The stack calculus

Alberto Carraro
DAIS, Università Ca' Foscari Venezia, Italia
acarraro@dsi.unive.it

Thomas Ehrhard
PPS, Université Denis Diderot Paris, France
thomas.ehrhard@pps.univ-paris-diderot.fr

Antonino Salibra
DAIS, Università Ca' Foscari Venezia, Italia
salibra@dsi.unive.it

We introduce a functional calculus with simple syntax and operational semantics in which the calculi introduced so far in the Curry–Howard correspondence for Classical Logic can be faithfully encoded. Our calculus enjoys confluence without any restriction. Its type system enforces strong normalization of expressions and it is a sound and complete system for full implicational Classical Logic. We give a very simple denotational semantics which allows easy calculations of the interpretation of expressions.

1 Introduction

The Curry–Howard correspondence [16] was first designed as the isomorphism between natural deduction for minimal Intuitionistic Logic [28] and the simply typed λ-calculus, and for a long time no one thought this isomorphism could be extended to Classical Logic, until Griffin [14] proposed that natural deduction for Classical Logic could be viewed as a type system for a λ-calculus extended with a control operator $\mathscr{C}$, introduced by Felleisen in his $\lambda\mathscr{C}$-calculus [10]. There are also other operators that correspond to logical axioms that, once added to minimal Intuitionistic Logic, give proof systems of different power, from minimal to full implicational Classical Logic. Felleisen's $\mathscr{C}$, corresponding to the *Double-Negation Elimination law*, gives full implicational Classical Logic; less powerful operators are $\mathscr{K}$ (a.k.a. $\mathsf{call/cc}$), typable with *Peirce's law*, and $\mathscr{A}$ (a.k.a. *abort*) typable with the *Ex-Falso Quodlibet law*. On the programming side, this classification corresponds to the different expressive power of the operators as control primitives. Ariola and Herbelin [1] survey and classify these logical systems and introduce a refinement of $\lambda\mathscr{C}$-calculus which aims at resolving a mismatch between the operational and proof-theoretical interpretation of Felleisen's $\lambda\mathscr{C}$-reduction theory.

Another extension of the λ-calculus is Parigot's $\lambda\mu$-calculus [27] which introduces a Natural Deduction with multiple conclusions. This system implements minimal Classical Logic and it is able to encode the primitive $\mathsf{call/cc}$; Ariola and Herbelin [1] extend it to cover full Classical Logic and compare their system with Felleisen's $\lambda\mathscr{C}$-calculus: similar studies are made by De Groote [7]. The correspondence between classical principles and functional control operators is further stressed by De Groote's extension of λ-calculus with *raise/handle* primitives [8]. While the untyped version of $\lambda\mu$-calculus enjoys confluence, its extensional version is only confluent on closed terms via the addition of a rewrite rule that destroys the strong normalization of typable terms [6].

Gentzen's sequent calculus LK [11] is put in correspondence with a reduction system by Urban [32]; the type system of Curien–Herbelin's $\bar{\lambda}\mu\tilde{\mu}$-calculus [3] corresponds to its implicational fragment. These two approaches are compared in detail by Lengrand [24]. These calculi highlight the duality between call-by-value and call-by-name cut-elimination (or evaluation): confluence is not achievable without

Submitted to:
LSFA 2012

choosing one of the two strategies. Other computational interpretations of Classical sequent calculus are Girard's LC [13] and the translations of Classical Logic in Linear Logic [5], based upon linear dual decomposition of classical implication.

In this paper we introduce the *stack calculus*. The idea of this calculus comes from a synthesis of Krivine's extension of the λ-calculus with *stacks* and $\mathsf{call/cc}$ [18] with Parigot's $\lambda\mu$-calculus. It also bears similarities with the call-by-name variant of $\bar{\lambda}\mu\tilde{\mu}$-calculus. In Krivine's Classical Realizability [18] classical implication is associated to a stack constructor, while in $\lambda\mu$-calculus (as in $\lambda\mathscr{C}$-calculus) the arrow-type is introduced by an intuitionistic λ-abstraction: the role of the μ-abstraction is to make it classical by "merging together" many intuitionistic arrows. The μ-abstraction can then be thought of as a functional abstraction over *lists of inputs*, corresponding to a list of consecutive λ-abstractions. This idea is used in the design of Löw–Streicher's CPS_∞-calculus [25] which is an infinitary version of λ-calculus that allows only infinite abstractions and infinite applications.

The stack calculus is a finitary functional language in which stacks are first-class entities, and many of the previously-mentioned calculi can be faithfully translated. The stack calculus enjoys confluence without any restriction, also in its extensional version. We type the stack calculus with a propositional language with implication and falsity, to be associated to stack construction and empty stack, respectively. As a consequence one obtains a sound and complete system for full implicational Classical Logic. In our case the realizability interpretation of types à la Krivine matches perfectly the logical meaning of the arrow in the type system: proofs of soundness and strong normalization of the calculus are both given by particular realizability interpretations. The simplicity of the stack calculus, which does not use at the same time λ- and μ-abstractions allows an easy encoding of control primitives like $\mathsf{call/cc}$, $\mathsf{label/resume}$, $\mathsf{raise/catch}$.

Many researchers contributed to the study of proof semantics of Classical Logic. From Girard [13], to Reus and Streicher [29], to Selinger [30] who gives a general presentation in terms of *control categories*. It is also very interesting the work by Laurent and Regnier [23] which shows in detail how to extract a control category out of a categorical model of Multiplicative Additive Linear Logic (MALL).

Inspired by Laurent and Regnier's work [23] we give a minimal framework in which the stack calculus can be soundly interpreted. The absence of the λ-abstraction, allows us to focus on the minimal structure required to interpret Laurent's Polarized Linear Logic [21] and to use it to interpret the stack calculus. The simplicity of the framework gives an easy calculation of the semantics of expressions.

2 The untyped stack calculus

The stack calculus has three syntactic categories: *terms* that are in functional position, *stacks* that are in argument position and represent streams of arguments, *processes* that are terms applied to stacks. The basis for the definition of the stack calculus language is a countably infinite set of *stack variables*, ranged over by the initial small letters $\alpha, \beta, \gamma, \ldots$ of the greek alphabet. The language is then given by the following grammar:

$$\begin{array}{lcll} \pi, \varpi & ::= & \alpha \mid \mathsf{nil} \mid M \cdot \pi \mid \mathsf{cdr}(\pi) & \text{stacks} \\ M, N & ::= & \mu\alpha.P \mid \mathsf{car}(\pi) & \text{terms} \\ P, Q & ::= & M \star \pi & \text{processes} \end{array}$$

We use letters E, E' to range over *expressions* which are either stacks, terms or processes. We denote by Σ^{p}, Σ^{s}, Σ^{t}, and Σ^{e} the sets of all processes, stacks, terms, and expressions respectively. The operator μ is

a binder. An occurrence of a variable α in an expression E is *bound* if it is under the scope of a $\mu\alpha$; the set $\mathrm{FV}(E)$ of *free variables* is made of those variables having a non-bound occurrence in E.
Stacks represent lists of terms: nil is the empty stack. A stack $M_1 \cdot \cdots \cdot M_k \cdot \mathsf{nil}$, stands for a finite list while a stack $M_1 \cdot \cdots \cdot M_k \cdot \alpha$ stands for a non-terminated list that can be further extended.
Terms are entities that wait for a stack to compute. A term $\mu\alpha.P$ is the μ-*abstraction* of α in P.
Processes result from the *application* $M \star \pi$ of a term M to a stack π. This application, unlike in λ-calculus, has to be thought as *exhaustive* and gives rise to an evolving entity that does not have any outcome.

Application has precedence over μ-abstraction and the stack constructor has precedence over application, so that the term $\mu\alpha.M \star N \cdot \pi$ unambiguously abbreviates $\mu\alpha.(M \star (N \cdot \pi))$. As usual, the calculus involves a substitution operator. By $E\{\pi/\alpha\}$ we denote the (capture-avoiding) substitution of the stack π for all free occurrences of α in E. The symbol '$\equiv$' stands for syntactic equality, while ':=' stands for definitional equality.

Lemma 1 (Substitution Lemma)**.** *For $E \in \Sigma^{\mathsf{e}}$, $\pi, \varpi \in \Sigma^{\mathsf{s}}$, $\alpha \notin \mathrm{FV}(\varpi)$ and $\alpha \not\equiv \beta$ we have* $E\{\pi/\alpha\}\{\varpi/\beta\} \equiv E\{\varpi/\beta\}\{\pi\{\varpi/\beta\}/\alpha\}$.

Definition 2. *The reduction rules of the stack calculus are the following ones:*

$$\begin{array}{ll} (\mu) & (\mu\alpha.P) \star \pi \to_{\mu} P\{\pi/\alpha\} \\ (\mathsf{car}) & \mathsf{car}(M \cdot \pi) \to_{\mathsf{car}} M \\ (\mathsf{cdr}) & \mathsf{cdr}(M \cdot \pi) \to_{\mathsf{cdr}} \pi \end{array}$$

Adding the following rules we obtain the extensional *stack calculus:*

$$\begin{array}{lll} (\eta_1) & \mu\alpha.M \star \alpha \to_{\eta_1} M & \text{if } \alpha \notin \mathrm{FV}(M) \\ (\eta_2) & \mathsf{car}(\pi) \cdot \mathsf{cdr}(\pi) \to_{\eta_2} \pi & \end{array}$$

We simply write $\to_{\mathsf{s}}$ for the contextual closure of the relation $(\to_{\mu} \cup \to_{\mathsf{car}} \cup \to_{\mathsf{cdr}})$. Moreover we write $\to_{\eta}$ for the contextual closure of the relation $(\to_{\eta_1} \cup \to_{\eta_2})$ and finally we set $\to_{\mathsf{s}\eta} = (\to_{\mathsf{s}} \cup \to_{\eta})$. For example, if $\mathbf{I} := \mu\alpha.\mathsf{car}(\alpha) \star \mathsf{cdr}(\alpha)$, then $\mathbf{I} \star \mathbf{I} \cdot \mathsf{nil} \to_{\mathsf{s}} \mathbf{I} \star \mathsf{nil} \to_{\mathsf{s}} \mathsf{car}(\mathsf{nil}) \star \mathsf{cdr}(\mathsf{nil})$ and the reduction does not proceed further. If $\omega := \mu\alpha.\mathsf{car}(\alpha) \star \alpha$, then $\omega \star \omega \cdot \mathsf{nil} \to_{\mathsf{s}} \omega \star \omega \cdot \mathsf{nil}$; this is an example of a non-normalizing process. The stack calculus enjoys confluence, even in its extensional version, as the following theorems state.

