma \vdash : T]\!]$, and the conclusion is sound.

4. Last rule was

$$\text{(BetaRed)}\ \frac{\Gamma \vdash N : S \qquad \Gamma, x : S \vdash M : T}{\Gamma \vdash !!(\mathrm{App}([x : S]\,T, \lambda x : S.M, N) = [x := N : S]M) : [x := N : S]T}$$

Going back to the proof of the original (BetaRed) rule, we can see that it carries over to the case of lax product types almost completely. However, the last equality is only fulfilled up to the 2-cell $L(S) \in \Xi$, proving soundness of the new (BetaRed) rule.

5. Last rule was

$$\text{(CompEqTyForm)}\ \frac{\Gamma \vdash M\colon S \qquad \Gamma \vdash N\colon S}{\Gamma \vdash \mathrm{CompEq}_S(M,N)\colon \mathrm{Prop}}$$

   This is proved in the same way as (RewTyForm).

6. Last rule was

$$\text{(CompEqTyIntro)}\ \frac{\Gamma \vdash ||(M = N)\colon S}{\Gamma \vdash \kappa(M,S)\colon\ \mathrm{CompEq}_S(M,N)}$$

   See (RewTyIntro) and the soundness theorem for OCC_1.

7. Last rule was

$$\text{(CompEqTyElim)}\ \frac{\Gamma \vdash P\colon\ \mathrm{CompEq}_A(M,N)}{\Gamma \vdash !!(M = N)\colon A}$$

   See (RewTyElim) and the definition of computational equality types.

□

6 Conclusion

We have presented four type theories, similar in spirit to Stehr's Open Calculus of Constructions, gradually widening our focus from a very simple framework of atoms, types, and universes to a rich system with support for dependent types, rewriting, and different forms of equality.

For every system, we have defined a class of structures in which they can be interpreted, and proved that this interpretation is sound. We have not, however, given any examples of such structures. In fact, both a set-theoretic model, similar to the set-theoretic model in [9], and a term model can be given for (a slight extension of) OCC_3. The construction is not particularly difficult, but the proofs are long-winded and the notation somewhat heavy, so we decided not to include them in this presentation. Based on the term model, a completeness result is easily proved, which in turn might open the way for proofs of further meta-theoretic results.

We also want to point out that although our last system, OCC_3, comes close to the original OCC in many respects, it falls short of it or takes different paths in others. Generally speaking, our systems are stricter than OCC, for example by requiring explicit typing annotations for function application and equality types. As Stehr remarks, "the ... semantics [of OCC] is inherently untyped" [20], so the original system relies on dynamic typechecking conditions. This is reflected in Stehr's set-based semantics, where all functions are "totalized" so that they are applicable to arbitrary arguments.

While such a measure of freedom is certainly interesting, it is very hard to support in a category theory based approach. Category theory, it has been remarked, is a very "strongly typed" formalism; for example, each morphism has a fixed domain and codomain, unlike in set-theory, where we can assign different codomains to the same function. As usual in categorical semantics, we interpret atoms as elements resp. sections of their types (which are interpreted as objects). But this means that an object can only belong to one type, which makes it very hard to model, e.g., subtyping (as provided by OCC's "$\rightarrow:$" typing judgement) in a satisfactory manner.

In the end we found it more important to retain the general flavor of OCC, in particular the combination of dependent types, rewriting, and multiple kinds of equality, than to try and stick as closely as possible to one particular formulation of the system.

We feel that the approach of modelling rewritings and non-structural equalities using 2-categorical features is quite elegant and perhaps warrants further investigation, especially the connection between lax 2-categorical constructions and datatypes like the lax product, which allow elimination only up to a computational equality.

Another intriguing question is whether one can derive an abstract machine for type checking and reducing OCC terms from our semantics in the way it was done for CC by Ritter [18]. Our semantics, like Ritter's, is based on a D-Category structure, but it remains to be seen whether features like computational equality can be integrated into his approach.

