Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 300 (2014) 71–87

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

J-Calc: A Typed Lambda Calculus for Intuitionistic Justification Logic

Konstantinos Pouliasis[1,2,5]

Department of Computer Science
The Graduate Center at City University of New York,
NY, USA

Giuseppe Primiero[3,4,5]

Department of Computer Science
Middlesex University
United Kingdom

Abstract

In this paper we offer a system J-Calc that can be regarded as a typed λ-calculus for the $\{\rightarrow, \bot\}$ fragment of Intuitionistic Justification Logic. We offer different interpretations of J-Calc, in particular, as a two phase proof system in which we proof check the validity of deductions of a theory T based on deductions from a stronger theory T' and computationally as a type system for separate compilations. We establish some first metatheoretic results.

Keywords: Typed λ-calculus, Justification Logic, Modular Programming.

1 Introduction

A plausible reading of Gödel's incompleteness results ([18]) is that the notion of "validity" diverges from that of "truth within a specific theory": given a theory that includes enough arithmetic, there are statements whose validity can only be established in a theory of larger proof-strength. This phenomenon can be shown even

[1] This research is part of Konstantinos Pouliasis' Phd study as a Enhanced Chancellor Fellow at the Graduate Center of the City University of New York under the supervision of Prof. Sergei Artemov. He is indebted to Prof. Sergei Artemov's advisement and guidance during his academic journey.

[2] Email: Kpouliasis@gc.cuny.edu

[3] This research was conducted while Giuseppe Primiero was a Post-Doctoral Fellow of the Research Foundation Flanders (FWO) at the Centre for Logic and Philosophy of Science, Ghent University Belgium. He gratefully acknowledges the financial support.

[4] Email: G.Primiero@mdx.ac.uk

[5] Both authors are indebted to the constructive comments of the anonymous referees of the Intuitionistic Modal Logic and Applications Workshop, 2013.

http://dx.doi.org/10.1016/j.entcs.2013.12.012

with non-Gödelian arguments in the relation e.g. between $\mathsf{I}\Delta_0$ and $\mathsf{I}\Sigma_1$ arithmetic [27], $\mathsf{I}\Sigma_1$ and PA, PA and ZF, etc. [29,15]. The very same issues arise in automated theorem proving. A good example is given by type systems and interactive theorem provers (e.g. Coq, Agda) of the typed functional paradigm. In such systems, when termination of functions has to be secured, one might need to invoke stronger proof principles. The need for reasoning about two kinds of proof objects within a type system is apparent most of all when one wants to establish non-admissibility results for a theory T that can, in contrast, be proved in some stronger T'. The type system, then, has to reconcile the existence of a proof object of some type ϕ in some T' and a proof object of type $\neg\exists s.Prov_T(s,\phi)$ that witnesses the non-provability of ϕ (in T).

In this work, we argue that the explicit modality of Justification Logic [7] can be used to axiomatize relations between objects of two different calculi such as those mentioned above. It is well known that the provability predicate can be axiomatized using a modality [14], [9]. The Logic of Proofs LP [3] goes further and provides explicit proof terms (*proof polynomials*) to inhabit judgments on validity. By translating reasoning in Intuitionistic Propositional Calculus (IPC) to classical proofs, LP obtains a classical semantics for IPC through a modality (inducing a BHK semantics). In this paper we axiomatize the relation between the two kinds of proof objects explicitly, by creating a modal type theory that reasons about bindings or linking of objects from two calculi: a lower-level theory T, formulated as IPC with Church-style λ-terms representing intuitionistic proof objects; and a higher-level, possibly stronger and classical (co-)theory T' fixed as foundational, with *justifications* expressing its proof objects. The axiomatization of such a (co-)theory follows directly the proof system of Justification Logic (here restricted to its applicative K-fragment) and is used to interpret classically (meaning *truth-functionally*) the constructions of the intuitionistic natural deduction. The underlying principle of our linking system is as follows:

$$constructive\ \ necessity = admissible\ \ validity = truth\ (in\ T) + validity\ (in\ T')$$

Necessity of a true (in T) proposition P is, thus, sensitive to the existence of a proof (witnessed by a justification) of its intended interpretation within T'. We assume an interpretation function on types $Just$ that maps the type universe of T into the type universe of T'. We employ judgments of the kind $M : P$ (read as "M is a proof of type P in T") that represent truths in T and judgments of the kind $j : \mathsf{Just}\ P$ (to be read as "j is a justification of the interpretation of P in T'") that represent truth in T' (validity). Incorporating them, the principle can be rewritten in a judgmental fashion:

$$M : P + j : \mathsf{Just}\ P \Rightarrow \Box^{\mathsf{j}} P\ true$$

Notice that the $\Box$-types are indexed by justifications ($\Box^{\mathsf{j}}P$) being sensitive to the interpretation (T') chosen. To complete the picture we need canonical elements of $\Box^{\mathsf{j}}$-types. Naturally, witnesses of this kind are *links* between proof objects from T

and T' with corresponding types (P and $\mathsf{Just}\ \mathsf{P}$). For that reason we introduce a *linking witness* constructor $Link$. This is how necessity is introduced: by proof-checking deductions of T with deductions of T', we reason constructively about admissibility of valid (via T') statements in T. The principle thus becomes:

$$M : P + j : \mathsf{Just}\ P \Rightarrow Link(M, j) : \Box^{\mathsf{j}} P$$

We show how this principle is admissible in our system.

A possible application of the presented type theory can be a refined type system for programming languages with modular programming constructs or external function calls as we show in section 5. In these kinds of languages (e.g. of the ML family) a program or module can call for external definitions that are implemented elsewhere (in another module or, even in another language) [6] . We can read functions within $\Box$-types indexed by justifications as linking processes for such languages that perform the mapping of well–typed constructs importing and using module signatures into their residual programs. By residual programs we mean programs where all instances of module types and function calls are replaced by (i.e. *linked* to) their actual implementations, which remain hidden in the module. We show with a real example how, with slight modifications, our type system can find a natural application in this setting. Here we focus on the type system itself and not on its operational semantics.

The backbone of this work is the idea of representing the proof theoretic semantics for IPC through modality that stems from [5],[6]. An operational approach to modality related to this work can be found in [4]. The modularity of LP, i.e. its ability to realize other kinds of modal reasoning with proper changes in the axiomatization of proof polynomials, was shown with the development of the family of Justification Logics [7]. This ability is easily seen to be preserved here. Our work incorporates the rich type system and modularity of Justification Logic within the proofs-as-programs doctrine. For that reason, we obtain an extension of the Curry-Howard correspondence ([30], [17]) and adopt the judgmental approach of Intuitionistic Type Theory ([21], [22], [23], [25], [11]). Our system borrows from other modal calculi developed within the judgmental approach (e.g. [28], [19],[1] and especially [13] for the modal logic K). A main difference of our system with those systems, as well as with previous λ-calculi for LP ([2], [10]) is that our type system hosts a two-kinded typing relation for proof objects of corresponding formulae. It can be viewed as an attempt to add proof terms for validity judgments as presented in [28]. The resulting type system adopts dependent typing ([12], [26]) to relate the two kinds of proof objects with modality. The construction of the type universe as well as of justificational terms draws a lot from ideas in [8] and from [16]. Extending typed modal calculi with additional (contextual) terms of dependent typing can be also found in [24].

