Freie Universität Berlin

Categorical Semantics of Intuitionistic Multiplicative Linear Logic and its Formalization in Isabelle/HOL

Master's thesis

17 May 2021

Name:	Alexey Gonus
Matriculation number:	5301263
E-mail address:	gonua96@zedat.fu-berlin.de
Department:	Mathematics and Computer Science
Institute:	Institute of Mathematics
Study program:	Master of Science in Mathematics
First supervisor:	Prof. Dr. Christoph Benzmüller
Second supervisor:	Prof. Dr. Klaus Altmann
External advisor and reviewer:	Prof. Dr. Dana Scott

Abstract

Linear logic (LL) is a substructural logic which captures the idea of linearity in proofs in a controlled manner. There are different fragments of linear logic which can be considered, and the simplest one is the intuitionistic multiplicative LL (IMLL) fragment. The goals of this thesis are (1) to provide a careful investigation of the categorical concepts connected to the denotational semantics of IMLL as a starting point in the view of free higher-order logic, and (2) to model IMLL semantics effectively in the interactive proof assistant Isabelle/HOL. This formalization then leaves us with the translation of IMLL into the classical higher-order logic and allows future developments and verification of other linear logic fragments.

Statutory Declaration

I hereby declare that I have written the present thesis independently, without enlisting any external assistance, and only using the specified aids. I additionally assert that this thesis has not been part of another examination process.

....

Date & Place Alexey Gonus

Contents

List of Tables

Chapter 1

Introduction

1.1 Motivation

Linear Logic (LL), generally speaking, carries the idea of treating mathematical "truths" as information resources. LL has found a large number of theoretical and practical applications, especially in computer science but even in linguistics, since in LL the concepts of interaction and resource awareness are of primary importance. Moreover, with LL there is a simple and natural way to describe many computational models [Lincoln, 1992]. LL ideas have even been applied in developing a new and promising kind of proof formalisms, namely in *deep inference* [Tubella and Straßburger, 2019]. This paper deals with a particular fragment of LL, i.e. intuitionistic multiplicative LL (IMLL), but nevertheless the method and formalization steps applied in this research can be continued to describe other more complex fragments. The denotational semantical model of IMLL, if we consider it as a proof system, is expressed through symmetrical monoidal closed categories. In the way of creating and describing the foundations of mathematics, there is always a choice between which logical systems with sufficient power to put into the foundation of mathematical language, and the battle between these systems seems likely to continue. This has been one of the motivations for this paper. Ultimately, there are several goals regarding IMLL. The first can be described as a thorough investigation of the categorical concepts associated with the denotational semantics of IMLL, in terms of higher order free logic (FHOL). The choice towards FHOL was made for the reason that theory of categories is more naturally described in the language of free logic due to the use of partial domains. The second goal of the work is to provide formalization of the studied concepts and notions into a computational system. This is accomplished by faithful shallow semantical embedding (SSE) of FHOL into the Isabelle/HOL [Nipkow et al., 2002] system, which is the interactive proof assistant of our choice for this work. The mentioned purpose was based on several considerations and ideas. Firstly, this formalization provides a kind of "bridge" or translation between the logics of interest, IMLL, FHOL and HOL, which can be regarded as logical systems for describing the foundations of mathematics, as mentioned above. With this translation in place, we can do the gluing of the mentioned logics and the proofs already constructed in them into a single Isabelle/HOL system. Secondly, there is a pragmatic motivation in such formalization and encoding, since in parallel we are testing how far we can get in modelling structures and logical theories in the modern theorem prover.

The methodology of such a study consists of the following steps. To begin with, we have described all the semantical concepts of intuitionistic multiplicative linear logic in the language of free logic in the world of object-free categories. In parallel, difficult to carry out proofs for computer were firstly carried out on paper and only then formalized into the Isabelle/HOL system. The main problem was to reformulate the language of category theory with two types of objects, based on the language of classical logic, into the language of free logic, where categories are constructed using only one

type, namely morphisms. Therefore, in the process of formalization it was necessary to redesign the system of axioms for categories, functors and other notions.

1.2 Isabelle/HOL

Isabelle is a generic system that provides an infrastructure for building deductive systems, logical formalisms and object logics. A special focus of the system is on the integrating of interactive and automated theorem proving and one of the implementations of such a system in particular is Isabelle/HOL [Nipkow et al., 2002], the one exploited in this thesis, aimed for the specification of *classical higher-order logic* (HOL) (equivalently referred to as *simple theory of types*). Isabelle/HOL uses polymorphic higher-order logic supplemented with axiomatic types which gives a broad module system for creating mathematical theories or program verification. The prover uses formal proof Isar language [Wenzel, 1999] that is an additional feature supporting user interaction. For proof construction different state-of-the-art first-order and higher-order automated theorem provers are integrated (modulo suitable logic translations) with Isabelle via the meta-prover Sledgehammer [Blanchette et al., 2013, Paulsson and Blanchette, 2012]. It invokes to prominent first-order automated theorem provers such as CVC4 [Deters et al., 2014], Z3 [de Moura and Bjørner, 2008], E [Schulz, 2013], Vampire [Kovács and Voronkov, 2013] and Spass [Blanchette et al., 2012]. Remotely, also higher-order provers such as Satallax [Brown, 2012] or LEO-III [Steen and Benzmüller, 2019] can be reached. There is an extremely useful finite model finder Nitpick [Blanchette and Nipkow, 2010] implemented inside Isabelle/HOL which helps with debugging theories and formal constructions and prevents from future mistakes. There are other alternatives aiming for mathematic's formalisation, e.g. Coq [Mahboubi and Tassi, 2021] and Lean [de Moura et al., 2015], but Isabelle/HOL seems the most suitable for the purpose of this work due to its minimalist machinery, long history and extensive supplementary library of mathematical structures at the same time. To mention here as well, there was a work by Chaudhuri and Pfenning on implementation of a theorem prover for first-order intuitionistic linear logic [Chaudhuri and Pfenning, 2005].

Chapter 2

Preliminaries

2.1 Classical higher-order logic

Simple theory of types (STT), also called Church's type theory, is an expressive logical formalism that builds classical higher-order logic (HOL)[1] on top of the simply typed λ-calculus [Barendregt et al., 2013]. STT was originally presented by Church in 1940 [Church, 1940], generalized by Henkin in 1950 to *extensional type theory*. It is presented and rephrased below with a reference to [Benzmüller and Andrews, 2019] with primitive equality and definite description operator.

2.1.1 Syntax

The main notions and building blocks of STT are types and terms, or more precisely, typed terms. The set of types is generated from the set of two primitive types $\{o, i\}$[2] and the right associative function constructor $\rightarrow$. One may see o as the type of Boolean truth values and i as the type of individuals.

Definition 2.1.1.1. The set $\mathcal{T}$ of simple types is given by the following abstract grammar:

$$\alpha, \beta \coloneqq o \mid i \mid \alpha \rightarrow \beta$$

Definition 2.1.1.2. The subset $\mathcal{T}_o \subset \mathcal{T}$ of simple types of type o is given by the following abstract grammar:

$$\beta \coloneqq o \mid \alpha \rightarrow \beta,$$

with $\alpha \in \mathcal{T}$.

The subset $\mathcal{T}_i \subset \mathcal{T}$ of simple types of type i is defined in a similar way.

The typed terms of HOL are finite sequences of primitive symbols and constructed by the improper symbols $(,), \lambda$, variables denoted by lower case letters, nonlogical constants or parameters denoted by capital letters and logical constants denoting equality, disjunction, negation, universal quantifier and definite description operator.

Definition 2.1.1.3 (Syntax of HOL)**.** The typed terms of HOL are constructed with the following formation rules:

$$s, t \ \coloneqq x_\alpha \mid P_\alpha \mid (s_{\alpha\rightarrow\beta} t_\alpha)_\beta \mid (\lambda x_\alpha . s_\beta)_{\alpha\rightarrow\beta}$$[3]

with $\alpha, \beta \in \mathcal{T}$.

[1] Here we spell out that HOL acronym will be used to refer to a classical higher-order logic and sometimes to its implementation in the face of Isabelle/HOL and not to some other systems, e.g. Gordon's "the HOL system" etc.

[2] In general, one may use the base set with more than two primitive types [Benzmüller and Miller, 2014, Farmer, 2008].

[3] $(s_{\alpha\rightarrow\beta} t_\alpha)_\beta$ is known as function application and $(\lambda x_\alpha . s_\beta)_{\alpha\rightarrow\beta}$ as function abstraction.

Moreover, we assume the existence of the following logical constant symbols in our *signature*[4]:

$$
\begin{array}{lll}
=_{\alpha\to\alpha\to o} & \in & C_{\alpha\to\alpha\to o} \\
\vee_{o\to o\to o} & \in & C_{o\to o\to o} \\
\neg_{o\to o} & \in & C_{o\to o} \\
\forall_{(\alpha\to o)\to o} & \in & C_{(\alpha\to o)\to o} \\
\iota_{(\alpha\to o)\to\alpha} & \in & C_{(\alpha\to o)\to\alpha} \\
SOME_{(\alpha\to o)\to\alpha} & \in & C_{(\alpha\to o)\to\alpha}
\end{array}
$$

The type of each term is given as a subscript. Terms of type o are formulas, nonformula terms of type $\alpha \in \mathcal{T}_o$ are predicates. Terms of type $\alpha \in \mathcal{T}_i$ except i itself serve as functions. V_α denotes the countably infinite set of variables x_α of type α and C_α denotes the countably infinite set of constant symbols of type α.

The presented logical constants are interpreted according to their intuitive meaning, for instance, $(\iota_{(\alpha\to o)\to\alpha}(\lambda x_\alpha.s_o)_{\alpha\to o})_\alpha$ is the *definite description operator* which returns the unique object of type α satisfying s_o and some arbitrary element of the same type if there is no such and its almost "twin" – *indefinite description operator* $(SOME_{(\alpha\to o)\to\alpha}(\lambda x_\alpha.s_o)_{\alpha\to o})_\alpha$ – which also returns an object of type α satisfying s_o (which might not be unique and therefore utilizing the *axiom of choice* to pick some object) if there are those and arbitrary element if not. Such $\iota_{(\alpha\to o)\to\alpha}$ operator is also useful for defining the $if-then-else$ operator:

$$ite_{o\to\alpha\to\alpha\to\alpha} \coloneqq \lambda s_o.\lambda x_\alpha.\lambda y_\alpha.\iota(\lambda z_\alpha.(s \to z = x) \wedge (\neg s \to z = y)).$$

Other logical constants can be introduced as shown below:

$$
\begin{array}{lll}
\wedge_{o\to o\to o} & \coloneqq & \lambda x_o.\lambda y_o.\neg(\neg x \vee \neg y) \\
\to_{o\to o\to o} & \coloneqq & \lambda x_o.\lambda y_o.\neg x \vee y \\
\leftrightarrow_{o\to o\to o} & \coloneqq & \lambda x_o.\lambda y_o.(x \to y) \wedge (y \to x) \\
\exists_{(\alpha\to o)\to o} & \coloneqq & \lambda p_{\alpha\to o}.\neg\forall(\lambda x_\alpha.\neg px) \\
\top_o & \coloneqq & \lambda x_\alpha.x = x
\end{array}
$$

with $\alpha \in \mathcal{T}$.

It should be mentioned that if one had started without the equality being a primitive logical symbol then one could have introduced it by Leibniz' formula encoding that two objects of the same type α are the same if they have the same properties [Benzmüller et al., 2004] :

$$=^L_{\alpha\to\alpha\to o} \coloneqq \lambda x_\alpha.\lambda y_\alpha.\forall(\lambda p_{\alpha\to o}.px \to py).$$

Definition 2.1.1.4. A *free* variable is a variable x that is out of the scope of any quantifier in a term s. A *bound* variable is a variable that is occurs in a scope of some quantifier in s.

Definition 2.1.1.5. A *closed term* is any term that has no free variables. A *sentence* is any closed formula.

[4]It is not necessary although to have all of them, i.e. having only equality as primitive logical symbol we can define all others except the *definite description operator* [Benzmüller and Andrews, 2019].

Definition 2.1.1.6. $s[x \to t]$ denotes a substitution in a term s_β of all free occurrences of variable x_α.

Substitutions are capture-avoiding meaning that there should be no binding of free variables, so the term is α-converted whenever needed. α-renaming, β, η-conversions are defined as usual:

$\lambda x_\alpha.s_\beta$	$\rightsquigarrow$	$\lambda y_\alpha.s[x \to y]$	y does not occur free in s	(α)
$(\lambda x_\alpha.s_\beta)y_\alpha$	$\rightsquigarrow$	$s[x \to y]$		(β)
$(\lambda x_\alpha.s_\beta x_\alpha)$	$\rightsquigarrow$	s_β	x does not occur free in s	(η)

For the sake of simplicity we may omit types in a subscript if it is obvious from the context and unambiguous, also any binary operator notation $((bop\ s)t)$ may be substituted by easier-to-read infix notation $(s\ bop\ t)$, the binder notation $\forall x.s$ or $\iota x.s$ may be used instead of $\forall(\lambda x.s)$ or $\iota(\lambda x.s)$.

2.1.2 Semantics

With the semantics of a language syntactically built terms obtain a meaning. Providing the HOL with intuitive set-theoretic standard semantics as is done for first-order logic lacks the judicious notion of completeness [Benzmüller et al., 2004]. However, there exists a more general notion of semantics [Henkin, 1950], namely Henkin models, which allows completeness results and will be presented below with strong reliance to [Benzmüller et al., 2004].

Definition 2.1.2.1 (A *frame* for HOL)**.** A *frame* $\mathcal{D}$ is a set $\{\mathcal{D}_\alpha : \alpha \in \mathcal{T}\}$ of nonempty sets (domains) $\mathcal{D}_\alpha$, such that $\mathcal{D}_o$ is $\{True,\ False\}$[5] (where $True$ and $False$ denotes truth and falsehood respectively), $\mathcal{D}_\iota$ is chosen freely, and $\mathcal{D}_{\alpha\to\beta}$ is the set of all total functions from $\mathcal{D}_\alpha$ to $\mathcal{D}_\beta$.

Definition 2.1.2.2 (A *standard model* for HOL)**.** A *standard model structure* is a tuple $\mathcal{M} = \langle \mathcal{D}, \mathcal{I} \rangle$, where $\mathcal{D}$ is a frame and $\mathcal{I}$ is a family of typed interpretation functions $\mathcal{I} = \{\mathcal{I}_\alpha : \alpha \in \mathcal{T}\}$, where each $\mathcal{I}_\alpha$ maps constant symbols p_α to appropriate elements in $\mathcal{D}_\alpha$ (denotations of p_α). Logical connectives are always given standard denotations which are shown below. It is also assumed that the domains $\mathcal{D}_{\alpha\to\alpha\to o}$ include the respective identity relations[6].

As was already pointed out, the logical connectives receive the following meaning in all standard interpretations:

$I(=_{\alpha\to\alpha\to o})$	$:=$	eq	$\in \mathcal{D}_{\alpha\to\alpha\to o}$	s.t.	for all $a, b \in \mathcal{D}_\alpha$, $eq(a,b) =$ T iff $a \equiv b$
$I(\vee_{o\to o\to o})$	$:=$	or	$\in \mathcal{D}_{o\to o\to o}$	s.t.	for all $a, b \in \mathcal{D}_o$,
					$or(a,b) =$ T iff $a =$ T or $b =$ T
$I(\neg_{o\to o})$	$:=$	not	$\in \mathcal{D}_{o\to o}$	s.t.	for all $a \in \mathcal{D}_o$, $not(a) =$ T iff $a =$ F
$I(\forall_{(\alpha\to o)\to o})$	$:=$	$fall$	$\in \mathcal{D}_{(\alpha\to o)\to o}$	s.t.	for all $p \in \mathcal{D}_{\alpha\to o}$,
					$fall(p) =$ T iff $p(a) =$ T for all $a \in \mathcal{D}_\alpha$
$I(\iota_{(\alpha\to o)\to\alpha})$	$:=$	$desc$	$\in \mathcal{D}_{(\alpha\to o)\to\alpha}$	s.t.	for all $p \in \mathcal{D}_{\alpha\to o}$, $desc(p) = a \in \mathcal{D}_\alpha$
					if $p(a) =$ T and for all $a' \in \mathcal{D}_\alpha$ if $p(a') =$ T

[5] Later on we might use the notation T for $True$ and F for $False$

[6] It was shown by Andrews [Andrews, 1972a] that it is important to include these relations [Benzmüller et al., 2004]

$$
\begin{array}{lcll}
 & & & \text{then } a' = a, \text{ otherwise } desc(p) = er, \\
 & & & \text{where } er \text{ is an arbitrary object of } \mathcal{D}_\alpha, \\
I(SOME_{(\alpha\to o)\to\alpha}) & := & indesc \in \mathcal{D}_{(\alpha\to o)\to\alpha} \text{ s.t.} & \text{for all } p \in \mathcal{D}_{\alpha\to o}, indesc(p) = a \in \mathcal{D}_\alpha \\
 & & & \text{if } p(a) = \text{T}, \text{ otherwise } indesc(p) = er, \\
 & & & \text{where } er \text{ is an arbitrary object of } \mathcal{D}_\alpha,
\end{array}
$$

where $\alpha \in \mathcal{T}$.

Definition 2.1.2.3. $g = \{g_\alpha : \alpha \in \mathcal{T}\}$ is a family of *variable assignment* functions, i.e. each g_α is a function $g_\alpha : V_\alpha \to \mathcal{D}_\alpha$ mapping variable symbols of type α to the corresponding elements in $\mathcal{D}$. $g[x \to a]$ denotes the variable assignment which is equivalent to g apart from x_α that is now mapped to a_α.

Definition 2.1.2.4 (*Evaluation function)*. The value $[\![s_\alpha]\!]^{\mathcal{M},g}$ of a HOL term s_α in a model $\mathcal{M}$ under the variable assignment g is a an element $d \in \mathcal{D}_\alpha$ and it is definied in the following way:

$$
\begin{array}{lcl}
[\![P_\alpha]\!]^{\mathcal{M},g} & := & I(P_\alpha) \\
[\![x_\alpha]\!]^{\mathcal{M},g} & := & g(x_\alpha) \\
[\![(s_{\alpha\to\beta}t_\alpha)_\beta]\!]^{\mathcal{M},g} & := & [\![s_{\alpha\to\beta}]\!]^{\mathcal{M},g}([\![t_\alpha]\!]^{\mathcal{M},g}) \\
[\![(\lambda x_\alpha.s_\beta)_{\alpha\to\beta}]\!]^{\mathcal{M},g} & := & \text{the function } f \text{ from } \mathcal{D}_\alpha \text{ to } \mathcal{D}_\beta \\
 & & \text{s.t. } f(d) = [\![s_\beta]\!]^{\mathcal{M},g[x\to d]} \text{ for all } d \in \mathcal{D}_\alpha
\end{array}
$$

Definition 2.1.2.5 (*Validity)*. A formula s_o if *true* in a model $\mathcal{M}$ under the assignment g if and only if $[\![s_o]\!]^{\mathcal{M},g} = \text{T}$, also denoted by $\mathcal{M}, g \models s_o$. A formula s_o is *valid in* a model $\mathcal{M}$ if and only if $\mathcal{M}, g \models s_o$ for all variable assignments g, which is also denoted by $\mathcal{M} \models s_o$. Finally, a formula s_o is *valid*, denoted by $\models s_o$, if and only if it is valid in all models $\mathcal{M}$, i.e. $\mathcal{M} \models s_o$ for all $\mathcal{M}$.

Definition 2.1.2.6 (*Standard* and *Henkin models*). A model $\mathcal{M} = \langle \mathcal{D}, \mathcal{I} \rangle$ is a *standard model* if and only if for all $\alpha, \beta \in \mathcal{T}$ we have $\mathcal{D}_{\alpha\to\beta} = \{f : \mathcal{D}_\alpha \longrightarrow \mathcal{D}_\beta\}$. In a *Henkin model* we only require that $\mathcal{D}_{\alpha\to\beta}$ is some nonempty subset of $\{f : \mathcal{D}_\alpha \longrightarrow \mathcal{D}_\beta\}$ and that evaluation function $[\![\cdot]\!]^{\mathcal{M},g}$ is total, meaning that every term denotes.

Every standard model is obviously a Henkin model as well. By the consequence of Gödel's incompleteness theorem, STT with respect to the ordinary semantics based on standard models is incomplete [Farmer, 2008]. On the other hand, Henkin's generalised semantics admits sound and complete proof calculi [Henkin, 1950, Andrews, 1972b, Benzmüller et al., 2004].

2.2 Free higher-order logic

Free logic (FL) is a logic which has less existential assumptions than the classical logic. Terms in free logic might denote some 'nonexistent objects', i.e. some terms, that refer to nothing or refer to objects outside the domain of discourse [Nolt, 2020]. Existential and universal quantifiers are assumed to range over the existent terms as in the usual case. This logic is interesting for mathematicians, in particular, because it helps to reason about *partiality*, *partial* functions. The axiomatic category theory that we start from to encode the semantics of IMLL bases precisely on free logic since the composition of morphisms inside the theory is not always defined. To describe different nature of existent and nonexistent terms we use the *dual domain* approach. To be more precise, its particular form, where there are two domains of objects, $\mathcal{D}$ which includes inside itself E. As one might

guess, the former describes the whole domain of terms, or objects, but the existent ones should lie in the domain E as well. This descriptive approach is due to Cocchiarella [Morscher and Simons, 2001]. Such choice is made due to its better fitting to our goals. In this section we present syntax and semantics of free higher-order logic (FHOL) with further intention to translate it into the simple theory of types (since the system where the formalization takes place is Isabelle/HOL). This question was already addressed in the works of Schütte [Schütte, 1960] and Farmer [Farmer, 1990] which approach we shall follow with few modifications. In our setting every function evaluating formulas and nonformula terms will be *total*, whereas in the work of Farmer chose to work with partial evaluation of nonformula terms. Nevertheless, there is still be a way to handle partiality artificially.

2.2.1 Syntax

The full set of simple types $\mathcal{T}$, the subsets $\mathcal{T}_o$ and $\mathcal{T}_i$ of simple types of type o and i respectively defined in the same way as for classical higher-order logic (see 2.1.1.12.1.1.2). But the set of typed terms for FHOL defined in the same way as for HOL 2.1.1.3 as well and the difference here lie in the presence of $E!_{\alpha\to o} \in C_{\alpha\to o}$ in the signature of language (in addition to the ones which we described for HOL) which aims to describe the existence properties of objects.

It should be mentioned that the existence predicate could have alternatively been expressed via the equality [Hintikka, 1959] as

$$E! \coloneqq \lambda x_\alpha.\exists y_\alpha.y = x,$$

where quantification is over only existent terms. Nevertheless, there either equality or this special symbol expressing the existence of terms should be primitive in the syntax, otherwise there would be no possibility to express free logic properties [Meyer et al., 1982].

2.2.2 Semantics

In order to give the semantical interpretation of terms of FHOL in a model one should decide whether it is dealing with the *positive*, *negative*, *neutral* or *supervaluational* semantics [Nolt, 2020]. As to this work, we exploit only the positive semantics and meanwhile present the general concepts and definitions concerning FHOL semantics which were developed in [Benzmüller and Scott, 2020].

The definition of *frame* $\mathcal{D} = \{\mathcal{D}_\alpha : \alpha \in \mathcal{T}\}$ here would be the same as for classical HOL 2.1.2.1. At the same time there is a need in additional domain E_α as was noted above.

Definition 2.2.2.1 (A *subframe* for FHOL)**.** A *subframe* E is a set $\{E_\alpha : \alpha \in \mathcal{T}\}$ of possibly empty sets (domains) E_α, such that $E_\alpha \subset \mathcal{D}_\alpha$ for each $\alpha \in \mathcal{T}_i$ and $E_\alpha = \mathcal{D}_\alpha$ for $\alpha \in \mathcal{T}_o$.