Of course, OCC is not the only system aiming at a unification of type theory and rewriting. A recent system under active research is the Rho-Calculus [5] developed by Kirchner and his colleagues, which embeds term rewriting capabilities into lambda calculus by allowing abstraction not only on variables but

also on patterns, which are given first-class status. This calculus is wider in scope than OCC in that it can be instantiated to different type systems and more powerful in that it allows the encoding of rewriting strategies. So far, however, the investigation of its semantics seems to be confined to operational semantics as presented in [4], and it might be interesting to see whether the approach of this thesis could be extended to yield a categorical semantics for the Rho-Calculus.

A Basic Category Theory

This appendix does not try to give an introduction to category theory, we just recall some of the basic concepts in an informal manner. The standard reference on (1-)category theory still seems to be Mac Lane's book [12], a good introduction for computer scientists is Pierce's short treatise [16]. An overview of 2-category theory is given in Kelly and Street's paper [11]. We stick closely to the notational conventions used in these references.

A.1 1-Category Theory

A.1.1 (1-)Categories

A (1-)category $\mathcal{C}$ consists of a collection $\mathrm{Ob}(\mathcal{C})$ of objects and a collection $\mathrm{Mor}(\mathcal{C})$ of morphisms (or arrows). Each morphism f has a source object or domain $\mathrm{dom}(f)$ and a target object or codomain $\mathrm{cod}(f)$, both of which must be unique; to express that $A = \mathrm{dom}(f)$ and $B = \mathrm{cod}(f)$, we also write $f\colon A \to B$. The collection of all arrows between two objects A and B is written $\mathcal{C}(A, B)$ and called the *homset* between A and B. For each object A, there is an identity morphism $\mathrm{id}_A\colon A \to A$.

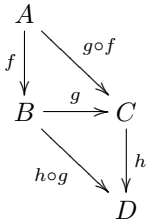
Two morphisms f and g with $\mathrm{cod}(f) = \mathrm{dom}(g)$ can be composed to yield $g \circ f\colon \mathrm{dom}(f) \to \mathrm{cod}(g)$. This composition must be associative, i.e. $h \circ (g \circ f) = (h \circ g) \circ f$, and identity arrows are neutral elements, i.e. $\mathrm{id}_{\mathrm{cod}(f)} \circ f = f = f \circ \mathrm{id}_{\mathrm{dom}(f)}$.

The paradigmatic example of a category is the category **Set** of sets and typed functions, whose objects are sets, and whose arrows are functions annotated with explicit domain and codomain (recall that in set theory every function has a unique domain, but its codomain is not fixed).

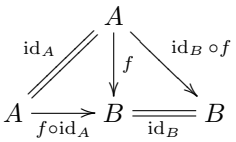
Any individual set can also be viewed as a category, with the set's elements as objects, and as morphisms just one identity morphism for each element. This arrangement also satisfies the category laws and is known as a *discrete category*.

A.1.2 Diagrams

Facts about objects and morphisms in a category are often expressed by commutative diagrams. For example, the associativity law corresponds to the following diagram:



Note that not all composites are depicted in the diagram, and identity arrows are omitted, except if we want to give them special emphasis as in this depiction of the identity laws, where they appear doubly stroked:



Often, we will not explicitly label identity arrows.

For emphasis or clarity, we will sometimes say $\mathcal{C}$-*diagram* for a diagram expressing some fact concerning a category $\mathcal{C}$.

A.1.3 Isomorphisms and Sections

An arrow $f\colon A \to B$ is called an isomorphism if it has an inverse arrow $f^{-1}\colon B \to A$ such that $f \circ f^{-1} = \mathrm{id}_B$ and $f^{-1} \circ f = \mathrm{id}_A$.

A right inverse to f is also called a section of f.