[6] See [20].

2 A road map for the type system

The present system can be viewed as a calculus of reasoning in three interleaving phases. Firstly, reasoning about proof objects in the implicational fragment of an intuitionistic theory T in absence of any metatheoretic assumptions of validity, introduced in Section 3. This calculus is formalized by the turnstile $\Gamma \vdash_{\mathsf{IPC}}$[7] where Γ contains assumptions on proofs of sentences in T. The underlying logic is intuitionistic, the system corresponding to the implicational fragment of simply typed lambda calculus. Secondly, reasoning with justifications, corresponding to reasoning about proof objects in some fixed foundational system: the (co-)theory T', introduced in Section 4. We suppose that T' provides the intended semantics for the intuitionistic system T. The corresponding turnstile is $\Delta \vdash_{\mathsf{J}}$. Abstracting from any specific metatheory, all that matters from a purely logical point of view is that the theory of the interpretation should – at least – include as much logic as the implicational fragment of T and it should satisfy some minimal conditions for the provability predicate of T. Finally, reasoning about existence of *links* between proof objects in the implicational fragment of *both* axiomatic systems, introduced in Section 6. This mode of reasoning is axiomatized within the full turnstile $\Delta; \Gamma \vdash_{\mathsf{JC}}$. The core of this system is the $\Box$-Introduction rule, which allows to express constructive reasoning on linking existence. The idea is – ignoring contextual reasoning for simplicity – that linking a construction in T with a justification of its corresponding type in T' we obtain a proof of a constructive (or, admissible in T) validity. The rule in full (i.e. including contexts) corresponds to the construction of a link for a compound term based on existing link on its subterms. The full turnstile $\Gamma; \Delta \vdash$ is, hence, a modal logic that "zips" mutual reasoning between the two calculi. Within this framework we obtain a computational reading for justification logic restricted to K modal reasoning. Before presenting this mutual reasoning at any arbitrary level of nesting (i.e. arbitrary modal types), we first introduce JCalc_1 which is a restriction of the type universe up to 1 level of $\Box$-nesting.

We fix a countable universe of propositions (P_i) that corresponds to sentences of T. The elements of this universe can be inhabited either by constructions or justifications. We will need, accordingly, two kinds of inhabitation relations for each proposition. We will be writing $M : \phi$ for a construction M of type ϕ in T. We will be writing $j : \mathsf{Just}\ \phi$ to express the fact that j is a justification (proof in T') of the proposition ϕ. When there is no confusion we will be abbreviating this by $j :: \phi$. A construction in $M : \phi$ in T does not entail its necessity: to this aim, a corresponding justification $j : \mathsf{Just}\ \phi$ from T' has to be obtained. Vice versa, the justification (j) of ϕ in T' alone entails its validity but not its admissibility in T (*constructive necessity*). This is expressed by the proposition – type $\Box^{\mathsf{j}}\phi$. A construction of $\Box^{\mathsf{j}}\phi$ can be obtained only when the (weaker) theory T actually "responds" with a construction M of the type ϕ to the valid fact ϕ known from T' by deducing j. Hence, once (and only if) we have $j :: \phi$ then $\Box^{\mathsf{j}}\phi$ can be regarded

[7] One could alternatively use an additional constant symbol null and write $\mathsf{null}; \Gamma \vdash_{\mathsf{IPC}}$ to denote reasoning purely in T and, thus, in absence of any metatheoretic environment.

as a well formed proposition. The stronger theory might be able to judge about $\Box^{\mathsf{j}}\phi$ (given $j :: \phi$) and prove e.g, $u :: \Box^{\mathsf{j}}\phi$. In that case T' "knows" that ϕ is admissible in T. In other words, when reasoning with justifications, the universe of types is *contextual*. To speak about an admissible (or, constructive) necessity of a proposition we require the existence of a corresponding proof object j in T' that establishes its validity.

3 Reasoning without foundational assumptions: IPC

Reasoning about the implicational fragment of the constructive theory (T), without formulating provability statements, is done within the implicational fragment of the simply typed lambda calculus. We start by giving the grammar for the metavariable ϕ used in the rules.

$$\phi := P_i | \phi \to \phi$$

The calculus is presented by introducing: the universe of types $\mathsf{Prop_0}$; rules for constructing well-formed contexts of simple propositional assumptions Γ_0; the rules governing $\vdash_{\mathsf{IPC}}$.

$$\frac{}{P_i \in \mathsf{Prop_0}}\ \textsc{Atom}_0 \qquad \frac{\phi_1 \in \mathsf{Prop_0} \qquad \phi_2 \in \mathsf{Prop_0}}{\phi_1 \to \phi_2 \in \mathsf{Prop_0}}\ \textsc{Impl}_0$$

$$\frac{}{\mathsf{nil} \vdash_{\mathsf{IPC}} \mathsf{wf}}\ \textsc{Nil}_0 \qquad \frac{\Gamma_0 \vdash_{\mathsf{IPC}} \mathsf{wf} \qquad \phi \in \mathsf{Prop_0}}{\Gamma_0, x : \phi \vdash_{\mathsf{IPC}} \mathsf{wf}}\ \Gamma_0\textsc{-Exp}$$

$$\frac{\Gamma_0 \vdash_{\mathsf{IPC}} \mathsf{wf} \qquad x : P_i \in \Gamma_0}{\Gamma_0 \vdash_{\mathsf{IPC}} x : P_i}\ \Gamma\textsc{-Refl}$$

$$\frac{\Gamma_0, x : \phi_1 \vdash_{\mathsf{IPC}} M : \phi_2}{\Gamma_0 \vdash_{\mathsf{IPC}} \lambda x : \phi_1.\ M : \phi_1 \to \phi_2}\ {\to}\mathrm{I} \qquad \frac{\Gamma_0 \vdash_{\mathsf{IPC}} M : \phi_1 \to \phi_2 \qquad \Gamma_0 \vdash_{\mathsf{IPC}} M' : \phi_1}{\Gamma_0 \vdash_{\mathsf{IPC}} (MM') : \phi_2}\ {\to}\mathrm{E}$$

4 Reasoning in the Presence of Foundations: A calculus of Justifications J

Reasoning in the presence of minimal foundations corresponds to reasoning on the existence of proof objects in the foundational theory T'. The minimal foundational assumptions from the logical point of view is that T' "knows" at least as much logic as T does. The more non-logical axioms in T, the more the specifications T' should satisfy (one needs stronger foundations to justify stronger theories). Abstracting

from any particular T and T', and assuming only that T incorporates minimal logic, the specifications about existence of proofs in T' are:

- to have "enough" types to provide – at least – an intended interpretation of every type ϕ of T to a unique type $\mathsf{Just}\ \phi$. In other words a subset of the types of T' should serve as interpretations of types in T;
- to have – at least – proof objects for all the instances of the axiomatic characterization of the IPC fragment described above; [8]
- to include some modus ponens rule which translates as: the existence of proof objects of types $\mathsf{Just}\ (\phi \to \psi)$ and of type $\mathsf{Just}\ \phi$ in T' should imply the existence of a proof object of the type $\mathsf{Just}\ \phi$.