Continuing ideas from [Farmer, 1990] we set $\bot_\alpha \in \mathcal{D}_\alpha \backslash E_\alpha$ for $\alpha \in \mathcal{T}_i$ where

$$\bot_{\alpha\to\beta}(d) \coloneqq \bot_\beta \quad \text{for all } d \in \mathcal{D}_\alpha.$$

Besides that, each $\mathcal{D}_\alpha$ with $\alpha \in \mathcal{T}_o$ contains the element F_α defined inductively via

$$\begin{array}{lcl} \mathrm{F}_o & \coloneqq & \mathrm{F} \\ \mathrm{F}_{\alpha\to\beta}(d) & \coloneqq & \mathrm{F}_\beta \ \text{ for all } d \in \mathcal{D}_\alpha \end{array} .$$

Here the symbol $\bot_i$ symbolizes the *undefinedness* and presented constructions ensure the propagation of nondenotation and falsehood of a term up through all terms containing it.

Definition 2.2.2.2 (A *standard model* for FHOL)**.** A *standard model structure* is a triple $\mathcal{M} = \langle \mathcal{D}, E, \mathcal{I} \rangle$, where $\mathcal{D}$ is a frame, E is a subframe and $\mathcal{I}$ is a family of typed interpretation functions $\mathcal{I} = \{\mathcal{I}_\alpha : \alpha \in \mathcal{T}\}$, where each $\mathcal{I}_\alpha$ maps constant symbols p_α to appropriate elements in $\mathcal{D}_\alpha$ (denotations of p_α).

The partiality of free logic is handled while the conservation of totality of functions via the utilization of $\perp_\alpha$. This ensures the possibility for FHOL to extend in the same way presented standard models to Henkin models.

Here is the point where one should make a choice of a particular semantics for FHOL. As we have already described, there are four different theories expressing the nondenotation and their value assignments, since one may choose, e.g. to allow terms with nondenoting terms to receive a T (*True*) value (as, in particular, we did in the work), which is known under the *positive semantics*. There one might have chosen a way where all terms which include nondenoting terms could only evaluate to F (*False*), or any value. For more discussion and exploration of such issues see [Nolt, 2020,Makarenko, 2020].

Free higher-order logic with positive semantics (PFHOL) allows atomic formulas with terms that do not denote to receive a T value. This kind of semantics seems to be the most common among others [Scott, 1967,Lambert, 1963,Lambert and Van Fraassen, 1972]. With respect to this discussion, the logical and nonlogical symbols are interpreted as follows:

$I(=_{\alpha\to\alpha\to o})$	$:=$	eq	$\in E_{\alpha\to\alpha\to o}$	s.t.	for all $a, b \in \mathcal{D}_\alpha$, $eq(a,b) =$ T iff $a \equiv b$
$I(\vee_{o\to o\to o})$	$:=$	or	$\in E_{o\to o\to o}$	s.t.	$or(a,b) =$ T iff $a =$ T or $b =$ T
$I(\neg_{o\to o})$	$:=$	not	$\in E_{o\to o}$	s.t.	$not(a) =$ T iff $a =$ F
$I(\forall_{(\alpha\to o)\to o})$	$:=$	$fall$	$\in E_{(\alpha\to o)\to o}$	s.t.	for all $p \in \mathcal{D}_{\alpha\to o}$,
					$fall(p) =$ T iff $p(a) =$ T for all $a \in E_\alpha$
$I(\iota_{(\alpha\to o)\to\alpha})$	$:=$	$desc$	$\in E_{(\alpha\to o)\to\alpha}$	s.t.	for all $p \in \mathcal{D}_{\alpha\to o}$, $desc(p) = a \in E_\alpha$
					if $p(a) =$ T and for all $a' \in E_\alpha$ if
					$p(a') =$ T then $a' = a$, otherwise
					$desc(p) = \perp_\alpha$ if $\alpha \in \mathcal{T}_i$
					and $desc(p) =$F$_\alpha$ if $\alpha \in \mathcal{T}_o$
$I(SOME_{(\alpha\to o)\to\alpha})$	$:=$	$indesc$	$\in E_{(\alpha\to o)\to\alpha}$	s.t.	for all $p \in \mathcal{D}_{\alpha\to o}$, $indesc(p) = a \in E_\alpha$
					if $p(a) =$ T, otherwise $indec(p) = \perp_\alpha$
					if $\alpha \in \mathcal{T}_i$ and $indec(p) = F_\alpha$ if $\alpha \in \mathcal{T}_o$
$I(E!_{\alpha\to o})$	$:=$	$ex?$	$\in E_{\alpha\to o}$	s.t.	for all $a \in \mathcal{D}_\alpha$,
					$ex?(a) =$T iff $a \in E_\alpha$

where $\alpha \in \mathcal{T}$.

Definition 2.2.2.3 (*Evaluation function* for PFHOL)**.** The value $[\![s_\alpha]\!]^{\mathcal{M},g}$ of a PFHOL term s_α in a model $\mathcal{M}$ under the variable assignment g is a an element $d \in \mathcal{D}_\alpha$ and it is definied in the following way:

$[\![P_\alpha]\!]^{\mathcal{M},g}$	$:=$	$I(P_\alpha)$
$[\![x_\alpha]\!]^{\mathcal{M},g}$	$:=$	$g(x_\alpha)$
$[\![(s_{\alpha\to\beta}t_\alpha)_\beta]\!]^{\mathcal{M},g}$	$:=$	$[\![s_{\alpha\to\beta}]\!]^{\mathcal{M},g}([\![t_\alpha]\!]^{\mathcal{M},g})$

$$[\![(\lambda x_\alpha . s_\beta)_{\alpha\to\beta}]\!]^{\mathcal{M},g} \quad := \quad \text{the function } f \text{ from } \mathcal{D}_\alpha \text{ to } \mathcal{D}_\beta \text{ s.t. } f(d) = [\![s_\beta]\!]^{\mathcal{M},g[x\to d]} \text{ for all } d \in \mathcal{D}_\alpha$$

Here the function application is defined in a nonstrict way. To define it strictly, put

$$[\![(s_{\alpha\to_i\beta} t_\alpha)_\beta]\!]^{\mathcal{M},g} \quad := \quad [\![s_{\alpha\to_i\beta}]\!]^{\mathcal{M},g}([\![t_\alpha]\!]^{\mathcal{M},g}) \text{ if } [\![t_\alpha]\!]^{\mathcal{M},g} \in E_\alpha \text{ and } \perp_\beta \text{ otherwise,}$$

where $\alpha \to_i \beta \ \in \mathcal{T}_i$.

In the second case once we encounter the nondenoting term the result of an application is undefined. That is the place where the need for introduced special symbols $\perp_\alpha$ in $\mathcal{D}_\alpha$ is fulfilled.

Chapter 3

Linear Logic

3.1 Intoduction

Linear Logic (LL) was proposed by Girard in 1987 [Girard, 1987] as a refinement of classical logic as well as intuitionistic logic by taking the fully involutive negation of the former and the idea of constructivism of the latter and was discovered through while studying *coherence spaces*[1]. The goal is done with the help of the new connectives (will be presented below) providing with the idea that LL is not a new logic but the extension of the classical one [Girard, 1995]. The idea came from the semantical analysis of the models of System F [Di Cosmo and Miller, 2019]. LL focuses on the role of formulas as *resources* (one could see a good intuition in physics where the resources can be used once in a reaction and that reaction cannot be iterated again) instead of emphasizing the truth, like in classical logic, and emphasizing the proof, as in intuitionistic. Being the outcome of the study of System F it also bears the trace of interaction between logic and computer science and computation [Beffara, 2013, Girard et al., 1989]. LL often called "resource-sensitive" logic due to the possibility of manipulating the formulas as resources what can be expressed from the proof-theoretical point of view as the absence of the usual structural rules of construction and weakening but recovering them in an other controlled way. Once such structural rules are eliminated we are left with two different types of conjunction and disjunction, i.e. *multiplicative* and *additive* parts. At this point one might see the difference and conflict of intuitionistic and classical logic, e.g. the law of excluded middle of the former one is expressed via the additive disjunction in LL and for the classical case it is expressed with the multiplicative disjunction. There is also an interesting extension to *non-commutative linear logic*, where the system does not possess Exchange structural rule, e.g. see [Guglielmi and Straßburger, 2001]. LL has a great range of applications, for instance, in computer science [Alexiev, 1994], in linguistics (where the non-commutative extension might also play a crucial role [Casadio, 2001]) and many other areas. For some additional ideas, discussions and thoughts which lead in some way to the development of LL one can also look through the [Girard, 2011] (part III).

3.2 Propositional Linear Logic

In this chapter we will introduce propositional linear logic, i.e. without the use of existential or universal quantifiers. We will present it as a formal proof system in a sequent calculus style.[2]

[1] See, for more information, [Girard et al., 1989]

[2] Development of sequent calculus began with works of Gentzen in 1935 [Gentzen, 1935a, Gentzen, 1935b].

Definition 3.2.0.1. *Sequents* are judgements[3] of the form $\Gamma \vdash \Delta$, where Γ and Δ are called *contexts*.

In the metalanguage $\Gamma \vdash \Delta$ can be read as "the conjunction of Γ entails the disjunction of Δ".[4] Moreover, we will present LL in a *one-sided* sequent calculus, i.e. the left context of sequents is empty, as did Girard in his work. There is no problem of representing or simulating the general sequents in a one-sided way, e.g. a sequent $\Gamma \vdash \Delta$ can be transformed into $\vdash \Gamma^{\perp}, \Delta$[5] (and we will exploit precisely such approach for the reason of its better space efficiency).

3.2.1 Syntax

As we did for HOL and FHOL, we begin with the introduction of the language of LL with a reliance on [Girard, 1995].

Definition 3.2.1.1 (*Syntax* of LL)**.** The *formulas* of LL are generated by the following abstract grammar:

$$\begin{aligned} A \;\coloneqq\; & p \mid p^{\perp} \mid \\ & \mathbf{1} \mid \mathbf{0} \mid \top \mid \perp \mid \\ & A \otimes A \mid A \parr A \mid A \,\&\, A \mid A \oplus A \\ & !A \mid ?A \end{aligned}$$

where p and $p^{\perp}$ range over atomic logical symbols.

Above, symbols $\mathbf{1}$, $\mathbf{0}$, $\top$, $\perp$ are *nullary constructors*, or *units* for $\otimes$, $\oplus$, $\&$, $\parr$ respectively, symbols $!A$, $?A$ are *unary constructors*, or *exponential modalities*, and there are four *binary constructors* $A \otimes A$, $A \oplus A$, $A \,\&\, A$, $A \parr A$. *Linear negation* and *linear implication* are defined as follows:

Definition 3.2.1.2 (*Linear negation)*. *Linear negation* of every formula A, also called a *dual* of a formula A, is defined by De Morgan equations:

$$\begin{array}{lcllcl} (p)^{\perp} & \coloneqq & p^{\perp} & (p^{\perp})^{\perp} & \coloneqq & p \\ \mathbf{1}^{\perp} & \coloneqq & \perp & \perp^{\perp} & \coloneqq & \mathbf{1} \\ \top^{\perp} & \coloneqq & \mathbf{0} & \mathbf{0}^{\perp} & \coloneqq & \top \\ (A \otimes B)^{\perp} & \coloneqq & A^{\perp} \parr B^{\perp} & (A \parr B)^{\perp} & \coloneqq & A^{\perp} \otimes B^{\perp} \\ (A \,\&\, B)^{\perp} & \coloneqq & A^{\perp} \oplus B^{\perp} & (A \oplus B)^{\perp} & \coloneqq & A^{\perp} \,\&\, B^{\perp} \\ (!A)^{\perp} & \coloneqq & ?A^{\perp} & (?A)^{\perp} & \coloneqq & !A^{\perp}. \end{array}$$

$(\cdot)^{\perp}$ is an *involution*, meaning that $A^{\perp\perp} = A$. In fact, as was noted by Girard himself, linear negation is the most important linear connective and expresses the notion of *duality* [Girard, 1995].

Definition 3.2.1.3 (*Linear implication)*. *Linear implication* is a defined connective:

$$A \multimap B \;\coloneqq\; A^{\perp} \parr B$$

All the units and binary connectives are subdivided into two groups for the reasons explained and discussed below: $\otimes, \parr, \mathbf{1}, \perp$ together with $\multimap$ are called *multiplicatives*; and $\&, \oplus, \top, \mathbf{0}$ are called *additives*.

[3] Judgements are the statements in metalanguage, expressing or knowledge about the mathematical objects. For more information see [Martin-Löf, 1987]

[4] Here it is meant "multiplicative conjunction" and "multiplicative disjunction".

[5] It is worth mentioning this translation is right in the presence of De Morgan duality rules for negation. In other words, we should have the law of "excluded middle" which is not presented in intuitionistic logics.

3.2.2 Sequent Style Presentation

To define a deductive system one needs to introduce axiom schemes or proof rules (sometimes called inference rules) to the setting. For the linear logic we will need the latter ingredients. In the table 3.1 these rules are combined and summarized.

Identity and Cut	
$\dfrac{}{\vdash A, A^{\perp}}$ Identity	$\dfrac{\vdash \Gamma, A \quad \vdash A^{\perp}, \Delta}{\vdash \Gamma, \Delta}$ Cut
Structure	
$\dfrac{\vdash \Gamma, A, B, \Delta}{\vdash \Gamma, B, A, \Delta}$ Exchange	
Logic	
$\dfrac{\vdash \Gamma, A \qquad \vdash B, \Delta}{\vdash \Gamma, A \otimes B, \Delta}\ \otimes$	$\dfrac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\ \parr$
$\dfrac{\vdash \Gamma, A \qquad \vdash \Gamma, B}{\vdash \Gamma, A \,\&\, B}\ \&$	$\dfrac{\vdash \Gamma, A}{\vdash \Gamma, A \oplus B}\ \oplus_l$ $\dfrac{\vdash \Gamma, B}{\vdash \Gamma, A \oplus B}\ \oplus_r$
$\dfrac{}{\vdash \mathbf{1}}\ \mathbf{1}$	$\dfrac{\vdash \Gamma}{\vdash \Gamma, \perp}\ \perp$
$\dfrac{}{\vdash \Gamma, \top}\ \top$	(no rule for $\mathbf{0}$)
$\dfrac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}$ Promotion	$\dfrac{\vdash \Gamma}{\vdash \Gamma, ?A}$ Weakening
$\dfrac{\vdash \Gamma, ?A}{\vdash \Gamma, A}$ Dereliction	$\dfrac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}$ Contraction

Table 3.1: Inference rules of linear logic

One might note that there is no proof rule for the formula $\mathbf{0}$ as might be seen through the computational interpretation and the fact $\mathbf{0}$ is neutral for $\oplus$ [Beffara, 2013, Abramsky, 1993]. To be more precise and clear, following the "Curry-Howard" isomorphism[6], formulas A in LL are seen as *types*, where types might be thought of as a set of proofs of A. Considering the rules for connective $\oplus$ we may conclude that the proof of $A \oplus B$ is either the proof of A or a proof of B and, therefore, the set of proofs for $A \oplus B$ is roughly the disjoint union of proofs for A and B. Taking into account the neutrality of $\mathbf{0}$, the sets of proofs for $A \oplus \mathbf{0}$ and A should be equivalent and essentially the same, it follows that there should be no proof rule for $\mathbf{0}$.

[6]There will be more comments on this notion a bit later in the work.

3.2.3 Cut-elimination

The main and standard method to establish consistency results in logical proof systems is to prove these systems possess *cut-elimination* property, or *Hauptsatz* (in German)[7]. To give it simpler, one may say that this property merely yields the possibility to use "lemmas" in proofs. The property is the following[8]:

Theorem 3.2.3.1. *The sequent* $\vdash \Gamma$ *is provable in LL proof system presented above if and only if it is provable without the use of Cut rule.*

This theorem also tells us that the *Cut* rule does not effect the expressiveness of the proof system. To see more about consistency result, assume we can proof $\vdash A$ and $\vdash \neg A$, then, using the *Cut* rule we can prove the empty sequent $\vdash$, and by using the theorem we should also be able to find the proof without the *Cut*, which is impossible.

3.3 Intuitionistic Multiplicative Linear Logic

The whole linear logic would be hard to formalize at once since there one has exponentials which require complex categorical constructions, e.g. monads [Melliès, 2009], to consider. There also should be some starting easier point to begin. That is why we deal in fact only with a fragment of full LL, namely, intuitionistic multiplicative linear logic (IMLL). In this case we should restrict the syntax and modify some rules and sequents' form.

Definition 3.3.0.1 (*Syntax* of IMLL)**.** The *formulas* of IMLL are generated by the following abstract grammar:

$$A \coloneqq p \mid \mathbf{1} \mid A \otimes A \mid A \multimap A$$

where p ranges over atomic symbols.

Negation $\neg A$ of a formula A here is presented as $A \multimap \bot$, where $\bot$ is some atomic symbol p. Sequents in this case are of the form $\Gamma \vdash A$, where the Γ is a context which is allowed to have several formulas, but now there is only one formula on the right. The proof rules are given in the table 3.2.

[7] This result was originally presented by Gentzen in his paper [Gentzen, 1935a] for the systems of intuitionistic and classical logics.

[8] For more information on the property and how to prove it see [Beffara, 2013, Melliès, 2009], the steps provided there are carefully explained.

Identity and Cut

$$\frac{}{A \vdash A}\ \text{Identity} \qquad \frac{\Gamma \vdash A \quad A, \Delta \vdash B}{\Gamma, \Delta \vdash, B}\ \text{Cut}$$

Structure

$$\frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C}\ \text{Exchange}$$

Logic

$$\frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}\ \otimes_r \qquad \frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, A \otimes B, \Delta \vdash C}\ \otimes_l$$

$$\frac{A, \Gamma \vdash B}{\Gamma \vdash A \multimap B}\ \multimap_r \qquad \frac{\Gamma \vdash A \qquad B, \Delta \vdash C}{\Gamma, A \multimap B, \Delta \vdash C}\ \multimap_l$$

$$\frac{}{\vdash \mathbf{1}}\ \mathbf{1}_r \qquad \frac{\Gamma, \Delta \vdash A}{\Gamma, \mathbf{1}, \Delta \vdash A}\ \mathbf{1}_l$$

Table 3.2: Inference rules of intuitionistic multiplicative linear logic

Chapter 4

Categorical Semantics of Linear Logic

There is an interesting way of how we can look at proofs. We may look on them through the lenses of categorical language considering such proofs, or derivations, of some facts B with the assumptions A as morphisms $f : A \longrightarrow B$. This derivability relation between formulas should be reflexive and transitive. The former one may be expressed as the existence of trivial proof $1_A : A \longrightarrow A$ and the latter one as the composition law of the derivation one $f : A \longrightarrow B$ and a derivation two $g : B \longrightarrow C$:

$$\frac{f : A \longrightarrow B \quad g : B \longrightarrow C}{g \circ f : A \longrightarrow C}^{1}$$

Having all of this we only come to the notion of a *deductive system* in the sense of [Došen and Petrić, 2004]:

Definition 4.0.0.1 (*Deductive System)*. A *deductive system* is a graph[2] with two additional laws, i.e. the identity law and composition law mentioned above.

Moreover, there is a question of whether two derivations $A \longrightarrow B$ are equal or no. E.g. one would want to have derivations $f \circ 1_A$ and $1_B \circ f$ to be the same and equal to f. Thus, it imposes the first equivalence relation on the such derivations/proofs:

$$f \circ 1_A \ = \ 1_B \circ f \ = \ f. \qquad (\text{Ax1}_{cat})$$

Likewise, it is natural to ask two derivations $h \circ (g \circ f) : A \longrightarrow D$ and $(h \circ f) \circ g) : A \longrightarrow D$ to be the same as before, where $f : A \longrightarrow B$, $g : B \longrightarrow C$ and $h : C \longrightarrow D$. Then, next equivalence relation arises in a theory:

$$h \circ (g \circ f) \ = \ (h \circ f) \circ g). \qquad (\text{Ax2}_{cat})$$

With all this information we come to the notion of a *category* [Došen and Petrić, 2004]:

Definition 4.0.0.2. [*Category*[3]] A *category* is a deductive system where two mentioned above axioms Ax1_{cat} and Ax2_{cat} hold between arrows.

[1] The composition used here is functional.

[2] Here we use the notion *graph* in the sense it is a pair of functions, where one is called the source function and the second one is the target function, between the set of elements called arrows to the set of elements called objects [Došen and Petrić, 2004].

[3] To be more precise, here we defined the notion of a *small category* where arrows and objects constitute sets but not the *proper classes*.

Therefore, one sees that it is achievable to investigate logical proof systems from the perspective of the categorical theory, and a suitable notion of equivalence between logical and categorical notions may be established [Cockett and Seely, 2017]. For a particular logical system there have to be additional structures in a category as, for example, the *cartesian closed categories* model the intuitionistic logic, or simply-typed λ-calculus[4] [Lambek and Scott, 1986].

4.1 Proof Invariants of IMLL

In order to explore the intuitionistic multiplicative linear logic as a proof system, we may look for the *denotations* $[\![\pi]\!]$ and $[\![A]\!]$ of proofs π and formulas A, respectively, as was informally sketched above[5]. Here we will give a more detailed description of the proof invariants of IMLL with a reliance to [Melliès, 2009].

First of all, we are looking for the *invariants* of proofs under cut-elimination. Aforementioned statement means that the denotation of a proof $[\![\pi]\!]$ and the denotation of a proof $[\![\pi']\!]$, that we receive after applying cut-elimination procedure, are the same. To say more, later coherence diagrams that will be constructed for *monoidal, braided* and *symmetric* categories precisely reflect parts of this procedure. Moreover, we ask the invariant to be *modular*, i.e. the denotation of the proof π may be deduced directly from the denotations $[\![\pi_1]\!]$ and $[\![\pi_2]\!]$ of the proofs π_1 and π_2. This requirement signifies our need to compose different proofs. As one can see that it is approximately described by the cut-rule which is presented in the logic. The associativity and the existence of left and right identities for denotations of proofs is then obtained from the fact that denotations of proofs are invariant under cut-elimination procedure.

Definition 4.1.0.1 (*Modular invariant)*. A *modular invariant* is a mathematical function $\pi \longrightarrow [\![\pi]\!]$, which assigns some mathematical entity $[\![\pi]\!]$, called *denotation*, to a proof π in such a way that cut-elimination procedure does not affect the assignment, and that one can get the denotation of a proof π out of the other two.

The denotation of the proof

$$\frac{}{A \vdash A}\ \text{Identity}$$

is natural and represented by $1_{[\![A]\!]}$. Then we want to give an example of the proof that $1_{[\![A]\!]}$ act like identities on the denotations of proofs. Imagine, we have two proofs[6]:

$$\dfrac{\begin{array}{c}\pi\\ \vdots\\ A \vdash B\end{array} \qquad \dfrac{}{B \vdash B}\ \text{Identity}}{A \vdash B}\ \text{Cut}$$

and

[4]The first idea to connect intuitionistic proofs of propositions to programs over types is due to Brouwer, Heyting, Kolmogorov, known as *Brouwer-Heyting-Kolmogorov interpretation*, but the precise form came in *Curry–Howard isomorphism* [Curry and Feys, 1958, Howard, 1980]. Some time later, J. Lambek showed that the proofs of intuitionistic propositional logic and the combinators of typed combinatory logic share a common equational theory which is the one of cartesian closed categories [Abramsky and Tzevelekos, 2011]

[5]Above discussion was presented in order to give an intuitive feeling of how the connection between the proof theory and category theory works. Since we are dealing with two-sided sequents in case of IMLL, but not with the raw formulas and derivations, we may "associate" the derivation f from A to B in the former explanation to the proof $[\![\pi]\!]$ of the sequent $A \vdash B$.

[6]Also to be noted here, the *proof* of a sequent $A \vdash B$ is a derivation with the topmost sequent to be empty.

$$\dfrac{\dfrac{}{A \vdash A}\ \text{Identity} \qquad \begin{array}{c}\pi\\ \vdots\\ A \vdash B\end{array}}{A \vdash B}\ \text{Cut}$$

Carrying out the cut-elimination procedure, one gets the following proof:

$$\begin{array}{c}\pi\\ \vdots\\ A \vdash B\end{array}$$

Since the invariant we are seeking for insensible to cut-elimination steps, denotations of all mentioned proofs are the same, i.e. $[\![\pi]\!] = [\![\pi]\!] \circ 1_{[\![A]\!]} = 1_{[\![B]\!]} \circ [\![\pi]\!]$.