A.1.4 Initial and Terminal Objects

An object $\mathbf{0}$ of $\mathcal{C}$ is called initial if for any object A, there is exactly one arrow from $\mathbf{0}$ to A. Symmetrically, $\mathbf{1} \in \mathrm{Ob}(\mathcal{C})$ is called terminal if there is exactly one arrow $!_A\colon A \to \mathbf{1}$ from any object A to $\mathbf{1}$.

While not unique, initial and terminal objects are determined up to isomorphism, i.e. if there are two terminal objects $\mathbf{1}$ and $\mathbf{1}'$, then there is a unique isomorphism $i\colon \mathbf{1} \to \mathbf{1}'$.

In **Set**, the (only) initial object is the empty set, while any one-element set is a terminal object. Note that an arrow *from* a terminal object to any other object A (i.e., a function from some one-element set to A) can be seen as "picking out" one element of A. Analogously, in any category $\mathcal{C}$, we call an arrow from a terminal object $\mathbf{1}$ to another object A a (global) element of A.

A.1.5 Subcategories

A subcategory $\mathcal{C}$ of a category $\mathcal{D}$ contains some of $\mathcal{D}$'s objects and morphisms such that it is itself a category under the same composition and with the same identities.

In particular, $\mathcal{C}$ is called full if for any $A, B \in \mathrm{Ob}(\mathcal{C})$ we have $\mathcal{C}(A,B) = \mathcal{D}(A,B)$.

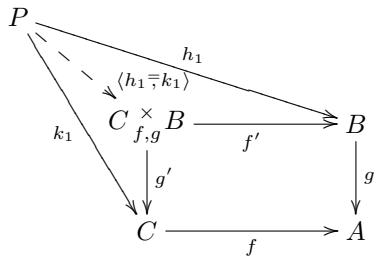
A.1.6 Pullbacks

Out of the plethora of constructions that can be performed in a category, we only need the notion of a pullback.

A pullback for two arrows $f\colon C \to A$ and $g\colon B \to A$ is given by

- an object $C \overset{\times}{_{f,g}} B$
- two arrows $f'\colon C \overset{\times}{_{f,g}} B \to B$ and $g'\colon C \overset{\times}{_{f,g}} B \to C$ such that $f \circ g' = g \circ f'$
- for any object P and arrows $h_1\colon P \to B$, $k_1\colon P \to C$ such that $f \circ k_1 = g \circ h_1$, there is a unique mediating arrow $\langle h_1 \overset{=}{,} k_1 \rangle$ with $f' \circ \langle h_1 \overset{=}{,} k_1 \rangle = h_1$ and $g' \circ \langle h_1 \overset{=}{,} k_1 \rangle = k_1$

The situation is summarized in the following diagram:



A.1.7 (1-)Functors

A (1-)functor F between categories $\mathcal{C}$ and $\mathcal{D}$ consists of two mappings F_0 and F_1, the first one mapping objects of $\mathcal{C}$ to objects of $\mathcal{D}$, and the second one mapping arrows $f\colon A \to B$ in $\mathcal{C}$ to arrows $F_1(f)\colon F_0(A) \to F_0(B)$ such that composition is preserved, i.e. $F_1(g \circ f) = F_1(g) \circ F_1(f)$, and identity arrows are also preserved, i.e. $F_1(\mathrm{id}_A) = \mathrm{id}_{F_0(A)}$. Normally, we write F for both F_0 and F_1.

For example, if $\mathcal{C}$ is a subcategory of $\mathcal{D}$, then there is an inclusion functor $\iota_{\mathcal{C},\mathcal{D}}$ from $\mathcal{C}$ to $\mathcal{D}$.

For any category $\mathcal{C}$, there is an identity functor $1_{\mathcal{C}}$ from $\mathcal{C}$ to itself, and two functors can be composed in the expected way; it is readily checked that this composition is associative and has identity functors as neutral elements. Thus, glossing over some cardinality considerations, we can speak of the category **Cat** of categories, which has categories as objects and functors as morphisms.