4.1 Minimal Justification Logic J-Calc$_1$

Under these minimal requirements, we develop a minimal justification logic that is able to realize modal reasoning as reasoning on the existence of links between proofs of T and T'. We first realize modal reasoning restricted to formulae of degree (i.e. level of $\Box$-nesting) 1. Such a calculus will be used as a base to build a full modal calculus with justifications for formulae of arbitrary degree. Here is the grammar for the metavariables appearing below:

$$
\begin{aligned}
\phi &:= P_i \mid \Box^{\mathsf{j}}\phi \mid \phi_1 \to \phi_2 \\
j &:= s_i \mid C \mid j_1 * j_2 \\
t &:= x_i \mid \lambda x_i : \phi.t \mid Js :: \phi.t \\
C &:= \mathsf{K}[\phi_1, \phi_2] \mid \mathsf{S}[\phi_1, \phi_2, \phi_3] \mid C_1 * C_2 \\
\pi &:= \Pi s :: \phi_1.\ \phi_2 \mid \Pi s :: \phi_1.\ \pi \\
\mathsf{T} &:= \phi \mid \mathsf{Just}\ \phi \mid \pi \\
s &:= s_i \\
x &:= x_i
\end{aligned}
$$

4.1.1 Reasoning on minimal foundations J_0

Reasoning about such a minimal metatheory is axiomatized in its own turnstile ($\vdash_{J_0}$). [9] Henceforth, judgments on the justificational type universe of J_0 (corresponding to formualae in the (co-)theory T') together with wf predicate for Δ_0 contexts go as follows:

[8] If we extend our fragment we should extend our specifications accordingly but this can be easily done directly as in full justification logic. We choose to remain within this fragment for economy of presentation.

[9] This is the part of the calculus that corresponds directly to the algebra of justifications restricted to the applicative fragment.

$$\frac{}{\mathsf{nil} \vdash_{\mathsf{J}_0} \mathsf{wf}}\ \textsc{Nil} \qquad \frac{\Delta_0 \vdash_{\mathsf{J}_0} \mathsf{wf} \qquad \Delta_0 \vdash_{\mathsf{J}_0} \phi \in \mathsf{Prop_0}}{\Delta_0 \vdash_{\mathsf{J}_0} \mathsf{Just}\ \phi \in \mathsf{jtype_0}}\ \textsc{Simple}$$

$$\frac{\Delta_0 \vdash_{\mathsf{J}_0} \mathsf{Just}\ \phi \in \mathsf{jtype_0} \qquad s \notin \Delta_0}{\Delta_0, s :: \phi \vdash_{\mathsf{J}_0} \mathsf{wf}}\ \Delta_0\text{-App} \qquad \frac{\Delta_0 \vdash_{\mathsf{J}_0} \mathsf{wf} \qquad s :: \phi \in \Delta}{\Delta_0 \vdash_{\mathsf{J}_0} s :: \phi}\ \Delta_0\text{-Refl}$$

We add logical constants to satisfy the requirement that J_0 includes an axiomatic characterization of – at least – a fragment of IPC. Following justification logic, we define a signature of polymorphic constructors including K, S from combinatory logic. The values of those constructors are axiomatic constants that witness existence of proofs in T' of all instances of the corresponding logical validities. This axiomatic characterization of intuitionistic logic in J_0 together with rule scheme Times (*applicativity of justifications*) satisfy the minimal requirement for T' to reason logically.

$$\frac{\Delta_0 \vdash_{\mathsf{J}_0} \mathsf{Just}\ \ \phi_1 \to \phi_2 \to \phi_1 \in \mathsf{jtype_0}}{\Delta_0 \vdash_{\mathsf{J}_0} \mathsf{K}[\phi_1, \phi_2] :: \phi_1 \to \phi_2 \to \phi_1}\ \mathsf{K}$$

$$\frac{\Delta_0 \vdash_{\mathsf{J}_0} \mathsf{Just}\ \ (\phi_1 \to \phi_2 \to \phi_3) \to (\phi_1 \to \phi_2) \to (\phi_1 \to \phi_3) \in \mathsf{jtype_0}}{\Delta_0 \vdash_{\mathsf{J}_0} \mathsf{S}[\phi_1, \phi_2, \phi_3] :: (\phi_1 \to \phi_2 \to \phi_3) \to (\phi_1 \to \phi_2) \to (\phi_1 \to \phi_3)}\ \mathsf{S}$$

$$\frac{\Delta_0 \vdash_{\mathsf{J}_0} j_2 :: \phi_1 \to \phi_2 \qquad \Delta_0 \vdash_{\mathsf{J}_0} j_1 :: \phi_1}{\Delta_0 \vdash_{\mathsf{J}_0} j_2 * j_1 :: \phi_2}\ \mathsf{Times}$$

4.1.2 Zipping: $\mathsf{J\text{-}Calc}_1 = \mathsf{IPC} + \mathsf{J}_0 + \Box\text{−Intro}$

In this section we introduce $\mathsf{J\text{-}Calc}_1$ for reasoning on the existence of links i.e. constructions that witness the existence of proofs both in IPC (T) and J_0 (T'). By constructing a link we have a proof of a constructive necessity of a formula, showing that it is true and valid. Links have types of the form $\Box^{\mathsf{j}}\phi$ where j is a justification of the appropriate type. $\mathsf{J\text{-}Calc}_1$ realizes modal logic theoremhood in K up to degree 1 (i.e. formulae where its subformula includes up to 1 level of $\Box$).