With the modular invariant then we get our denotations of proofs and formulas to be organised in a category, as we wanted earlier.

4.1.1 Tensor product

To model and explore IMLL further one needs to express in a categorical way the tensor product as well as linear implication, which will be discussed below. The usual cartesian product in a category does not fit this frame because it allows one to use the diagonal and projection maps, $\Delta_A : A \longrightarrow A \times A$ and $pr_A : A \times B \longrightarrow A$, which aren't linear. Since the tensor product in IMLL behaves similar to the one in linear algebra, we enforce the invariant of proofs to be *tensorial* as well. To be more clear, it means that there is a new operation on denotations, and the denotation of the formula $[\![A \otimes B]\!]$ coincides with the $[\![A]\!] \otimes [\![B]\!]$. Moreover, this statement also highlights that the tensor product acts on the denotations too. E.g. one has two proofs:

$$\begin{array}{c}\pi_1\\ \vdots\\ A_1 \vdash B_1\end{array} \qquad\qquad \begin{array}{c}\pi_2\\ \vdots\\ A_2 \vdash B_2\end{array}$$

Then out of them it constructs the proof π:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ A_1 \vdash B_1\end{array} \qquad \begin{array}{c}\pi_2\\ \vdots\\ A_2 \vdash B_2\end{array}}{A_1, B_1 \vdash A_2 \otimes B_2}\ \otimes_R}{A_1 \otimes B_1 \vdash A_2 \otimes B_2}\ \otimes_L$$

and the denotation of this proof π should follow from the denotations of π_1 and π_2 by applying the tensor:

$$[\![\pi]\!] \ = \ [\![\pi_1]\!] \otimes [\![\pi_2]\!].$$

Proposition 4.1.1.1. *This operation on the modular and invariant denotations is a bifunctor*[7]*. I.e. it obeys two bifunctorial equalities: 1)* $(f_3 \otimes f_4) \circ (f_1 \otimes f_2) = (f_3 \circ f_1) \otimes (f_4 \circ f_2)$ *and 2)* $1_{[\![A \otimes B]\!]} = 1_{[\![A]\!]} \circ 1_{[\![B]\!]}$.

Proof. The proof of the first equality is done via careful exploration of the cut-elimination procedure of the denotation $(f_3 \otimes f_4) \circ (f_1 \otimes f_2)$ that is transformed to the denotation $(f_3 \circ f_1) \otimes (f_4 \circ f_2)$.
The proof of the second equality is easier and can be managed in a similar way. ✠

[7] In the next chapter there will be a more careful investigation of categorical definitions and concepts. This section is an introduction to the connection between linear logic concepts and categorical constructions.

Interestingly, this operation/bifunctor $\otimes$ almost defines a monoidal category. For this purpose we still need to make some choices. First of all, we have to pick the *unit* $\mathbf{e}$ in a category of denotations but the selection is almost immediate since in IMLL the is a formula $\mathbf{1}$ and the provable sequents $A \otimes \mathbf{1} \vdash A$ and $A \vdash A \otimes \mathbf{1}$. Then here we just claim that $\mathbf{e} = [\![\mathbf{1}]\!]$.

Along these lines, we have to introduce three more morphisms to the category of denotations to finish the construction of the monoidal category, namely α, ρ and λ, which are indexed by the objects A, B, C of a category and are denotations of the specific proofs:

$$\begin{aligned} \alpha_{A,B,C} &: (A \otimes B) \otimes C \longrightarrow A \otimes (B \otimes C), \\ \rho_A &: A \otimes \mathbf{e} \longrightarrow A, \\ \lambda_A &: \mathbf{e} \otimes A \longrightarrow A. \end{aligned}$$

For the ongoing development, the morphism $\alpha_{A,B,C}$ should be natural in all three objects and the morphisms ρ_A and λ_A should be natural in A. Specified features, as well as coherence property[8], are checked using the same technigue with the invariance under cut-elimination transformations.

What is more, morphisms α, ρ and λ should also be isomorphisms, meaning there are inverses α^{-1}, ρ^{-1} and λ^{-1}, such that

$$\begin{aligned} \alpha \circ \alpha^{-1} &= 1_{A\otimes(B\otimes C)}, & \alpha^{-1} \circ \alpha &= 1_{(A\otimes B)\otimes C}, \\ \rho \circ \rho^{-1} &= 1_A, & \rho^{-1} \circ \rho &= 1_{A\otimes \mathbf{e}}, \\ \lambda \circ \lambda^{-1} &= 1_A, & \lambda^{-1} \circ \lambda &= 1_{\mathbf{e}\otimes A}. \end{aligned}$$

But only two of these equalities may be proved, $\lambda \circ \lambda^{-1}$ and $\rho \circ \rho^{-1}$. Other four equalities are not automatically satisfied in the category of denotations. For this reason one merely adds a series of standard equalities on several denotations of proofs [Melliès, 2009]. In fact, they are not very strong, natural and this little convergence is manly connected to a "bureaucracy" of sequent calculus.

The requirement which imposes the morphisms α, ρ and λ to be in fact isomorphisms reflects the case that in the IMLL proof system where, for example, formulas $(A \otimes B) \otimes C$ and $A \otimes (B \otimes C)$ are equivalent (for ρ and λ there are equivalences $A \otimes \mathbf{e} \equiv A$ and $\mathbf{e} \otimes A \equiv A$ respectively) and in some sense "isomorphic" and even in some denotational semantics are identified (see, for more information, [Beffara, 2013]).

4.2 Interpretation of IMLL in monoidal categories

The next question which comes to one trying to interpret linear logic in categories is about the actual encoding of sequents into categorical language. The goal of this particular work is to encode intuitionistic sequents of multiplicative linear logic, i.e. the sequents of the form

$$A_1, ..., A_n \vdash B,$$

where only one formula on the right and may be many on the left. There might be encoding of these sequents to the language of *multicategories*, or *colored operads*[9], but also there is a way merely to interpret them as the morphism in an ordinary monoidal category:

[8]The coherence properties of the mentioned morphisms in a monoidal category are $\alpha_{A,B,C\otimes D} \circ \alpha_{A\otimes B,C,D} = A \otimes \alpha_{B,C,D} \circ \alpha_{A,B\otimes C,D} \circ \alpha_{A,B,C} \otimes D$ and $A \otimes \lambda_B \circ \alpha_{A,\mathbf{e},B} = \rho_A \otimes B$.

[9]See, for more information, e.g. [Leinster, 2004, Cockett and Seely, 2017]

$$(...([\![A_1]\!] \otimes A_2) \otimes ... \otimes [\![A_n]\!]) \longrightarrow B$$[10]

To be more accurate, we interpret not the sequents themselves as morphisms, but the proofs of such sequents. Then we say that the context $\Gamma = A_1, ..., A_n$ is interpreted as

$$[\![\Gamma]\!] = [\![A_1]\!] \otimes ... \otimes [\![A_n]\!]$$

The denotations $[\![A]\!]$ of formulas are defined inductively on their size as

$$[\![A \otimes B]\!] = [\![A]\!] \otimes [\![B]\!]$$

In a similar way, the interpretations of proofs π are defined inductively on the depths of their derivation trees. Some basic derivation steps or rules of inference are the identity rule

$$\frac{}{A \vdash A}\ \text{Identity}$$

which is typically interpreted as the identity morphism

$$\mathbf{1}_{[\![A]\!]} : [\![A]\!] \longrightarrow [\![A]\!]$$

and the right tensor rule[11]. For the latter case there are two proofs at the beginning

$$\begin{array}{cc} \pi_1 & \pi_2 \\ \vdots & \vdots \\ \Gamma \vdash A & \Delta \vdash B \end{array}$$

which are interpreted as

$$f : [\![\Gamma]\!] \longrightarrow [\![A]\!] \qquad\qquad g : [\![\Delta]\!] \longrightarrow [\![B]\!]$$

and one applies the right tensor rule to them to get a proof π

$$\frac{\begin{array}{cc} \pi_1 & \pi_2 \\ \vdots & \vdots \\ \Gamma \vdash A & \Delta \vdash B \end{array}}{\Gamma, \Delta \vdash A \otimes B}\ \otimes_R$$

Then such proof π is interpreted as a morphism

$$f \otimes g : [\![\Gamma]\!] \otimes [\![\Delta]\!] \longrightarrow [\![A]\!] \otimes [\![B]\!].$$

From this point on, the additional axioms monoidal category should obey come from the consideration of various versions of LL under investigation. In case of this thesis, we will need *symmetrical monoidal closed category* in order to interpret IMLL. *Symmetry* is needed due to the commutativity of IMLL, i.e. due to the presence of the *exchange* rule

$$\frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C}\ Exchange$$

The *closed structure* is essential to interpret the linear implication in monoidal category and will be developed with more attention in the next chapter.

[10] Later on, if it is obvious from the context will omit the double-square brackets $[\![\cdot]\!]$ for formulas, to write the denotations of these formulas in categories. Moreover, it is worth noting that in the presence of associativity isomorphisms $\alpha_{A,B,C}$ all such objects with different permutations of brackets are almost the same in a categorical view, therefor we might omit them as well if there will be no ambiguity.

[11] The $\otimes_L$-rule will be explained further because basically the left-hand side of the sequents is already interpreted as a tensor product of the formulas in it.

Chapter 5

Encoding of the Categorical Model for IMLL into Isabelle/HOL

The formalizations of the categorical model for the IMLL is done with the help of an interactive theorem prover Isabelle/HOL, as was noted above. In other words, the base logic of the formalization is classical higher-order logic (HOL), which is described in some details in the beginning of the work. There are several steps to achieve the goal. Firstly, we used a technique, called *shallow semantical embedding*, to reason in a realm of FHOL inside Isabelle/HOL to avoid the exploration of many different proof systems aimed for the target logic. Secondly, we took the axioms which define the object-free notion of categories and develop them on the level of FHOL in Isabelle/HOL. Finally, there is a development of categorical notions inside FHOL language until the notion of *symmetric closed monoidal categories* where the proof rules of the IMLL are shown.

5.1 Shallow Semantical Embedding of FHOL into Isabelle/HOL

In order to reason automatically within free logics inside a classical higher-order logic a meta-logical approach, which is known under the name *shallow semantical embedding* (SSE), is utilized within this work. The ideas of the approach began with the works [Gabbay, 2014, Ohlbach et al., 2001] before and, more recently, was used successfully by Benzmüller, usually announced under the name *universal reasoning framework* [Benzmüller, 2019], for many nonclassical logics, such as modal logic, many-valued logic, dyadic deontic logic, many others and especially first-order free logic [Benzmüller and Scott, 2016]. The approach was extended to embed also free higher-order logic in the work [Makarenko and Benzmüller, 2020], which, in particular, we will rely on to use it for many higher-level categorical constructions. In SSE the semantics of the language of interest, e.g. FHOL, or PFHOL to be more precise, is mapped to the corresponding syntax constructs of the target language, HOL in our case. It may be viewed as a translation between the logics, where only semantical differences are targeted, e.g. existential features of free semantics. The SSE approach showed itself as a simple in use, readable and understandable way for implementing a translation of variety of nonclassical logics and therefore it is used in this work to make a basis for axiomatic category theory.

5.1.1 Interpretation of PFHOL terms in HOL

For the purpose of defining a translation of positive FHOL a nonlogical constant, unary predicate $E_{\alpha\to o}$ should be added to the language of HOL, which aims to distinguish between existent and

nonexistent objects in the domain D_α. Here we follow the embedding that is done in [Makarenko and Benzmüller, 2020]. Moreover, an error value $\mathbf{e}_\alpha$ is included to each domain D_α, which is the output of the definite/indefinite description operator in case there is no such object. This way, we should redefine slightly the interpretations of ι and $SOME$ as shown further:

$$
\begin{array}{lcll}
I(\iota_{(\alpha\to o)\to\alpha}) & := & desc \in \mathcal{D}_{(\alpha\to o)\to\alpha} & \text{s.t. for all } p \in \mathcal{D}_{\alpha\to o},\ desc(p) = a \in \mathcal{D}_\alpha \\
 & & & \text{if } p(a) = \mathrm{T} \text{ and for all } a' \in \mathcal{D}_\alpha \text{ if } p(a') = \mathrm{T} \\
 & & & \text{then } a' = a, \text{ otherwise } desc(p) = \mathbf{e}_\alpha \\
I(SOME_{(\alpha\to o)\to\alpha}) & := & indesc \in \mathcal{D}_{(\alpha\to o)\to\alpha} & \text{s.t. for all } p \in \mathcal{D}_{\alpha\to o},\ indesc(p) = a \in \mathcal{D}_\alpha \\
 & & & \text{if } p(a) = \mathrm{T}, \text{ otherwise } desc(p) = \mathbf{e}_\alpha
\end{array}
$$

where $\alpha \in \mathcal{T}$.

A HOL term $[s_\alpha]$ is assigned to the corresponding term s_α in PFHOL according to the following translation function:

$$
\begin{array}{lcl}
[P_\alpha] & = & P_\alpha \\
[x_\alpha] & = & x_\alpha \\
[(E^F_{\alpha\to o} s_\alpha)_o] & = & (E_{\alpha\to o}[s_\alpha])_o \\
[((=^F_{\alpha\to\alpha\to o}\ s_\alpha)_{\alpha\to o} t_\alpha)_o] & = & ((=^H_{\alpha\to\alpha\to o}\ [s_\alpha])_{\alpha\to o}[t_\alpha])_o \\
[(\neg^F_{o\to o} s_o)_o] & = & (\neg^H_{o\to o}[s_o])_o \\
[((\wedge^F_{o\to o\to o} s_o)_{o\to o} t_o)_o] & = & ((\wedge^H_{o\to o\to o}[s_o])_{o\to o}[t_o])_o \\
[(\forall^F_{(\alpha\to o)\to o}(\lambda x_\alpha.s_o)_{\alpha\to o})_o] & = & (\forall^H_{(\alpha\to o)\to o}(\lambda x_\alpha.((E_{\alpha\to o}x_\alpha)_o \to^H_{o\to o\to o} [s_o])_o)_{\alpha\to o})_o \\
[(\iota^F_{(\alpha\to o)\to\alpha}(\lambda x_\alpha.s_o)_{\alpha\to o})_\alpha] & = & (\iota^H_{(\alpha\to o)\to\alpha}(\lambda x_\alpha.((E_{\alpha\to o}x_\alpha)_o \wedge^H_{o\to o\to o} [s_o])_o)_{\alpha\to o})_\alpha \\
[(SOME^F_{(\alpha\to o)\to\alpha}(\lambda x_\alpha.s_o)_{\alpha\to o})_\alpha] & = & (SOME^H_{(\alpha\to o)\to\alpha}(\lambda x_\alpha.((E_{\alpha\to o}x_\alpha)_o \wedge^H_{o\to o\to o} [s_o])_o)_{\alpha\to o})_\alpha \\
[(s_{\alpha\to\beta} t_\alpha)_\beta] & = & ([s_{\alpha\to\beta}][t_\alpha])_\beta \\
[(\lambda x_\alpha.s_\beta)_{\alpha\to\beta}] & = & (\lambda x_\alpha.[s_\beta])_{\alpha\to\beta}
\end{array}
$$

where $\alpha, \beta \in \mathcal{T}$.

Here we note that the term superscripts ^{H} and ^{F} reflect whether the term correspond to HOL or to PFHOL, respectively. The provided translation is faithful, i.e. sound and complete.

5.1.2 Encoding PFHOL into Isabelle/HOL

As was mentioned above, to reach the goal of encoding the categorical constructions the Isabelle/HOL [Nipkow et al., 2002] system was chosen. Then the necessary translation is provided by the following code[1][2]:

[1] For more detailed discussion, see [Makarenko and Benzmüller, 2020]

[2] Here the bold font of the logical symbol and predicate should refer to the PFHOL and regular one to the HOL, even if they are hardly recognizable here. But even though we are aiming to build a translation of categorical notions using a free logic reasoning we still avoid where it is possible FL abbreviations in the Isabelle/HOL encoding itself in order to keep the possibility of finding more proofs via automation, e.g. in cases where we have some formulas of the type $\exists f.f : A \longrightarrow B$ for some identities A and B, but the predicate $(\_) : (\_) \longrightarrow (\_)$ is constructed in such a way that the morphism f should be existent anyway.

SSE of PFHOL into HOL

typedecl *i* — The base type for individuals
consts *fExistence :: "'a ⇒ bool" ("E")*

consts *fStar :: "i" ("⋆")*

consts *fUndef :: "'a" ("e")*
axiomatization where *fUndefIAxiom: "¬E (e::i)"*
axiomatization where *fFalsehoodBAxiom: "(e::bool) = False"*
axiomatization where *fTrueAxiom: "E True"*
axiomatization where *fFalseAxiom: "E False"*

abbreviation *fNot :: "bool⇒bool" ("¬")*
  where *"¬φ ≡ ¬φ"*
abbreviation *fImplies (***infixr** *"→" 13)*
 where *"φ → ψ ≡ φ ⟶ ψ"*
abbreviation *fIdentity (***infixr** *"=" 13)*
 where *"l = r ≡ l = r"*
abbreviation *fForall ("∀")*
 where *"∀Φ ≡ ∀x. E x ⟶ Φ x"*
abbreviation *fForallBinder (***binder** *"∀" [8] 9)*
 where *"∀x. φ x ≡ ∀φ"*
abbreviation *fThat:: "('a⇒bool)⇒'a" ("I")*
 where *"IΦ ≡ if ∃x. E(x) ∧ Φ(x) ∧ (∀y. (E(y) ∧ Φ(y)) ⟶ (y = x))*
 then THE x. E(x) ∧ Φ(x)
 else e"

abbreviation *fThatBinder:: "('a⇒bool)⇒'a" (***binder** *"I" [8] 9)*
  where *"Ix. φ(x) ≡ I(φ)"*

abbreviation *fSome :: "('a ⇒ bool) ⇒ 'a" ("SOME")* — Here we added the corresponding definition in Free Logic for indefinite description SOME operator, but as for this work, everywhere it could be replaced by the ordinary one
  where *"SOMEΦ ≡ if ∃x. E(x) ∧ Φ(x)*
 then SOME x. E(x) ∧ Φ(x)
 else e"

abbreviation *fSomeBinder:: "('a ⇒ bool) ⇒ 'a" (***binder** *"SOME" [8]9)*
  where *"SOMEx. φ(x) ≡ SOME(φ)"*

abbreviation *fOr (***infixr** *"∨" 11)*
 where *"φ ∨ ψ ≡ (¬φ) → ψ"*
abbreviation *fAnd (***infixr** *"∧" 12)*
 where *"φ ∧ ψ ≡ ¬(¬φ ∨ ¬ψ)"*
abbreviation *fImplied (***infixr** *"←" 13)*
 where *"φ ← ψ ≡ ψ → φ"*
abbreviation *fEquiv (***infixr** *"↔" 15)*
 where *"φ ↔ ψ ≡ (φ → ψ) ∧ (ψ → φ)"*
abbreviation *fExists ("∃")*
 where *"∃Φ ≡ ¬(∀(λy. ¬(Φ y)))"*
abbreviation *fExistsBinder (***binder** *"∃" [8]9)*
  where *"∃x. φ x ≡ ∃φ"*

5.2 Basic Category Theory

Once there is an encoding of PFHOL inside Isabelle/HOL one can proceed further to utilize it for the definition of theories. In case of this work, we apply it to axiomatic category theory definitions. The first effort based on these ideas was done by C. Benzmüller and D. Scott in [Benzmüller and Scott, 2020, Benzmüller and Scott, 2016] where the authors formalized the notion of object-free category using the SSE of free first-order logic (FFOL). The object-free category means that the concept is described with the use of only one kind of objects, i.e. morphisms, instead of two, i.e. objects and morphisms. This course is better aligned with the ideas of partiality and undefinedness which are central in free logics.

5.2.1 Categories

The final notion of a category will be taken from the work [Benzmüller and Scott, 2016] and slightly modified to meet the needs of further higher categorical constructions[3]. Firstly, recall the more standard definition of a category[4] [Adamek et al., 1990]:

Definition 5.2.1.1. A *category* is a quadruple $\mathcal{C} = (\mathcal{O}, \text{hom}, \mathbf{1}, \circ)$, where
1) $\mathcal{O}$ is a class[5] of objects,
2) for each pair (A, B) of objects from $\mathcal{O}$, a (possible empty) set $\text{hom}(A, B)$, whose members are morphisms from A to B,
3) for each object A there is an identity morphism $\mathbf{1}_A$,
4) a binary operation of composition that associates to each pair of morphisms $f : A \longrightarrow B$ and $g : B \longrightarrow C$ a morphism $g \circ f : A \longrightarrow C$, in such a way that the following axioms hold:
(a) the composition $\circ$ is associative,
(b) identities $\mathbf{1}_A$ act like identities with respect to composition,
(c) the sets $\text{hom}(A, B)$ are pairwise disjoint.

As one could have noted from the definition, there is a one-to-one correspondence between the objects of a category and identity morphisms. Therefore, it is possible to define a category without appealing to the objects. In particular, it will be based on a free logic. For the formalization one needs to introduce three more equalities[6], which are defined as follows:

```
abbreviation KlEq (infixr "≅" 56)
 where "x ≅ y ≡ (E x ∨ E y) → x = y"

abbreviation ExId (infixr "≃" 56)
  where "x ≃ y ≡ E x ∧ E y ∧ x = y"

abbreviation dirEq (infixr "≥" 56)
  where "x ≥ y ≡ E x → (x = y)"
```

[3]This small modification is in the use of polymorphism in functions which define categories to be able later to build categories not only from objects of type of individuals.

[4]The less common one was already given in the previous part of the work, i.e Definition 4.0.0.2

[5]The classes are needed in case of work with big categories, e.g. category of all sets, in order to prevent the origination of Russell-Zermelo paradox, see [Irvine and Deutsch, 2021]. Later in the work we deal with smaller collections since we formalize the notions in Isabelle/HOL, so categories are small here.

[6]The third one will only be needed to define a *functor* in 5.2.2.

The first one is *Kleene equality*. This equality asserts that if either side is defined then so is the other and they are equal. From the view of axiomatic category theory exploited here it may be extremely useful due to the presence of partiality in functions and nonexistent morphisms which now can be handled with the same equalities describing the behaviour of "true" existent morphisms. The second identity is called *existing identity*, and as one might note it might be only true for existing equal elements. Therefore, this one is not an equivalence relation anymore (in contrast to Kleene equality) since for nonexistent object x we now have $\neg(x \simeq x)$. The third one, *directed equality*, states that if the left-hand side is existing then the right-hand side is so and they are equal (and might be seen as a nonsymmetric version of Kleene equality in some sense).

Once there are these equalities one can start defining the theory of categories. The morphisms inside them are simulated as objects inside domains D (respectively, in D_α). The categories are given by providing two total unary operations, or functions, namely, *domain* and *codomain*, which represent the source and target of the morphisms, and by one partial binary operation, *composition*[7]. The partiality of composition is handled as desired with the help of existence predicate E. Therefore, the category can be defined in Isabelle/HOL as follows[8]:

Category

locale *category* =
fixes
  domain :: "'a ⇒ 'a" ("dom _" [108] 109) **and**
  codomain :: "'a ⇒ 'a" ("cod _" [110] 111) **and**
  composition :: "'a ⇒ 'a ⇒ 'a" (**infix** "·" 110) **and**
  nonex :: "'a" ("*") — Designates some non-existent object in a category
assumes
  S1: "E(dom x) ⟶ E x" **and**
  S2: "E(cod y) ⟶ E y" **and**
  S3: "E(x·y) ⟷ dom x ≃ cod y" **and**
  S4: "x·(y·z) ≅ (x·y)·z" **and**
  S5: "x·(dom x) ≅ x" **and**
  S6: "(cod y)·y ≅ y" **and**
  SN: "¬(E *)"

The formalization of many concepts investigated throughout this work and requiring the notion of category as a whole is done inside the *locales*, which were created to provide a module system for Isabelle/HOL [Ballarin, 2010, Ballarin, 2004]. Such a choice was done due to its advances in formalizing algebra concepts [Ballarin, 2020].