Theorem 3. *The $\to_{\mathsf{s}}$-reduction is Church-Rosser.*

Theorem 4. *The $\to_{\mathsf{s}\eta}$-reduction is Church-Rosser.*

We observe that Theorem 4 holds despite the non left-linearity of the reduction rules of the extensional stack calculus. In other calculi, like the λ-calculus with surjective pairing, the interaction of the extensionality rule with the projection rules breaks the Church-Rosser property for the calculus [17].

2.1 Translation of lambda-mu-calculus

Many calculi have been introduced so far to extend the Curry–Howard correspondence to classical logic [14, 27, 8, 32, 3]. Since we cannot attempt to report a comparison with the stack calculus for each one of them, so we choose probably the best known, i.e. Parigot's $\lambda\mu$-calculus. In this section we show how $\lambda\mu$-calculus can be faithfully encoded into the stack calculus (in the precise sense of the forthcoming Theorem 6).

The basis for the definition of the $\lambda\mu$-calculus language are two (disjoint) sets $\lambda\mathrm{Var}$ and $\mu\mathrm{Var}$ of λ-variables and μ-variables (a.k.a. *names*), respectively. The names, ranged over by $\alpha,\beta,\gamma,\ldots$, are are taken from $\mu\mathrm{Var}$ and the usual variables, taken in $\lambda\mathrm{Var}$, are ranged over by $x,y,z,\ldots$. The expressions belonging to the language of $\lambda\mu$-calculus are often divided into two categories, *terms* and *named terms*, produced by the following grammar:

$$\begin{array}{lcll} s,t & ::= & x \mid \lambda x.t \mid st \mid \mu\alpha.p & \text{terms} \\ p,q & ::= & [\alpha]t & \text{named terms} \end{array}$$

We use letters e,e' to range over *expressions* which are either terms or named terms. We denote by Λ^{t}, Λ^{p}, and Λ^{e} the sets of all terms, named terms and expressions, respectively.

We briefly recall the operational semantics of $\lambda\mu$-calculus. In addition to the usual capture-free substitution $e\{t/x\}$ of a term t for a variable x in e, $\lambda\mu$-calculus uses the *renaming* $e\{\beta/\alpha\}$ of α with β in e and the *structural substitution* $e\{s/^*\alpha\}$ that replaces all named subterms $[\alpha]t$ of e with the named term $[\alpha]ts$: for example $(\lambda y.\mu\beta.[\alpha]z)\{\lambda x.x/^*\alpha\} \equiv \lambda y.\mu\beta.[\alpha]z(\lambda x.x)$ (see [27]). Note that we adopt here the notations of David and Py [6] instead of Parigot's original ones. The reduction relation characterizing the $\lambda\mu$-calculus is given by the contextual closure of the following rewrite rules:

$$\begin{array}{llll} (\beta) & (\lambda x.t)s \to_\beta t\{s/x\} & \text{logical reduction} & \\ (\mu) & (\mu\alpha.p)s \to_\mu \mu\alpha.p\{s/^*\alpha\} & \text{structural reduction} & \\ (\rho) & [\beta](\mu\alpha.p) \to_\rho p\{\beta/\alpha\} & \text{renaming} & \\ (\theta) & \mu\alpha.[\alpha]t \to_\theta t & \text{if } \alpha \notin \mathrm{FN}(t) & \end{array}$$

The reduction $\to_{\beta\mu\rho\theta}$ was proved to enjoy the Church-Rosser property by Parigot [27]. The extensional $\lambda\mu$-calculus is obtained by adding the contextual closure of the following reduction rules:

$$\begin{array}{lll} (\eta) & \lambda x.tx \to_\eta t & \text{if } x \notin \mathrm{FV}(t) \\ (\nu) & \mu\alpha.p \to_\nu \lambda x.\mu\alpha.p\{x/^*\alpha\} & \text{if } x \notin \mathrm{FV}(p) \end{array}$$

We are now going to translate $\lambda\mu$-expressions into expressions of the stack calculus (stack-expressions, for short). A minor technical detail for the translation is the need of regarding all λ-variables and all names as stack variables.

Definition 5. *Define a mapping* $(\cdot)^\circ : \Lambda^{\mathsf{e}} \to \Sigma^{\mathsf{e}}$ *by induction as follows:*

$$\begin{array}{ll} x^\circ = \mu\beta.\mathsf{car}(x)\star\beta & \\ (\lambda x.t)^\circ = \mu x.t^\circ \star \mathsf{cdr}(x) & \\ (ts)^\circ = \mu\beta.t^\circ \star s^\circ \centerdot \beta & \beta \notin \mathrm{FV}(t^\circ)\cup\mathrm{FV}(s^\circ) \\ ([\alpha]t)^\circ = t^\circ \star \alpha & \\ (\mu\alpha.p)^\circ = \mu\alpha.p^\circ & \end{array}$$

The translation of Definition 5 preserves the convertibility of expressions and in this sense provides an embedding of $\lambda\mu$-calculus into the stack calculus.

Theorem 6. *Let* $e,e' \in \Lambda^{\mathsf{e}}$.

(i) If $e \to_{\beta\mu\rho\theta} e'$*, then* e° *and* $(e')^\circ$ *have a common reduct in the stack calculus.*

(ii) If $e \to_{\beta\mu\rho\theta\eta\nu} e'$*, then* e° *and* $(e')^\circ$ *have a common reduct in the extensional stack calculus.*

Note that the extensional $\lambda\mu$-calculus does not enjoy a full Church-Rosser theorem, as witnessed by the following counterexample [6]: $[\gamma]y \;{}_{\eta\rho}\!\!\leftarrow [\beta]\lambda x.(\mu\alpha.[\gamma]y)x \to_\mu [\beta]\lambda x.\mu\alpha.[\gamma]y$.

However this kind of situations do not arise in the stack calculus (by Theorem 4): in this case for example we have $([\gamma]y)^\circ \twoheadrightarrow_{\mathsf{s}} \mathsf{car}(y)\star\gamma \;{}_{\mathsf{s}\eta}\!\!\twoheadleftarrow ([\beta]\lambda x.\mu\alpha.[\gamma]y)^\circ$.

For example $(\lambda x.x)^\circ = \mu x.\mathsf{car}(x)\star\mathsf{cdr}(x)$ and $(\mathsf{call/cc})^\circ = \mu\alpha.\mathsf{car}(\alpha)\star(\mu\beta.\mathsf{car}(\beta)\star\mathsf{cdr}(\alpha))\centerdot\mathsf{cdr}(\alpha)$, where $\mathsf{call/cc} \equiv \lambda f.\mu\alpha.[\alpha](f(\lambda x.\mu\delta.[\alpha]x))$.

3 The typed stack calculus

We are now going to look at the stack calculus in the light of the Curry–Howard isomorphism. Since the stack calculus can encode calculi with control features (such as $\lambda\mu$-calculus), it can be given a deductive system of full classical implicational propositional logic ($\{\to,\bot\}$-fragment).

The type system has judgements that come in three forms: $\pi : A \vdash \Delta$, $\vdash M : A \mid \Delta$, and $\vdash P \mid \Delta$, where as usual greek capital letters Δ, Δ' are used to denote *contexts*, that is sets of assumptions $\{\alpha_1 : A_1, \ldots, \alpha_n : A_n\}$ (also abbreviated by $\vec{\alpha} : \vec{A}$). In a judgement like $\vdash M : A \mid \Delta$, the semicolon separates the context Δ from the *active formula* A; Theorem 9 can sharpen its role via a comparison with judgements in typed $\lambda\mu$-calculus.

$$\frac{\vdash M : A \mid \Delta \qquad \pi : B \vdash \Delta}{M \cdot \pi : A \to B \vdash \Delta}\,[\to i] \qquad \frac{\alpha : A \in \Delta}{\alpha : A \vdash \Delta}\,[\mathsf{ax}] \qquad \frac{\pi : A \to B \vdash \Delta}{\mathsf{cdr}(\pi) : B \vdash \Delta}\,[\to e_l] \qquad \frac{}{\mathsf{nil} : \bot \vdash \Delta}\,[\bot i]$$

$$\frac{\pi : A \to B \vdash \Delta}{\vdash \mathsf{car}(\pi) : A \mid \Delta}\,[\to e_r] \qquad \frac{\vdash P \mid \Delta, \alpha : A}{\vdash \mu\alpha.P : A \mid \Delta}\,[\mu, \alpha] \qquad \frac{\vdash M : A \mid \Delta \qquad \pi : A \vdash \Delta}{\vdash M \star \pi \mid \Delta}\,[\mathsf{cut}]$$

Fig 2: Typed stack calculus - propositional $\{\to,\bot\}$-fragment.

The choice for the forms of the judgements is justified by the forthcoming Theorem 9, where it will appear that the role of contexts is analogous to that of *name contexts* (i.e. right contexts) in typed $\lambda\mu$-calculus (see Figure 3).

It is very well-known that by restricting Gentzen's sequent calculus LK [11] to manage at most one formula on the right-hand side of sequents one gets the intuitionistic sequent calculus. On the other hand, the symmetric restriction (which, by symmetry, is well behaved with respect to cut elimination) is not so popular. One can find an explicit study of the induced system in Czermak [4]. In [22] Laurent studies a slight variation of Czermak's system, that he calls LD_0, and explores the logical duality between LD_0 and its symmetrical calculus LJ_0. The existence of these two symmetrical (and equivalent, via duality) systems has its roots in the dual "decomposition" of LK into Danos et. al's [5] LKQ and LKT systems, corresponding to call-by-value and call-by-name evaluation of classical proofs, respectively. Both systems are as powerful as LK, and LKT can be encoded into LD_0, in which the *stoup* disappears, since there is at most one formula on the left-hand side of sequents. There is a close relationship between LD_0, LKT and the stack calculus, but indeed while the first two are formulated as a sequent calculus (i.e., with introduction rules only) the latter has elimination rules. One can translate both LKT and LD_0 into the stack calculus (and viceversa), somewhat as Gentzen's LK can be translated into Prawitz's natural decuction [28] (and viceversa) but the translations are not mere inclusions.