We start by importing well-formedness judgments for contexts and justificational types (Δ_0Wf, JustWf respectively), and for the Prop_1 universe and its contexts:

$$\frac{\Delta_0 \vdash_{\mathsf{J}_0} \mathsf{wf}}{\Delta_0; \mathsf{nil} \vdash_{\mathsf{JC}_1} \mathsf{wf}}\ \Delta_0\text{WF} \qquad \frac{\Delta_0; \Gamma_1 \vdash_{\mathsf{JC}_1} \mathsf{wf} \qquad \Delta_0 \vdash_{\mathsf{J}_0} j :: \phi}{\Delta_0; \Gamma_1 \vdash_{\mathsf{JC}_1} j :: \phi}\ \textsc{Just}_0\textsc{Wf}$$

$$\frac{\phi \in \mathsf{Prop_0} \qquad \Delta_0; \Gamma_1 \vdash_{\mathsf{JC}_1} j :: \phi}{\Delta_0; \Gamma_1 \vdash_{\mathsf{JC}_1} \Box^j \phi \in \mathsf{Prop_1}}\ \textsc{Prop}_1\text{-}\textsc{Intro}$$

$$\frac{\Delta_0; \Gamma_1 \vdash_{\mathsf{JC}_1} \phi \in \{\mathsf{Prop_0}, \mathsf{Prop_1}\} \qquad x \notin \Gamma_1}{\Delta_0; \Gamma_1, x : \phi \vdash_{\mathsf{JC}_1} \mathsf{wf}}\ \Gamma_1\text{-}\textsc{App}$$

From justifications of formulas in $\mathsf{Prop_0}$, we can reason about their admissibility in T. Hence, Γ_1 might include assumptions from the sorts $\mathsf{Prop_0}$ and $\mathsf{Prop_1}$. For the inhabitation of $\mathsf{Prop_0}, \mathsf{Prop_1}$, we first accumulate intuitionistic reasoning extended to the new type universe ($\mathsf{Prop_1}$), adapting the rules from Section 3:

$$\frac{\Delta_0; \Gamma_1 \vdash_{\mathsf{JC}_1} \mathsf{wf} \qquad x : \phi \in \Gamma_1}{\Delta_0; \Gamma_1 \vdash_{\mathsf{JC}_1} x : \phi}\ \Gamma_1\text{-}\textsc{Refl} \qquad \frac{\Delta_0; \Gamma_1, x : \phi_1 \vdash_{\mathsf{JC}_1} M : \phi_2}{\Delta_0; \Gamma_1 \vdash_{\mathsf{JC}_1} \lambda x : \phi_1.\ M : \phi_1 \to \phi_2}\ {\to}\text{I}$$

$$\frac{\Delta_0; \Gamma_1 \vdash_{\mathsf{JC}_1} M : \phi_1 \to \phi_2 \qquad \Delta_0; \Gamma_1 \vdash_{\mathsf{JC}_1} M' : \phi_1}{\Delta_0; \Gamma \vdash_{\mathsf{JC}_1} (MM') : \phi_2}\ {\to}\text{E}$$

For relating the two calculi, a lifting rule is formulated for turning strictly $\mathsf{Prop_0}$ judgments to judgments on proof links ($\mathsf{Prop_1}$). In the rule, the $\lfloor$-operator ensures that context list $\lfloor\ \Gamma$ includes assumptions strictly in $\mathsf{Prop_0}$. The operator $\lfloor$ can be viewed as the opposite of *lift* operation applied on context lists erasing one level of boxed assumptions at the top level as described below.

$$\begin{aligned}
\lfloor\ \Gamma\ :=\ & \textbf{match}\ \Gamma\ \textbf{with} \\
& \mathsf{nil} \Rightarrow \mathsf{nil} \\
& \mid\ \Gamma', x'_i : \Box^j \phi_i \Rightarrow \lfloor\ \Gamma',\ x_i : \phi_i \\
& \mid \Gamma', _ \Rightarrow \lfloor\ \Gamma'
\end{aligned}$$

A corresponding iterative *let-binding* construct (let^*) is introduced simultaneously with the context lifting. The purpose of the iterative let binding is to extract the target(s) (T' terms) of existing links on subterms ($x_1 \ldots x_n$) of some composite term M in T and compose them to the target of the whole term M creating its residual. We show the operation of this construct in the example from section 5.

$$
\begin{array}{l}
\quad let^*\ \Gamma\ := \\
\textbf{match}\ \Gamma\ \textbf{with} \\
\qquad\qquad \mathsf{nil} \Rightarrow let\ () = () \\
\qquad\qquad \mid \Gamma', x_i' : \Box^j \phi_i \Rightarrow\ (let^*\ \Gamma')\ in\ let\ link(x_i, j_i) = x_i' \\
\qquad\qquad \mid \Gamma', \_ \Rightarrow let^*\ \Gamma'
\end{array}
$$

The $\Box$-Introduction rule goes as follows:

$$\frac{;\lfloor \Gamma_1 \vdash_{\mathsf{JC}_1} M : \phi \qquad \Delta_0; \Gamma_1 \vdash_{\mathsf{JC}_1} j :: \phi}{\Delta_0; \Gamma_1 \vdash_{\mathsf{JC}_1} (let^*\ \Gamma)\ in\ link\ (M, j) : \Box^j \phi}\ \Box\text{-Intro}$$

Finally, under empty Γ_1, we are permitting abstraction from a non-empty Δ_0. The resulting abstractions ($J-$terms), as we will see, are the inhabitants of modal types and correspond to linking processes. Their typing is, naturally, of Π-kind since the typing of a link is sensitive to its target code. We introduce Π-formation and inhabitation rules:

$$\frac{\Delta_0, s :: \phi_1; \vdash_{\mathsf{JC}_1} \phi_2 \in \{\mathsf{Prop_0}, \mathsf{Prop_1}\}}{\Delta_0; \vdash_{\mathsf{JC}_1} \Pi s :: \phi_1.\phi_2 \in \Pi}\ \Pi\ \text{TYPE}_0 \qquad \frac{\Delta_0, s :: \phi_1; \vdash_{\mathsf{JC}_1} \pi \in \Pi}{\Delta_0; \vdash_{\mathsf{JC}_1} \Pi s :: \phi_1.\pi \in \Pi}\ \Pi\ \text{TYPE}_1$$

$$\frac{\Delta_0, s :: \phi; \vdash_{\mathsf{JC}_1} t : \mathsf{T}}{\Delta_0; \vdash_{\mathsf{JC}_1} Js :: \phi.\ t : \Pi s :: \phi.\mathsf{T}}\ \Pi\text{-Intro}$$

$$\frac{\Delta_0; \vdash_{\mathsf{JC}_1} t : \Pi s :: \phi.\mathsf{T} \qquad \Delta_0; \vdash_{\mathsf{JC}_1} j :: \phi}{\Delta_0; \vdash_{\mathsf{JC}_1}\ (t\ j) : \mathsf{T}[\mathsf{s} := \mathsf{j}]}\ \Pi\text{-Elim}$$

5 Computational Motivation: A type system for separate compilation

In this section, we show how J-Calc can be viewed as a type system for program generation in typed languages that support separate compilation (modular programming or external function calls). These languages follow the client/ sever approach to programming: client code, can refer to code definitions implemented by the server elsewhere; the server can be some module or even another language providing the required function calls, but it needs not know the details of the implementation (*encapsulation*). A challenge in such a system is to provide a mechanism of separate compilation such that the client (or, source) code is compiled independently of changes in the implementation of the server. In what follows we present J-Calc as

a type system for linking processes in such a setting. Following our language: the constructs of T represent here client (or source) expressions and constructs of T' represent target (or server) code expressions. Our linking by way of the $\Box$-Intro rule linking processes generators that consume different implementations from the server and link them with constructs of the source. We show, following a textbook example for modules, that our type system provides the abstraction required for such a language so that client code needs to be compiled once and only, independently of the different implementations that the server module might provide.