Afterwards, some well-known and obvious facts such as double application of the domain function etc. are shown by invoking the Sledgehammer for automating theorem proving. The predicate which checks out which morphisms are identities is defined in a following way [Benzmüller and Scott, 2020]:

[7]Additionally, we have added the information on nonexistent morphisms to every domain where categories might be defined for further convenience because, for example, where we would need to define a function from one domain to another and then prove this function is a functor. The information on nonexistent object will only be presented with the encodings to leave the usual feeling of categories. Moreover, we note that *domain, codomain* and *composition* functions will be denoted as dom, cod and $\cdot$.

[8]The axioms are taken from D. Scott's paper [Scott, 1979] except the last one for nonexistent object.

Identity Predicate

definition *I :: "'a ⇒ bool" ("I _")*
  where *"(I i) ≡ ((∀x. (E(i·x) → (i·x ≅ x))) ∧ (∀x. (E(x·i) → (x·i ≅ x))))"*

In other words, it checks whether the morphism acts as a left and right unit for the composition but only for the existent ones. Later we showed that this definition is equivalent to the one which expresses the invariability under *Kleene equality* of the morphism under the *domain* function and the same is true for *codomain*. Therefore, we defined the new notion of being identity for the sake of simplicity for computers to search for proofs. Moreover, we introduced the *existing identity* predicate in order to separate out the existing morphisms due to their use in further axiomatizations[9]:

Modified Identity Predicates

definition *Id :: "'a ⇒ bool" ("1(_)")*
  where *"(1$_x$) ≡ (dom x ≅ x)"*

abbreviation *IdEx :: "'a ⇒ bool"*
  where *"(IdEx a) ≡ (dom a ≃ a)"*

lemma *Id2:*
  assumes *"cod x ≅ x"*
  shows *"1$_x$"*
  by *(metis S1 assms dom_cod local.Id_def)*

lemma *Id3:*
  assumes *"1$_x$"*
  shows *"cod x ≅ x"*
  by *(metis S2 assms cod_dom local.Id_def)*

To translate many notions of the categorical language we will need several more predicates and one function to be introduced here and which will be in use throughout the work:

definition *from_hom_set ("(_):$_d$(_)→(_)")* — Predicate for testing morphism has specific domain and codomain
  where *"from_hom_set x a b ≡ ((dom x ≃ a) ∧ (cod x ≃ b))"*

abbreviation *from_hom_seta ("(_):(_)→(_)")* — The same as before, but due to Isabelle/HOL architecture, abbreviations suit better for automated theorem proving (as an observation)
  where *"x:a→b ≡ ((dom x ≃ a) ∧ (cod x ≃ b))"*

abbreviation *commSquareEx ("commSquareE")*
  where *"commSquareE g p q f ≡ g·p ≃ f·q"*

The first one here presents the predicate defining a morphism x that is existing one and goes from identity a to identity b. It merely translates the usual requirement in many definitions, lemmas and

[9] We also added the lemmas which justify that an identity might have been defined through the *cod* function as well.

so on that a morphism has specific domain and codomain. The next one describes the *isomorphisms* in a category, i.e.

Definition 5.2.1.2. A morphism $x : A \longrightarrow B$ is an *isomorphism* in a category $\mathcal{C}$ provided that there exists a morphism $y : B \longrightarrow A$, such that $\mathbf{1}_A \simeq y \cdot x$ and $\mathbf{1}_B \simeq x \cdot y$.

Isomorphisms and Inverses

```
definition Iso :: "'a ⇒ bool" ("Iso _")
  where "(Iso x) ≡ (∃y.(E(y·x) ∧ (Id (y·x)) ∧ (Id (x·y))))"

abbreviation Iso_a :: "'a ⇒ bool" ("Iso_a _")
  where "(Iso_a x) ≡ (∃y.(E(y·x) ∧ (Id (y·x)) ∧ (Id (x·y))))"

definition inv :: "'a ⇒ 'a" ("(_)^-1")
  where "x^-1 = (SOME y. E(y·x) ∧ (Id(y·x)) ∧ Id(x·y))"

abbreviation isomorphic :: "'a ⇒ 'a ⇒ bool"
  where "isomorphic x y ≡ ∃f. dom f ≃ x ∧ cod f ≃ y ∧ (Iso_a f)"

abbreviation inv_arrowsa :: "'a ⇒ 'a ⇒ bool" (infix "⇌" 56)
  where "x⇌y ≡ (E(y·x) ∧ (Id(y·x)) ∧ Id(x·y))"
```

And morphism y is called an *inverse* of x, which is presented by the last indefinite description operator. The $SOME$ operator works in such a way here that it returns an inverse if there is one and returns arbitrary element if there is no such. We could have used the introduced operator for PFHOL - $\boldsymbol{S}OME$ - but the condition of this operator prohibits the morphism to be nonexistent. Also it is well-known and might be easily checked that the inverse for an isomorphism is unique. Moreover, in addition to the mentioned fact about inverses in the category *locale* we have added as well many lemmas which establish the correctness of introduced predicates and operators and shows simple facts about the morphisms in order to use them later on for automating proving.

Later on, we will need several more specific kinds of categories, *opposite category* and *product category*.

Definition 5.2.1.3. An *opposite category* $\mathcal{C}^{op}$ for a given category $\mathcal{C}$ is a category whose domain of morphisms is the same as for $\mathcal{C}$ and three functions are defined as follows:
$dom^{op}(x) \coloneqq cod(x)$
$cod^{op}(x) \coloneqq dom(x)$
$x \cdot^{op} y \coloneqq y \cdot x$.

Definition 5.2.1.4. A *product category* $\mathcal{C} \times \mathcal{D}$ for two given categories $\mathcal{C}$ and $\mathcal{D}$ is a category whose domain of morphisms is a product of domains for $\mathcal{C}$ and $\mathcal{D}$ and the functions are defined in a following fashion[10]:
$dom_{\mathcal{C}\times\mathcal{D}}(x_1, x_2) \coloneqq (dom_{\mathcal{C}}(x_1), dom_{\mathcal{D}}(x_2))$
$cod_{\mathcal{C}\times\mathcal{D}}(x_1, x_2) \coloneqq (cod_{\mathcal{C}}(x_1), cod_{\mathcal{D}}(x_2))$
$(x_1, x_2) \cdot_{\mathcal{C}\times\mathcal{D}} (y_1, y_2) \coloneqq (x_1 \cdot_{\mathcal{C}} y_1, x_2 \cdot_{\mathcal{D}} y_2)$

The former one is encoded inside the Isabelle/HOL system with:

[10]Here and later on, subscripts with the names of categories will refer to the respective dom, cod, $\cdot$ functions but after some time we will omit them where it is obvious from the context what the underlying categories are.

Opposite Category

```
locale opposite_category =
A: category domain codomain composition *
for
  domain :: "'a ⇒ 'a" ("dom _") and
  codomain :: "'a ⇒ 'a" ("cod _") and
  composition :: "'a ⇒ 'a ⇒ 'a" (infix "·" 110) and
  nonex :: "'a" ("*")
begin

abbreviation domop :: "'a ⇒ 'a"
  where "(domop x) ≡ (codomain x)"

abbreviation codop :: "'a ⇒ 'a"
  where "(codop x) ≡ (domain x)"

abbreviation compositionop :: "'a ⇒ 'a ⇒ 'a" (infix "·op" 110)
  where "x ·op y ≡ y · x"

end
```

Formally, one should show that defined functions indeed form a category, but the proof of each axiom is about one line length, therefore we only included that in the Isabelle/HOL file and present here. As a consequence, any theorem proved for *locale* category should be available for *opposite category*, which is encoded through *sublocale* instrument outside the opposite category *locale*.

```
interpretation category domop codop compositionop *
  apply unfold_locales
  using A.S2 apply blast
  using A.S1 apply blast
  apply (metis A.S3)
  using A.S4 apply metis
  using A.S6 apply blast
  using A.S5 apply blast
  using A.SN by auto

lemma iscategory:
  shows "category domop codop compositionop *"
  using category_axioms by blast
```

Almost the same discussion applies for the latter concept, i.e., for the *product category*, but in the latter case we would need to handle the arbitrary behaviour of existence predicate to translate the idea that a morphism (x, y) in product category is existent if and only if morphisms x and y are existent. Then the proof of being a category again is quite obvious and straightforward and is given (written in Isabelle/HOL) in the Appendix A.

5.2.2 Functors

Roughly speaking, the *functor* is a morphism of categories, i.e. a function on categories which preserves their structure. The definition is taken from the book [Freyd and Scedrov, 1990]:

Definition 5.2.2.1. A *functor* **F** between two categories $\mathcal{C}$ and $\mathcal{D}$ is a function $\mathbf{F} : \mathcal{C} \longrightarrow \mathcal{D}$ which satisfies the following axioms:
$\mathbf{F}(dom_{\mathcal{C}}(x)) \cong dom_{\mathcal{D}}(\mathbf{F}(x))$,
$\mathbf{F}(cod_{\mathcal{C}}(x)) \cong cod_{\mathcal{D}}(\mathbf{F}(x))$,
$\mathbf{F}(x \cdot_{\mathcal{C}} y) \geqslant \mathbf{F}(x) \cdot_{\mathcal{D}} \mathbf{F}(y)$

The corresponding notion in Isabelle/HOL will look like:

Functor

```
locale functorCat =
A: category domain codomain composition nonex +
B: category domain' codomain' composition' nonex'
for
  domain :: "'a ⇒ 'a" ("dom _") and
  codomain :: "'a ⇒ 'a" ("cod _") and
  composition :: "'a ⇒ 'a ⇒ 'a" (infix "·" 110) and
  nonex :: "'a" ("*") and
  domain' ("dom2 _") and
  codomain' ("cod2 _") and
  composition' (infix "·2" 110) and
  nonex' :: "'b" ("*2") +
fixes
  F :: "'a ⇒ 'b"
assumes
  exists: "(E x) ⟶ E(F x)" and
  nexists:"(¬(E x)) ⟶ (¬(E(F x)))" and
  pres1: "(F (dom x)) ≅ dom2 (F x)" and
  pres2: "(F (cod x)) ≅ cod2 (F x)" and
  pres3: "(F (x·y)) ≥ (F x)·2(F y)"
```

Here one might note that there are two additional axioms describing the mappings of existing and nonexistent objects which are needed due to the totality of functions inside Isabelle/HOL system, which is based on classical HOL and, therefore, we need to separate the domains of existent and nonexistent morphisms[11]. As a little test, we show several simple facts such as preservation of identities and isomorphisms under the functor application:

```
lemma pres_identity:
  assumes "1_x"
  shows "B.Id (F x)"
  by (metis A.Id_def B.Id_def assms nexists pres1)

lemma iso_pres:
  assumes "A.Iso x"
```

[11] This part of the work where we should deal with the proper preservation of an existence was one of the main difficulties later to present the notions.

```
  shows "B.Iso (F x)"
  by (smt A.Id_def A.S3 A.S4 A.S5 A.category_axioms B.Iso_def assms category.Iso_def
category.S2 exists pres3 pres_identity)
```

In addition to the ordinary definition we will need some special kinds of functors to model the semantics of IMLL. Here they are:

Definition 5.2.2.2. The *identity functor* is a functor $id_{\mathcal{C}} : \mathcal{C} \longrightarrow \mathcal{C}$ which maps each morphism of $\mathcal{C}$ to itself.

Identity Functor

```
locale identityfunc =
A: category
begin
definition map :: "'a ⇒ 'a"
  where "map x = x"

lemma isfunctor:
  shows "functorCat domain codomain composition nonex domain codomain composition nonex
map"
   by (simp add: A.category_axioms functorCat.intro functorCat_axioms_def local.map_def)
end

sublocale identityfunc ⊆ functorCat domain codomain composition nonex domain codomain
composition nonex map
  using isfunctor by auto

sublocale category ⊆ identityfunc domain codomain composition nonex
  by (simp add: category_axioms identityfunc.intro)
```

As one might have already noted from the encoding, each category in fact may be seen as an identity functor and vice versa.

Definition 5.2.2.3. A *binary functor* **BF**, or more shortly *bifunctor*, is a functor of two variables, i.e. a functor between the product category $\mathcal{C}_1 \times \mathcal{C}_2$ and the category $\mathcal{D}$.

Moreover, for a given bifunctor $\mathbf{BF} : \mathcal{C}_1 \times \mathcal{C}_2 \longrightarrow \mathcal{D}$ we define an *exchange bifunctor* $\mathbf{BF}_{ex} : \mathcal{C}_2 \times \mathcal{C}_1 \longrightarrow \mathcal{D}$, which swaps the arguments and then the function is applied to the arguments, i.e.

$$\mathbf{BF}_{ex}(x, y) = \mathbf{BF}(y, x).$$

It is not difficult to make sure that the $\mathbf{BF}_{ex}$ function defined thereby is in fact a bifunctor. In the Isabel/HOL environment, this can be done by applying `unfold_locales` to split the lemma on axioms and then invoking the Sledgehammer to each of them, see Appendix B for more.

There are two facts [MacLane, 1965] about bifunctors that will be useful later on in formalization, because they concern the fixation of one of the arguments, and, for example, the tensor product is just such a bifunctor.

Proposition 5.2.2.4. *Given a bifunctor* **BF**: $\mathcal{C}_1 \times \mathcal{C}_2 \longrightarrow \mathcal{D}$ *and an object, existent identity morphism,* A *in a category* $\mathcal{C}_1$*, one gets a functor* $\mathbf{BF}(A, \bullet) : \mathcal{C}_2 \longrightarrow \mathcal{D}$.

Proof. There are three axioms of the functor that are to be checked. For the first one, given a morphism g in $\mathcal{C}_2$, we have:

$$dom_{\mathcal{D}}(\mathbf{BF}(A, g)) \cong \mathbf{BF}(dom_{\mathcal{C}_1 \times \mathcal{C}_2}(A, g)) = \mathbf{BF}(A, dom_{\mathcal{C}_2}(g)).$$

Here the first equality holds for the reason of **BF** being a functor on a product category, and the second one is the definition of the dom function for the product category. The same result holds obviously for the second axiom. And for the third one assume that $\mathbf{BF}(A, x \cdot y)$ exists. Therefore, $(A, x \cdot y)$ is an existent morphism, and $x \cdot y,\ \mathbf{BF}(A, x),\ \mathbf{BF}(A, y)$ are so, then one could write[12]:

$$\mathbf{BF}(A, x) \cdot_{\mathcal{D}} \mathbf{BF}(A, y) = \mathbf{BF}((A, x) \cdot_{\mathcal{C}_1 \times \mathcal{C}_2} (A, y)) = \mathbf{BF}(A, x \cdot_{\mathcal{C}_2} y).$$

In case $\mathbf{BF}(A, x \cdot y)$ does not exist, then the second part of the equation $\mathbf{BF}(A, x \cdot_{\mathcal{C}_2} y)$ neither and we now are able to state the third axiom then.

✠

As a proof of encoding validity here are the same results obtained in Isabelle/HOL:

```
lemma fix_arg1:
  assumes "A.Id a ∧ E a "
  shows "functorCat domain'' codomain'' composition'' nonex'' domain''' codomain'''
composition''' nonex''' (λx2. BF (a, x2))"
  using assms
  apply unfold_locales
  using AxB.Ex exists apply blast
  using AxB.Ex nexists apply blast
  apply (metis A.Id_def fst_conv pres1 snd_conv)
  apply (metis A.Id3 fst_conv pres2 snd_conv)
  by (smt A.Id3 A.category_axioms category.S6 fst_conv pres3 snd_conv)

lemma fix_arg2:
  assumes "B.Id a ∧ E a"
  shows "functorCat domain codomain composition nonex domain''' codomain''' composition'''
nonex''' (λx1. BF (x1, a))"
  using assms
  apply unfold_locales
  using AxB.Ex exists apply blast
  using AxB.Ex nexists apply blast
  apply (metis B.Id_def fst_conv pres1 snd_conv)
  apply (metis B.Id3 fst_conv pres2 snd_conv)
  by (metis B.Id3 B.category_axioms category.S6 fst_conv pres3 snd_conv)
```

Proposition 5.2.2.5. *Given a bifunctor* **BF**: $\mathcal{C}_1 \times \mathcal{C}_2 \longrightarrow \mathcal{D}$ *and morphisms* $x_1 : A_1 \longrightarrow B_1$ *in a category* $\mathcal{C}_1$ *and* $x_2 : A_2 \longrightarrow B_2$ *in a category* $\mathcal{C}_2$*, one gets two natural transformations* $\mathbf{BF}(x_1, \bullet) : \mathbf{BF}(A_1, \bullet) \longrightarrow \mathbf{BF}(B_1, \bullet)$ *and* $\mathbf{BF}(\bullet, x_2) : \mathbf{BF}(\bullet, A_2) \longrightarrow \mathbf{BF}(\bullet, B_2)$.

Proof. Since the notion of natural transformation is given a bit later, the proof will be presented in the Appendix B. ✠

[12]We should note again that in a proof process one should pay attention on the different properties of equalities in order to state the result in such setting. That is omitted in usual books on category theory unless one starts working with axiomatic category theory and partiality of composition.

5.2.3 Natural Transformations

There are two definitions of *natural transformations* to be presented here. The first one suits more the frame we are working in. But the idea is simple, in a similar way a functor is a morphism between categories, a natural transformation is a morphism between the functors. The next definition is taken from [nLab authors, 2021b] and a bit modified according to the equalities that were introduced here.

Definition 5.2.3.1. A *natural transformation* η between the functors **F**: $\mathcal{C} \longrightarrow \mathcal{D}$ and **G**: $\mathcal{C} \longrightarrow \mathcal{D}$ is a function $\eta : \mathcal{C} \longrightarrow \mathcal{D}$ such that:
$dom_{\mathcal{D}}(\eta(x)) \cong dom_{\mathcal{D}}(\mathbf{F}(x))$,
$cod_{\mathcal{D}}(\eta(y)) \cong cod_{\mathcal{D}}(\mathbf{F}(y))$,
$\eta(x) \cdot_{\mathcal{D}} \mathbf{F}(y) \simeq \mathbf{G}(x) \cdot_{\mathcal{D}} \eta(y)$,
whenever $x \cdot y$ is defined.

Here we also give the corresponding encoding of the notion to Isabelle/HOL with a remark, that again, as in the case with functors, we add two axioms separating existent and nonexistent morphisms:

Natural Transformation

```
locale naturalTransformation =
A: category domain codomain composition nonex +
B: category domain' codomain' composition' nonex' +
F: functorCat domain codomain composition nonex domain' codomain' composition' nonex' F
+
G: functorCat  domain codomain composition nonex domain' codomain' composition' nonex' G
for
  domain::"'a⇒'a" ("dom _") and
  codomain::"'a⇒'a" ("cod _") and
  composition::"'a⇒'a⇒'a" (infix "·" 110) and
  nonex ::"'a" ("*") and
  domain'::"'b⇒'b" ("dom2 _") and
  codomain'::"'b⇒'b" ("cod2 _") and
  composition'::"'b⇒'b⇒'b" (infix "·2" 110) and
  nonex' ::"'b" ("*2") and
  F :: "'a ⇒ 'b"  and
  G :: "'a ⇒ 'b"  +
fixes
  η :: "'a ⇒ 'b"
assumes
  ntExists: "(E x) ⟶ E(η x)" and
  nntExists: "¬(E x) ⟶ ¬(E(η x))" and
  presDom: "(dom2 (η x)) ≅ (dom2 (F x))" and
  presCod: "(cod2 (η x)) ≅  (cod2 (G x))" and
  naturality: "(E(z·w)) ⟶ ((η z)·2(F w)) ≃ ((G z)·2(η w))"
```

As a quick check, we may show that a functor **F**: $\mathcal{C} \longrightarrow \mathcal{D}$ defines a natural transformation **F**: **F** $\Rightarrow$ **F** : $\mathcal{C} \longrightarrow \mathcal{D}$ in Appendix C.

And more usual definition which is defined on identity morphisms will also be presented due to its necessity in the definition of *inverse transformations* and more.

Definition 5.2.3.2. A *natural transformation* $\eta : \mathbf{F} \Rightarrow \mathbf{G}$ between the functors $\mathbf{F}$: $\mathcal{C} \longrightarrow \mathcal{D}$ and $\mathbf{G}$: $\mathcal{C} \longrightarrow \mathcal{D}$ is an assignment to every object A a morphism $\eta(A) : \mathbf{F}(A) \longrightarrow \mathbf{G}(A)$ such that for any morphism $x : A \longrightarrow B$ in $\mathcal{C}$ the following diagram, called *naturality square*, commutes:

$$\begin{array}{ccc} \mathbf{F}(A) & \xrightarrow{\eta(A)} & \mathbf{G}(A) \\ \downarrow{\scriptstyle \mathbf{F}(x)} & & \downarrow{\scriptstyle \mathbf{G}(x)} \\ \mathbf{F}(B) & \xrightarrow{\eta(B)} & \mathbf{G}(B), \end{array}$$

or, in terms of the axioms, η satisfies:
$\mathbf{G}(x) \cdot_{\mathcal{D}} \eta(dom_{\mathcal{C}}(x)) \cong \eta(cod_{\mathcal{C}}(x)) \cdot_{\mathcal{D}} \mathbf{F}(x)$.

As a complement, the translation to the computer system is done as follows:

Natural Transformation

```
locale naturalTransformationViaIdentities =
A: category domain codomain composition nonex +
B: category domain' codomain' composition' nonex' +
F: functorCat domain codomain composition nonex domain' codomain' composition' nonex' F
+
G: functorCat domain codomain composition nonex domain' codomain' composition' nonex' G

for
  domain::"'a⇒'a" ("dom _") and
  codomain::"'a⇒'a" ("cod _") and
  composition::"'a⇒'a⇒'a" (infix "·" 110) and
  nonex ::"'a" ("*") and
  domain'::"'b⇒'b" ("dom2 _") and
  codomain'::"'b⇒'b" ("cod2 _") and
  composition'::"'b⇒'b⇒'b" (infix "·2" 110) and
  nonex' ::"'b" ("*2") and
  F :: "'a ⇒ 'b"  and
  G :: "'a ⇒ 'b"  +
fixes
  η :: "'a ⇒ 'b"
assumes
  exists: "((dom a) ≃ a) ⟶ B.from_hom_set (η a) (F a) (G a)" and
  naturality: "(((G x)·2(η(dom x))) ≅ (η(cod x))·2(F x))"
```

One might observe that it is possible to extend this latter definition according to the setting we are working in through the description of how this natural transformation acts on the full domain of morphisms and not on identities only. The main idea here is to assign to the morphism x the diagonal of the commutative square above, i.e. for $x : A \longrightarrow B$ $\eta(x) \coloneqq \mathbf{G}(x) \cdot_{\mathcal{D}} \eta(dom_{\mathcal{C}}(x))$[13][14]:

$$\begin{array}{ccc} \mathbf{F}(A) & \xrightarrow{\eta(A)} & \mathbf{G}(A) \\ \downarrow{\scriptstyle \mathbf{F}(x)} & \overset{\eta(x)}{\searrow} & \downarrow{\scriptstyle \mathbf{G}(x)} \\ \mathbf{F}(B) & \xrightarrow{\eta(B)} & \mathbf{G}(B), \end{array}$$

[13] Or, as the axiom hints, one might have started with $\eta(x) \coloneqq \eta(cod_{\mathcal{C}}(x)) \cdot_{\mathcal{D}} \mathbf{F}(x)$.
[14] $\eta(A)$ is usually called a *component* of the natural transformation, where A is an identity morphism, or object.

or in the formal terms of Isabelle[15]:

```
definition map :: "'a ⇒ 'b"
  where "map x = (if (E x)
                then (η(cod x))·2(F x)
                else *2)"
```

Lemma 5.2.3.3. *The latter definition of natural transformation with extended specification of how it acts on the morphisms defines a natural transformation in the former sense.*

Proof. Given an existent morphism $x : A \longrightarrow B$ we have:

$$\begin{aligned} dom_{\mathcal{D}}(\eta(x)) &= dom_{\mathcal{D}}(\mathbf{G}(x) \cdot_{\mathcal{D}} \eta(dom_{\mathcal{C}}(x))) = dom_{\mathcal{D}}(\eta(dom_{\mathcal{C}}(x))) \\ &= dom_{\mathcal{D}}(\eta(A)) = \mathbf{F}(A) = dom_{\mathcal{D}}(\mathbf{F}(x)). \end{aligned}$$

For nonexistent morphism we sure get the desired behaviour. The same equality is true for the cod function describing the second axiom.