For a category $\mathcal{C}$, $\mathrm{Sub}(\mathcal{C})$ is the category of all full subcategories of $\mathcal{C}$, which itself is a full subcategory of **Cat**.

An example of a functor from **Set** to itself (i.e. an *endofunctor* on **Set**) is the *powerset functor* $\mathscr{P}$ which takes each set to its powerset and each function $f\colon A \to B$ to the function $\mathscr{P}(f)$ defined by

$$\mathscr{P}(f)(A') = \{f(a) \mid a \in A'\}$$

for any $A' \subseteq A$.

A.1.8 (1-)Natural Transformations

A (1-)natural transformation between two functors F and G, both between the same categories $\mathcal{C}$ and $\mathcal{D}$, is a map $\zeta\colon \mathrm{Ob}(\mathcal{C}) \to \mathrm{Mor}(\mathcal{C})$ such that for any arrow $f\colon A \to B$ in $\mathcal{C}$ we have

$$\begin{array}{ccc} F(A) & \xrightarrow{\zeta(A)} & G(A) \\ {\scriptstyle F(f)}\downarrow & & \downarrow{\scriptstyle G(f)} \\ F(B) & \xrightarrow[\zeta(B)]{} & G(B) \end{array}$$

We write this as $\zeta\colon F \xrightarrow{\bullet} G$.

For any functor F, there is an identity natural transformation $1_F \colon F \xrightarrow{\bullet} F$, given by $1_F(A) = \mathrm{id}_A$. Natural transformations $\zeta \colon F \xrightarrow{\bullet} G$ and $\vartheta \colon G \xrightarrow{\bullet} H$ can be composed in the obvious way to yield $\zeta \circ \vartheta \colon F \xrightarrow{\bullet} H$.

Perhaps unexpectedly, natural transformations can also be composed with functors. Indeed, if we have a pair of functors F, G between categories $\mathcal{C}$ and $\mathcal{D}$ with a natural transformation $\zeta \colon F \xrightarrow{\bullet} G$, and another pair of functors H, K between categories $\mathcal{D}$ and $\mathcal{E}$ with another natural transformation $\vartheta \colon H \xrightarrow{\bullet} K$, then we can form composites $H\zeta \colon H \circ F \xrightarrow{\bullet} H \circ G$ and $\vartheta F \colon H \circ F \xrightarrow{\bullet} K \circ F$ defined as

$$H\zeta(B) = H(\zeta(B))$$

and

$$\vartheta F(A) = \vartheta(F(A))$$

A.1.9 (1-)Adjoints

A (1-)adjunction between two categories $\mathcal{C}$ and $\mathcal{D}$ is a quadruple $(F, G, \eta, \varepsilon)$ where

- $F : \mathcal{C} \to \mathcal{D}$ is a functor
- $G : \mathcal{D} \to \mathcal{C}$ is a functor
- $\eta : 1_{\mathcal{C}} \xrightarrow{\bullet} G \circ F$ is a natural transformation
- $\varepsilon : F \circ G \xrightarrow{\bullet} 1_{\mathcal{D}}$ is a natural transformation
- we have:
$$(G\varepsilon) \circ (\eta G) = 1_G$$
$$(\varepsilon F) \circ (F\eta) = 1_F$$

More succinctly, this can be understood as a bijective correspondence between certain pairs of arrows, suggestively written as

$$\frac{F(A) \xrightarrow{u} B}{\overline{\overline{A \xrightarrow{v} U(B)}}}$$

A.2 2-Category Theory

A.2.1 2-Categories

A 2-category $\mathcal{C}$ has 0-cells/objects like A, 1-cells/arrows like $f \colon A \to B$ between objects A, B and 2-cells like $\alpha \colon f \Rightarrow g$ between arrows $f, g \colon A \to B$, more fully written as $\alpha \colon f \Rightarrow g \colon A \to B$.