5.1 Producing generic code

As an example, we will use ML-like module definitions. We start with a definition of a module's public signature (i.e. the operations provided by the server to the client). Here we provide the signature for a stack of integers.

```
module type INTSTACK =
            sig
            type intstack
            val Empty: intstack
            val push : int->intstack->intstack
            val pop: int->intstack->intstack
end;;
```

This signature can be implemented in various ways but our goal is to produce generic code from compiling source code only once. We take for example the source code expression $\vdash$ (`push 2 Empty`) : `intstack` and show step-by-step the construction of generic code following our calculus. First we factorize the usage of the signature by rewriting the term:

$$\lfloor \Gamma = \mathtt{x}_1 : \mathtt{int} \to \mathtt{intstack} \to \mathtt{intstack}, \mathtt{x}_2 : \mathtt{intstack} \vdash (\mathtt{x}_1\ 2\ \mathtt{x}_2) : \mathtt{intstack}$$

Secondly, we assume implementations of "missing" code in the validity context, i.e.

$$\Delta = \mathtt{s}_1 :: \mathtt{int} \to \mathtt{intstack} \to \mathtt{intstack}, \mathtt{s}_2 :: \mathtt{intstack} \vdash \mathtt{s1} * 2 * \mathtt{s}_2 :: \mathtt{intstack}$$

Using the $\Box$-Intro rule we obtain:

$$\Delta; \Gamma = \mathtt{x}'_1 : \Box^{\mathtt{s}_1}(\mathtt{int} \to \mathtt{intstack} \to \mathtt{intstack}), \mathtt{x}'_2 : \Box^{\mathtt{s}_2}\mathtt{intstack} \vdash$$
$$\mathtt{let}\ \mathtt{link}(\mathtt{x}_1, \mathtt{s}_1) = \mathtt{x}'_1\ \mathtt{in}$$
$$\mathtt{let}\ \mathtt{link}(\mathtt{x}_2, \mathtt{s}_2) = \mathtt{x}'_2\ \mathtt{in}$$
$$\mathtt{link}(\mathtt{x}_1\ 2\ \mathtt{x}_2, \mathtt{s}_1 * 2 * \mathtt{s}_2) : \Box^{\mathtt{s}_1 * 2 * \mathtt{s}_2}\mathtt{intstack}$$

Finally, abstracting we get a linking process generator of typing that is sensitive to the different implementations provided by the server:

$\vdash \mathtt{Js_1.Js_2.}\ \lambda \mathtt{x}'_1.\lambda \mathtt{x}'_2.\ \mathtt{let}^*\ \Gamma\ \mathtt{in}\ \mathtt{link}(\mathtt{x_1}\ 2\ \mathtt{x_2},\ \mathtt{s_1} * 2 * \mathtt{s_2})$

of type:

$\Pi \mathtt{s_1}.\Pi \mathtt{s_2}.\Box^{\mathtt{s_1}}(\mathtt{int} \to \mathtt{intstack} \to \mathtt{intstack}) \to \Box^{\mathtt{s_2}}\mathtt{intstack} \to \Box^{\mathtt{s_1}*2*\mathtt{s_2}}\mathtt{intstack}$

where

$\mathtt{let}^*\ \Gamma =^{\mathtt{def}} \mathtt{let}\ \mathtt{link}(\mathtt{x_1}, \mathtt{s_1}) = \mathtt{x}'_1\ \mathtt{in}\ \mathtt{let}\ \mathtt{link}(\mathtt{x_2},\ \mathtt{s_2}) = \mathtt{x}'_2$

5.2 Providing implementations

The server might provide different implementations of the *instack* module signature. The two textbook approaches use lists or arrays of integers. Given different implementations, the initial source code has different computational value since the links that it induces change. Schematically:

$\mathtt{push} \xrightarrow{\mathtt{link}} \mathtt{Cons} : \Box^{\mathtt{Cons}}(\mathtt{int} \to \mathtt{intstack} \to \mathtt{intstack})$
$\mathtt{Empty} \xrightarrow{\mathtt{link}} [] : \Box^{[]}\mathtt{intstack}$
$\mathtt{push\ 2\ Empty} \xrightarrow{\mathtt{link}} \mathtt{Cons\ 2\ []} : \Box^{\mathtt{Cons}*2*[]}\mathtt{intstack}$

$\mathtt{push} \xrightarrow{\mathtt{link}} \mathtt{Addarr} : \Box^{\mathtt{Addarr}}(\mathtt{int} \to \mathtt{intstack} \to \mathtt{intstack})$
$\mathtt{Empty} \xrightarrow{\mathtt{link}} \mathtt{create()} : \Box^{\mathtt{create()}}\mathtt{intstack}$
$\mathtt{push\ 2\ Empty} \xrightarrow{\mathtt{link}} \mathtt{Addarray\ 2\ create()} : \Box^{\mathtt{Addarr}*2*\mathtt{create()}}\mathtt{intstack}$

Both cases are captured by the generic code we produced giving us the ability of *separate compilation* of source and implementation. In the first case we have:

```
Just[intstack]=List,
Just[push]= Cons,
Just[Empty]=[]
```

From which we obtain using $\Box-$Intro the following:

$$\vdash \mathtt{link}(\mathtt{Empty}, []) : \Box^{[]}\mathtt{intstack}$$
$$\vdash \mathtt{link}(\mathtt{push}, \mathtt{Cons}) : \Box^{\mathtt{Cons}}(\mathtt{int} \to \mathtt{intstack} \to \mathtt{intstack})$$

Finally, using linking process generator obtained in the previous section and under standard operational semantics for application (β-reduction) and *let*-binding evaluation we link the source code to its residual program using the `list` implementation:

$\vdash \mathtt{link}(\mathtt{push\ 2\ Empty}, \mathtt{Cons}*2*[]) : \Box^{\mathtt{Cons}*2*[]}\mathtt{intstack}$

Analogously, using the exact same code generator, closing Δ with implementations

```
Just[intstack]= Array
Just[push]= addarr
Just[Empty]= create()
```

We obtain the links:

$\vdash \texttt{link}(\texttt{push}, \texttt{addarr}) : \Box^{\texttt{addarr}}(\texttt{int} \to \texttt{intstack} \to \texttt{intstack})$

$\vdash \texttt{link}(\texttt{Empty}, \texttt{create()}) : \Box^{\texttt{create()}}\texttt{intstack}$

And from the previous generic judgment under standard operational semantics for application (β-reduction) and *let*-binding evaluation we link the source code to its residual program using the `Array` implementation:

$\vdash \texttt{link}(\texttt{push 2 Empty}, \texttt{addarr * 2 * create()}) : \Box^{\texttt{addarr*2* create()}}\texttt{intstack}$

Note that the client code does not need to recompile. Our generic code construction provides the expressive means to evaluate source *contextually* given different implementations of the module signature.