The judgements in stack calculus have the following intuitive logical interpretation, in terms of the classical (boolean) notion of semantic entailment "$\models$". For those of the form $\pi : A \vdash \beta_1 : B_1, \ldots, \beta_n : B_n$, read "$\neg B_1, \ldots, \neg B_n \models \neg A$"; for those of the form $\vdash M : A \mid \beta_1 : B_1, \ldots, \beta_n : B_n$, read "$\neg B_1, \ldots, \neg B_n \models A$"; for those of the form $\vdash P \mid \beta_1 : B_1, \ldots, \beta_n : B_n$, read "$\neg B_1, \ldots, \neg B_n \models \bot$". The above indications will be restated and proved precisely in Theorem 17.

We now show that the reduction rules specified in Section 2 are indeed reduction rules for the proofs of the typed system.

Lemma 7 (Typed substitution lemma)**.** *Suppose* $\pi : B \vdash \Delta$.

(i) If $\varpi : A \vdash \beta : B, \Delta$, *then* $\varpi\{\pi/\beta\} : A \vdash \Delta$

(ii) *if* $\vdash M : A \mid \beta : B, \Delta$, *then* $\vdash M\{\pi/\beta\} : A \mid \Delta$

(iii) *if* $\vdash P \mid \beta : B, \Delta$, *then* $\vdash P\{\pi/\beta\} \mid \Delta$.

Using Lemma 7, we can prove that the reduction of a typed term preserves the type.

Theorem 8. *For all* $\pi, \pi' \in \Sigma^{\mathsf{s}}$, *all* $P, P' \in \Sigma^{\mathsf{p}}$ *and* $M, M' \in \Sigma^{\mathsf{t}}$ *we have that*

(i) *if* $\vdash P \mid \Delta$ *and* $P \to_{\mathsf{s}\eta} P'$, *then* $\vdash P' \mid \Delta$

(ii) *if* $\pi : A \vdash \Delta$ *and* $\pi \to_{\mathsf{s}\eta} \pi'$, *then* $\pi' : A \vdash \Delta$

(iii) *if* $\vdash M : A \mid \Delta$ *and* $M \to_{\mathsf{s}\eta} M'$, *then* $\vdash M' : A \mid \Delta$.

Another way to type the stack calculus is to choose a language with negation, conjunction and falsity, to be associated to abstraction, stack construction and empty stack, respectively. This approach mirrors the one used by Lafont et al. [19] to type the λ-calculus with explicit pair constructor and projections. The result is an intuitionistic proof system that can be seen as the target of a CPS translation that embeds Classical Logic into a fragment of Intuitionistic Logic via a mapping that transforms the types but not the proofs; this can be done by two translations $(\cdot)^+$ and $(\cdot)^-$ from $\{\to, \bot\}$-formulas into $\{\wedge, \neg, \bot\}$-formulas as follows: $\bot^+ = \neg\bot$ and $a^+ = a$, for every atom a; $(A \to B)^+ = A^- \wedge B^+$; $A^- = \neg A^+$. One obtains a "rule-per-rule" correspondence: under this point of view, the stack calculus is the target-language of a CPS translation from itself that alters the types but not the proofs, while the translation of Lafont et al. does change also the terms.

3.1 Translation of typed lambda-mu-calculus

The $\lambda\mu$-calculus is endowed with a type system that is a sound and complete Natural Deduction system for purely implicational classical logic.

The type system has judgements that come in two forms: $\Gamma \vdash_{\lambda\mu} t : A \mid \Delta$ and $\Gamma \vdash_{\lambda\mu} p \mid \Delta$. On the left-hand side, Γ represents a context $\vec{x}{:}\vec{A}$ of assumptions for the free λ-variables, while on the right-and side, Δ represents a context $\vec{\alpha} : \vec{B}$ of assumptions for the free names.

$$\frac{x{:}A \in \Gamma}{\Gamma \vdash_{\lambda\mu} x : A \mid \Delta}\,[\mathsf{ax}] \qquad \frac{\Gamma, x{:}A \vdash_{\lambda\mu} t : B \mid \Delta}{\Gamma \vdash_{\lambda\mu} \lambda x.t : A \to B \mid \Delta}\,[\to i, x] \qquad \frac{\Gamma \vdash_{\lambda\mu} t : A \to B \mid \Delta \quad \Gamma \vdash_{\lambda\mu} s : A \mid \Delta}{\Gamma \vdash_{\lambda\mu} ts : B \mid \Delta}\,[\to e]$$

$$\frac{\Gamma \vdash_{\lambda\mu} t : A \mid \Delta}{\Gamma \vdash_{\lambda\mu} [\alpha]t \mid \alpha : A, \Delta}\,[\bot i] \qquad \frac{\Gamma \vdash_{\lambda\mu} p \mid \beta : B, \Delta}{\Gamma \vdash_{\lambda\mu} \mu\beta.p : B \mid \Delta}\,[\bot e, \beta]$$

Fig. 3: Typed $\lambda\mu$-calculus - propositional $\{\to\}$-fragment.

Given a context $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ and a sequence of formulas $\vec{C} = C_1, \ldots, C_n$ we write $\Gamma \to \vec{C}$ as an abbreviation for $x_1 : A_1 \to C_1, \ldots, x_n : A_n \to C_n$.

Theorem 9. *(i) If* $\Gamma \vdash_{\lambda\mu} t : B \mid \Delta$, *then for all sequences* $\vec{C}$ *of formulas we have* $\vdash t^\circ : B \mid \Gamma \to \vec{C}, \Delta$.

(ii) If $\Gamma \vdash_{\lambda\mu} p \mid \Delta$, *then for all sequences* $\vec{C}$ *of formulas we have* $\vdash p^\circ \mid \Gamma \to \vec{C}, \Delta$.

From Theorem 9 results clearly that when the λ-variables are looked at as stack variables, they are endowed with a stream type of which only the type of the head is uniquely determined.

Finally we observe that the empty stack nil does not appear in the translations of $\lambda\mu$-terms. It is needed if we want to translate the so-called $\lambda\mu$-top calculus [1]: in fact one can naturally set $([top]t)^\circ = t^\circ \star \mathsf{nil}$.

3.2 Realizability interpretation of classical logic via stack calculus

In this section we set up a framework which is the analogue of Krivine's Classical Realizability [18]. Krivine's idea is to interpret implicational formulas at the same time as sets of stacks and sets of terms of his modified λ-calculus obtaining, respectively, falsehood and truth values for the formulas. This method has many applications, among which the extraction of programs *realizing* mathematical theorems in the context of relevant logical theories such as Zermelo–Frenkel Set Theory and Analysis [18]. We will apply particular instances of realizability interpretation in Sections 4 and 3.3 to prove soundness and strong normalization of our typed calculus.

Let $\mathbf{T} \subseteq \Sigma^{\mathsf{t}}$ and $\mathbf{0} \subseteq \Sigma^{\mathsf{s}}$ be given sets of terms and stacks, respectively, such that $\mathsf{nil} \in \mathbf{0}$ and if $M \in \mathbf{T}$ and $\pi \in \mathbf{0}$, then $M\cdot\pi \in \mathbf{0}$ and $\mathsf{cdr}(\pi) \in \mathbf{0}$.

We define three binary relations $\succ_{\mathsf{s}}$, $\succ_{\mathsf{t}}$, $\succ_{\mathsf{p}}$ on Σ^{s}, Σ^{t} and Σ^{p}, respectively, as the smallest reflexive relations satisfying the following conditions:

- $\succ_{\mathsf{s}}$ is transitive;
- if $M \in \mathbf{T}$, $\pi \in \mathbf{0}$ and $\varpi \succ_{\mathsf{s}} M\cdot\pi$, then $\mathsf{car}(\varpi) \succ_{\mathsf{t}} M$ and $\mathsf{cdr}(\varpi) \succ_{\mathsf{s}} \pi$;
- if $\pi \in \mathbf{0}$, then $(\mu\alpha.P)\star\pi \succ_{\mathsf{p}} P\{\pi/\alpha\}$;
- if $M' \succ_{\mathsf{t}} M$, then $M'\star\pi \succ_{\mathsf{p}} M\star\pi$.

Moreover we let $\succ_{\mathsf{e}} = \succ_{\mathsf{p}} \cup \succ_{\mathsf{s}} \cup \succ_{\mathsf{t}}$ and we say that a set $X \subseteq \Sigma^{\mathsf{e}}$ is *saturated* if $E \in X$ and $E' \succ_{\mathsf{e}} E$ imply $E' \in X$. For $X \subseteq \Sigma^{\mathsf{e}}$, we let $\mathscr{P}_{\mathrm{s}}(X)$ denote the family of all saturated subsets of X.

Definition 10. *A triple* $(\perp\!\!\!\perp, \mathbf{T}, \mathbf{0})$ *of sets is a* realizability triple *if* $\perp\!\!\!\perp \subseteq \Sigma^{\mathsf{p}}$, $\mathbf{T} \subseteq \Sigma^{\mathsf{t}}$, $\mathbf{0} \subseteq \Sigma^{\mathsf{s}}$ *are all saturated.*

Definition 11 (Realizability relation)**.** *Let* $(\perp\!\!\!\perp, \mathbf{T}, \mathbf{0})$ *be a realizability triple. We define a binary relation* $\Vdash \subseteq \mathbf{T} \times \mathscr{P}_{\mathrm{s}}(\mathbf{0})$ *as* $M \Vdash X$ *iff* $\forall \pi \in X.\ M\star\pi \in \perp\!\!\!\perp$.

If $M \Vdash X$, we say that M *realizes* X, or that M is a *realizer* of X; the set of realizers of X is $\mathsf{rea}(X) = \{M \in \mathbf{T} : M \Vdash X\}$. We define the following binary operation on $\mathscr{P}(\Sigma^{\mathsf{s}})$ as follows:
$X \Rightarrow Y = \{\varpi \in \mathbf{0} : \exists M \in \mathsf{rea}(X).\exists \pi \in Y.\ \varpi \succ_{\mathsf{s}} M\cdot\pi\}$.

We indicate by At the set of all atomic formulas, which includes $\perp$ and a countable set of atoms. We indicate by Fm the set of all formulas built from At with the connective $\to$. We use the following conventions: letters $A, B, C, \ldots$ range over Fm, and $F, G, H, \ldots$ range over At. We let arrows associate to the right, so that $A \to B \to C \equiv A \to (B \to C)$. Every formula is of the form $B_1 \to \cdots \to B_n \to G$, where G is atomic. As usual the negation is defined as $\neg A := A \to \perp$.