$$A \overset{f}{\underset{g}{\Downarrow \alpha}} B$$

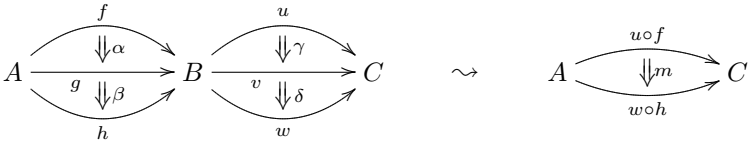
Objects and arrows form a (1-)category $\mathcal{C}_0$ (often just written $\mathcal{C}$). For two objects A, B, all the arrows between A and B form a (1-)category $\mathcal{C}(A, B)$, called

the *hom-category* between A and B. In particular, there must be an identity 2-cell id_f for any arrow $f\colon A \to B$, and for $\alpha\colon f \Rightarrow g$ and $\beta\colon g \Rightarrow h$ there must be a composite $\beta \circ \alpha\colon f \Rightarrow h$, called the *vertical composite* of α and β.

Where there is a vertical composition, there is also a horizontal composition: 2-cells $\alpha\colon f \Rightarrow g\colon A \to B$ and $\gamma\colon u \Rightarrow v\colon B \to C$ have a *horizontal composite* $\gamma * \alpha\colon u \circ f \Rightarrow v \circ g\colon A \to C$.

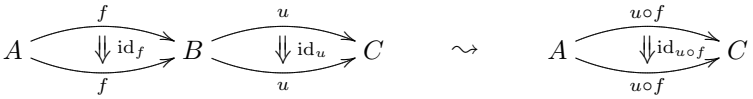
These two compositions must be compatible in the sense that for $\alpha\colon f \Rightarrow g\colon A \to B$, $\beta\colon g \Rightarrow h\colon A \to B$, $\gamma\colon u \Rightarrow v\colon B \to C$, $\delta\colon v \Rightarrow w\colon B \to C$ we have

$$(\delta * \beta) \circ (\gamma * \alpha) = (\delta \circ \gamma) * (\beta \circ \alpha) =: m$$

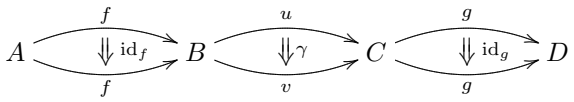


and

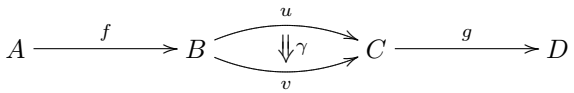
$$\mathrm{id}_u * \mathrm{id}_f = \mathrm{id}_{u \circ f}$$



When drawing diagrams, identity 2-cells are usually omitted; instead of



just use



The paradigmatic example of a 2-category is **Cat**, with categories as 0-cells, functors as 1-cells, and natural transformations as 2-cells. This shows that a 1-category can, of course, be a 2-category at the same time.

Every 1-category itself can also be considered as a 2-category in which there are only identity 2-cells; such a 2-category is called *discrete*.

A.2.2 2-Functors, 2-Natural Transformations, and 2-Adjoints

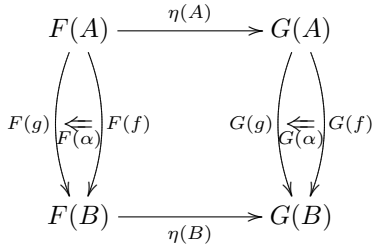
A 2-functor $F\colon \mathcal{C} \to \mathcal{D}$ sends objects of $\mathcal{C}$ to objects of $\mathcal{D}$, arrows of $\mathcal{C}$ to arrows of $\mathcal{D}$, and 2-cells of $\mathcal{C}$ to 2-cells of $\mathcal{D}$, preserving domains, codomains, identities, and compositions.