6 The Full Calculus: J-Calc

$\mathsf{J\text{-}Calc}_1$ motivates the generalization to modal reasoning of arbitrary nesting: J-Calc. To allow such generalization, we need justifications of types of the form $\mathsf{Just}\ \Box^{\mathsf{j}}\phi$. Let us revise: If ϕ is a proposition (or, a sentence in the language of T), then $\mathsf{Just}\ \phi$ corresponds to the intended interpetation of ϕ in some (co-)theory T'. In $\mathsf{J\text{-}Calc}_1$ we could reason logically about the constructive admissibility of (valid according to T') facts of T. The existence of a link of a proof in T with an existing proof of the same type in T' would lead to constructions of a type of the form $\Box^{j}\phi$ with ϕ a simple type. To get modal theoremhood of degree 2 or more we have to assume that T' can express the existence of such links in itself. That is to say that T' can express the provability predicates both of T *and* of itself. Hence, supposing that $j :: \phi$, we can read a justification term of type $\mathsf{Just}\ \Box^{\mathsf{j}}\phi$ as a witness of a proof in T' of the fact $\exists x.Proof_T(x, \underline{\phi}) \wedge \exists x.Proof_{T'}(x,\ \underline{\mathsf{Just}\ \phi})$ expressed in T'. We will specify which of those types $\overline{T'}$ is expected to *capture* by introducing additional appropriate constants. Having this kind of justifications we can obtain $\mathsf{Prop_i}$ for any finite i as slices of a type universe in a mutual inductive construction. Schematically: $\mathsf{Prop_0} \Rightarrow \mathsf{Just\ Prop_0} \Rightarrow \mathsf{Prop_1} \Rightarrow \mathsf{Just\ Prop_1}$ and so on. This way we obtain full minimal justification logic. As different kinds of judgments are kept separated by the different typing relations, we do not need to provide distinct calculi as we did for $\mathsf{J\text{-}Calc}_1$ but we provide one "zipped" calculus directly. [10]

[10] In fact, adjoining Γ contexts when reasoning within justifications is pure weakening so we could have kept those judgments separated in a single–context $\vdash$ relation. We gain something though: we can squeeze two

6.1 Justificational (Validity) Judgments

The justificational type system has to include: judgments on the wellformedness of contexts (wf);[11] judgments on what T' can reason about (jtype) under the requirement that it is a metatheory of T; judgments on the construction of the justificational type universe (jtype) and minimal requirements about its inhabitation (i.e, *a minimal signature of logical constants*). The grammar of terms is the same as in section 4.1, the difference now is that the restrictions on the *Prop* universe are dropped.

We introduce progressively: formation rules for Prop; the formation rule for jtype; rules to build well-formed contexts of propositions and justifications (where we will be abbreviating using the following equational rule: $\mathsf{nil}, s_1 :: \phi_1, s_2 :: \phi_2, \ldots =^{def} s_1 :: \phi_1, s_2 :: \phi_2, \ldots$).

$$\frac{}{\mathsf{nil};\mathsf{nil} \vdash_{\mathsf{JC}} \mathsf{wf}}\ \textsc{Nil} \qquad \frac{\Delta;\Gamma \vdash_{\mathsf{JC}} \mathsf{wf}}{\Delta;\Gamma \vdash_{\mathsf{JC}} P_i \in \mathsf{Prop}}\ \textsc{Atom}$$

$$\frac{\Delta;\Gamma \vdash_{\mathsf{JC}} \phi_1 \in \mathsf{Prop} \qquad \Delta;\Gamma \vdash_{\mathsf{JC}} \phi_2 \in \mathsf{Prop}}{\Delta;\Gamma \vdash_{\mathsf{JC}} \phi_1 \to \phi_2 \in \mathsf{Prop}}\ \textsc{Impl} \qquad \frac{\Delta;\Gamma \vdash_{\mathsf{JC}} j :: \phi}{\Delta;\Gamma \vdash_{\mathsf{JC}} \Box^j \phi \in \mathsf{Prop}}\ \textsc{Box}$$

$$\frac{\Delta;\Gamma \vdash_{\mathsf{JC}} \phi \in \mathsf{Prop}}{\Delta;\Gamma \vdash_{\mathsf{JC}} \mathsf{Just}\ \phi \in \mathsf{jtype}}\ \textsc{Jtype} \qquad \frac{\Delta;\Gamma \vdash_{\mathsf{JC}} \mathsf{Just}\ \phi \in \mathsf{jtype} \qquad s \notin \Delta}{\Delta, s :: \phi;\Gamma \vdash_{\mathsf{JC}} \mathsf{wf}}\ \Delta\textsc{-App}$$

$$\frac{\Delta;\Gamma \vdash_{\mathsf{JC}} \phi \in \mathsf{Prop} \qquad x \notin \Gamma}{\Delta_0;\Gamma, x : \phi \vdash \mathsf{wf}}\ \Gamma\textsc{-App}$$

6.1.1 Prop *Inhabitation*

Here is the first part of logical propositional reasoning of the system.

$$\frac{\Delta;\Gamma \vdash_{\mathsf{JC}} \mathsf{wf} \qquad x : \phi \in \Gamma}{\Delta;\Gamma \vdash_{\mathsf{JC}} x : \phi}\ \Gamma\textsc{-Refl}$$

$$\frac{\Delta;\Gamma, x : \phi_1 \vdash_{\mathsf{JC}} M : \phi_2}{\Delta;\Gamma \vdash_{\mathsf{JC}} \lambda x : \phi_1.\ M : \phi_1 \to \phi_2}\ \to\!\textsc{I}$$

$$\frac{\Delta;\Gamma \vdash_{\mathsf{JC}} M : \phi_1 \to \phi_2 \qquad \Delta;\Gamma \vdash_{\mathsf{JC}} M' : \phi_1}{\Delta;\Gamma \vdash_{\mathsf{JC}} (MM') : \phi_2}\ \to\!\textsc{E}$$

premises ($\Delta \vdash j :: \phi$, $\Delta;\Gamma \vdash \mathsf{wf}$) to a single one ($\Delta;\Gamma \vdash j :: \phi$).

[11] Analogous treatments of judgments on the validity of contexts can be found e.g. in [26].