For the third axiom, let $y \cdot x : A \longrightarrow B \longrightarrow C$ be defined. Then x, y and all the morphisms in the axiom are so too. Thus, the equality

$$\begin{aligned} \eta(y) \cdot_{\mathcal{D}} \mathbf{F}(x) &= \mathbf{G}(y) \cdot_{\mathcal{D}} \eta(dom_{\mathcal{C}}(y)) \cdot_{\mathcal{D}} \mathbf{F}(x) = \mathbf{G}(y) \cdot_{\mathcal{D}} \eta(cod_{\mathcal{C}}(x)) \cdot_{\mathcal{D}} \mathbf{F}(x) \\ &= \mathbf{G}(y) \cdot_{\mathcal{D}} \mathbf{G}(x) \cdot_{\mathcal{D}} \eta(dom_{\mathcal{C}}(x)) = \mathbf{G}(y) \cdot_{\mathcal{D}} \eta(x) \end{aligned}$$

holds.

✠

This fact is established with formal methods as well.

As with functors before, we will need several more kinds of natural transformations which are presented further.

Definition 5.2.3.4. A *natural isomorphism* η is a natural transformation whose components $\eta(A)$ for every identity morphism are all isomorphisms[16].

Since every component of natural isomorphism is an isomorphism, it is possible to invert every such morphism to get a new natural transformation.

Definition 5.2.3.5. An *inverse natural transformation* η^{-1} for a given natural isomorphism η is a natural transformation defined through: for every identity morphism A its component $\eta^{-1}(A)$ is $(\eta(A))^{-1}$.

Ideally, one should show that this definition indeed defines a natural transformation, but a bit more effort is needed in proving that in Isabelle/HOL. The first axiom which states, roughly, that for $\eta^{-1}(A)$[17] the domain and codomain are codomain and domain for $\eta(A)$, respectively, is easily seen on the diagram but isn't obvious for the Sledgehammer. The second one is shown by composing each side from left and right with inverses of isomorphisms.

[15] As one might note here, the definition also describes the behaviour on nonexistent morphisms as well.

[16] The code of formalization of a given definition and of the next one, i.e. of inverse natural transformations, see in Appendix C.

[17] Where A is an identity morphism.

5.3 Towards the Categorical Model of IMLL

The crucial point in modelling the categorical semantics of IMLL is the shift to the language of monoidal categories, where there is no diagonal maps which allow the duplication of the objects, or, equivalently, resources [Abramsky and Tzevelekos, 2011]. At this point we are equipped with anything that would be needed in the construction of monoidal category[18]:

Definition 5.3.0.1. A *monoidal category* $\mathcal{C}$ is a category $\mathcal{C}$ equipped with:
1) a *bifunctor*, which represents the tensor product:

$$\otimes : \mathcal{C} \times \mathcal{C} \longrightarrow \mathcal{C},$$

associative up to a natural isomorphism

$$\alpha_{A,B,C} : (A \otimes B) \otimes C \longrightarrow A \otimes (B \otimes C),$$

2) an *identity morphism* $\mathtt{e}$, which is a *unit* up to two natural isomorphisms:

$$\lambda_A : \mathbf{e} \otimes A \longrightarrow A \qquad\qquad \rho_A : A \otimes \mathbf{e} \longrightarrow A.$$

All these natural transformations must satisfy two axioms[19]:
a) the *triangular* identity

$$(A \otimes \lambda_B) \cdot \alpha_{A,\mathbf{e},B} \simeq \rho_A \otimes B,$$

b) the *pentagonal* identity

$$(A \otimes \alpha_{B,C,D}) \cdot (\alpha_{A,B\otimes C,D} \cdot \alpha_{A,B,C} \otimes D) \simeq \alpha_{A,B,C\otimes D} \cdot \alpha_{A\otimes B,C,D}$$

for all identity morphisms A, B, C, D.

Now we have to clarify several details in order to construct the encoding of above definition. Firstly, consider the natural isomorphism α. It has to be clear between which functors α and over which categories these functors act. Two functors here are $(\bullet \otimes \bullet) \otimes \bullet : \mathcal{C} \times \mathcal{C} \times \mathcal{C} \longrightarrow \mathcal{C}$ and $\bullet \otimes (\bullet \otimes \bullet) : \mathcal{C} \times \mathcal{C} \times \mathcal{C} \longrightarrow \mathcal{C}$ which domain is a product category $\mathcal{C} \times \mathcal{C} \times \mathcal{C}$ and the codomain is the category $\mathcal{C}$ itself. The similar discussion works in the case of maps λ and ρ, where the functors have the following form: $\mathbf{e} \otimes \bullet$ and $\bullet \otimes \mathbf{e}$ respectively for the domains of λ and ρ, and $\mathbf{id}_{\mathcal{C}}$, the identity functor on $\mathcal{C}$, for the codomains. All the encodings of these definitions and facts are presented in Appendix D. Moreover, for all these three natural isomorphisms we construct the respective inverse transformations due to their need in the building of *braided* and *symmetric* monoidal categories.

It is also worth mentioning that two axioms, triangle and pentagon, which describe the behaviour of structural maps ensure the existence of a *coherence property* in such monoidal categories, i.e. that 'every "formal" diagram in a monoidal category made up of associative and unity isomorphisms commutes'[20]. These coherence diagrams reflect the transformations of proofs under cut-elimination procedure steps [Melliès, 2009].

Definition 5.3.0.2. A *braided monoidal category* is a monoidal category $\mathcal{C}$ equipped with a *braiding*, i.e. a natural isomorphism

$$\gamma_{A,B} : A \otimes B \longrightarrow B \otimes A,$$

making two hexagonal axioms hold:

[18] The definitions that are needed to describe the model of IMLL are taken from [Melliès, 2009] and adopted to our frame.

[19] They take such names from diagrammatical representations where the first one looks as a triangle and the second one as a pentagon.

[20] For the proof of the statement as well as for deeper discussion, see [MacLane, 1971, nLab authors, 2021a].

$$\alpha_{B,C,A} \cdot (\gamma_{A,B\otimes C} \cdot \alpha_{A,B,C}) \simeq (B \otimes \gamma_{A,C}) \cdot (\alpha_{B,A,C} \cdot (\gamma_{A,B} \otimes C)) \text{ and}$$
$$\alpha^{-1}_{C,A,B} \cdot (\gamma_{A\otimes B,C} \cdot \alpha^{-1}_{A,B,C}) \simeq (\gamma_{A,C} \otimes B) \cdot (\alpha^{-1}_{A,C,B} \cdot (A \otimes \gamma_{B,C}))$$

for all identity morphisms A, B, C.

Definition 5.3.0.3. A *symmetric monoidal category* is a braided monoidal category $\mathcal{C}$, whose braiding is a *symmetry*, i.e. $\gamma_{A,B} \simeq \gamma^{-1}_{B,A}$ for all identity morphisms A, B.

The respective encodings are presented in the Appendix D. At this point it is already possible to interpret the *Exchange* rule of IMLL and the next step would be to determine the structure for *linear implication*.

5.3.1 Left and Right Closed Structures

There are three equivalent definitions of left, or right, closed structure, and the way to define one chooses depends on the goals and the frame one works in due to the ease of use in some cases. We will give two of them here and will give some comments on their connection. The first one was chosen since it will directly be used in further developments of IMLL semantics, and the second one will give more feeling of the notion and the understanding of why we introduced $eval$ function later to describe the translation of formulas to categorical constructs.

Definition 5.3.1.1. A *left monoidal closed category* is a monoidal category $\mathcal{C}$ endowed with a *left closed structure*, i.e. with a data of:
1) a *bifunctor* $\multimap\colon \mathcal{C}^{op} \times \mathcal{C} \longrightarrow \mathcal{C}$ and
2) a *bijection* $\mathcal{C}(A \otimes B, C) \cong_b \mathcal{C}(B, A \multimap C)$ which is natural in A, B and C[21].

In the respective formalization in Isabelle/HOL E.1 the mentioned bijection between the Hom-sets will be described via two functions Φ and Ψ acting as inverses of each other and having more inputs than just one, namely, the morphism itself. The functions have the following types:

$$\Phi ::' a \Rightarrow' a \Rightarrow' a \Rightarrow' a \qquad \text{and} \qquad \Psi ::' a \Rightarrow' a \Rightarrow' a \Rightarrow' a.$$

The reason we are using two functions describing the bijection lies in the fact that in the Isabelle/HOL system the functions defined on some types are total, but in our framework we are working with the so-called existent morphisms which constitute only part of some type. Therefore, in order to handle in a precise way the behaviour of this bijective function, which, on top of everything else, has several additional arguments, we should introduce anyway additional axioms which would control it. As a consequence, it might add more complexity to the formalization compared to the one we have chosen.

Moreover, this additional data is needed to be specified when we want to apply mentioned bijection in order to know the exact structure of the domain of input morphism for the bijection going right and the same for the codomain of input morphism for the bijection going left. In other words, given some morphism f, and a necessity to check whether the domain of f is exactly $A \otimes B$, we cannot merely apply the dom function to find the hidden parts. Therefore, this additional information recovers the missing data. The same reasoning clarifies the same features of Ψ. And the bijectivity behaviour of the Hom-sets for the needed morphisms $f : A \otimes B \longrightarrow C$ and $g : B \longrightarrow A \multimap C$ axiomatized through

[21] The sign $\cong_b$ here means the bijection between the sets and we introduced it in order to distinguish it from the Kleene duality.

$$\Psi(A, C, \Phi(A, B, f)) = f \qquad \text{and} \qquad \Phi(A, B, \Psi(A, C, g)) = g.^{22}$$

Now it has to be noted that being "natural in A, B, C" means that the bijection here is in fact a natural isomorphism ϕ

$$\mathcal{C}^{op} \times \mathcal{C}^{op} \times \mathcal{C} \xrightarrow[\mathcal{C}(\_,\_\multimap\_)]{\mathcal{C}(\_\otimes_{ex}\_,\_)} Set,$$

or, in a clearer form, for every morphisms $f : B_2 \longrightarrow B_1$, $g : A_2 \longrightarrow A_1$ and $h : C_1 \longrightarrow C_2$, the diagram

$$\begin{array}{ccc}
(B_1, A_1, C_1) & \mathcal{C}(A_1 \otimes B_1, C_1) & \xrightarrow{\phi_{B_1,A_1,C_1}} \mathcal{C}(B_1, A_1 \multimap C_1) \\
\downarrow{\scriptstyle (f^{op},g^{op},h)} & \downarrow{\scriptstyle \mathcal{C}(g\otimes f,h)} & \downarrow{\scriptstyle \mathcal{C}(f,g\multimap h)} \\
(B_2, A_2, C_2) & \mathcal{C}(A_2 \otimes B_2, C_2) & \xrightarrow{\phi_{B_2,A_2,C_2}} \mathcal{C}(B_2, A_2 \multimap C_2)
\end{array}$$

commutes. To put it even simpler, for every those morphisms f, g, h and a morphism $x \in \mathcal{C}(A_1 \otimes B_1, C_1)$ the equality

$$\phi_{B_2,A_2,C_2}(h \cdot x \cdot (g \otimes f)) \simeq (g \multimap h) \cdot \phi_{B_1,A_1,C_1}(x) \cdot f.$$

Thus, the naturality in all three arguments of the bijection might be described without invoking the notion of Hom-functors and the category Set of sets, which would have to be encoded firstly in Isabelle/HOL. Moreover, the natural isomorphism ϕ defines a corresponding inverse natural isomorphism ϕ^{-1}[23], which should be showed due to the choice of direct encoding path:

Proposition 5.3.1.2. *The function Ψ defined above in the `LeftClosedMonCatBifunctor` locale E.1 possesses the following property: for every morphisms $x : A' \longrightarrow A$, $y : B' \longrightarrow B$ and $z : C \longrightarrow C'$ and every $f : B \longrightarrow A \multimap C$ the following equality*

$$z \cdot \Psi(f) \cdot (x \otimes y) \simeq \Psi((x \multimap z) \cdot f \cdot y)$$

holds.

As was pointed out above, the other definition of *left closed structure* is the following:

Definition 5.3.1.3. A *left closed structure* in a monoidal category $\mathcal{C}$ is the data of:

1) an *identity morphism* $A \multimap B$ and

2) a *left evaluation morphism* $eval_{A,B} : A \otimes (A \multimap B) \longrightarrow B$,

for every identity morphisms A and B. The $eval_{A,B}$ morphism satisfies the following universal property:

$$\forall f : A \otimes X \longrightarrow B.(\exists! h : X \longrightarrow A \multimap B) \wedge (f \simeq eval_{A,B} \cdot (A \otimes h))^{24}.$$

This evaluation morphism is merely an interpretation of elimination rule for $\multimap$ if the proof system is designed with it instead of *Cut* rule.

[22] Later we will omit first two argument of Φ and Ψ using them not in the encoding since it is usually obvious from the context what morphisms are taken into consideration.

[23] The proof is presented in Appendix E.2.

[24] Here we in fact introduced the new abbreviation $\exists! h.P(h)$ which means $\exists h.(P(h) \wedge (\forall t.P(t) \to t = h))$.

Proposition 5.3.1.4. *The left closed structure (LCS) in 5.3.1.1 defines the left closed structure in the sense of 5.3.1.3*[25].

Proof. Assume $\Phi,\ \Psi,\ \multimap$ defines a LCS in the sense of 5.3.1.1. Then since $\multimap$ is a bifunctor (which preserves the identity morphisms) $A \multimap B$ is an identity morphism for A and B being so.

For the second piece of data, define a function $eval_{A,B} \coloneqq \Psi(A \multimap B, A \multimap B, A \multimap B)$, i.e. a morphism which is returned after applying the bijection to the left on the identity morphism from $\mathcal{C}(A \multimap B, A \multimap B)$. Obviously, we have $eval_{A,B} : A \otimes (A \multimap B) \longrightarrow B$. Now take arbitrary $f : A \otimes X \longrightarrow B$. We want to show next that $\Phi(f)$ is the unique desired h for which the universal property is satisfied. Consider the naturality diagram formed by the morphism $\Phi(f)$:

$$\begin{array}{ccc} \mathcal{C}(A \otimes X, B) & \underset{\Phi}{\overset{\Psi}{\leftrightarrows}} & \mathcal{C}(X, A \multimap B) \\ \uparrow{\scriptstyle (A \otimes \Phi(f))^*} & & \uparrow{\scriptstyle \Phi(f)^*} \\ \mathcal{C}(A \otimes (A \multimap B), B) & \underset{\Phi}{\overset{\Psi}{\leftrightarrows}} & \mathcal{C}(A \multimap B, A \multimap B) \end{array}$$

Here the star * near the morphisms mean that they define a precomposition. And now, starting from bottom right with the identity morphism $A \multimap B$, we get $eval_{A,B} \cdot (A \otimes h) \simeq \Psi(h) = \Psi(\Phi(f)) = f$. ✠

To show the converse statement, one would firstly need to show that from the universal property one can get a endofunctor $A \multimap \bullet$ for every identity A [Melliès, 2009] and then to show that it is possible to construct a binary functor from it satisfying the properties of the Def. 5.3.1.1. And for this one should present a Yoneda Lemma [MacLane, 1971], which would require a lot more things to encode in Isabelle/HOL.

As the next step, we may check whether this encoding reflects some properties of IMLL. It is well known that formulas A and $\mathbf{e} \multimap A$ are *isomorphic* in the sense given in [Beffara, 2013]. Translating it into the language of monoidal categories we come to the statement that:

Proposition 5.3.1.5. *In a monoidal category $\mathcal{C}$ with a left closed structure $\multimap$ it is true that for any identity A it is isomorphic to $\mathbf{e} \multimap A$.*

Firstly, we will give a proof using the small steps with the usual means of pen and paper here, and then we present the corresponding encoding in E.4.

Proof of proposition 5.3.1.5. We should construct a morphism $x : A \longrightarrow \mathbf{e} \multimap A$ and $y : \mathbf{e} \multimap A \longrightarrow A$, s.t. $x \cdot y \simeq \mathbf{e} \multimap A$ and $y \cdot x \simeq A$.

1) There is always a morphism λ_A for every identity A. Thus, we apply on it the bijection Φ to get a morphism

$$\Phi(\lambda_A) : A \longrightarrow \mathbf{e} \multimap A.$$

This one will play a role of x.

2) Then we get two morphisms $\Psi(\mathbf{e} \multimap A)$ and $\lambda^{-1}_{\mathbf{e} \multimap A}$ and compose them to get:

$$\Psi(\mathbf{e} \multimap A) \cdot \lambda^{-1}_{\mathbf{e} \multimap A} : \mathbf{e} \multimap A \longrightarrow \mathbf{e} \otimes (\mathbf{e} \multimap A) \longrightarrow A.$$

This morphism will be y. Mentioned morphisms always exist for every identities. Now we claim that they act as inverses of each other.

[25] For the Isabelle/HOL code, see E.3.

3) Consider the λ^{-1} naturality diagram between the functors $\mathbf{id}_{\mathcal{C}}$ and $\mathbf{e} \otimes \_$ induced by the morphism $\Phi(\lambda_A)$:

$$\begin{array}{ccc} \mathbf{e} \otimes A & \xleftarrow[\lambda_A^{-1}]{} & A \\ \downarrow{\scriptstyle \mathbf{e} \otimes \Phi(\lambda_A)} & & \downarrow{\scriptstyle \Phi(\lambda_A)} \\ \mathbf{e} \otimes (\mathbf{e} \multimap A) & \xleftarrow{\lambda^{-1}_{\mathbf{e} \multimap A}} & \mathbf{e} \multimap A \end{array}$$

From this one we can conclude that $\lambda^{-1}_{\mathbf{e} \multimap A} \cdot \Phi(\lambda_A) \simeq (\mathbf{e} \otimes \Phi(\lambda_A)) \cdot \lambda_A^{-1}$. Now merely postcompose it with $\Psi(\mathbf{e} \multimap A)$ to get:

$$\Psi(\mathbf{e} \multimap A) \cdot \lambda^{-1}_{\mathbf{e} \multimap A} \cdot \Phi(\lambda_A) \simeq \Psi(\mathbf{e} \multimap A) \cdot (\mathbf{e} \otimes \Phi(\lambda_A)) \cdot \lambda_A^{-1}.$$

If we prove that the right-hand side is A then one direction of composition inverse morphisms will be done.

4) Consider the naturality diagram for bijections defined again by the morphism $\Phi(\lambda_A)$ and take on the upper left corner the identity $\mathbf{e} \multimap A$:

$$\begin{array}{ccc} \mathcal{C}(\mathbf{e} \otimes (\mathbf{e} \multimap A), A) & \xleftarrow[\Psi]{} & \mathcal{C}(\mathbf{e} \multimap A, \mathbf{e} \multimap A) \\ \downarrow{\scriptstyle (\mathbf{e} \otimes \Phi(\lambda_A))^*} & & \downarrow{\scriptstyle \Phi(\lambda_A)^*} \\ \mathcal{C}(\mathbf{e} \otimes A, A) & \xleftarrow{\Psi} & \mathcal{C}(A, \mathbf{e} \multimap A) \end{array}$$

From this diagram we may conclude that $\Psi(\mathbf{e} \multimap A) \cdot (\mathbf{e} \otimes \Phi(\lambda_A)) \simeq \lambda_A$ and therefore

$$\Psi(\mathbf{e} \multimap A) \cdot \lambda^{-1}_{\mathbf{e} \multimap A} \cdot \Phi(\lambda_A) \simeq A.$$

5) Now, if we take the naturality diagram formed by $\Psi(\mathbf{e} \multimap A) \cdot \lambda^{-1}_{\mathbf{e} \multimap A}$:

$$\begin{array}{ccc} \mathbf{e} \otimes (\mathbf{e} \multimap A) & \xrightarrow[\lambda_{\mathbf{e} \multimap A}]{} & \mathbf{e} \multimap A \\ \downarrow{\scriptstyle \mathbf{e} \otimes (\Psi(\mathbf{e} \multimap A) \cdot \lambda^{-1}_{\mathbf{e} \multimap A})} & & \downarrow{\scriptstyle \Psi(\mathbf{e} \multimap A) \cdot \lambda^{-1}_{\mathbf{e} \multimap A}} \\ \mathbf{e} \otimes A & \xrightarrow{\lambda_A} & A \end{array}$$

we will get that $\lambda_A \cdot (\mathbf{e} \otimes (\Psi(\mathbf{e} \multimap A) \cdot \lambda^{-1}_{\mathbf{e} \multimap A})) \simeq \Psi(\mathbf{e} \multimap A)$.

6) And if we consider the other naturality diagram formed by the same morphism as in the previous step and start with a morphism $\Phi(\lambda_A)$ in the bottom-left:

$$\begin{array}{ccc} \mathcal{C}(\mathbf{e} \multimap A, \mathbf{e} \multimap A) & \xrightarrow[\Psi]{} & \mathcal{C}(\mathbf{e} \otimes (\mathbf{e} \multimap A), A) \\ \uparrow{\scriptstyle (\Psi(\mathbf{e} \multimap A) \cdot \lambda^{-1}_{\mathbf{e} \multimap A})^*} & & \uparrow{\scriptstyle (\mathbf{e} \otimes (\Psi(\mathbf{e} \multimap A) \cdot \lambda^{-1}_{\mathbf{e} \multimap A}))^*} \\ \mathcal{C}(A, \mathbf{e} \multimap A) & \xrightarrow{\Psi} & \mathcal{C}(\mathbf{e} \otimes A, A) \end{array}$$

we will get the equality $\lambda_A \cdot (\mathbf{e} \otimes (\Psi(\mathbf{e} \multimap A) \cdot \lambda^{-1}_{\mathbf{e} \multimap A})) \simeq \Psi(\Phi(\lambda_A) \cdot \Psi(\mathbf{e} \multimap A) \cdot \lambda^{-1}_{\mathbf{e} \multimap A})$. As was shown on the step 5) we can rewrite this equation as

$$\Psi(\mathbf{e} \multimap A) \simeq \Psi(\Phi(\lambda_A) \cdot \Psi(\mathbf{e} \multimap A) \cdot \lambda^{-1}_{\mathbf{e} \multimap A}).$$

And taking into account that Ψ is a bijective function we may conclude that $\mathbf{e} \multimap A \simeq \Phi(\lambda_A) \cdot \Psi(\mathbf{e} \multimap A) \cdot \lambda^{-1}_{\mathbf{e} \multimap A}$.

✠

In the same way one defines a *left closed structure* it is possible to define the *right closed struture*[26] [Melliès, 2009] E.5.

Definition 5.3.1.6. A *right monoidal closed category* is a monoidal category $\mathcal{C}$ endowed with a *right closed structure*, i.e. with a data of:
1) a *bifunctor* $\multimap_r$: $\mathcal{C}^{op} \times \mathcal{C} \longrightarrow \mathcal{C}$ and
2) a *bijection* $\mathcal{C}(B \otimes A, C) \cong_b \mathcal{C}(B, A \multimap_r C)$ which is natural in A, B and C.

Definition 5.3.1.7. A *monoidal biclosed category* is a monoidal category $\mathcal{C}$ endowed with the left and right closed structures.

With these ingredients we are now ready to set up and describe the categorical model of IMLL.

5.3.2 Symmetric Monoidal Closed Categories

Definition 5.3.2.1. A *symmetric monoidal closed category* (SMCC) is a monoidal category $\mathcal{C}$ equipped with a *symmetry* γ and a *left closed structure* $\multimap$[27].

It is needed to show that in any such monoidal category there is also a *right closed structure* as well. Define the functions $\bullet \multimap_r \bullet := \bullet \multimap \bullet$, $\Phi_r(f) := \Phi(f \cdot \gamma_{A,B})$ and $\Psi_r(g) := \Psi(g) \cdot \gamma_{B,A}$ for morphisms $f : B \otimes A \longrightarrow C$ and $g : B \longrightarrow A \multimap C$[28], and they will provide the needed properties:

Proposition 5.3.2.2. *In any symmetric monoidal category previously defined functions* $(\multimap_r, \Phi_r, \Psi_r)$ *give rise to the right closed structure (RCS) F.1.*

Proof. It is easy to see that all the properties except for naturality of Φ_r for the RCS are satisfied. The direct check applies here.