A 2-natural transformation $\eta\colon F \xrightarrow{\bullet} G\colon \mathcal{C} \to \mathcal{D}$ gives for every $A \in \mathrm{Ob}(\mathcal{C})$ an arrow $\eta(A)\colon F(A) \to G(A)$ such that for all $f, g\colon A \to B$ and $\alpha\colon f \Rightarrow g$, we have both

$$\eta(B) \circ F(f) = G(f) \circ \eta(A)$$

and

$$\mathrm{id}_{\eta(B)} * F(\alpha) = G(\alpha) * \mathrm{id}_{\eta(A)}$$



A 2-adjunction, finally, is simply a 1-adjointness with 1-functors replaced by 2-functors, and 1-natural transformations by 2-natural transformations.

B Formal System of OCC_3

For purposes of reference, we list here again the complete formal system of OCC_3.

$$\text{(Ax)}\ \frac{}{\Gamma \vdash s_1 \colon s_2}\ , \mathcal{A}(s_1) = s_2$$

$$\text{(Start1)}\ \frac{}{\Gamma, x \colon A \vdash x^0 \colon \uparrow_{(x \colon A)} A}$$

$$\text{(Start2)}\ \frac{\Gamma \vdash x^i \colon A}{\Gamma, x \colon B \vdash x^{i+1} \colon \uparrow_{(x \colon B)} A}$$

$$\text{(Start3)}\ \frac{\Gamma \vdash y^m \colon A}{\Gamma, x \colon B \vdash y^m \colon \uparrow_{(x \colon B)} A}\ , x \not\equiv y$$

$$\text{(UnitForm)}\ \frac{}{\Gamma \vdash \mathrm{unit} \colon \mathrm{Prop}}$$

$$\text{(UnitIntro)}\ \frac{}{\Gamma \vdash * \colon \mathrm{unit}}$$

$$\text{(UnitElim)}\ \frac{\Gamma \vdash Q \colon \mathrm{unit}}{\Gamma \vdash ||(Q = *) \colon \mathrm{unit}}$$

$$\text{(StrRefl)}\ \frac{\Gamma \vdash M \colon A}{\Gamma \vdash ||(M = M) \colon A}$$

$$\text{(StrSymm)}\ \frac{\Gamma \vdash ||(M = N) \colon A}{\Gamma \vdash ||(N = M) \colon A}$$

$$\text{(StrTrans)}\ \frac{\Gamma \vdash ||(P = Q) \colon A \qquad \Gamma \vdash ||(Q = R) \colon A}{\Gamma \vdash ||(P = R) \colon A}$$

$$\text{(Pi)}\ \frac{\Gamma \vdash S \colon s_1 \qquad \Gamma, x \colon S \vdash T \colon s_2}{\Gamma \vdash \Pi x \colon S.T \colon s_3}\ , \mathcal{R}(s_1, s_2) = s_3$$

$$\text{(Lda)}\ \frac{\Gamma \vdash S \colon s \qquad \Gamma, x \colon S \vdash M \colon T}{\Gamma \vdash \lambda x \colon S.M \colon \Pi x \colon S.T}\ , s \in \mathcal{S}$$

$$\text{(App)}\ \frac{\Gamma \vdash M \colon \Pi x \colon S.T \qquad \Gamma \vdash N \colon S}{\Gamma \vdash \mathrm{App}([x \colon S]\, T, M, N) \colon [x := N \colon S]T}$$

$$\text{(StrPi)}\ \frac{\Gamma \vdash ||(S = S') \colon s_1 \qquad \Gamma, x \colon S \vdash ||(T = T') \colon s_2}{\Gamma \vdash ||(\Pi x \colon S.T = \Pi x \colon S'.T') \colon s_3}\ , \mathcal{R}(s_1, s_2) = s_3$$

$$\text{(StrLda)}\ \frac{\Gamma \vdash ||(S = S') \colon s \qquad \Gamma, x \colon S \vdash ||(M = M') \colon S}{\Gamma \vdash ||(\lambda x \colon S.M = \lambda x \colon S'.M') \colon \Pi x \colon S.T}\ , s \in \mathcal{S}$$