Let $\mathbf{R} = (\perp\!\!\!\perp, \mathbf{T}, \mathbf{0})$ be a realizability triple. An *atomic* $\mathbf{R}$*-interpretation* is a function $\mathscr{I} : \mathsf{At} \to \mathscr{P}_{\mathrm{s}}(\mathbf{0})$ such that $\mathscr{I}(\perp) = \mathbf{0}$. Then $\mathscr{I}$ extends uniquely to a map $\|\cdot\|_{\mathscr{I}} : \mathsf{Fm} \to \mathscr{P}(\Sigma^{\mathsf{s}})$ by setting $\|A \to B\|_{\mathscr{I}} = \|A\|_{\mathscr{I}} \Rightarrow \|B\|_{\mathscr{I}}$. The set $\|A\|_{\mathscr{I}}$ is called the *falsehood value* of the formula A under $\mathscr{I}$. The *truth value* $|A|_{\mathscr{I}}$ of a formula A under $\mathscr{I}$ is given by $|A|_{\mathscr{I}} = \mathsf{rea}(\|A\|_{\mathscr{I}})$.

Proposition 12. *For every formula* A, $\|A\|_{\mathscr{I}} \in \mathscr{P}_{\mathrm{s}}(\mathbf{0})$ *and* $|A|_{\mathscr{I}} \in \mathscr{P}_{\mathrm{s}}(\mathbf{T})$.

Proof. By induction on the structure of formulas. For falsehood values is suffices to observe that $\mathscr{P}_{\mathrm{s}}(\mathbf{0})$ is closed under the $\Rightarrow$ operation. For truth values, use the fact that $M' \succ_{\mathsf{t}} M$ implies $M'\star\pi \succ_{\mathsf{p}} M\star\pi$ and the saturation of $\perp\!\!\!\perp$. □

If $\vec{\pi} = \pi_1, \ldots, \pi_n$ and $\vec{B} = B_1, \ldots, B_n$ are sequences, we write $\vec{\pi} \in \|\vec{B}\|_{\mathscr{I}}$ as an abbreviation for $\pi_1 \in \|B_1\|_{\mathscr{I}}, \ldots, \pi_n \in \|B_n\|_{\mathscr{I}}$. The next theorem is the stack calculus analogue of Krivine's Adequacy Theorem [18], which shows that realizability is compatible with deduction in classical logic. It is an essential tool that will be used to obtain, in a uniform way, both soundness and strong normalization of the typed calculus.

Theorem 13 (Adequacy theorem)**.** *Let* $\mathbf{R} = (\perp\!\!\!\perp, \mathbf{T}, \mathbf{0})$ *be a realizability triple and let* $\mathscr{I}$ *be an* $\mathbf{R}$*-interpretation. If* $\vec{\pi} \in \|\vec{B}\|_{\mathscr{I}}$ *then*

(i) If $\varpi : A \vdash \vec{\alpha} : \vec{B}$*, then* $\varpi\{\vec{\pi}/\vec{\alpha}\} \in \|A\|_{\mathscr{I}}$*;*

(ii) If $\vdash M : A \mid \vec{\alpha} : \vec{B}$*, then* $M\{\vec{\pi}/\vec{\alpha}\} \in |A|_{\mathscr{I}}$*;*

(iii) If $\vdash P \mid \vec{\alpha} : \vec{B}$*, then* $P\{\vec{\pi}/\vec{\alpha}\} \in \perp\!\!\!\perp$*.*

One proves all items simultaneously proceeding by induction on the depth of type derivations.

3.3 Normalization in the typed stack calculus

We are now going to prove that the typed stack calculus is strongly normalizing. We prove this fact by adapting the reducibility candidates technique to our setting. It becomes a sort of instance of Krivine's adequacy theorem in the context of Classical Realizability. We let $\mathrm{SN}^{\mathsf{e}} \subseteq \Sigma^{\mathsf{e}}$ be the set of all strongly normalizing expressions of the stack calculus (w.r.t. $\to_{\mathsf{s}\eta}$-reduction); SN^{t}, SN^{p}, SN^{s} denote the sets all strongly normalizing terms, processes and stacks, respectively.

Proposition 14. $\mathbf{S} = (\mathrm{SN}^{\mathsf{p}}, \mathrm{SN}^{\mathsf{t}}, \mathrm{SN}^{\mathsf{s}})$ *is a realizability triple.*

The proof of Proposition 14 consists in showing that if $E' \succ_{\mathsf{e}} E$ and $E \in \mathrm{SN}^{\mathsf{p}}$ (resp. $E \in \mathrm{SN}^{\mathsf{t}}$, $E \in \mathrm{SN}^{\mathsf{s}}$), then also $E' \in \mathrm{SN}^{\mathsf{p}}$ (resp. $E' \in \mathrm{SN}^{\mathsf{t}}$, $E' \in \mathrm{SN}^{\mathsf{s}}$). One can proceed by induction on the definition of $\succ_{\mathsf{e}}$. The main point of such a proof is when we consider the case in which $P \equiv M \star \pi \in \mathrm{SN}^{\mathsf{p}}$ and $P' \equiv M' \star \pi$ with $M' \succ_{\mathsf{t}} M$ because there there exist ϖ and π' such that $\varpi \succ_{\mathsf{s}} M \cdot \pi'$ and $M' \equiv \mathsf{car}(\varpi)$. Here one can show that if $M' \star \pi$ has an infinite reduction path, then $M \star \pi$ has an infinite reduction path too. Note that it is crucial that for the terms $M' \equiv \mu\alpha.(\mu\beta.\beta[1] \star \beta) \star (\mu\gamma.\alpha[0] \star \alpha) \cdot \mathsf{nil}$ and $M \equiv \mu\alpha.\mathsf{nil}[0] \star \mathsf{nil}$ we have $M' \not\succ_{\mathsf{t}} M$. In fact, setting $\pi \equiv (\mu\delta.\delta[0] \star \delta) \cdot \mathsf{nil}$, we obtain that $M \star \pi$ is strongly normalizing but $M' \star \pi$ is not strongly normalizing.

Let A be a formula. We define its *arity* $\mathsf{ar}(A)$ by induction setting $\mathsf{ar}(G) = 0$ and $\mathsf{ar}(A \to B) = 1 + \mathsf{ar}(B)$. It is convenient sometimes to use abbreviations $\pi[n] := \mathsf{cdr}(\cdots \mathsf{cdr}(\pi) \cdots)$ (n times) and $\pi[n] := \mathsf{car}(\pi[n])$, in order to make some expressions more readable.

Theorem 15 (Strong normalization)**.** *Let* $M \in \Sigma^{\mathsf{t}}$*,* $\pi \in \Sigma^{\mathsf{s}}$ *and* $P \in \Sigma^{\mathsf{p}}$*.*

(i) If there exist Δ, A *such that* $\pi : A \vdash \Delta$*, then* $\pi \in \mathrm{SN}^{\mathsf{s}}$*;*

(ii) If there exist Δ, A *such that* $\vdash M : A \mid \Delta$*, then* $M \in \mathrm{SN}^{\mathsf{t}}$*;*

(iii) If there exist Δ *such that* $\vdash P \mid \Delta$*, then* $P \in \mathrm{SN}^{\mathsf{p}}$*.*

Proof. Let $\Delta = \vec{\alpha} : \vec{B}$, where $\vec{\alpha} = \alpha_1, \ldots, \alpha_n$ and $\vec{B} = B_1, \ldots, B_n$. Let $\mathscr{I}$ be the $\mathbf{S}$-interpretation sending every atom to SN^{s} and set $\pi_i := \alpha_i[0] \cdot \ldots \cdot \alpha_i[\mathsf{ar}(B_i) - 1] \cdot \alpha_i[\mathsf{ar}(B_i))$, for each $i = 1, \ldots, n$ and $\vec{\pi} = \pi_1, \ldots, \pi_n$. An easy induction on the arity of formulas shows that $\vec{\pi} \in \|\vec{B}\|_{\mathscr{I}}$. By Theorem 13 (i),(ii),(iii) respectively we get that
(i) $\varpi\{\vec{\pi}/\vec{\alpha}\} \in \|A\|_{\mathscr{I}} \subseteq \mathrm{SN}^{\mathsf{s}}$, (ii) $M\{\vec{\pi}/\vec{\alpha}\} \in |A|_{\mathscr{I}} \subseteq \mathrm{SN}^{\mathsf{t}}$ and (iii) $P\{\vec{\pi}/\vec{\alpha}\} \in \mathrm{SN}^{\mathsf{p}}$.
Finally in each of the above cases we have $E\{\vec{\pi}/\vec{\alpha}\} \twoheadrightarrow_{\eta} E$ and since $E\{\vec{\pi}/\vec{\alpha}\}$ is strongly normalizing, then so is E. □

4 Soundness and completeness of typed stack calculus w.r.t. classical semantics

The present section provides soundness and completeness proofs of the stack calculus for the two-valued semantics of classical propositional logic. We find interesting to report the full completeness proof, which resembles very much a completeness proof for a tableaux calculus [31]. In fact, as in a tableaux system there are labeled formulas (with *true* and *false* labels), in the stack calculus we have terms and stacks which play, respectively, the role of proofs and counter-proofs, exactly in the spirit of Krivine's Classical Realizability.

It is easy matter to check that $\mathbf{B} = (\emptyset, \Sigma^{\mathsf{t}}, \Sigma^{\mathsf{s}})$ is a realizability triple. For every formula A and $\mathbf{B}$-interpretation $\mathscr{I}$ we have

$$|A|_{\mathscr{I}} = \begin{cases} \Sigma^{\mathsf{t}} & \text{if } \|A\|_{\mathscr{I}} = \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

The induced function $|\cdot|_{\mathscr{I}}$ maps formulas into elements of the two-element boolean algebra $\{\Sigma^{\mathsf{t}}, \emptyset\}$, where the ordering is set-inclusion and the operators are $\cup$, $\cap$ and complement. In other words Σ^{t} represents "true" and $\emptyset$ represents "false". The truth values behave as expected w.r.t. negation: $|A|_{\mathscr{I}} = \emptyset \Leftrightarrow |\neg A|_{\mathscr{I}} = \Sigma^{\mathsf{t}}$.

Definition 16. *Let Φ be a set of formulas and let A be a formula. We say that Φ* semantically entails *A, notation $\Phi \vDash A$, if for every atomic $\mathbf{B}$-interpretation $\mathscr{I}$ we have that $\bigcap_{B\in\Phi} |B|_{\mathscr{I}} \subseteq |A|_{\mathscr{I}}$.*

Theorem 17 (Soundness)**.**

(i) If $\vdash M : A \mid \vec{\beta} : \vec{B}$ is provable (where $\mathrm{FV}(M) \subseteq \vec{\beta}$), then $\neg B_1, \ldots, \neg B_n \vDash A$.