6.1.2 jtype *Inhabitation*

Now we move to the core of the system. In the judgments below we provide the constructions of canonical elements of justificational types (jtype). The judgments reflect the minimal requirements for T' to be a metatheory of some T as presented in Section 4.1.1 together with specifications on internalizing proof links reasoning in itself. More specifically, we demand that T' can *capture* reasoning on links (between proof objects of T and itself) *within* itself and also, internalize modus ponens of T. To capture these provability conditions we add the constant constructors ! (*bang*) and Kappa. Although introduction of links is axiomatized in the next section, the judgments concerning the ! and Kappa constructors should be viewed in conjunction with $\Box - Intro$. They witness the fact that T' internalizes modus ponens (of T) and linking existence (again of T).

$$\frac{\Delta;\Gamma \vdash_{\mathsf{JC}} \mathsf{Just}\ \ \phi_1 \to \phi_2 \to \phi_1 \in \mathsf{jtype}}{\Delta;\Gamma \vdash_{\mathsf{JC}} \mathsf{K}[\phi_1,\phi_2] :: \phi_1 \to \phi_2 \to \phi_1}\ \text{K}$$

$$\frac{\Delta;\Gamma \vdash_{\mathsf{JC}} \mathsf{Just}\ \ (\phi_1 \to \phi_2 \to \phi_3) \to (\phi_1 \to \phi_2) \to (\phi_1 \to \phi_3) \in \mathsf{jtype}}{\Delta;\Gamma \vdash_{\mathsf{JC}} \mathsf{S}[\phi_1,\phi_2,\phi_3] :: (\phi_1 \to \phi_2 \to \phi_3) \to (\phi_1 \to \phi_2) \to (\phi_1 \to \phi_3)}\ \text{S}$$

$$\frac{\Delta;\Gamma \vdash_{\mathsf{JC}} j_2 :: \phi_1 \to \phi_2 \qquad \Delta;\Gamma \vdash_{\mathsf{JC}} j_1 :: \phi_1}{\Delta \vdash_{\mathsf{J}} j_2 * j_1 :: \phi_2}\ \textsc{Times} \qquad \frac{\Delta;\mathsf{nil} \vdash_{\mathsf{JC}} M : \Box^{\mathsf{C}}\phi}{\Delta;\Gamma \vdash_{\mathsf{JC}} !C :: \Box^{\mathsf{C}}\phi}\ \textsc{Bang}$$

$$\frac{\Delta;\Gamma \vdash_{\mathsf{JC}} \mathsf{Just}\ \Box^{\mathsf{j}'}\phi_1 \in \mathsf{jtype} \qquad \Delta;\Gamma \vdash_{\mathsf{JC}} \mathsf{Just}\ \ \Box^{\mathsf{j}}(\phi_1 \to \phi_2) \in \mathsf{jtype}}{\Delta;\Gamma \vdash_{\mathsf{JC}} \mathsf{Kappa}[j,j',\phi_1,\phi_2] :: \Box^{\mathsf{j}}(\phi_1 \to \phi_2) \to \Box^{\mathsf{j}'}\phi_1 \to \Box^{\mathsf{j}*\mathsf{j}'}\phi_2}\ \textsc{Kappa}$$

6.2 Proof Links

Our next task is to formulate the main rule for the K modality as a lifting rule for going from reasoning about constructions to reasoning about admissibility of validities via proof linking. To reflect the modal axiom K in Natural Deduction we have to obtain a rule that reflects the following provability principle:

$$\frac{\phi_1\ \mathsf{true},\ldots,\phi_n\ \mathsf{true} \vdash \phi\ \mathsf{true} \qquad \phi_1\ \mathsf{valid},\ldots,\phi_n\ \mathsf{valid} \vdash \phi\ \mathsf{valid}}{\Box\phi_1\ \mathsf{true},\ldots,\Box\phi_n\ \mathsf{true},\ldots \vdash \Box\phi\ \mathsf{true}}\ \Box\text{-}\textsc{Intro}$$

We proceed with giving inhabitants analogously to what was explained in Section

4.1.2:[12]

$$\frac{\Delta;\lfloor\Gamma\vdash_{\mathsf{JC}} M:\phi \qquad \Delta;\Gamma\vdash_{\mathsf{JC}} j::\phi}{\Delta;\Gamma\vdash_{\mathsf{JC}} (let^*\ \Gamma)\ in\ link\ (M,j):\Box^j\phi}\ \Box\text{-Intro}$$

Finally, abstraction from Δ contexts over empty Γ contexts applies in the extended type universe:

$$\frac{\Delta,s::\phi;\vdash_{\mathsf{JC}} t:\mathsf{T}}{\Delta;\vdash_{\mathsf{JC}} Js::\phi.\ t:\Pi s::\phi.\mathsf{T}}\ \Pi\text{-Intro} \qquad \frac{\Delta;\vdash_{\mathsf{JC}} t:\Pi s::\phi.\mathsf{T} \qquad \Delta_0;\vdash_{\mathsf{JC}} j::\phi}{\Delta;\vdash_{\mathsf{JC}}\ (t\ j):\mathsf{T}[\mathsf{s}:=\mathsf{j}]}\ \Pi\text{-Elim}$$

7 Further Results and Conclusions

Standard meta-theoretical results can be proven for J-Calc. We just mention here that the iterative *let* operator satisfies standard commutativity with the substitution rule for justifications and that structural rules can be proven. We will be skipping the index in $\vdash_{\mathsf{JC}}$.

Theorem 7.1 (Weakening) J-Calc *satisfies Weakening in both modes of reasoning:*

(i) *If* $\Delta;\mathsf{nil}\vdash j::\phi$, *and* $\Delta;\Gamma\vdash\mathsf{wf}$ *then,* $\Delta;\Gamma\vdash j::\phi$.

(ii) *If* $\Delta;\Gamma\vdash j::\phi$, *then* $\Delta,s::\phi';\Gamma\vdash j::\phi$, *with* s *fresh.*

(iii) *If* $\Delta;\Gamma\vdash M:\phi$, *then* $\Delta;\Gamma,x{:}\phi'\vdash M:\phi$, *with* x *fresh.*

Proof. For all items by structural induction on the derivation trees of the two kinds of constructions. The proof of the first is vacuous since Γ contexts are irrelevant in justification formation. As a result, its inverse can also be shown. □

Theorem 7.2 (Contraction) J-Calc *satisfies Contraction:*

(i) *If* $\Delta,s::\phi,t::\phi;\mathsf{nil}\vdash j::\phi'$, *then* $\Delta,u::\phi;\mathsf{nil}\vdash j[s\equiv t/u]::\phi'$.

(ii) *If* $\Delta,s::\phi,t::\phi;\Gamma\vdash\mathsf{wf}$, *then,* $\Delta,u::\phi;\Gamma[s\equiv t/u]\vdash\mathsf{wf}$.

(iii) *If* $\Delta,s::\phi,t::\phi;\Gamma\vdash M:\phi'$, *then,* $\Delta,u::\phi;\Gamma[s\equiv t/u]\vdash M[s\equiv t/u]:\phi'[s\equiv t/u]$.

(iv) *If* $\Delta;\Gamma,x:\phi,y:\phi\vdash M:\phi'$, *then* $\Delta;\Gamma,z:\phi\vdash M[x\equiv y/z]:\phi'$.