For the naturality property we need to prove that for every morphisms $f : A_2 \longrightarrow A_1$, $g : B_2 \longrightarrow B_1$ and $h : C_1 \longrightarrow C_2$ and given a morphism $x : B_1 \otimes A_1 \longrightarrow C_1$ the equality

$$(f \multimap_r h) \cdot \Phi_r(x) \cdot g \simeq \Phi_r(h \cdot x \cdot (f \otimes g))$$

holds. This equality under the given definitions translates into

$$(f \multimap h) \cdot \Phi(x \cdot \gamma_{A_1,B_1}) \cdot g \simeq \Phi(h \cdot x \cdot (f \otimes g) \cdot \gamma_{A_2,B_2})$$

Here we see, that using the equality for naturality of γ, i.e. $(f \otimes g) \cdot \gamma_{A_2,B_2} \simeq \gamma_{A_1,B_1} \cdot (g \otimes f)$, this is equivalent to the natural property of Φ in the LCS applied to the morphism $x \cdot \gamma_{A_1,B_1}$. ✠

[26] The definition of the right closed structure is translated in a similar way in the encoding as was proposed for the left closed structure.

[27] For the encoding, see Appendix F.

[28] That is exactly the place where we use in the encoding those additional first two parameters of Φ and Ψ to specify the morphisms.

5.3.3 Interpretation of IMLL in SMCC

At this point we are ready to give the interpretation of *intuitionistic multiplicative linear logic* formulas in *symmetric monoidal closed categories* and present the respective formalization encoded into Isablle/HOL system.

First of all, we start with the discussion of the well-known principle in intuitionistic logic, which in particular holds in IMLL, that every formula A implies its double negation $\neg\neg A$. For this purpose there should be some formula $\bot$[29], which helps to define the negation of a formula as $A \multimap \bot$ [Melliès, 2009]. Thus, the mentioned principle translates to the fact in SMCC as:

Proposition 5.3.3.1. *There is always a morphism $\delta_A : A \longrightarrow (A \multimap \bot) \multimap \bot$ for every identity A in any SMCC.*

Proof. There is always an identity morphism $A \multimap \bot$ in SMCC. Thus, we always can get a morphism $\delta_A \coloneqq \Phi_r(\Psi(A \multimap \bot))$ whose domain is A and whose codomain is $(A \multimap \bot) \multimap \bot$. ✠

It is worth to mention and check that there is no difference how we came to the formula $(A_2 \multimap \bot) \multimap \bot$ given a derivation π of $A_1 \vdash A_2$. In other words, we could have come to this formula via double negating A_1 and then applying the derivation π translated for double negation, or we double negate A_2. This fact corresponds in the SMCC to the:

Proposition 5.3.3.2. *In every SMCC the constructed morphism δ_A is in fact a natural transformation F.2.*

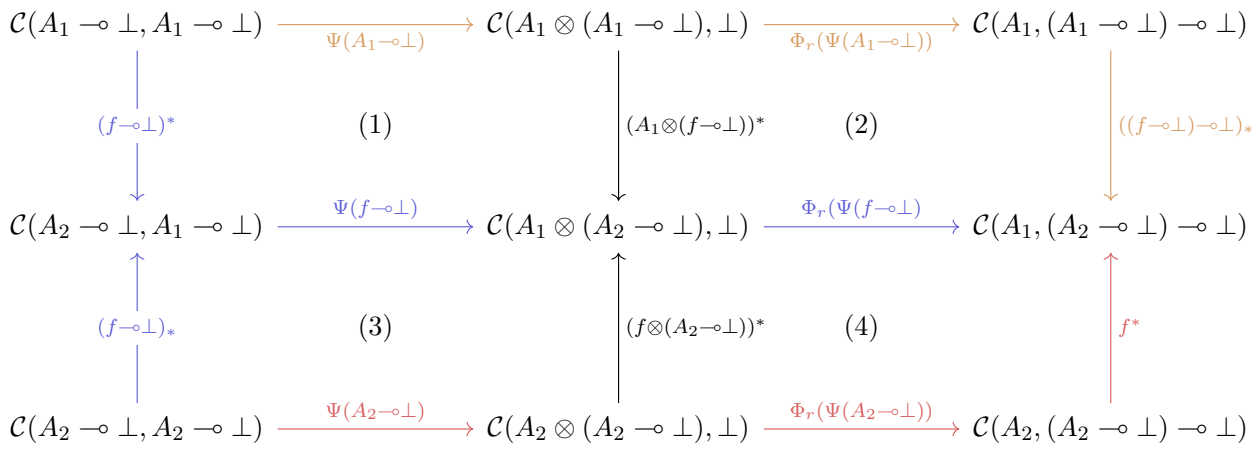
Proof. It is easy to see that δ should be the natural transformation between the functors $\mathbf{id}_{\mathcal{C}} \Rightarrow (\bullet \multimap \bot) \multimap \bot : \mathcal{C} \longrightarrow \mathcal{C}$ and the first property is satisfied by the definition of δ.

Now it is left to show that the naturality square diagram

$$\begin{array}{ccc} A_1 & \xrightarrow[\delta_{A_1}]{} & (A_1 \multimap \bot) \multimap \bot \\ \downarrow{\scriptstyle f} & & \downarrow{\scriptstyle (f \multimap \bot) \multimap \bot} \\ A_2 & \xrightarrow{\delta_{A_2}} & (A_2 \multimap \bot) \multimap \bot \end{array}$$

holds, i.e. $((f \multimap \bot) \multimap \bot) \cdot \Phi_r(\Psi(A_1 \multimap \bot)) \simeq \Phi_r(\Psi(A_2 \multimap \bot)) \cdot f$, where $f : A_1 \longrightarrow A_2$. To do so, we consider several naturality diagrams for functions Φ_r and Ψ glued together to see the whole picture.

[29] It worth pointing out that as in IMLL $\bot$ designates merely some atomic formula, $\bot$ in SMCC also designates just some fixed object, or identity morphism.



On this diagram square (1) represents the naturality of Ψ induced by the morphism $f \multimap \bot$ and applied for the identity morphism $A_1 \multimap \bot$. (2) - the naturality for the function Φ_r induced by the same $f \multimap \bot$. It might be almost immediately shown that the resulting rectangle is also a commutative square. Thus, starting with the identity and going via the orange arrows is the same as going through the purple ones, i.e. $((f \multimap \bot) \multimap \bot) \cdot \Phi_r(\Psi(A_1 \multimap \bot)) \simeq \Phi_r(\Psi(f \multimap \bot))$. The same reasoning applies for the diagrams (3) and (4) again but starting with the identity $A_2 \multimap \bot$ leaving us with $\Phi_r(\Psi(f \multimap \bot)) \simeq \Phi_r(\Psi(A_2 \multimap \bot)) \cdot f$. As we see here, starting with different identities and using respective naturality diagrams the ways inside them thorugh the purple arrows coincide providing us with the result we were seeking for

$$((f \multimap \bot) \multimap \bot) \cdot \Phi_r(\Psi(A_1 \multimap \bot)) \simeq \Phi_r(\Psi(A_2 \multimap \bot)) \cdot f$$

✠

Now we are going to provide the interpretation function $[\![\cdot]\!]^{SMCC}$ for the formulas in IMLL and for the canonical proof $\pi_{\bot\bot}$ of the sequent $A \vdash (A \multimap \bot) \multimap \bot$:

$$\begin{aligned}
[\![\mathbf{1}]\!]^{SMCC} &= \mathbf{e} \\
[\![\bot]\!]^{SMCC} &= \bot \\
[\![A \otimes B]\!]^{SMCC} &= [\![A]\!]^{SMCC} \otimes [\![B]\!]^{SMCC} \\
[\![A \multimap B]\!]^{SMCC} &= [\![A]\!]^{SMCC} \multimap [\![B]\!]^{SMCC}. \\
[\![\pi_{\bot\bot}]\!]^{SMCC} &= \delta_A.
\end{aligned}$$

The interpretation of proofs of IMLL in SMCC was discussed in the beginning and here we recall that they are defined inductively on the depth of their derivation tree. The most basic building blocks are proof rules which we now translate to the categorical language[30] with the help of [Abramsky and Tzevelekos, 2011] and summarize them in the Table 5.1. These translated rules were written down in Isabelle/HOL system as well F.3 proving that we can reason about proof-theoretical constructions (invariant under cut-elimination procedure) of IMLL inside higher-order logic (simple type theory).

[30]The left linear implication rule required a bit of creativity since it does not naturally arise from given functions. Moreover, we slightly modified the right rule for **1** because otherwise direct categorical translation would seem useless. This rule merely transcribes into the fact of the existence of λ_A and ρ_A functions for every identity A because $\mathbf{1}_r$ rule allows us to plug **1** into the right side of the sequent (via applying the $\otimes_r$ rule on $\vdash \mathbf{1}$ and $\Gamma \vdash \Delta$).

Identity and Cut

$$\frac{}{A : A \longrightarrow A}\ \text{Identity} \qquad \frac{f : \Gamma \longrightarrow A \qquad g : (A \otimes \Delta) \longrightarrow B}{g \cdot (f \otimes \Delta) : (\Gamma \otimes \Delta) \longrightarrow B}\ \text{Cut}$$

Structure

$$\frac{f : (\Gamma \otimes (A \otimes B)) \otimes \Delta \longrightarrow C}{f \cdot ((\Gamma \otimes \gamma_{B,A}) \otimes \Delta) : (\Gamma \otimes (B \otimes A)) \otimes \Delta \longrightarrow C}\ \text{Exchange}$$

Logic

$$\frac{f : \Gamma \longrightarrow A \qquad g : \Delta \longrightarrow B}{f \otimes g : \Gamma \otimes \Delta \longrightarrow A \otimes B}\ \otimes_r \qquad \frac{f : ((\Gamma \otimes A) \otimes B) \otimes \Delta \longrightarrow C}{f \cdot (\alpha^{-1}_{\Gamma,A,B} \otimes \Delta) : (\Gamma \otimes (A \otimes B)) \otimes \Delta \longrightarrow C}\ \otimes_l$$

$$\frac{f : A \otimes \Gamma \longrightarrow B}{\Phi(f) : \Gamma \longrightarrow A \multimap B}\ \multimap_r$$

$$\frac{f : \Gamma \longrightarrow A \qquad g : B \otimes \Delta \longrightarrow C}{(g \cdot (eval_{A,B} \otimes \Delta)) \cdot ((f \otimes (A \multimap B)) \otimes \Delta) : (\Gamma \otimes (A \multimap B)) \otimes \Delta \longrightarrow C}\ \multimap_l$$

$$\frac{}{\lambda^{-1}_A : A \longrightarrow \mathbf{1} \otimes A}\ \mathbf{1}_\lambda \qquad \frac{f : \Gamma \otimes \Delta \longrightarrow A}{f \cdot (\Gamma \otimes \lambda_\Delta) : \Gamma \otimes (\mathbf{1} \otimes \Delta) \longrightarrow A}\ \mathbf{1}_l$$

$$\frac{}{\rho^{-1}_A : A \longrightarrow A \otimes \mathbf{1}}\ \mathbf{1}_\rho$$

Table 5.1: IMLL proof rules in SMCC

Chapter 6

Conclusion and Discussions

As a result of a long path of intuitionistic multiplicative linear logic semantics investigation and formalization we finally were able to provide the description of all the proof rules seen as objects in a denotational categorical model in the Isabelle/HOL computer proof system. This translation shows, in particular, that we constructed a sound semantical model with the help of formal tools. The way we have chosen to do so might seem strange and winding but all the choices we've made were based on an idea of finding an axiomatic approach. The interest began with the exploration of the linear logic and how it might be connected to existing formal computer systems to apply the developing methods of the latter to analyze the features of the former. It is also known that LL may naturally express many features that classical logic do only through complex constructions [Girard, 1995]. One way to investigate logics is to express their formulas and proofs inside categories. Therefore, as a fundament of the work, as the main block we chose a perspective of an axiomatic category theory based on free logic (due to the utilization of partial functions) the beginnings of formalization of which in Isabelle/HOL began in the works [Benzmüller and Scott, 2020, Benzmüller and Scott, 2016] where free logic firstly shallow semantically embedded. Along the way the formalization and the encoding of many categorical notions was carried out testing the possibilities and capabilities of simple type theory on behalf of Isabelle system. Thus even such a system based on STT might share the tools for building and reasoning about complex theories. This being said, there were a lot of difficulties with handling partial behaviour of functions of categorical constructions which required a bit of creativity in translation and formalization of definitions and axioms. Moreover, all new constructions were tested on properties that they are known to possess. Approaching the end of intended formalization we not only checked the proof rules of IMLL, which, of course, lay the foundations of formal logical reasoning, but on top of that some special logical features, such as the isomorphism between several formulas and properties of double negation. It is worth noting as well that the whole formalization might not be the most efficient and it seems that there is a way to overcome this. What else is intriguing about this approach is that there is a possibility of further gluing of IMLL categorical proofs with other logics, i.e. the presented constructions which reflect in a way the nature of IMLL formulas and their connections through proofs exist as higher abstraction layers on top of HOL in the Isabelle system. Therefore, if there are some other logics embedded in the same system either through shallow semantical embedding or with the categorical semantics (as done in the work) we may unfold all these constructions to stay with the pure terms in HOL which now have the same essence, or we may work on the level of categories and functors to navigate between them.

However, IMLL is the simplest part of the whole family of linear logic fragments that might be studied, e.g. adding more connective, adding the axiom of excluded middle for $\otimes$ which boils down to the statement that previously defined δ_A for an identity morphism A is in fact an isomorphism,

adding the 'infinitary' properties of logic in the face of exponential modalities. And as these mentioned logical constructions might be captured in the categorical language as well (showing finally that exponential modalities in fact might be decomposed to even smaller parts [Melliès, 2009]), even though more complex than those presented here, there is a way to develop this study further to handle the full linear logic.

Appendices

Appendix A

Product Category

Product Category

locale *productCategory =*
A: category domain codomain composition nonex +
B: category domain' codomain' composition' nonex'
for
domain :: "'a ⇒ 'a" ("dom _") **and**
codomain :: "'a ⇒ 'a" ("cod _") **and**
*composition :: "'a ⇒ 'a ⇒ 'a" (***infix** *"·" 110)* **and**
nonex :: "'a" ("")* **and**
domain' :: "'b ⇒ 'b" ("dom2 _") **and**
codomain' :: "'b ⇒ 'b" ("cod2 _") **and**
*composition' :: "'b ⇒ 'b ⇒ 'b" (***infix** *"·2" 110)* **and**
*nonex' :: "'b" ("*2") +*
assumes
Ex:"(E (x::'a) ∧ E (y::'b)) ⟷ E(x,y)" **and**
Ex2:"(E (x::'a) ∧ E (y::'b)) ⟷ E(y,x)"

abbreviation *(input) Dom :: "('a * 'b) ⇒ ('a * 'b)"*
where *"Dom x ≡ ((dom (fst x)), dom2 (snd x))"*

abbreviation *(input) Cod :: "('a * 'b) ⇒ ('a * 'b)"*
where *"Cod x ≡ ((cod (fst x)), cod2 (snd x))"*

abbreviation *(input) Composition :: "('a * 'b) ⇒ ('a * 'b) ⇒ ('a * 'b)" (***infix** *"·" 110)*
where *"Composition x y ≡ ((fst x)·(fst y),(snd x)·2(snd y))"*

abbreviation *(input) Nonex :: "'a*'b"*
where *"Nonex ≡ (*,*2)"*

theorem *product_is_category:*
shows *"category Dom Cod Composition (*,*2)"*
...

Appendix B

Binary Functor

B.1 Definition

Binary Functor

locale *biFunctor =*
A: category domain codomain composition nonex +
B: category domain' codomain' composition' nonex' +
AxB: productCategory domain codomain composition nonex domain' codomain' composition' nonex' +
C: category domain'' codomain'' composition'' nonex'' +
functorCat AxB.Dom AxB.Cod AxB.Composition AxB.Nonex domain'' codomain'' composition'' nonex'' BF
for
domain::"'a⇒'a" ("dom _") **and**
codomain::"'a⇒'a" ("cod _") **and**
*composition::"'a⇒'a⇒'a" (***infix** *"·" 110)* **and**
nonex ::"'a" ("")* **and**
domain'::"'b⇒'b" ("dom2 _") **and**
codomain'::"'b⇒'b" ("cod2 _") **and**
*composition'::"'b⇒'b⇒'b" (***infix** *"·2" 110)* **and**
nonex' ::"'b" ("2") **and**
domain''::"'c⇒'c" ("dom3 _") **and**
codomain''::"'c⇒'c" ("cod3 _") **and**
*composition''::"'c⇒'c⇒'c" (***infix** *"·3" 110)* **and**
*nonex'' ::"'c" ("*3")* **and**
*BF :: "'a * 'b ⇒ 'c"*

B.2 Exchange binary functor

abbreviation *Exch*
where *"Exch ≡ (λx. BF(snd x, fst x))"*
lemma *Exch_is_binary:*
shows *"biFunctor domain' codomain' composition' nonex' domain codomain composition nonex domain'' codomain'' composition'' nonex'' Exch"*

```
  apply unfold_locales
...
```

B.3 Binary functor defines a natural transformation

Proof of proposition 5.2.2.5. We will prove the axioms of natural transformations which are defined through their components. For the first one, assume we are given an identity morphism C_2 in $\mathcal{C}_2$, then obviously $\mathbf{BF}(x_1, C_2)$ is an existent morphism and $dom_{\mathcal{D}}(\mathbf{BF}(x, C_2)) = \mathbf{BF}(dom_{\mathcal{C}_1}, C_2) = \mathbf{BF}(A_1, C_2)$ and $cod_{\mathcal{D}}(\mathbf{BF}(x_1, C_2)) = \mathbf{BF}(cod_{\mathcal{C}_1}, C_2) = \mathbf{BF}(B_1, C_2)$. The same might be shown for the other entry and C_1 being an identity in $\mathcal{C}_1$.

Now assume we are given a morphism $y : C \longrightarrow C'$ in $\mathcal{C}_2$. We should prove that the following naturality diagram

$$\begin{array}{ccc} \mathbf{BF}(A, C) & \xrightarrow[\mathbf{BF}(x,C)]{} & \mathbf{BF}(B, C) \\ \downarrow{\scriptstyle \mathbf{BF}(A,y)} & & \downarrow{\scriptstyle \mathbf{BF}(B,y)} \\ \mathbf{BF}(A, C') & \xrightarrow{\mathbf{BF}(x,C')} & \mathbf{BF}(B, C') \end{array}$$

commutes. For this part consider the chain of transformations:

$$\mathbf{BF}(x, C') \cdot \mathbf{BF}(A, y) \overset{(1)}{=} \mathbf{BF}(x \cdot A, C' \cdot y) \overset{(2)}{=} \mathbf{BF}(B \cdot x, y \cdot C) \overset{(3)}{=} \mathbf{BF}(B, y) \cdot \mathbf{BF}(x, C)$$

where (1) and (3) are the use of the third axiom of a functor **BF** and (2) is merely the fifth and sixth axioms of a category. That was the case when one of the either sides is existent. When one of it is nonexistent, then the other is so and we can then write

$$\mathbf{BF}(x, C') \cdot \mathbf{BF}(A, y) \cong \mathbf{BF}(B, y) \cdot \mathbf{BF}(x, C).$$

✠

The code for the lemmas expresses this fact is the following[1]:

```
lemma fix_arr1n:
  assumes "E x1"
  shows "naturalTransformation domain' codomain' composition' nonex' domain'' codomain''
composition'' nonex'' (λx2. BF (domain x1, x2)) (λx2. BF (codomain x1, x2))
(λx2. BF (x1,x2))"
  using assms
  apply unfold_locales
...
lemma fix_arr2n:
  assumes "E x2"
  shows "naturalTransformation domain codomain composition nonex domain'' codomain''
composition'' nonex'' (λx1. BF (x1, dom2 x2)) (λx1. BF (x1, cod2 x2)) (λx1. BF (x1,x2))"
  using assms
  apply unfold_locales
...
```

[1]Here we once more heavily used the automated tools after unfolding again the `naturalTransformationViaIdentities` *locale* axioms.

Appendix C

Natural Transformations

C.1 Functor defines the natural transformation

```
lemma functors_is_nat_transformation:
  assumes "functorCat domain codomain composition nonex domain' codomain' composition'
nonex' F"
  shows "naturalTransformation domain codomain composition nonex domain' codomain'
composition' nonex' F F F"
  using assms
  apply unfold_locales
...

sublocale functorCat ⊆ naturalTransformation domain codomain composition nonex domain'
codomain' composition' nonex' F F F
  by (simp add: functorCat_axioms functors_is_nat_transformation)
```

Here, as in many cases where we hided the code after invoking `unfold_locales`, one may use Sledgehammer tools easily to prove each statement, since they aren't hard for it.

C.2 Natural isomorphism

Natural Isomorphism

```
locale naturalIsomorphism =
A: category domain codomain composition nonex +
B: category domain' codomain' composition' nonex' +
F: functorCat domain codomain composition nonex domain' codomain' composition' nonex' F
+
G: functorCat  domain codomain composition nonex domain' codomain' composition' nonex' G
+
η: naturalTransformation domain codomain composition nonex domain' codomain' composition'
nonex' F G η
for
  domain :: "'a ⇒ 'a" ("dom _" [108] 109) and
  codomain :: "'a⇒'a" ("cod _" [110] 111) and
  composition :: "'a⇒'a⇒'a" (infix "·" 110) and
```

```
  nonex :: "'a" ("*") and
  domain' :: "'b ⇒ 'b" ("dom2 _" [108] 109) and
  codomain' :: "'b⇒'b" ("cod2 _" [110] 111) and
  composition' :: "'b⇒'b⇒'b" (infix "·2" 110) and
  nonex' :: "'b" ("*2") and
  F :: "'a ⇒ 'b" and
  G :: "'a ⇒ 'b" and
  η :: "'a ⇒ 'b" +
assumes
  comp_iso: "A.IdEx x ⟶ (B.Iso (η x))"
```

C.3 Inverse natural transformation

Inverse Natural Transformation

```
locale inverse_natural_tr =
A: category domain codomain composition nonex +
B: category domain' codomain' composition' nonex' +
F: functorCat domain codomain composition nonex domain' codomain' composition' nonex' F
+
G: functorCat  domain codomain composition nonex domain' codomain' composition' nonex' G
+
η: naturalIsomorphism domain codomain composition nonex domain' codomain' composition'
nonex' F G η
for
  domain :: "'a ⇒ 'a" ("dom _" [108] 109) and
  codomain :: "'a⇒'a" ("cod _" [110] 111) and
  composition :: "'a⇒'a⇒'a" (infix "·" 110) and
  nonex :: "'a" ("*") and
  domain' :: "'b ⇒ 'b" ("dom2 _" [108] 109) and
  codomain' :: "'b⇒'b" ("cod2 _" [110] 111) and
  composition' :: "'b⇒'b⇒'b" (infix "·2" 110) and
  nonex' :: "'b" ("*2") and
  F :: "'a ⇒ 'b" and
  G :: "'a ⇒ 'b" and
  η :: "'a ⇒ 'b"

interpretation η': naturalTransformationViaIdentities domain codomain composition nonex
domain' codomain' composition' nonex' G F "λa. B.inv (η a)"
  apply (unfold_locales, simp_all)
...

definition map
  where "map = η'.map"
```

It might be worth noting here how this construction works. We are given a natural isomorphism and would like to construct an inverse through their components. For this purpose, we should prove that taking inverses with indefinite description operator indeed returns us isomorphisms as well but going in reverse. Then we use the extended construction of a *locale* describing the natural transformations through identities and this function $\eta'.\mathtt{map}$ is our desired one.