(ii) If $\pi : A \vdash \vec{\beta} : \vec{B}$ is provable (where $\mathrm{FV}(\pi) \subseteq \vec{\beta}$), then $\neg B_1, \ldots, \neg B_n \vDash \neg A$.

(iii) If $\vdash P \mid \vec{\beta} : \vec{B}$ is provable (where $\mathrm{FV}(P) \subseteq \vec{\beta}$), then $\neg B_1, \ldots, \neg B_n \vDash \bot$.

Proof. (i) Let $\mathscr{I}$ be a $\mathbf{B}$-interpretation. By Theorem 13 (Adequacy) if for all $i \in [1,n]$ $\|B_i\|_{\mathscr{I}} \neq \emptyset$, then $M\{\vec{\pi}/\vec{\alpha}\} \in |A|_{\mathscr{I}}$, i.e., $|A|_{\mathscr{I}} \neq \emptyset$. Since $\|B_i\|_{\mathscr{I}} \neq \emptyset \Leftrightarrow |B_i|_{\mathscr{I}} = \emptyset \Leftrightarrow |\neg B_i|_{\mathscr{I}} = \Sigma^{\mathsf{t}}$, we conclude that every derivable judgement $\vdash M : A \mid \vec{\beta} : \vec{B}$ has the following property: for every $\mathscr{I}$, if $|\neg B_i|_{\mathscr{I}} = \Sigma^{\mathsf{t}}$ for all $i \in [1,n]$, then $|A|_{\mathscr{I}} = \Sigma^{\mathsf{t}}$. This means, by definition, that $\neg B_1, \ldots, \neg B_n \vDash A$.
(ii),(iii) Similar to (i), again applying Theorem 13. □

The main goal of the rest of the section is to prove that every classical tautology is the type of some term of the stack-calculus. The proof is supported by some auxiliary definitions and lemmas.

Definition 18. *Let A be a formula. We define its* terminal $\mathsf{tmn}(A)$ *by induction setting $\mathsf{tmn}(G) = G$ and $\mathsf{tmn}(A \to B) = \mathsf{tmn}(B)$. We also define its* premisses $\mathsf{pr}(A)$ *by induction setting $\mathsf{pr}(G) = \emptyset$ and $\mathsf{pr}(A \to B) = \{A\} \cup \mathsf{pr}(B)$.*

Definition 19. *Let Φ be a set of formulas. We define three sets $\mathsf{tmn}(\Phi) = \{\mathsf{tmn}(A) : A \in \Phi\}$, $\mathsf{pr}(\Phi) = \bigcup_{A\in\Phi} \mathsf{pr}(A)$, and $\mathsf{prt}(\Phi) = \{A \in \mathsf{pr}(\Phi) : \mathsf{tmn}(A) \in (\mathsf{tmn}(\Phi) \cup \{\bot\})\}$.*

Definition 20. *A set Φ of formulas is* saturated *if for every formula $A \in \mathsf{prt}(\Phi)$ we have $\mathsf{pr}(A) \cap \Phi \neq \emptyset$.*

It will turn out that, by applying an iterative process, it is possible to construct saturated sets of formulas starting from finite sets of formulas which cannot be proved by a sequent of the stack calculus. The forthcoming Lemmas 21 and 22 are the fundamental ingredients for such construction. We write $\nvdash - : A \mid - : \vec{B}$ to express the fact that there are no variables $\vec{\beta}$ and no term M such that $\vdash M : A \mid \vec{\beta} : \vec{B}$.

Lemma 21. *Let $\Phi = \{B_0, \ldots, B_n\}$ be a finite set of formulas and suppose $\nvdash - : B_0 \mid - : B_1, \ldots, - : B_n$. Then $\mathsf{prt}(\Phi) \cap \mathsf{At} = \emptyset$.*

Proof. We prove the contrapositive statement. Supposing $A \in \mathsf{prt}(\Phi) \cap \mathsf{At}$, we distinguish two possible cases: (1) and (2). We write $\vec{\beta} : \vec{B}$ for the context $\beta_1 : B_1, \ldots, \beta_n : B_n$. Let ε be a fresh variable.

(1) There exist some $j, k \in [0,n]$ such that $B_j = C'_1 \to \cdots \to C'_i \to \cdots \to C'_{m'} \to G'$, $B_k = C''_1 \to \cdots \to C''_{m''} \to G''$, and $C'_i = G'' = A$. Then $\vdash \mu\beta_0.(\mu\varepsilon.\beta_j[i-1] \star \varepsilon[m'')) \star \beta_k : B_0 \mid \vec{\beta} : \vec{B}$.

(2) There exist some $j \in [0,n]$ such that $B_j = C'_1 \to \cdots \to C'_i \to \cdots \to C'_{m'} \to G'$, and $C'_i = \bot = A$. Then $\vdash \mu\beta_0.(\mu\varepsilon.\beta_j[i-1] \star \mathsf{nil}) \star \beta_k : B_0 \mid \vec{\beta} : \vec{B}$.

□

Lemma 22. *Let $\Phi = \{B_0, \ldots, B_n\}$ be a finite set of formulas and suppose $\nvdash - : B_0 \mid - : B_1, \ldots, - : B_n$. Then for every $A \in \mathsf{prt}(\Phi)$ there exists a formula $C \in \mathsf{pr}(A)$ such that $\nvdash - : B_0 \mid - : B_1, \ldots, - : B_n, - : C$.*

Proof. We prove the contrapositive statement. To this end, suppose $A \in \mathsf{prt}(\Phi)$ is a formula that is a counterexample to the conclusion of the statement. First note that $\mathsf{pr}(A) \neq \emptyset$, otherwise $A \in \mathsf{At}$, in contradiction with Lemma 21. Therefore $A = C_1 \to \cdots \to C_m \to G$, with $m \geq 1$. We write $\vec{\beta} : \vec{B}$ for the context $\beta_1 : B_1, \ldots, \beta_n : B_n$.

By our assumption for every $i = 1, \ldots, m$ ($m \geq 1$) there exist M_i, γ_i such that $\vdash M_i : B_0 \mid \vec{\beta} : \vec{B}, \gamma_i : C_i$ and thus we derive $\vdash \mu\gamma_i.M_i \star \beta_0 : C_i \mid \beta_0 : B_0, \vec{\beta} : \vec{B}$ for each $i = 1, \ldots, m$. Moreover, since $A \in \mathsf{prt}(\Phi)$, there are two cases:

(1) there exist some $k, h \in [0,n]$ such that $B_h = C'_1 \to \cdots \to C'_j \to \cdots \to C'_{m'} \to G'$, $B_k = C''_1 \to \cdots \to C''_{m''} \to G''$, $A = C'_j$, and $G = G''$.

(2) $G = \bot$ and there exist some $h \in [0,n]$ such that $B_h = C'_1 \to \cdots \to C'_j \to \cdots \to C'_{m'} \to G'$ and $A = C'_j$.

Let ε be a fresh variable. In both cases (1) and (2) there exists a stack π such that $\pi : G \vdash \varepsilon : B_k$ is derivable, where π is either nil or $\varepsilon[\mathsf{ar}(B_k))$.

Let $\gamma_1, \ldots, \gamma_m, \delta$ be fresh variables and let $\varpi := (\mu\gamma_1.M_1 \star \beta_0) \cdot \ldots \cdot (\mu\gamma_m.M_m \star \beta_0) \cdot \pi$. Then we finally derive $\vdash \mu\beta_0.(\mu\delta.(\mu\varepsilon.\delta[j-1] \star \varpi) \star \beta_k) \star \beta_h : B_0 \mid \vec{\beta} : \vec{B}$. □

The *complexity* of a formula A is the total number of implications and atomic sub-formulas occurring in A. The formulas of complexity one are exactly the atomic ones.

Lemma 23. *Let Φ be a saturated set of formulas. Then there exists a* **B***-interpretation $\mathscr{I}$ such that $|A|_{\mathscr{I}} = \emptyset$, for all $A \in \Phi$.*

Proof. The case in which $\Phi = \emptyset$ is trivial, so for the rest of the proof we assume $\Phi \neq \emptyset$. We define a **B**-interpretation $\mathscr{I}$ as follows:

$$\mathscr{I}(G) = \begin{cases} \emptyset & \text{if } G \in \mathsf{tmn}(\Phi) \\ \Sigma^{\mathsf{t}} & \text{otherwise} \end{cases}$$

We now prove that $|A|_{\mathscr{I}} = \emptyset$, for all $A \in \Phi$. The proof is by induction on the complexity of formulas. Suppose $A \in \mathsf{At}$. If $A = \bot$ the result is obvious; otherwise, since $A \in \mathsf{tmn}(\Phi)$, we have $|A|_{\mathscr{I}} = \emptyset$. Suppose $A = C_1 \to \cdots \to C_m \to G$ (with $m \geq 1$). We now prove that

(1) $|C_1|_{\mathscr{I}} = \cdots = |C_m|_{\mathscr{I}} = \Sigma^{\mathsf{t}}$; (2) $|G|_{\mathscr{I}} = \emptyset$.

The items (1) and (2) together yield $|A|_{\mathscr{I}} = \emptyset$.

(1) For $C_i \in \mathsf{pr}(A)$ we distinguish two cases.
Suppose $C_i \notin \mathsf{prt}(\Phi)$. Then $\mathsf{tmn}(C_i)$ is not a terminal of a formula in Φ. By definition of $\mathscr{I}$ we have $|\mathsf{tmn}(C_i)|_{\mathscr{I}} = \Sigma^{\mathsf{t}}$. We conclude observing that $|C_i|_{\mathscr{I}} \supseteq |\mathsf{tmn}(C_i)|_{\mathscr{I}} = \Sigma^{\mathsf{t}}$.
Suppose $C_i \in \mathsf{prt}(\Phi)$. Then, by saturation of Φ, $C_i = C'_1 \to \cdots \to C'_{m'} \to G'$ (with $m' \geq 1$) and there exists $j \in [1, m']$ such that $C'_j \in \Phi$. Since C'_j has strictly lower complexity than A, by induction hypothesis $|C'_j|_{\mathscr{I}} = \emptyset$. This implies $|C_i|_{\mathscr{I}} = \Sigma^{\mathsf{t}}$.

(2) Since $G \in \mathsf{tmn}(\Phi) \cup \{\bot\}$, evidently $|G|_{\mathscr{I}} = \emptyset$ by the definition of the interpretation $|\cdot|_{\mathscr{I}}$.

□

Next we give the second main theorem of this section, concerning completeness. The idea of its proof is the *counter-model construction*, typical of Smullyan's analytic tableaux [31].