Proof. First item by structural induction on the derivation trees of justifications (validity judgments). Note, as mentioned in the previous theorem, that it can be

[12] We prefer this to the mouthful but equivalent:

$$\frac{\begin{array}{c}\Delta;x_1:\phi_1,\ldots,x_i:\phi_i\ \ as\ \Gamma\vdash M:\phi\\ \forall\phi_i\in\Gamma.\ \Delta';\mathsf{nil}\vdash j_i::\phi_i \qquad \Delta';\mathsf{nil}\vdash j::\phi \qquad \Delta';x_1:\Box^{j_1}\phi_i,\ldots,\Box^{j_i}\phi_i\vdash\mathsf{wf}\end{array}}{\Delta';x_1:\Box^{j_1}\phi_i,\ldots,\Box^{j_i}\phi_i\vdash JBox\ j:\Box^{\mathsf{j}}\phi}\ \Box\text{-Intro}$$

shown for arbitrary Γ. For the second, nested induction on the structure of context Γ (treated as list) and the complexity of formulas. Vacuously in the nil case. For the non-empty case: case analysis on the complexity of the head formula using the inductive hypothesis on the tail. Cases of interest are with $\Box^{\mathsf{s}}\phi$ or $\Box^{\mathsf{t}}\phi$ as subformulae. Use the previous item and judgments for wf contexts. For the third and the fourth, again by structural induction on the derivation. □

In a similar fashion we can show the more general:

Theorem 7.3 (Preservations of Types under Substitution) J-Calc *preserves types under substitution and simultaneous substitution:*

(i) *If* $\Delta;\Gamma, x:\phi \vdash t:\mathsf{T}$, *and* $\Delta;\Gamma \vdash M:\phi$ *then* $\Delta;\Gamma \vdash t[x/M]:\mathsf{T}$

(ii) *If* $\Delta, s::\phi, \Delta';\Gamma \vdash t:\mathsf{T}$, *and* $\Delta;\vdash j::\phi$ *then* $\Delta, \Delta'[s/j];\Gamma[s/j] \vdash t[s/j]:\mathsf{T}[s/j]$

We additionally mention that the calculus satisfies *permutation* for both contexts Δ and Γ with the restriction that the permutations in Δ should not break the chain of dependencies. Lastly, we mention here that under standard *let*-binding evaluation and application as β-reduction within a dependently typed framework, a small step operational semantics has been developed and progress and preservation can be shown.

For future work, we plan to extend the computational relevance of the full calculus (JCalc) by establishing its connection with higher-order module systems (e.g. where module signatures can refer to other module signatures which, in turn, are implemented by a third module). Linking processes in such systems would utilize our type system in full. Cut-elimination results are currently under development.

References

[1] Zine El abidine Benaissa, Eugenio Moggi, Walid Taha, and Tim Sheard. Logical modalities and multi-stage programming.

[2] Jesse Alt and Sergei Artemov. Reflective λ-calculus. Technical Report CFIS 2000-06, Cornell University, 2000.

[3] Sergei N. Artemov. Logic of proofs. *Annals of Pure and Applied Logic*, 67(1–3):29–59, May 1994.

[4] Sergei N. Artemov. Operational modal logic. Technical Report MSI 95–29, Cornell University, December 1995.

[5] Sergei N. Artemov. Explicit provability and constructive semantics. *Bulletin of Symbolic Logic*, 7(1):1–36, March 2001.

[6] Sergei N. Artemov. Unified semantics for modality and λ-terms via proof polynomials. In Kees Vermeulen and Ann Copestake, editors, *Algebras, Diagrams and Decisions in Language, Logic and Computation*, volume 144 of *CSLI Lecture Notes*, pages 89–118. CSLI Publications, Stanford, 2002.

[7] Sergei N. Artemov. Justification logic. In *JELIA*, pages 1–4, 2008.

[8] Sergei N. Artemov. The ontology of justifications in the logical setting. *Stud. Log.*, 100(1-2):17–30, April 2012.

[9] Sergei N. Artemov and Lev D. Beklemishev. Provability logic. In D. M. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, 2nd Edition*, volume 13, pages 189–360. Springer, 2005.

[10] Sergei N. Artemov and Eduardo Bonelli. The intensional lambda calculus. In Sergei N. Artemov and Anil Nerode, editors, *Logical Foundations of Computer Science, International Symposium, LFCS 2007, New York, NY, USA, June 4–7, 2007, Proceedings*, volume 4514 of *Lecture Notes in Computer Science*, pages 12–25. Springer, 2007.

[11] Steven Awodey and Florian Rabe. Kripke Semantics for Martin-Löf's Extensional Type Theory. In *TLCA'09*, pages 249–263, 2009.

[12] Gilles Barthe and Thierry Coquand. An Introduction to Dependent Type Theory. *Lecture Notes in Computer Science*, pages 1–41.

[13] Gianluigi Bellin, Valeria de Paiva, and Eike Ritter. Extended curry-howard correspondence for a basic constructive modal logic. In *Proceedings of Methods for Modalities*, 2001.

[14] George S. Boolos. *The Logic of Provability.*

[15] Samuel R. Buss. Chapter ii first-order proof theory of arithmetic. In Samuel R. Buss, editor, *Handbook of Proof Theory*, volume 137 of *Studies in Logic and the Foundations of Mathematics*, pages 79 – 147. Elsevier, 1998.

[16] Melvin Fitting. The logic of proofs, semantically. *Annals of Pure and Applied Logic*, vol. 132(1), pp. 1-25, 2005.

[17] Jean Y. Girard, Paul Taylor, and Yves Lafont. *Proofs and types.* Cambridge University Press, New York, NY, USA, 1989.

[18] Kurt Gödel. *On Formally Undecidable Propositions of Principia Mathematica and Related Systems.* Dover Publications, April 1992.

[19] Jean Goubault-Larrecq. On computational interpretations of the modal logic S4. Technical report, Institut fur Logik, Komplexität und Deduktionssysteme, Universitat, 1996.

[20] Robert Harper. *Programming in Standard ML.* 1998.

[21] Per Martin-Löf. Constructive Mathematics and Computer Programming. In *6-th International Congress for Logic, Methodology and Philosophy of Science, 1979*, pages 153–175. North–Holland, 1982.

[22] Per Martin-Löf. *Intuitionistic Type Theory.* Bibliopolis, Naples, 1984.

[23] Per Martin-Löf. On the Meanings of the Logical Constants and the Justifications of the Logical Laws. *Nordic Journal of Philosophical Logic*, 1(1):11–60, May 1996.

[24] Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. Contextual modal type theory. *ACM Trans. Comput. Logic*, 9(3):23:1–23:49, June 2008.

[25] Bengt Nordstrom, Kent Petersson, and Jan M. Smith. *Programming in Martin-Löf's Type Theory: An Introduction.* Oxford University Press, USA, July 1990.

[26] Ulf Norell. Dependently typed programming in Agda. In *Lecture Notes from the Summer School in Advanced Functional Programming*, 2008.

[27] Rohit Parikh. Existence and feasibility in arithmetic. *J. Symb. Log.*, pages 494–508, 1971.

[28] Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical. Structures in Comp. Sci.*, 11(04):511–540, August 2001.

[29] Peter Smith. An introduction to Gödel's theorems, Cambridge University Press, 2007.

[30] Morten Heine B. Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard Isomorphism*, volume 149 of *Studies in Logic and the Foundations of Mathematics.* Elsevier, 2006.