$$\text{(StrApp)}\ \frac{\Gamma \vdash ||(M = M') \colon \Pi x \colon S.T \qquad \Gamma \vdash ||(N = N') \colon S}{\Gamma \vdash ||(\mathrm{App}([x \colon S]\, T, M, N) = \mathrm{App}([x \colon S]\, T, M', N')) \colon [x := N \colon S]T}$$

$$\text{(BetaRed)}\ \frac{\Gamma \vdash N : S \qquad \Gamma, x : S \vdash M : T}{\Gamma \vdash !!(\mathrm{App}([x : S]\,T, \lambda x : S.M, N) = [x := N : S]M) : [x := N : S]T}$$

$$\text{(StrEqTyForm)}\ \frac{\Gamma \vdash M : S \qquad \Gamma \vdash N : S}{\Gamma \vdash \mathrm{StrEq}_S(M, N) : \mathrm{Prop}}$$

$$\text{(StrEqTyIntro)}\ \frac{\Gamma \vdash M : S}{\Gamma \vdash \sigma(M, S) : \ \mathrm{StrEq}_S(M, M)}$$

$$\text{(StrEqTyElim)}\ \frac{\Gamma \vdash P : \ \mathrm{StrEq}_A(M, N)}{\Gamma \vdash ||(M = N) : A}$$

$$\text{(RewRefl)}\ \frac{\Gamma \vdash M : A}{\Gamma \vdash M \to M : A}$$

$$\text{(RewCong)}\ \frac{\begin{array}{c}\Gamma \vdash M \to N : S \qquad \Gamma, x : S \vdash A : T \\ \Gamma \vdash ||([x := M : S]T = [x := N : S]T) : s\end{array}}{\Gamma \vdash [x := M : S]A \to [x := N : S]A : [x := M : S]T}\ , s \in \mathcal{S}$$

$$\text{(RewTrans)}\ \frac{\Gamma \vdash M \to M' : A \qquad \Gamma \vdash M' \to M'' : A}{\Gamma \vdash M \to M'' : A}$$

$$\text{(RwTyForm)}\ \frac{\Gamma \vdash M : S \qquad \Gamma \vdash N : S}{\Gamma \vdash M \xrightarrow{S} N : \mathrm{Prop}}$$

$$\text{(RwTyIntro)}\ \frac{\Gamma \vdash M : S}{\Gamma \vdash \rho(M, S) : M \xrightarrow{S} M}$$

$$\text{(RwTyElim)}\ \frac{\Gamma \vdash R : M \xrightarrow{A} N}{\Gamma \vdash M \to N : A}$$

$$\text{(RedRefl)}\ \frac{\Gamma \vdash M : A}{\Gamma \vdash !!(M = M) : A}$$

$$\text{(RedTrans)}\ \frac{\Gamma \vdash !!(P = Q) : A \qquad \Gamma \vdash !!(Q = R) : A}{\Gamma \vdash !!(P = R) : A}$$

$$\text{(TypeRed)}\ \frac{\Gamma \vdash M : S \qquad \Gamma \vdash !!(S = T) : s}{\Gamma \vdash M : T}\ , s \in \mathcal{S}$$

$$\text{(CompEqTyForm)}\ \frac{\Gamma \vdash M : S \qquad \Gamma \vdash N : S}{\Gamma \vdash \mathrm{CompEq}_S(M, N) : \mathrm{Prop}}$$

$$\text{(CompEqTyIntro)}\ \frac{\Gamma \vdash ||(M = N) : S}{\Gamma \vdash \kappa(M, S) : \ \mathrm{CompEq}_S(M, N)}$$

$$\text{(CompEqTyElim)}\ \frac{\Gamma \vdash P : \ \mathrm{CompEq}_A(M, N)}{\Gamma \vdash !!(M = N) : A}$$

References

[1] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean J. Lèvy. Explicit substitutions. In *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco, California*, pages 31–46. ACM, 1990.