Theorem 24 (Completeness)**.** *Let A be a formula and let $\vec{B}$ be a sequence of formulas. If $\neg B_1, \ldots, \neg B_n \vDash A$, then there exist M and $\vec{\beta}$ such that $\vdash M : A \mid \vec{\beta} : \vec{B}$ is provable.*

Proof. We proceed to prove the contrapositive statement. Suppose $\nvdash - : A \mid - : \vec{B}$. Then we can construct a saturated set Φ of formulas containing $\{A, B_1, \ldots, B_n\}$ as $\Phi := \bigcup_{n \geq 0} \Phi_n$, where the family $\{\Phi_n\}_{n \geq 0}$ is inductively defined as follows:

- $\Phi_0 := \{A, B_1, \ldots, B_n\}$;
- If $\mathsf{prt}(\Phi_n) = \emptyset$, then we define $\Phi_{n+1} := \Phi_n$. If $\mathsf{prt}(\Phi_n) = \{C_1, \ldots, C_k\} \neq \emptyset$, by Lemma 22 for each C_i there exists a formula $D_i \in \mathsf{pr}(C_i)$ such that $\nvdash - : A \mid - : B_1, \ldots, - : B_n, - : D_i$. Let $\Psi_n = \{D_1, \ldots, D_k\}$, where each D_i is the leftmost premiss of C_i having the property that $\nvdash - : A \mid - : B_1, \ldots, - : B_n, - : D_i$. Then we define $\Phi_{n+1} := \Phi_n \cup \Psi_n$.

By construction Φ is a saturated set of formulas containing $\{A, B_1, \ldots, B_n\}$. Finally applying Lemma 23 we obtain some $\mathscr{I}$ such that $|B_1|_{\mathscr{I}} = \cdots = |B_n|_{\mathscr{I}} = |A|_{\mathscr{I}} = \emptyset$, meaning that $\neg B_1, \ldots, \neg B_n \nvDash A$. □

Of course Theorem 24 implies that every classical propositional tautology (of the $\{\to, \bot\}$-fragment) is provable by the type derivation of a term.

5 The Krivine machine for stack calculus

In the present section we sketch the definition of a Krivine machine that executes the terms of stack calculus. Similar machines have been defined by de Groote [9], Laurent [20], Reus and Streicher [29] for the $\lambda\mu$-calculus. Using this machine we show how to encode control mechanisms like *label/resume* and *raise/handle* in the stack calculus.

In order to define the states of the machine, we need the following mutually inductive definitions. A *stack closure* is a pair $p = (\pi, e)$ consisting of a stack π and an environment e; a *term closure* is a pair $m = (M, e)$ consisting of a stack π and an environment e; an *environment* is a partial function (with finite domain) from the set of stack variables to the set of stack closures. We write $e[\alpha \leftarrow\!\shortmid p]$ for the environment e' which assumes the same values as e except at most on α, where $e'(\alpha) = p$.

A *state* is a pair $\langle m, p \rangle$ and the machine consists of the following (deterministic) transitions between states:

$$\begin{array}{lcll} \langle (N, e), p \rangle & \longrightarrow & \langle (\pi'[n], e'), p \rangle & \text{if } \alpha[n] \text{ is the } \twoheadrightarrow_{\mathsf{car,cdr}} \text{-normal form of } N \text{ and } e(\alpha) = (\pi', e') \\ \langle (N, e), p \rangle & \longrightarrow & \langle (M, e'), (\pi, e') \rangle & \text{if } \mu\alpha.M \star \pi \text{ is the } \twoheadrightarrow_{\mathsf{car,cdr}} \text{-normal form of } N \text{ and } e' = e[\alpha \leftarrow\!\shortmid p] \end{array}$$

We let $\longrightarrow\!\!\!\!\!\rightarrow$ be the reflexive and transitive closure of the relation $\longrightarrow$. Consider a state $\langle (M,e),p\rangle$. The closure p is the current context of evaluation of M; the next state may discard p and restore a context appeared in the past. The environment e is the current state of the memory: it takes into account all side effects caused by the previous stages of computation. The term M is said to be in *execution position* and it is the current program acting on p evaluated in e. A *computation* is a sequence of states sequentially related by the transition rules.

To explain how stack calculus achieves the control of the execution flow, we define label/resume and raise/handle instructions and show that the machine soundly executes them. We set

$$\begin{array}{ll}
\mathsf{lab}_\varepsilon\{M\} := \mu\beta.(\mu\varepsilon.M\star\beta)\star(\mu\delta.\delta[0]\star\beta)\mathbf{.}\beta & \text{with } \beta\notin \mathrm{FV}(M)\\
\mathsf{res}_\varepsilon\{M\} := \mu\gamma.\varepsilon[0]\star N\mathbf{.}\gamma & \text{with } \varepsilon,\gamma\notin \mathrm{FV}(M)\\
\mathsf{throw}_\varepsilon\{M\} := \mu\gamma.\varepsilon[0]\star M\mathbf{.}\mathsf{nil} & \text{with } \varepsilon,\gamma\notin \mathrm{FV}(M)\\
\mathsf{try}_\varepsilon\{M\}\mathsf{catch}\{N\} := \mu\beta.(\mu\varepsilon.M\star\beta)\star(\mu\delta.N\star\delta[0]\mathbf{.}\beta)\mathbf{.}\mathsf{nil} & \text{with } \beta\notin(\mathrm{FV}(M)\cup\mathrm{FV}(N)),\ \delta\notin\mathrm{FV}(N)
\end{array}$$

We now discuss briefly and informally how the machine executes the above instructions.

Suppose to start the machine in a state $S=\langle(\mathsf{lab}_\varepsilon\{M\},e_0),p_0\rangle$. If no term $\mathsf{res}_\varepsilon\{N\}$ ever reaches the execution position, then the computation starting at S is equivalent to that starting at $S'=\langle(M,e_0),p_0\rangle$. Otherwise $S\longrightarrow\!\!\!\!\!\rightarrow^n\langle(\mu\gamma.\varepsilon[0]\star N\mathbf{.}\gamma,e_n),p_n\rangle\longrightarrow\!\!\!\!\!\rightarrow^2\langle(N,e_{n+1}),p_{n+2}\rangle$, and we notice that the computation starting at $\langle(\mathsf{res}_\varepsilon\{N\},e_n),p_n\rangle$ is equivalent to that starting at $\langle(N,e_{n+1}),p_{n+2}\rangle$.

Suppose to start the machine in a state $S=\langle(\mathsf{try}_\varepsilon\{M\}\mathsf{catch}\{N\},e_0),p_0\rangle$. If no term $\mathsf{throw}_\varepsilon\{M'\}$ ever reaches the execution position, then the computation starting at S is equivalent to that starting at $S'=\langle(M,e_0),p_0\rangle$. Otherwise $S\longrightarrow\!\!\!\!\!\rightarrow^n\langle(\mu\gamma.\varepsilon[0]\star M'\mathbf{.}\mathsf{nil},e_n),p_n\rangle\longrightarrow\!\!\!\!\!\rightarrow^3\langle(N,e_{n+2}),(\delta[0]\mathbf{.}\beta,e_{n+2})\rangle$ and we can see that the exception handler N goes on with the computation, and the value M' returned by the exception is at use of N, since it is stored in the in the current environment e_{n+2} in a cell that is present in the current evaluation context.

We conclude remarking that all the above constructions can be typed by derived rules. Informally one may assert that Theorem 17 and Theorem 8, together, ensure that the execution of well-typed term always ensures that all the "resume" and "raise" instructions are always handled correctly.

6 Denotational semantics of stack calculus

Girard's *correlation spaces* [13] are (one of) the first denotational model of Classical Logic: they refine coherence spaces [12] with some additional structure. Intuitively, these richer objects come with the information required to interpret structural rules (weakening and contraction) on the right-hand side of sequents in classical sequent calculus. Girard's construction hints that Classical Logic may be encoded into Linear Logic, a result achieved by Danos et al. [5] via a dual linear decomposition of classical implication. In [29] the authors interpret the $\lambda\mu$-calculus in the Cartesian closed category of "negated domains", i.e. the full subcategory of **CPO** determined by the objects of the form R^A, where A is a predomain and R is some fixed domains of "responses". The category of negated domains is a particular *category of continuations* [19] and categories of continuations are *complete* [15] for the $\lambda\mu$-calculus, in the sense that every equational theory for $\lambda\mu$-calculus is given by the kernel relation of the interpretation in some category of continuations. Selinger [30] gives a general presentation in terms of *control categories*, which are easily seen to subsume categories of continuations. However via a categorical structure theorem he also shows that every control category is equivalent to a category of continuations. This structure theorem implies the soundness and completeness of the categorical interpretation of the $\lambda\mu$-calculus with respect to a natural CPS semantics.

In brief, a control category is a Cartesian closed category $(\mathbf{C},\&,\top,\Rightarrow)$ which is also a symmetric premonoidal category $(\mathbf{C},\parr,\bot)$. The binoidal functor $\parr$ distributes over $\&$ and there is a natural isomorphism $s_{A,B,C} : B^A \parr C \to (B \parr C)^A$ in A,B and C satisfying some coherence conditions. Selinger distinguishes a subcategory $\mathbf{C}^\sharp$ of $\mathbf{C}$, called the *focus* of $\mathbf{C}$, which have the same objects as $\mathbf{C}$ but fewer arrows. On $\mathbf{C}^\sharp$ the functor $\parr$ restricts to a coproduct. It is very important to remark that in any control category $\mathbf{C}$ there exists an isomorphism $\varphi : \mathbf{C}(\top, B \parr A) \cong \mathbf{C}^\sharp(\bot^A, B)$ natural in central B (see [30] for the details). If $\mathbf{C}$ is a control category we map falsity to the object $\bot$ and set $|A \to B| = \bot^{|A|} \parr |B|$; a context $\Delta = \vec{\alpha} : \vec{A}$ is mapped to $|\Delta| = |A_1| \parr \cdots \parr |A_n|$. Then the judgements are interpreted as morphisms $[\![\pi : A \vdash \Delta]\!] : |A| \to |\Delta|$, $[\![\vdash M : A \mid \Delta]\!] : \bot^{|A|} \to |\Delta|$ and $[\![\vdash P \mid \Delta]\!] : \top \to |\Delta|$, using the coproduct structure and the isomorphism φ. The above intepretation is *sound*, in the sense that it is invariant under $\to_{\mathtt{s}\eta}$-reduction of expressions.