Appendix D

Monoidal Category

D.1 Binary endofunctor

As a first step of formalization, we create a *locale* `biEndoFunctor`[1] which will later be used to describe the associative behaviour of a tensor binary functor, since there we would need to create two different multiplications out of three morphisms.

```
locale biEndoFunctor =
A: category domain codomain composition nonex +
AxA: productCategory domain codomain composition nonex domain codomain
composition nonex +
AxAxA: productCategory domain codomain composition nonex AxA.Dom AxA.Cod
AxA.Composition AxA.Nonex +
BiF: biFunctor domain codomain composition nonex domain codomain composition
nonex domain codomain composition nonex T
for
  domain::"'a⇒'a" ("dom _") and
  codomain::"'a⇒'a" ("cod _") and
  composition::"'a⇒'a⇒'a" (infix "·" 110) and
  nonex :: "'a" ("*") and
  T :: "'a * 'a ⇒ 'a"

definition TTA
  where "TTA x ≡ T (T (fst x, fst (snd x)), snd (snd x))"

lemma TTA_is_functor:
  shows "functorCat AxAxA.Dom AxAxA.Cod AxAxA.Composition AxAxA.Nonex domain codomain
composition nonex TTA"
  using TTA_def
  apply unfold_locales
  apply (metis AxA.Ex AxAxA.Ex BiF.ntExists prod.collapse)
  apply (metis AxA.Ex AxAxA.Ex BiF.reflects_morp prod.collapse)
  apply (metis AxA.Ex BiF.nntExists BiF.pres1 fst_conv snd_conv)
  apply (metis AxA.Ex BiF.nntExists BiF.pres2 fst_conv snd_conv)
  by (metis AxA.Ex BiF.nntExists BiF.pres3 fst_conv snd_conv)
```

[1] *Endofunctor* simply means that domain and codomain categories are the same, but with the word *binary* we merely create a product category as a domain.

lemma *TTA_simp [simp]:*
shows *"(TTA (x,y,z)) ≅ (T (T (x,y),z))"*
by *(simp add: TTA_def)*

definition *TAT*
where *"TAT x ≡ T (fst x, T (fst (snd x), snd (snd x)))"*

lemma *TAT_is_functor:*
shows *"functorCat AxAxA.Dom AxAxA.Cod AxAxA.Composition AxAxA.Nonex domain codomain composition nonex TAT"*
using *TAT_def*
apply *unfold_locales*
apply *(metis AxA.Ex AxAxA.Ex BiF.ntExists prod.collapse)*
apply *(metis AxA.Ex AxAxA.Ex BiF.reflects_morp prod.collapse)*
apply *(metis AxA.Ex BiF.nntExists BiF.pres1 fst_conv snd_conv)*
apply *(metis AxA.Ex BiF.nntExists BiF.pres2 fst_conv snd_conv)*
by *(metis AxA.Ex BiF.nntExists BiF.pres3 fst_conv snd_conv)*

D.2 Definition of monoidal category

Monoidal Category

locale *Monoidal_Category =*
A: category domain codomain composition nonex +
AxA: productCategory domain codomain composition nonex domain codomain composition nonex +
AxAxA: productCategory domain codomain composition nonex AxA.Dom AxA.Cod AxA.Composition AxA.Nonex +
T: biEndoFunctor domain codomain composition nonex T +
α: naturalIsomorphism AxAxA.Dom AxAxA.Cod AxAxA.Composition AxAxA.Nonex domain codomain composition nonex T.TTA T.TAT α +
μ: naturalIsomorphism domain codomain composition nonex domain codomain composition nonex "λx::'a. T (e,x)" A.map μ +
ϱ: naturalIsomorphism domain codomain composition nonex domain codomain composition nonex "λx::'a. T (x,e)" A.map ϱ
for
domain::"'a⇒'a" ("dom _") **and**
codomain::"'a⇒'a" ("cod _") **and**
*composition::"'a⇒'a⇒'a" (***infix** *"·" 110)* **and**
nonex :: "'a" ("")* **and**
*T :: "'a * 'a ⇒ 'a"* **and**
*α :: "'a * 'a * 'a ⇒ 'a"* **and**
μ :: "'a ⇒ 'a" **and**
ϱ :: "'a ⇒ 'a" **and**
e :: "'a" +
assumes
unit_object: "A.IdEx e" **and**
triangle: "((A.Id x) ∧ (A.Id y)) ⟶ (((T (x, (μ y)))·(α(x,e,y))) ≅ (T(ϱ x,y)))" **and**
pentagon: "((A.Id x) ∧ (A.Id y) ∧ (A.Id z) ∧ (A.Id w)) ⟶ (((T (x,α(y,z,w))) · ((α(x,T(y,z),w)) · (T(α(x,y,z),w)))) ≅

```
        (α(x,y,T(z,w))) · (α(T(x,y),z,w)))"

abbreviation TensorProd (infix "⊗" 56)
  where "x⊗y ≡ T(x,y)"

abbreviation from_hom_seta ("(_):(_)→_(_)")
  where "(x:a→_A b) ≡ A.from_hom_seta x a b"
```

D.3 Inverse transformations in monoidal category

```
interpretation μ_inv: inverse_natural_tr domain codomain composition nonex
domain codomain composition nonex "λx::'a. T (e,x)" A.map μ
  by (simp add: μ.naturalIsomorphism_axioms nat_transform_impl_inverse)

interpretation ϱ_inv: inverse_natural_tr domain codomain composition nonex
domain codomain composition nonex "λx::'a. T (x,e)" A.map ϱ
  by (simp add: ϱ.naturalIsomorphism_axioms nat_transform_impl_inverse)

interpretation α_inv: inverse_natural_tr AxAxA.Dom AxAxA.Cod AxAxA.Composition AxAxA.Nonex
domain codomain composition nonex T.TTA T.TAT α
  by (simp add: α.naturalIsomorphism_axioms nat_transform_impl_inverse)

abbreviation μ_inv ("μ⁻¹")
  where "μ⁻¹ ≡ μ_inv.map"

abbreviation ϱ_inv ("ϱ⁻¹")
  where "ϱ⁻¹ ≡ ϱ_inv.map"

abbreviation α_inv ("α⁻¹")
  where "α⁻¹ ≡ α_inv.map"
```

D.4 Braided monoidal category

Braided Monoidal Category

```
locale Braiding =
A: category domain codomain composition nonex +
AxA: productCategory domain codomain composition nonex domain codomain
composition nonex +
AxAxA: productCategory domain codomain composition nonex AxA.Dom AxA.Cod
AxA.Composition AxA.Nonex +
T: biEndoFunctor domain codomain composition nonex T +
α: naturalIsomorphism AxAxA.Dom AxAxA.Cod AxAxA.Composition AxAxA.Nonex domain codomain
composition nonex T.TTA T.TAT α +
Br: naturalIsomorphism AxA.Dom AxA.Cod AxA.Composition AxA.Nonex domain
codomain composition nonex T T.BiF.Exch γ +
α_inv: inverse_natural_tr AxAxA.Dom AxAxA.Cod AxAxA.Composition AxAxA.Nonex
```

```
domain codomain composition nonex T.TTA T.TAT α +
Br_inv: inverse_natural_tr AxA.Dom AxA.Cod AxA.Composition AxA.Nonex domain
codomain composition nonex T T.BiF.Exch γ
for
  domain::"'a⇒'a" ("dom _") and
  codomain::"'a⇒'a" ("cod _") and
  composition::"'a⇒'a⇒'a" (infix "·" 110) and
  nonex :: "'a" ("*") and
  T :: "'a * 'a ⇒ 'a" and
  α :: "'a * 'a * 'a ⇒ 'a" and
  γ :: "'a * 'a ⇒ 'a" +
assumes
  hexagon1: "((A.Id x) ∧ (A.Id y) ∧ (A.Id z)) ⟶ ((α(y,z,x) · (γ(x,T(y,z))
α(x,y,z))) ≅
             (T(y,γ(x,z)) · (α(y,x,z) · T(γ(x,y),z))))" and
  hexagon2: "((A.Id x) ∧ (A.Id y) ∧ (A.Id z)) ⟶
((α_inv.map(z,x,y) · (γ(T(x,y),z) · α_inv.map(x,y,z))) ≅
             (T(γ(x,z),y) · (α_inv.map(x,z,y) · T(x,γ(y,z)))))"

locale Braided_Mon_Cat =
  M: Monoidal_Category domain codomain composition nonex T α μ ϱ e +
  Braid: Braiding domain codomain composition nonex T α γ
for
  domain::"'a⇒'a" ("dom _") and
  codomain::"'a⇒'a" ("cod _") and
  composition::"'a⇒'a⇒'a" (infix "·" 110) and
  nonex :: "'a" ("*") and
  T :: "'a * 'a ⇒ 'a" and
  α :: "'a * 'a * 'a ⇒ 'a" and
  μ :: "'a ⇒ 'a" and
  ϱ :: "'a ⇒ 'a" and
  e :: "'a" and
  γ :: "'a * 'a ⇒ 'a"
```

D.5 Symmetric monoidal category

Symmetric Monoidal Category

```
locale Symmetry =
Braid: Braiding domain codomain composition nonex T α γ
for
  domain::"'a⇒'a" ("dom _") and
  codomain::"'a⇒'a" ("cod _") and
  composition::"'a⇒'a⇒'a" (infix "·" 110) and
  nonex :: "'a" ("*") and
  T :: "'a * 'a ⇒ 'a" and
  α :: "'a * 'a * 'a ⇒ 'a" and
  γ :: "'a * 'a ⇒ 'a" +
assumes
  symmetry: "((Braid.A.Id a) ∧ (Braid.A.Id b)) ⟶
(γ(b,a) ≅ Braid.Br_inv.map(a,b))"
```

```
locale Symmetric_Mon_Cat =
  M: Monoidal_Category domain codomain composition nonex T α μ ϱ e +
  Sym: Symmetry domain codomain composition nonex T α γ
for
  domain::"'a⇒'a" ("dom _") and
  codomain::"'a⇒'a" ("cod _") and
  composition::"'a⇒'a⇒'a" (infix "·" 110) and
  nonex :: "'a" ("*") and
  T :: "'a * 'a ⇒ 'a" and
  α :: "'a * 'a * 'a ⇒ 'a" and
  μ :: "'a ⇒ 'a" and
  ϱ :: "'a ⇒ 'a" and
  e :: "'a" and
  γ :: "'a * 'a ⇒ 'a"
```

Appendix E

Closed Monoidal Categories

E.1 Left closed structure

Left Closed Monoidal Category

locale *LeftClosedMonCatBifunctor =*
A: category domain codomain composition nonex +
Aop: opposite_category domain codomain composition nonex +
M: Monoidal_Category domain codomain composition nonex T α μ ϱ e *+*
Impl: biAopAEndoFunctor domain codomain composition nonex Impl
for
domain::"'a⇒'a" ("dom _") **and**
codomain::"'a⇒'a" ("cod _") **and**
*composition::"'a⇒'a⇒'a" (***infix** *"·" 110)* **and**
nonex :: "'a" ("")* **and**
*T :: "'a * 'a ⇒ 'a"* **and**
α *:: "'a * 'a * 'a ⇒ 'a"* **and**
μ *:: "'a ⇒ 'a"* **and**
ϱ *:: "'a ⇒ 'a"* **and**
e *:: "'a"* **and**
*Impl :: "'a * 'a ⇒ 'a"* **and**
Φ *:: "'a ⇒ 'a ⇒ 'a ⇒ 'a"* **and**
Ψ *:: "'a ⇒ 'a ⇒ 'a ⇒ 'a" +*
assumes
FunctionSetΦ*: "((f:(a⊗b)→*$_A$*c) ∧ A.IdEx a ∧ A.IdEx b) ⟶*
((Φ *a b f):b→*$_A$*(Impl(a,c)))"* **and**
FunctionSetΨ*: "((g:b→*$_A$*(Impl(a,c))) ∧ A.IdEx a ∧ A.IdEx c) ⟶*
((Ψ *a c g):(a⊗b)→*$_A$*c)"* **and**
BijectionOne: "((f:(a⊗b)→$_A$*c) ∧ A.IdEx a ∧ A.IdEx b) ⟶*
(Ψ *a c (*Φ *a b f) ≃ f)"* **and**
BijectionTwo: "((g:b→$_A$*(Impl(a,c))) ∧ A.IdEx a ∧ A.IdEx c) ⟶*
(Φ *a b (*Ψ *a c g) ≃ g)"* **and**
NaturalityDΦ*: "((f:(a⊗b)→*$_A$*c) ∧ A.IdEx a ∧ A.IdEx b ∧ ((cod x) = a) ∧*
((cod y) = b) ∧ ((dom z) = c)) ⟶
(Φ *(dom x) (dom y) (z·(f·(x⊗y)))) ≃ (Impl(x,z))·((*Φ *a b f)·y)"*

abbreviation *Implication (***infix** *"-∘" 56)*
where *"(x -∘ y) ≡ Impl(x,y)"*

```
lemma NaturalityA:
  shows "((f:(a⊗b)→_A c) ∧ A.IdEx a ∧ A.IdEx b ∧ ((cod x) = a)) ⟶
         (Φ (dom x) b (f · (x⊗b)) ≃ (x-∘c) · (Φ a b f))"
  using NaturalityDΦ[where y = ‹b› and z = ‹c›]
...

lemma NaturalityB:
  shows "((f:(a⊗b)→_A c) ∧ A.IdEx a ∧ A.IdEx b ∧ ((cod y) = b)) ⟶
         ((Φ a (dom y) (f · (a⊗y)) ≃ (Φ a b f)·y))"
  using NaturalityDΦ[where x = ‹a› and z = ‹c›]
...

lemma NaturalityC:
  shows "((f:(a⊗b)→_A c) ∧ A.IdEx a ∧ A.IdEx b ∧ ((dom z) = c)) ⟶
         (Φ a b (z · f) ≃ (a-∘z)·(Φ a b f))"
  using NaturalityDΦ[where x = ‹a› and y = ‹b›]
...
```

Here one might have noted that we are using yet to be defined *locale* `biAopAEndoFunctor`, which is the description of a functor `F`: $\mathcal{C}^{op} \times \mathcal{C} \longrightarrow \mathcal{C}$:

```
locale biAopAEndoFunctor =
A: category domain codomain composition nonex +
Aop: opposite_category domain codomain composition nonex +
AopxA: productCategory Aop.domop Aop.codop Aop.compositionop nonex domain
codomain composition nonex +
BiF: biFunctor Aop.domop Aop.codop Aop.compositionop nonex domain
codomain
composition nonex domain codomain composition nonex Il
for
  domain::"'a⇒'a" ("dom _") and
  codomain::"'a⇒'a" ("cod _") and
  composition::"'a⇒'a⇒'a" (infix "·" 110) and
  nonex :: "'a" ("*") and
  Il :: "'a * 'a ⇒ 'a"
```

E.2 Proof of the naturality property for Ψ

Proof of Proposition 5.3.1.2. $\Psi(y) : A \otimes B \longrightarrow C$ and therefore we can apply the naturality property of Φ to this morphism:

$$\Phi(z \cdot \Psi(f) \cdot (x \otimes y)) \simeq (x \multimap z) \cdot \Phi(\Psi(f)) \cdot y.$$

Applying the bijection Ψ back we get the desired result.

✠

```
lemma NaturalityInvA:
  assumes "((x:b→_A(a-∘c)) ∧ LM.A.Id a ∧ LM.A.Id c) ∧ ((cod f) = a)"
  shows "((Ψ a c x)·T(f,b)) ≃ (Ψ (dom f) c ((f-∘c)·x))"
```

```
...

lemma NaturalityInvB:
  assumes "(x:b→A(a-∘c)) ∧ LM.A.Id a ∧ LM.A.Id c ∧ ((cod f) = b)"
  shows "(Ψ a c x)·T(a,f) ≃ Ψ a c (x·f)"
...

lemma NaturalityInvC:
  assumes "((x:b→A(Impl(a,c))) ∧ LM.A.Id a ∧ LM.A.Id c) ∧ ((dom f) = c)"
  shows "(f·(Ψ a c x)) ≃ (Ψ a (cod f) ((a-∘f)·x))"
...
```

E.3 Universal property of LCS

```
abbreviation Ev :: "'a * 'a => 'a"
where "Ev AB ≡ Ψ (fst AB) (snd AB) (fst AB -∘ snd AB)"

lemma LCS1:
  assumes "((LM.A.IdEx x) ∧ (LM.A.IdEx y))"
  shows "((Ev(x,y)):(x⊗(x-∘y))→Ay)"
...

lemma LCS2:
  assumes "((x:(a⊗b)→Ac) ∧ (LM.A.Id a) ∧ (LM.A.Id b))"
  shows "(∃!h. ((h:b→A(a-∘c)) ∧ (x ≃ Ev(a,c)·(a⊗h))))"
...

lemma LCS31:
  assumes "((LM.A.IdEx a) ∧ (E g))"
  shows "((a-∘g):(a -∘ dom g)→A(a-∘ cod g)) ∧ (g·Ev(a,dom g)) ≃
(Ev(a,cod g)·T(a,Impl(a,g)))"
...

lemma LCS3:
  shows "((LM.A.IdEx a) ∧ (E g)) ⟶ (((a-∘g):(a -∘ dom g)→A(a -∘ cod g)) ∧
(g·Ev(a,dom g)) ≃ (Ev(a,cod g)·(a⊗(a-∘g))))"
...
```

E.4 Isomorphism between A and e $\multimap$ A

```
lemma Ae1:
  assumes "A.IdEx a"
  shows "(Φ e a (μ a)):a→A(e-∘a)"
...
lemma Ae2:
  assumes "A.IdEx a"
```

shows *"∃f. (f:a→$_A$(e-∘a))"*

...

lemma *Ae3:*

assumes *"A.IdEx a"*

shows *"Φ e (e-∘a) (Ψ e a (e-∘a)) ≃ e-∘a"*

...

lemma *Ae4:*

assumes *"A.IdEx a"*

shows *"(A.inv (μ (e-∘a))):(e-∘a)→$_A$(e⊗(e-∘a))"*

...

lemma *Ae5:*

assumes *"A.IdEx a"*

shows *"(Ψ e a (Impl(e,a))):(e⊗(e-∘a))→$_A$a"*

...

lemma *Ae6:*

assumes *"A.IdEx a"*

shows *"((Ψ e a (e-∘a))·(A.inv (μ (e-∘a)))):(e-∘a)→$_A$a"*

...

lemma *Ae7:*

assumes *"A.IdEx a"*

shows *"(((Ψ e a (e-∘a))·(A.inv (μ (e-∘a))))·(Φ e a (μ a))):a→$_A$a"*

...

lemma *Ae8:*

assumes *"A.IdEx a"*

shows *"(Φ e a (μ a))·(μ a) ≃ (μ (e-∘a))·(e⊗(Φ e a (μ a)))"*

...

lemma *Ae9:*

assumes *"A.IdEx a"*

shows *"A.IdEx ((A.inv (μ a))·(μ a))"*

...

lemma *Ae10:*

assumes *"A.IdEx a"*

shows *"A.IdEx ((μ a)·(A.inv (μ a)))"*

...

lemma *Ae11:*

assumes *"A.IdEx a"*

shows *"Φ e a (μ a) ≃ ((μ (e-∘a))·(e⊗(Φ e a (μ a))))·(A.inv (μ a))"*

...

lemma *Ae12:*

assumes *"A.IdEx a"*

shows *"(A.inv (μ (e-∘a)))·(Φ e a (μ a)) ≃*
(e⊗(Φ e a (μ a)))·(A.inv (μ a))"

...

lemma *Ae13:*

assumes *"A.IdEx a"*

shows *"(Ψ e a (e-∘a))·((A.inv (μ (e-∘a)))·(Φ e a (μ a))) ≃*
(Ψ e a (e-∘a))·((e⊗(Φ e a (μ a)))·(A.inv (μ a)))"

...

lemma *Ae14:*

assumes *"A.IdEx a"*

shows *"(Φ e a ((Ψ e a (e-∘a))·(e⊗(Φ e a (μ a))))) ≃*
(e-∘a)·(Φ e a (μ a))"

...

lemma *Ae15:*

```
  assumes "A.IdEx a"
  shows "(Φ e a ((Ψ e a (e-∘a))·(e⊗(Φ e a (μ a))))) ≃ (Φ e a (μ a))"
...
lemma Ae16:
  assumes "A.IdEx a"
  shows "Ψ e a (Φ e a ((Ψ e a (e-∘a))·(e⊗(Φ e a (μ a))))) =
(Ψ e a (e-∘a))·(e⊗(Φ e a (μ a)))"
...
lemma Ae17:
  assumes "A.IdEx a"
  shows "Ψ e a (Φ e a (μ a)) = μ a"
...
lemma Ae18:
  assumes "A.IdEx a"
  shows "((Ψ e a (e-∘a))·(e⊗(Φ e a (μ a)))) ≃ (μ a)"
...
lemma Ae19:
  assumes "A.IdEx a"
  shows "(Ψ e a (e-∘a))·((A.inv (μ (e-∘a)))·(Φ e a (μ a))) ≃ a"
...
lemma Ae20:
  assumes "A.IdEx a"
  shows "(Φ e a (μ a))⇌((Ψ e a (e-∘a))·((A.inv (μ (e-∘a)))))"
...
lemma Ae21:
  assumes "A.IdEx a"
  shows "A.Iso_a (Φ e a (μ a))"
...
lemma Ae:
  assumes "A.IdEx a"
  shows "A.isomorphic a (e-∘a)"
...
```

The reason we do not presented all unfolded proofs here and along the work is that the Sledgehammer tool also uses additional lemmas and facts that we prove while the debugging process of the encoded definitions and also for the sake of saving space.