[2] Henk Barendregt. Lambda calculi with types. In Abramsky, Gabbay, and Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2. Clarendon Press, Oxford, 1992.

[3] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development*. Texts in Theoretical Computer Science. Springer-Verlag, 2004.

[4] Horatiu Cirstea, Claude Kirchner, and Luigi Liquori. Rewriting calculus with(out) types. In Fabio Gadducci and Ugo Montanari, editors, *Proceedings of the fourth workshop on rewriting logic and applications*, Pisa (Italy), 2002. Electronic Notes in Theoretical Computer Science.

[5] Horatiu Cirstea, Luigi Liquori, and Benjamin Wack. Rewriting calculus with fixpoints: Untyped and first-order systems. In *Types for Proofs and Programs*, volume 3085 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.

[6] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and José F. Quesada. The Maude system. In P. Narendran and M. Rusinowitch, editors, *Rewriting Techniques and Applications, 10th International Conference, RTA'99, Trento, Italy, July 2–4, 1999, Proceedings*, volume 1631 of *Lecture Notes in Computer Science*, pages 240–243. Springer-Verlag, 1999.

[7] Thierry Coquand and Gerard Huet. The calculus of constructions. *Information and Computation*, 76(2–3), 1988.

[8] Thierry Coquand and Christine Paulin-Mohring. Inductively defined types. In *Proceedings of Colog'88*, volume 417 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[9] Martin Hofmann. Syntax and semantics of dependent types. In A. M. Pitts and P. Dybjer, editors, *Semantics and Logics of Computation*, volume 14, pages 79–130. Cambridge University Press, Cambridge, 1997.

[10] Bart Jacobs. *Categorical Type Theory*. PhD thesis, University of Nijmegen, The Netherlands, September 1991.

[11] G. M. Kelly and Ross Street. *Review of the elements of 2-categories*, volume 420 of *Lecture Notes in Mathematics*, pages 75–103. Springer-Verlag, 1974.

[12] Saunders Mac Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer-Verlag, 1971.

[13] José Meseguer. Functorial semantics of rewrite theories. *Lecture Notes in Computer Science : Formal Methods in Software and Systems Modeling*, pages 220–235, 2005.

[14] Bengt Nordström, Kent Petersson, and Jan M. Smith. *Programming in Martin-Löf's Type Theory*, volume 7 of *International Series of Monographs on Computer Science*. Clarendon Press, Oxford, 1990.

[15] Martí N. Oliet and J. Meseguer. Rewriting logic as a logical and semantic framework, 1993.

[16] Benjamin C. Pierce. *Basic Category Theory for Computer Scientists*. Foundations of Computing. The MIT Press, 1991.

[17] John C. Reynolds. Polymorphism is not set-theoretic. In *Semantics of Data Types*, volume 173 of *Lecture Notes in Computer Science*, pages 145–156. Springer-Verlag, 1984.

[18] Eike Ritter. *Categorical Abstract Machines for Higher-Order Type Lambda Calculi*. PhD thesis, University of Cambridge, September 1992.

[19] Mark-Oliver Stehr. Cinni - a generic calculus of explicit substitutions and its application to lambda-, sigma- and pi-calculi. *Electronic Notes in Theoretical Computer Science*, 36, 2000.

[20] Mark-Oliver Stehr. *Programming, Specification, and Interactive Theorem Proving*. PhD thesis, University of Hamburg, September 2002.

[21] Thomas Streicher. *Semantics of Type Theory: Correctness and Completeness*. Progress in Theoretical Computer Science. Birkhäuser, December 1991.

[22] Benjamin Werner. Sets in types, types in sets. In *Proceedings of TACS'97*, volume 1281 of *Lecture Notes in Computer Science*, pages 530–546. Springer-Verlag, 1997.

Index