Very interesting is the work of Laurent and Regnier [23] which shows in detail how to extract a control category out of a categorical model of MALL. This constribution gives a general framework under which falls the correlation spaces model construction by Girard and at the same time constitutes the categorical counterpart of Danos–Joinet–Schellinx's [5] call-by-name encoding of Classical logic into Linear Logic.

A $*$-autonomous category is a symmetric monoidal category with two monoidal structures $(\mathbf{C},\otimes,\mathbf{1})$ and $(\mathbf{C},\parr,\bot)$ possessing a *dualizing* endofunctor $(\cdot)^\bot$ which maps $f : A \to B$ to $f^\bot : B^\bot \to A^\bot$.

Let $\mathbf{C}$ be a $*$-autonomous category. When the forgetful functor from the category $\mathrm{Mon}_\parr(\mathbf{C})$ (of $\parr$-monoids and $\parr$-monoid morphisms) to the category $\mathbf{C}$ has a right adjoint, then $\mathbf{C}$ is a *Lafont category*. We recall that the co-Kleisli category $\mathbb{K}_\mathbf{C}$ of a monoidal category $\mathbf{C}$ via a comonad $(!,\delta,\varepsilon)$ has the same objects as $\mathbf{C}$ and $\mathbb{K}_\mathbf{C}(A,B) = \mathbf{C}(!A,B)$; the composition of morphisms is defined using the monad structure (see [26]).One of the main results of [23] is that if $\mathbf{C}$ is a $*$-autonomous Lafont category with finite products, then then the co-Kleisli category $\mathbb{K}_{\mathbf{C}'}$ of the full-subcategory $\mathbf{C}'$ of $\mathbf{C}$ whose objects are the $\parr$-monoids is a control category.

6.1 A simple interpretation of stack calculus

Inspired by Laurent and Regnier's work [23] we give a minimal framework in which the stack calculus can be soundly interpreted. The absence of the λ-abstraction, allows us to focus on the minimal structure required to interpret Laurent's Polarized Linear Logic [21] and to use it to interpret the stack calculus.

Let $\mathbf{C}$ be a $*$-autonomous category. We denote by $\rho_A : A \to A \parr \bot$, $\lambda_A : A \to \bot \parr A$, α, γ and τ the usual natural isomorphisms related to the monoidal structure of $(\mathbf{C},\parr,\bot)$.

A *linear category* is a symmetric monoidal category together with a symmetric monoidal comonad $((!,\mathsf{m}),\delta,\varepsilon)$ such that there are monoidal natural transformations with components $\mathbf{e}_A : !A \to \mathbf{1}$ and $\mathbf{d}_A : !A \to !A \otimes !A$ which are coalgebra morphisms and make each free !-coalgebra a commutative $\otimes$-comonoid $(!A, d_A, e_A)$; moreover $\delta_A : !A \to !!A$ is a comonoid morphism, for every object A.

In the sequel we let $\mathbf{C}$ be a $*$-autonomous linear category, so that by duality we can turn the above definition in terms of a monad $((?,\mathsf{m}),\delta,\varepsilon)$, ?-algebras and $\parr$-monoids. In this case there are monoidal natural transformations with components $w_A : \bot \to ?A$ and $c_A : ?A \parr ?A \to ?A$ which are ?-algebra morphisms and make each free ?-algebra a commutative $\parr$-monoid $(?A, \mathbf{c}_A, \mathbf{w}_A)$; $\delta_A : ??A \to ?A$ is a monoid morphism, for every object A. Under these hypotheses all ?-algebras A, being retract of a the free algebra $?A$, have a multiplication c_A, and a unit w_A (see [26] for further details). The category $\mathbf{C}^?$ of Eilenberg-Moore algebras is symmetric monoidal, with (co)tensor product of (A,alg_A), (B,alg_B) given by $(A \parr B, (\mathsf{alg}_A \parr \mathsf{alg}_B) \circ \mathsf{m}^2)$ and unit given by $(\bot,\mathsf{m}^1)$.

The $*$-autonomous structure of $\mathbf{C}$ yields a natural isomorphism $\Lambda : \mathbf{C}(\mathbf{1}, B \parr A) \to \mathbf{C}(A^\perp, B)$ that we will use to interpret abstraction (a natural retraction $\mathbf{C}(\mathbf{1}, B \parr A) \lhd \mathbf{C}(A^\perp, B)$ would suffice anyway).

Starting from a valuation that associates ?-algebras to atomic types and the object $\perp$ to falsity, the arrow-types are mapped as follows: $|A \to B| = ?|A|^\perp \parr |B|$. Given a context $\Delta = \vec{\alpha} : \vec{A}$ we set $|\Delta| = |A_1| \parr \cdots \parr |A_n|$. Note that all types are interpreted by ?-algebras. Then the type judgements with assumptions Δ can be easily interpreted as morphisms with target $|\Delta|$; for example $[\![\mathsf{nil} : \perp \vdash \Delta]\!] : \perp \to |\Delta|$ is the unit of the monoid $|\Delta|$.

We describe such interpretation for the particular case of the untyped stack calculus, for which we need a ?-algebra U of $\mathbf{C}$ together with two ?-algebra morphisms $\mathrm{La} : ?U^\perp \parr U \to U$ and $\mathrm{Ap} : U \to ?U^\perp \parr U$ satisfying $\mathrm{Ap} \circ \mathrm{La} = \mathsf{id}_{?U^\perp \parr U}$ and a ?-algebra morphism $\vartheta : U \to \perp$ (needed for the stack nil).

We write U^n for the n-fold $\parr$-product of U. Such product inherits a ?-algebra structure alg_{U^n} defined using the algebra alg_U and the monoidality of the monad; as a consequence it also inherits a multiplication c_{U^n} and a unit w_{U^n}. We also define $\iota_j{}^n : U \cong \perp^{j-1} \parr U \parr \perp^{n-j} \xrightarrow{w_{U^{j-1}} \parr \mathsf{id}_U \parr w_{U^{n-j}}} U^n$.

For all expressions E with $\mathrm{FV}(E) \subseteq \vec{\alpha}$ we define the interpretation $[\![M]\!]_{\vec{\alpha}} : U^\perp \to U^n$, $[\![\pi]\!]_{\vec{\alpha}} : U \to U^n$ and $[\![P]\!]_{\vec{\alpha}} : \mathbf{1} \to U^n$ as follows ($n = \sharp\vec{\alpha}$):

$$[\![\alpha_i]\!]_{\vec{\alpha}} = \iota_j{}^n \qquad [\![M \cdot \pi]\!]_{\vec{\alpha}} = [\mathsf{alg}_{U^n} \circ ?[\![M]\!]_{\vec{\alpha}}, [\![\pi]\!]_{\vec{\alpha}}] \circ \mathrm{Ap} \qquad [\![\mathsf{cdr}(\pi)]\!]_{\vec{\alpha}} = [\![\pi]\!]_{\vec{\alpha}} \circ \mathrm{La} \circ (w_{U^\perp} \parr \mathsf{id}_U) \circ \rho_U$$

$$[\![\mathsf{nil}]\!]_{\vec{\alpha}} = w_{U^n} \circ \vartheta \qquad [\![\mathsf{car}(\pi)]\!]_{\vec{\alpha}} = [\![\pi]\!]_{\vec{\alpha}} \circ \mathrm{La} \circ (\varepsilon_{U^\perp} \parr w_U) \circ \lambda_{U^\perp} \qquad [\![\mu\beta.P]\!]_{\vec{\alpha}} = \Lambda([\![P]\!]_{\vec{\alpha},\beta})$$

$$[\![M \star \pi]\!]_{\vec{\alpha}} = [\mathsf{id}_{U^n}, [\![\pi]\!]_{\vec{\alpha}}] \circ \Lambda^{-1}([\![M]\!]_{\vec{\alpha}})$$

Note that the denotations of stacks are ?-algebra morphisms and it is not difficult to verify that the above interpretation is invariant under $\to_{\mathsf{s}}$-reduction. To see that check before that $[\![E\{\pi/\beta\}]\!]_{\vec{\alpha}} = [\mathsf{id}_{\vec{\alpha}}, [\![\pi]\!]_{\vec{\alpha}}] \circ [\![E]\!]_{\vec{\alpha},\beta}$.

The category **Rel** of sets and relations is a $*$-autonomous linear category that satisfies all our requirements [26]. If S is a set, we denote by $\mathscr{M}_{\mathrm{f}}(S)^{(\omega)}$ the set of all the $\mathbb{N}$-indexed sequences $\sigma = (a_1, a_2, \dots)$ of multisets over S such that $a_i = []$ holds for all but a finite number of indices $i \in \mathbb{N}$. The set $\mathscr{M}_{\mathrm{f}}(S)^{(\omega)}$ is a simple example of ?-algebra of **Rel**. For $\sigma = (a_1, a_2, \dots)$ and $\tau = (b_1, b_2, \dots)$, we define $\sigma + \tau = (a_1 \uplus b_1, a_2 \uplus b_2, \dots)$ and $* = ([], [], \dots)$. Then the relations $w = \{(1, *)\}$ and $c = \{((\sigma, \tau), \sigma + \tau) : \sigma, \tau \in \mathscr{M}_{\mathrm{f}}(S)^{(\omega)}\}$ make $(\mathscr{M}_{\mathrm{f}}(S)^{(\omega)}, c, w)$ a $\parr$-monoid in **Rel**. The operation $+$ on $\mathscr{M}_{\mathrm{f}}(S)^{(\omega)}$ can also be extended componentwise to $(\mathscr{M}_{\mathrm{f}}(S)^{(\omega)})^k$ (whose elements are ranged over by $\vec{\sigma}, \vec{\tau}, \dots$) transferring thereby the monoid structure.