E.5 Right closed structure

Right Closed Monoidal Category

```
locale RightClosedMonCatBifunctor =
A: category domain codomain composition nonex +
Aop: opposite_category domain codomain composition nonex +
M: Monoidal_Category domain codomain composition nonex T α μ ϱ e +
Impr: biAopAEndoFunctor domain codomain composition nonex Impr
for
domain::"'a⇒'a" ("dom _") and
codomain::"'a⇒'a" ("cod _") and
composition::"'a⇒'a⇒'a" (infix "·" 110) and
nonex :: "'a" ("*") and
```

*T :: "'a * 'a ⇒ 'a"* **and**
α *:: "'a * 'a * 'a ⇒ 'a"* **and**
μ *:: "'a ⇒ 'a"* **and**
ϱ *:: "'a ⇒ 'a"* **and**
e *:: "'a"* **and**
*Impr :: "'a * 'a ⇒ 'a"* **and**
Φr *:: "'a ⇒ 'a ⇒ 'a ⇒ 'a"* **and**
Ψr *:: "'a ⇒ 'a ⇒ 'a ⇒ 'a" +*
assumes
FunctionSetΦ*: "((f:(b*a)→$_A$c) ∧ (A.Id a) ∧ (A.Id b)) ⟶*
((Φr *b a f):b→$_A$(Impr(a,c)))"* **and**
FunctionSetΨ*: "((g:b→$_A$(Impr(a,c))) ∧ (A.Id a) ∧ (A.Id c)) ⟶*
((Ψr *a c g):(b*a)→$_A$c)"* **and**
*BijectionOne: "((f:(b*a)→$_A$c) ∧ (A.IdEx a) ∧ (A.IdEx b)) ⟶*
(Ψr *a c (*Φr *b a f) = f)"* **and**
BijectionTwo: "((g:b→$_A$(Impr(a,c))) ∧ (A.IdEx a) ∧ (A.IdEx c)) ⟶
(Φr *b a (*Ψr *a c g) = g)"* **and**
NaturalityDΦ*: "((f:(b*a)→$_A$c) ∧ (A.IdEx a) ∧ (A.IdEx b) ∧ ((cod x) = a) ∧ ((cod y) = b) ∧ ((dom z) = c)) ⟶*
(Φr *(dom y) (dom x) (z·(f·(y*x))) ≃ (Impr(x,z))·((*Φr *b a f)·y))"*
...
abbreviation *RImplication (***infix** *"○-" 56)*
where *"x ○- y ≡ Impr(x,y)"*

Appendix F

Symmetric Monoidal Closed Category - a Model for IMLL

Symmetric Monoidal Closed Category

locale *SymmetricMonClosedCat =*
LM: LeftClosedMonCatBifunctor domain codomain composition nonex T α μ ϱ e *Impl* Φ Ψ *+*
Sym: Symmetry domain codomain composition nonex T α γ
for
domain::"'a⇒'a" ("dom _") **and**
codomain::"'a⇒'a" ("cod _") **and**
*composition::"'a⇒'a⇒'a" (***infix** *"·" 110)* **and**
nonex :: "'a" ("")* **and**
*T :: "'a * 'a ⇒ 'a"* **and**
α *:: "'a * 'a * 'a ⇒ 'a"* **and**
μ *:: "'a ⇒ 'a"* **and**
ϱ *:: "'a ⇒ 'a"* **and**
e *:: "'a"* **and**
γ *:: "'a * 'a ⇒ 'a"* **and**
*Impl :: "'a * 'a ⇒ 'a"* **and**
Φ *:: "'a ⇒ 'a ⇒ 'a ⇒ 'a"* **and**
Ψ *:: "'a ⇒ 'a ⇒ 'a ⇒ 'a" +*
fixes
FalseM :: "'a" ("$\bot_m$*")*
assumes
FM: "LM.A.IdEx ($\bot_m$*)"*

F.1 RCS defined via LCS and symmetry

abbreviation *Impr*
where *"Impr AB ≡ Impl AB"*

abbreviation *ImprA (***infix** *"○-" 56)*
where *"y ○- x ≡ Impr(y,x)"*

```
abbreviation Φr :: "'a ⇒ 'a ⇒ 'a ⇒ 'a"
  where "Φr b a x ≡ Φ a b (x·γ(a,b))"

abbreviation Ψr :: "'a ⇒ 'a ⇒ 'a ⇒ 'a"
  where "Ψr a c x ≡ (Ψ a c x)·γ(dom x,a)"

lemma RCS_ABC:
  assumes "((f:(b⊗a)→_A c) ∧ (LM.A.IdEx a) ∧ (LM.A.IdEx b) ∧ ((cod x) = a) ∧ ((cod y)
= b) ∧ ((dom z) = c))"
  shows "(Φr (dom y) (dom x) (z·(f·(y⊗x))) ≃ (Impr(x,z))·((Φr b a f)·y))"
proof-
  have 1: "(((f·γ(a,b)):(a⊗b)→_A c) ∧ (LM.A.IdEx a) ∧ (LM.A.IdEx b) ∧ ((cod x) = a) ∧
((cod y) = b) ∧ ((dom z) = c))"
  then have 2: "(Φ (dom x) (dom y) (z·((f·γ(a,b))·(x⊗y)))) ≃
(Impl(x,z))·((Φ a b (f·γ(a,b)))·y)" using LM.NaturalityDΦ by blast
  have 3: "(z·((f·γ(a,b))·(x⊗y))) ≃ (z·(f·(γ(a,b)·(x⊗y))))"
  have 4: "γ(cod x,cod y)·(x⊗y) ≅ (y⊗x)·γ(dom x, dom y)"
using Sym.Braid.Br.η.naturality
  then show ?thesis using 2 3 by (metis "1" LM.A.S2 LM.A.S3 LM.A.S4)
qed

interpretation RightClosedStr: RightClosedMonCatBifunctor domain codomain
composition nonex T α μ ϱ e Impr Φr Ψr
  apply unfold_locales
...
  using RCS_ABC by blast
```

F.2 Double negation

$\delta_A : A \longrightarrow (A \multimap \bot) \multimap \bot$

```
lemma Delta1:
  assumes "f:a→_A a'"
  shows "((f-○⊥_m)-○⊥_m)·(Φr a (a-○⊥_m) (Ψ a ⊥_m (a-○⊥_m)))
       = (Φr a' (a'-○⊥_m) (Ψ a' ⊥_m (a'-○⊥_m)))·f"
proof-
  have 1: "((Φr a' (a'-○⊥_m) (Ψ a' ⊥_m (a'-○⊥_m)))·f):a→_A((a'-○⊥_m)-○⊥_m)" ...
  have 2: "((((f-○⊥_m)-○⊥_m)·(Φr a (a-○⊥_m) (Ψ a ⊥_m (a-○⊥_m)))):a→_A
((a'-○⊥_m)-○⊥_m))" ...
  have 3: "Ψ a ⊥_m (f-○⊥_m) ≃ (Ψ a' ⊥_m (a'-○⊥_m))·(f⊗(a'-○⊥_m))"
    using NaturalityInvA[where ?x = ‹(a'-○⊥_m)› and ?b = ‹(a'-○⊥_m)› and ?a = ‹a'› and ?c
= ‹⊥_m› and ?f = ‹f›] ...
  have 4: "(Ψ a ⊥_m (a-○⊥_m))·(a⊗(f-○⊥_m)) ≃ Ψ a ⊥_m (f-○⊥_m)"
    using NaturalityInvB[where ?x = ‹(a-○⊥_m)› and ?b = ‹(a-○⊥_m)›
and ?c = ‹⊥_m› and ?a = ‹a› and ?f = ‹(a'-○⊥_m)›] ...
  have 5: "(Ψ a' ⊥_m (a'-○⊥_m)):(T(a',(a'-○⊥_m)))→_A⊥_m" ...
  have 6: "(Ψ a ⊥_m ((a-○⊥_m))):(T(a,(a-○⊥_m)))→_A⊥_m" ...
  obtain a1 where pa1: "a1 = (Ψ a' ⊥_m ((a'-○⊥_m)))" by blast
  obtain a2 where pa2: "a2 = (Ψ a ⊥_m ((a-○⊥_m)))" by blast
  have 7: "(Φr a' ((a'-○⊥_m)) a1)·f ≃ Φr a ((a-○⊥_m)) (a1·T(f,(a'-○⊥_m)))" ...
  have 8: "(((f-○⊥_m)-○⊥_m))·(Φr a ((a-○⊥_m)) a2) ≃
```

```
Φr a ((a'-∘⊥m)) (a2·T(a,(f-∘⊥m)))" ...
  have 9: "(Φr a' ((a'-∘⊥m)) (Ψ a' ⊥m ((a'-∘⊥m))))·f ≃
           (Φr a ((a'-∘⊥m)) (Ψ a ⊥m ((f-∘⊥m))))" by (metis "3" "7" pa1)
  have 10: "(((f-∘⊥m)-∘⊥m))·(Φr a ((a-∘⊥m)) (Ψ a ⊥m ((a-∘⊥m)))) ≃
           (Φr a ((a'-∘⊥m)) (Ψ a ⊥m ((f-∘⊥m))))" by (metis "4" "8" pa2)
  show ?thesis using "10" "9" by presburger
qed

abbreviation δ :: "'a ⇒ 'a"
  where "δ a ≡  Φr a (a-∘⊥m) (Ψ a ⊥m (a-∘⊥m))"

lemma Delta20:
  assumes "LM.A.from_hom_seta f a a'"
  shows "LM.A.commSquareEx (((f-∘⊥m)-∘⊥m)) (δ a) f (δ a')"
...

lemma Delta3:
  assumes "LM.A.IdEx a"
  shows "(δ a):a→A((a-∘⊥m)-∘⊥m)"
...
interpretation DeltaNatId: naturalTransformationViaIdentities domain codomain
  composition nonex domain codomain composition nonex
  LM.A.map "λf. ((f-∘⊥m)-∘⊥m)" δ
```

F.3 Proof rules

IMLL proof rules inside SMCC

```
lemma RuleIdEx:
  assumes "LM.A.IdEx a"
  shows "a:a→Aa"
...
lemma RuleExchange:
  assumes "f:(((Γ⊗(A⊗B))⊗Δ))→AC" and "LM.A.Id A" and "LM.A.Id B"
and "LM.A.Id Γ" and "LM.A.Id Δ"
  shows "(f·(((Γ⊗γ(B,A))⊗Δ))):(((Γ⊗(B⊗A))⊗Δ))→AC"
...
lemma RuleCut:
  assumes "f:Γ→AA" and "g:(A⊗Δ)→AB" and "LM.A.Id Δ"
  shows "(g·T(f,Δ)):(T(Γ,Δ))→AB"
...
lemma RuleTensorR:
  assumes "f:Γ→AA" and "g:Δ→AB"
  shows "(f⊗g):(Γ⊗Δ)→A(A⊗B)"
...
lemma RuleTensorL:
  assumes "f:(((Γ⊗A)⊗B)⊗Δ)→AC" and "LM.A.Id A" and "LM.A.Id B"
and "LM.A.Id Γ" and "LM.A.Id Δ"
  shows "(f·((α⁻¹(Γ,A,B))⊗Δ)):((Γ⊗(A⊗B))⊗Δ)→AC"
...
lemma RuleImplicationR:
```

assumes *"f:(A⊗Γ)→$_A$B"* **and** *"LM.A.Id A"* **and** *"LM.A.Id Γ"*
shows *"(Φ A Γ f):Γ→$_A$(A-∘B)"*
...
lemma *RuleImplicationL:*
assumes *"f:Γ→$_A$A"* **and** *"g:B⊗Δ→$_A$C"* **and** *"LM.A.Id B"* **and** *"LM.A.Id Δ"*
shows *"((g·(Ev(A,B)⊗Δ))·((f⊗(A-∘B))⊗Δ)):((Γ⊗(A-∘B))⊗Δ)→$_A$C"*
...
lemma *RuleUnitRμ:*
assumes *"LM.A.IdEx a"*
shows *"(μ^{-1} a):a→$_A$(e⊗a)"*
...
lemma *RuleUnitRϱ:*
assumes *"LM.A.IdEx a"*
shows *"(ϱ^{-1} a):a→$_A$(a⊗e)"*
...
lemma *RuleUnitL:*
assumes *"f:(Γ⊗Δ)→$_A$A"* **and** *"LM.A.Id Γ"* **and** *"LM.A.Id Δ"*
shows *"(f·(Γ⊗(μ Δ))):(Γ⊗(e⊗Δ))→$_A$A"*
...

Bibliography

[Abramsky, 1993] Abramsky, S. (1993). Computational interpretations of linear logic. *Theoretical Computer Science*, 111(1):3–57.

[Abramsky and Tzevelekos, 2011] Abramsky, S. and Tzevelekos, N. (2011). *Introduction to Categories and Categorical Logic*, pages 3–94. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Adamek et al., 1990] Adamek, J., Herrlich, H., and Strecker, G. (1990). *Abstract and Concrete Categories: The Joy of Cats*. A Wiley-Interscience publication. Wiley.

[Alexiev, 1994] Alexiev, V. (1994). Applications of Linear Logic to Computation: An Overview. *Logic Journal of the IGPL*, 2(1):77–107.

[Andrews, 1972a] Andrews, P. B. (1972a). General Models and Extensionality. *Journal of Symbolic Logic*, 37,2:395–397.

[Andrews, 1972b] Andrews, P. B. (1972b). General Models, Descriptions, and Choice in Type Theory. *Journal of Symbolic Logic*, 37,2:385–394.

[Ballarin, 2004] Ballarin, C. (2004). Locales and Locale Expressions in Isabelle/Isar. In Berardi, S., Coppo, M., and Damiani, F., editors, *Types for Proofs and Programs*, pages 34–50, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Ballarin, 2010] Ballarin, C. (2010). Tutorial to Locales and Locale Interpretation. *Contribuciones científicas en honor de Mirian Andrés Gómez, 2010-01-01, ISBN 978-84-96487-50-5, pages. 123-140*.

[Ballarin, 2020] Ballarin, C. (2020). Exploring the Structure of an Algebra Text with Locales. *Journal of Automated Reasoning*, 64(6):1093–1121.

[Barendregt et al., 2013] Barendregt, H., Dekkers, W., and Statman, R. (2013). *Lambda Calculus with Types*. Perspectives in Logic. Cambridge University Press.

[Beffara, 2013] Beffara, E. (2013). Introduction to linear logic. Lecture.

[Benzmüller and Miller, 2014] Benzmüller, C. and Miller, D. (2014). Automation of Higher-Order Logic. In Siekmann, J., editor, *Computational Logic*, volume 9 of *Handbook of the History of Logic*, pages 215–254. North Holland.

[Benzmüller and Scott, 2016] Benzmüller, C. and Scott, D. (2016). Automating Free Logic in Isabelle/HOL. In Greuel, G.-M., Koch, T., Paule, P., and Sommese, A., editors, *Mathematical Software – ICMS 2016*, pages 43–50, Cham. Springer International Publishing.

[Benzmüller and Scott, 2016] Benzmüller, C. and Scott, D. S. (2016). Axiomatizing Category Theory in Free Logic. *CoRR*, abs/1609.01493.

[Benzmüller and Scott, 2020] Benzmüller, C. and Scott, D. S. (2020). Automating Free Logic in HOL, with an Experimental Application in Category Theory. *Journal of Automated Reasoning*, 64(1):53–72.

[Benzmüller, 2019] Benzmüller, C. (2019). Universal (meta-)logical reasoning: Recent successes. *Science of Computer Programming*, 172:48–62.

[Benzmüller and Andrews, 2019] Benzmüller, C. and Andrews, P. (2019). Church's Type Theory. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2019 edition.

[Benzmüller et al., 2004] Benzmüller, C., Brown, C. E., and Kohlhase, M. (2004). Higher-Order Semantics and Extensionality. *The Journal of Symbolic Logic*, 69(4):1027–1088.

[Blanchette et al., 2013] Blanchette, J., Böhme, S., and Paulson, L. C. (2013). Extending Sledgehammer with SMT Solvers. *Journal of Automated Reasoning*, 51:109–128.

[Blanchette and Nipkow, 2010] Blanchette, J. C. and Nipkow, T. (2010). Nitpick: A Counterexample Generator for Higher-Order Logic Based on a Relational Model Finder. In Kaufmann, M. and Paulson, L. C., editors, *Interactive Theorem Proving*, pages 131–146, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Blanchette et al., 2012] Blanchette, J. C., Popescu, A., Wand, D., and Weidenbach, C. (2012). More SPASS with Isabelle. In Beringer, L. and Felty, A., editors, *Interactive Theorem Proving*, pages 345–360, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Brown, 2012] Brown, C. E. (2012). Satallax: An Automatic Higher-Order Prover. In Gramlich, B., Miller, D., and Sattler, U., editors, *Automated Reasoning*, pages 111–117, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Casadio, 2001] Casadio, C. (2001). Non-Commutative Linear Logic in Linguistics. *Grammars*, 4(3):167–185.

[Chaudhuri and Pfenning, 2005] Chaudhuri, K. and Pfenning, F. (2005). A Focusing Inverse Method Prover for First-Order Linear Logic. In R.Nieuwenhuis, editor, *Proceedings of the 20th International Conference on Automated Deduction (CADE-20)*, pages 69–83, Tallinn, Estonia. Springer Verlag LNCS 3632.

[Church, 1940] Church, A. (1940). A Formulation of the Simple Theory of Types. *The Journal of Symbolic Logic*, 5(2):56–68.

[Cockett and Seely, 2017] Cockett, J. R. B. and Seely, R. A. G. (2017). *Proof Theory of the Cut Rule*. Categories for the Working Philosopher. Oxford University Press, Oxford.

[Curry and Feys, 1958] Curry, H. H. and Feys, R. (1958). *Combinatory Logic*, volume 1 of *Studies in Logic and the Foundations of Mathematics*. North Holland Publishing Company, Amsterdam.

[de Moura and Bjørner, 2008] de Moura, L. and Bjørner, N. (2008). Z3: An Efficient SMT Solver. In Ramakrishnan, C. R. and Rehof, J., editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg. Springer Berlin Heidelberg.

[de Moura et al., 2015] de Moura, L., Kong, S., Avigad, J., van Doorn, F., and von Raumer, J. (2015). The Lean Theorem Prover (System Description). In Felty, A. P. and Middeldorp, A., editors, *Automated Deduction - CADE-25*, pages 378–388, Cham. Springer International Publishing.

[Deters et al., 2014] Deters, M., Reynolds, A., King, T., Barrett, C. W., and Tinelli, C. (2014). A tour of CVC4: How it works, and how to use it. *2014 Formal Methods in Computer-Aided Design (FMCAD)*, pages 7–7.

[Di Cosmo and Miller, 2019] Di Cosmo, R. and Miller, D. (2019). Linear Logic. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2019 edition.

[Došen and Petrić, 2004] Došen, K. and Petrić, Z. (2004). *Proof-theoretical Coherence*. Studies in logic. King's College Publications.

[Farmer, 1990] Farmer, W. M. (1990). A Partial Functions Version of Church's Simple Theory of Types. *The Journal of Symbolic Logic*, 55(3):1269–1291.

[Farmer, 2008] Farmer, W. M. (2008). The seven virtues of simple type theory. *Journal of Applied Logic*, 6(3):267–286.

[Freyd and Scedrov, 1990] Freyd, P. and Scedrov, A. (1990). *Categories, Allegories*, volume 39 of *North-Holland Mathematical Library*. Elsevier Science.

[Gabbay, 2014] Gabbay, D. M. (2014). *Introduction to Labelled Deductive Systems*, volume 17 of *Handbook of Philosophical Logic*, pages 179–266. Springer Netherlands, Dordrecht.

[Gentzen, 1935a] Gentzen, G. (1935a). Untersuchungen über das logische Schließen. I. *Mathematische Zeitschrift*, 39(1):176–210.

[Gentzen, 1935b] Gentzen, G. (1935b). Untersuchungen über das logische Schließen. II. *Mathematische Zeitschrift*, 39(1):405–431.

[Girard, 2011] Girard, J. (2011). *The Blind Spot: Lectures on Logic*. European Mathematical Society, Zürich, Switzerland.

[Girard et al., 1989] Girard, J., Lafont, Y., and Taylor, P. (1989). *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.

[Girard, 1987] Girard, J.-Y. (1987). Linear logic. *Theoretical Computer Science*, 50(1):1–101.

[Girard, 1995] Girard, J.-Y. (1995). *Linear Logic: its syntax and semantics*, page 1–42. London Mathematical Society Lecture Note Series. Cambridge University Press.

[Guglielmi and Straßburger, 2001] Guglielmi, A. and Straßburger, L. (2001). Non-commutativity and MELL in the Calculus of Structures. In Fribourg, L., editor, *Computer Science Logic*, pages 54–68, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Henkin, 1950] Henkin, L. (1950). Completeness in the Theory of Types. *Journal of Symbolic Logic*, 15,2:81–91.

[Hintikka, 1959] Hintikka, J. (1959). Existential Presuppositions and Existential Commitments. *The Journal of Philosophy*, 56(3):125–137.

[Howard, 1980] Howard, W. A. (1980). The Formulae-as-Types Notion of Construction. In Curry, H., B., H., Roger, S. J., and Jonathan, P., editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 479–490. Academic Press.

[Irvine and Deutsch, 2021] Irvine, A. D. and Deutsch, H. (2021). Russell's Paradox. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2021 edition.

[Kovács and Voronkov, 2013] Kovács, L. and Voronkov, A. (2013). First-Order Theorem Proving and Vampire. In Sharygina, N. and Veith, H., editors, *Computer Aided Verification*, pages 1–35, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Lambek and Scott, 1986] Lambek, J. and Scott, P. J. (1986). *Introduction to Higher Order Categorical Logic*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, USA.

[Lambert, 1963] Lambert, K. (1963). Existential import revisited. *Notre Dame Journal of Formal Logic*, 4(4):288 – 292.

[Lambert and Van Fraassen, 1972] Lambert, K. and Van Fraassen, B. (1972). *Derivation and Counterexample: An Introduction to Philosophical Logic*. Dickenson Publishing Company.

[Leinster, 2004] Leinster, T. (2004). *Higher Operads, Higher Categories*. London Mathematical Society Lecture Note Series. Cambridge University Press.

[Lincoln, 1992] Lincoln, P. (1992). Linear Logic. *ACM SIGACT News*, 23.2:29–37.

[MacLane, 1965] MacLane, S. (1965). Categorical algebra. *Bulletin of the American Mathematical Society*, 71(1):40 – 106.

[MacLane, 1971] MacLane, S. (1971). *Categories for the Working Mathematician*. Springer-Verlag, New York. Graduate Texts in Mathematics, Vol. 5.

[Mahboubi and Tassi, 2021] Mahboubi, A. and Tassi, E. (2021). *Mathematical Components*. Zenodo.

[Makarenko, 2020] Makarenko, I. (2020). Free Higher-Order Logic: Notion, Definition and Embedding in HOL (Master's Thesis).

[Makarenko and Benzmüller, 2020] Makarenko, I. and Benzmüller, C. (2020). Positive Free Higher-Order Logic and Its Automation via a Semantical Embedding. KI 2020: Advances in Artificial Intelligence, pages 116–131, Cham. Springer International Publishing.

[Martin-Löf, 1987] Martin-Löf, P. (1987). Truth of a proposition, evidence of a judgement, validity of a proof. *Synthese*, 73(3):407–420.

[Melliès, 2009] Melliès, P.-A. (2009). *Categorical semantics of linear logic*, pages 1 – 196. Number 27 in Panoramas et Synthèses. Société Mathématique de France.

[Meyer et al., 1982] Meyer, R. K., Bencivenga, E., and Lambert, K. (1982). The Ineliminability of E! In Free Quantification Theory without Identity. *Journal of Philosophical Logic*, 11(2):229–231.

[Morscher and Simons, 2001] Morscher, E. and Simons, P. (2001). *Free Logic: A Fifty-Year Past and an Open Future*, pages 1–34. New Essays in Free Logic: In Honour of Karel Lambert. Springer Netherlands, Dordrecht.

[Nipkow et al., 2002] Nipkow, T., Paulson, L. C., and Wenzel, M. (2002). *A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg.

[nLab authors, 2021a] nLab authors (2021a). coherence theorem for monoidal categories. http://ncatlab.org/nlab/show/coherence%20theorem%20for%20monoidal%20categories. Revision 17.

[nLab authors, 2021b] nLab authors (2021b). single-sorted definition of a category. http://ncatlab.org/nlab/show/single-sorted%20definition%20of%20a%20category. Revision 8.

[Nolt, 2020] Nolt, J. (2020). Free Logic. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2020 edition.

[Ohlbach et al., 2001] Ohlbach, H. J., Nonnengart, A., de Rijke, M., and Gabbay, D. M. (2001). *Encoding Two-Valued Nonclassical Logics in Classical Logic*, page 1403–1486. Elsevier Science Publishers B. V., NLD.

[Paulsson and Blanchette, 2012] Paulsson, L. C. and Blanchette, J. C. (2012). Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In *Proceedings of the 8th International Workshop on the Implementation of Logics (IWIL-2010), Yogyakarta, Indonesia. EPiC*, volume 2.

[Schulz, 2013] Schulz, S. (2013). System Description: E 1.8. In McMillan, K., Middeldorp, A., and Voronkov, A., editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 735–743, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Schütte, 1960] Schütte, K. (1960). Syntactical and Semantical Properties of Simple Type Theory. *The Journal of Symbolic Logic*, 25(4):305–326.

[Scott, 1967] Scott, D. (1967). Existence and Description in Formal Logic. In Schoenman, R., editor, *Bertrand Russell, Philosopher of the Century*, pages 181–200. Allen and Unwin, London.

[Scott, 1979] Scott, D. (1979). Identity and existence in intuitionistic logic. In Fourman, M., Mulvey, C., and Scott, D., editors, *Applications of Sheaves: Proceedings of the Research Symposium on Applications of Sheaf Theory to Logic, Algebra, and Analysis*, pages 660–696, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Steen and Benzmüller, 2019] Steen, A. and Benzmüller, C. (2019). The Higher-Order Prover Leo-III (Extended Abstract). In Benzmüller, C. and Stuckenschmidt, H., editors, *KI 2019: Advances in Artificial Intelligence*, pages 333–337, Cham. Springer International Publishing.

[Tubella and Straßburger, 2019] Tubella, A. A. and Straßburger, L. (2019). Introduction to Deep Inference. Lecture.

[Wenzel, 1999] Wenzel, M. (1999). Isar — A Generic Interpretative Approach to Readable Formal Proof Documents. In Bertot, Y., Dowek, G., Théry, L., Hirschowitz, A., and Paulin, C., editors, *Theorem Proving in Higher Order Logics*, pages 167–183, Berlin, Heidelberg. Springer Berlin Heidelberg.