In order to model the untyped calculus we need a $\parr$-monoid U of together with two relations $\mathrm{La} \subseteq (\mathscr{M}_{\mathrm{f}}(U) \times U) \times U$ and $\mathrm{Ap} \subseteq U \times (\mathscr{M}_{\mathrm{f}}(U) \times U)$ satisfying $\mathrm{Ap} \circ \mathrm{La} = \mathsf{id}_{\mathscr{M}_{\mathrm{f}}(U) \times U}$ and a relation $\vartheta \subseteq U \times \{1\}$. In the category **Rel** lives one such object $\mathscr{D} = (D, \mathrm{Ap}, \mathrm{La})$ that has already been encountered many times in the literature (see for example [2]) as a model of the ordinary λ-calculus (as well as of some of its extensions). The object is constructed as union $D = \bigcup_{n \in \mathbb{N}} D_n$ of a family of sets $(D_n)_{n \in \mathbb{N}}$ defined by $D_0 = \emptyset$ and $D_{n+1} = \mathscr{M}_{\mathrm{f}}(D_n)^{(\omega)}$. Given $\sigma = (a_1, a_2, a_3, \dots) \in D$ and $a \in \mathscr{M}_{\mathrm{f}}(D)$, we write $a :: \sigma$ for the element $(a, a_1, a_2, a_3, \dots) \in D$. Since $D = \mathscr{M}_{\mathrm{f}}(D)^{(\omega)}$, as previously observed it has a standard monoid structure and we can set $\mathrm{La} = \{((a, \sigma), a :: \sigma) : a \in \mathscr{M}_{\mathrm{f}}(D),\ \sigma \in D\}$ and $\mathrm{Ap} = \{(a :: \sigma, (a, \sigma)) : a \in \mathscr{M}_{\mathrm{f}}(D),\ \sigma \in D\}$ satisfying the desired equation; as a matter of fact also the equation $\mathrm{La} \circ \mathrm{Ap} = \mathsf{id}_U$ holds and the interpretation of expressions is invariant under $\to_{\mathsf{s}\eta}$-reduction. Finally $\vartheta = \{(*, 1)\}$.

The isomorphism $\Lambda : \mathbf{C}(\mathbf{1}, U \parr U) \to \mathbf{C}(U^\perp, U)$ is trivially given by $\Lambda(f) = \{(\alpha, \beta) : (1, (\beta, \alpha)) \in f\}$.

The interpretation is concretely defined as follows:

$$[\![\alpha_i]\!]_{\vec{\alpha}} = \{(\sigma,(*,..,\sigma,..,*)) : \sigma \in D\}; \qquad [\![\mathsf{cdr}(\pi)]\!]_{\vec{\alpha}} = \{(\sigma,\vec{\tau}) : ([]::\sigma,\vec{\tau}) \in [\![\pi]\!]_{\vec{\alpha}}\};$$
$$[\![\mathsf{car}(\pi)]\!]_{\vec{\alpha}} = \{(\sigma,\vec{\tau}) : ([\sigma]::*,\vec{\tau}) \in [\![\pi]\!]_{\vec{\alpha}}\}; \qquad [\![\mu\beta.P]\!]_{\vec{\alpha}} = \{(\sigma,\vec{\tau}) : (1,(\vec{\tau},\sigma)) \in [\![P]\!]_{\vec{\alpha},\beta}\};$$
$$[\![M\centerdot\pi]\!]_{\vec{\alpha}} = \{([\sigma_1,..,\sigma_k]::\sigma,\Sigma_{i=0}^{k}\vec{\tau}_i) : k \geq 0, \forall i = 1,..,k.\ (\sigma_i,\vec{\tau}_i) \in [\![M]\!]_{\vec{\alpha}},\ (\sigma,\vec{\tau}_0) \in [\![\pi]\!]_{\vec{\alpha}}\};$$
$$[\![M \star \pi]\!]_{\vec{\alpha}} = \{(1,\vec{\tau}+\vec{\tau}') : \exists\sigma \in D.\ (\sigma,\vec{\tau}) \in [\![M]\!]_{\vec{\alpha}},\ (\sigma,\vec{\tau}') \in [\![\pi]\!]_{\vec{\alpha}}\}; \qquad [\![\mathsf{nil}]\!]_{\vec{\alpha}} = \{(*,(*,..,*))\}.$$

For example for the stack calculus version of call/cc we have
$[\![\mu\alpha.\alpha[0]\star(\mu\beta.\beta[0]\star\alpha[1])\centerdot\alpha[1]]\!] = \{[[[\sigma_1]::*,..,[\sigma_k]::*]::\sigma_0]::(\Sigma_{i=0}^{k}\sigma_i) : k \geq 0,\ \sigma_0,..,\sigma_k \in D\}$.

7 Conclusions

We introduced the stack calculus, a finitary functional calculus with simple syntax and rewrite rules in which the calculi introduced so far in the Curry–Howard correspondence for classical logic can be faithfully encoded; instead of exhibiting comparisons with all the existing formalisms, we just showed how Parigot's $\lambda\mu$-calculus can be translated into our calculus.

We proved that the untyped stack calculus enjoys confluence, and that types enforce strong normalization. The typed fragment is a sound and complete system for full implicational Classical Logic. The type system that Lafont et al. [19] use for the λ-calculus with pairs may be used to type stack expressions within the $\{\wedge,\neg,\bot\}$-fragment of Intuitionistic Logic: under this point of view, the stack calculus is the target-language of a CPS translation from itself that alters the types but not the expressions of the calculus. In the classically-typed system ($\{\to,\bot\}$-fragment of Classical Logic) the arrow type corresponds to the stack constructor; for this reason the realizability interpretation of types à la Krivine matches perfectly the logical meaning of the arrow in the type system. The proofs of soundness and strong normalization of the calculus are both given by particular realizability interpretations.

We defined a Krivine machine that executes the terms of stack calculus. We showed how to encode control mechanisms like *label/resume* and *raise/handle* in the stack calculus which are soundly executed by our machine. This approach seems to be simpler than the extension of ML with exceptions studied in De Groote [8].

Inspired by Laurent and Regnier's work [23], we give a simple categorical framework to interpret the expressions of both typed and untyped stack calculus. We show how, in the case of a relational semantics, this famework allows a simple calculation of the interpretation of expressions.

References

[1] Z.M. Ariola & H. Herbelin (2003): *Minimal classical logic and control operators*. In: *ICALP*, pp. 871–885.

[2] A. Bucciarelli, T. Ehrhard & G. Manzonetto (2007): *Not Enough Points Is Enough*. In: *CSL*, *LNCS* 4646, pp. 298–312.

[3] P.-L Curien & H. Herbelin (2000): *The duality of computation*. In: *ACM SIGPLAN International Conference on Functional Programming*, pp. 233–243.

[4] J. Czermak (1977): *A Remark on Gentzen's Calculus of Sequents*. Notre Dame Journal of Formal Logic 18(3), pp. 471–474.

[5] V. Danos, J.-B. Joinet & H. Schellinx (1995): *LKQ and LKT: Sequent calculi for second order logic based upon dual linear decompositions of classical implication*. In J.-Y. Girard, Y. Lafont & L. Regnier, editors: *Advances in linear logic*, *London Math. Society Lecture Note Series* 222.

[6] R. David & W. Py (2001): *$\lambda\mu$-Calculus and Böhm's Theorem*. *J. Symb. Log.* 66(1), pp. 407–413.

[7] P. De Groote (1994): *On the relation between the $\lambda\mu$-calculus and the syntactic theory of sequential control*. In: *LPAR*, pp. 31–43.

[8] P. De Groote (1995): *A Simple Calculus of Exception Handling*. In: *TLCA*, pp. 201–215.

[9] P. De Groote (1998): *An environment machine for the lambda-mu-calculus*. *Math. Struct. in Comp. Sci.* 8(6), pp. 637–669.

[10] M. Felleisen & R. Hieb (1992): *The Revised Report on the Syntactic Theories of Sequential Control and State*. *Theor. Comput. Sci.* 103, pp. 235–271.

[11] G. Gentzen (1935): *Investigations into logical deduction*.

[12] J.-Y. Girard (1986): *The system F of variable types, fifteen years later*. *Theor. Comput. Sci.* 45, pp. 159–192.

[13] J.-Y. Girard (1991): *A new constructive logic: Classical Logic*. *Math. Struct. in Comp. Sci.* 1(3), pp. 255–296.

[14] T. Griffin (1990): *A Formulae-as-Types Notion of Control*. In: *POPL*, pp. 47–58.

[15] M. Hofmann & T. Streicher (1997): *Continuation Models are Universal for lambda-mu-Calculus*. In: *LICS*, pp. 387–395.

[16] W.A. Howard (1980): *The formulae-as-types notion of construction*. In J.R. Hindley & J.P. Seldin, editors: *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pp. 479–490.

[17] J.W. Klop & R.C. de Vrijer (1989): *Unique normal forms for lambda calculus with surjective pairing*. *Information and Computation* 2, pp. 97–113.

[18] J.-L. Krivine (2001): *Typed lambda-calculus in classical Zermelo-Frænkel set theory*. *Arch. Math. Log.* 40(3), pp. 189–205.

[19] Y. Lafont, B. Reus & T. Streicher (1993): *Continuations Semantics or Expressing Implication by Negation*. Technical Report 9321, Ludwig-Maximilians-Universitat, Munchen. Technical Report.

[20] O. Laurent (2003): *Krivine's abstract machine and the lambda mu-calculus (an overview)*. Unpublished note.

[21] O. Laurent (2003): *Polarized proof-nets and lambda-mu calculus*. *Theor. Comput. Sci.* 290(1), pp. 161–188.

[22] O. Laurent (2011): *Intuitionistic Dual-intuitionistic Nets*. *J. Log. Comput.* 21(4), pp. 561–587.

[23] O. Laurent & L. Regnier (2003): *About Translations of Classical Logic into Polarized Linear Logic*. In: *LICS*, pp. 11–20.

[24] S. Lengrand (2003): *Call-by-value, call-by-name, and strong normalization for the classical sequent calculus*. *Elec. Notes in Theor. Comp. Sci.* 86. WRS.

[25] T. Low & T. Streicher (2006): *Universality Results for Models in Locally Boolean Domains*. In: *CSL*, pp. 456–470.

[26] P.-A. Melliès: *Categorical semantics of linear logic*. Available at `http://www.pps.jussieu.fr/~mellies/papers/panorama.pdf`. Panoramas et Synthèses 27, Société Mathématique de France, 2009.

[27] M. Parigot (1992): *$\lambda\mu$-calculus: An Algorithmic Interpretation of Classical Natural Deduction*. In: *LPAR*, pp. 190–201.

[28] D. Prawitz (1965): *Natural Deduction - a proof theoretical study*. Almqvist & Wiksell, Stokholm.

[29] B. Reus & T. Streicher (1998): *Classical Logic, Continuation Semantics and Abstract Machines*. *J. Funct. Program.* 8(6), pp. 543–572.

[30] P. Selinger (2001): *Control categories and duality: on the categorical semantics of the lambda-mu calculus*. *Math. Struct. in Comp. Sci.* 11, pp. 207–260.

[31] R. Smullyan (1968): *First-order logic*. Springer-Verlag, New York.

[32] C. Urban (2000): *Classical Logic and Computation*. Ph.D. thesis, University of Cambridge Comp. Laboratory.