Characterization of strong normalizability for a sequent lambda calculus with co-control

José Espírito Santo
Centro de Matemática, Universidade do Minho
Portugal
jes@math.uminho.pt

Silvia Ghilezan
University of Novi Sad, Mathematical Institute SANU
Serbia
gsilvia@uns.ac.rs

ABSTRACT

We study strong normalization in a lambda calculus of proof-terms with co-control for the intuitionistic sequent calculus. In this sequent lambda calculus, the management of formulas on the left hand side of typing judgements is "dual" to the management of formulas on the right hand side of the typing judgements in Parigot's lambda-mu calculus - that is why our system has first-class "co-control". The characterization of strong normalization is by means of intersection types, and is obtained by analyzing the relationship with another sequent lambda calculus, without co-control, for which a characterization of strong normalizability has been obtained before. The comparison of the two formulations of the sequent calculus, with or without co-control, is of independent interest. Finally, since it is known how to obtain bidirectional natural deduction systems isomorphic to these sequent calculi, characterizations are obtained of the strongly normalizing proof-terms of such natural deduction systems.

CCS CONCEPTS

• **Theory of computation → Proof theory**; **Type theory**;

KEYWORDS

intersection types, strong normalization, sequent calculus, bidirectional natural deduction, co-control

ACM Reference format:
José Espírito Santo and Silvia Ghilezan. 2017. Characterization of strong normalizability for a sequent lambda calculus with co-control. In *Proceedings of PPDP'17, Namur, Belgium, October 9–11, 2017,* 12 pages.
https://doi.org/10.1145/3131851.3131867

1 INTRODUCTION

We study strong normalization in the system $\overline{\lambda\tilde{\mu}}$, a lambda calculus of proof-expressions with co-control for the intuitionistic sequent calculus [9]. In this sequent lambda calculus, the management of formulas on the left hand side of typing judgements is "dual" to the management of formulas on the right hand side of the typing judgements in Parigot's $\lambda\mu$-calculus [17] - that is why our system has first-class "co-control".

In $\overline{\lambda\tilde{\mu}}$, there is a variant of the $\tilde{\mu}$-operator of Curien and Herbelin's $\overline{\lambda\mu\tilde{\mu}}$-calculus [3] which, when appearing in the hole of a "co-continuation", may trigger a kind of dual "structural substitution" (a "co-continuation" is, roughly, a non-value with a hole in "tail" position, while a continuation is a non-value with a hole in head position). This is the reduction rule that defines the behavior of $\tilde{\mu}$ and where co-control operation is concentrated. Such a rule coexists with four other reduction rules which, together, reduce expressions of $\overline{\lambda\tilde{\mu}}$ to a form corresponding to the cut-free proofs of LJT [12] - hence, logically, the reduction rules express a combination of cut-elimination and focalization [15]. It is known [9] that the typable $\overline{\lambda\tilde{\mu}}$-terms are strongly normalizing.

Since the seminal work of Coppo and Dezani [2], intersection types became a powerful tool for characterizing strong normalization in different frameworks [1, 4, 5]. We employ them to obtain a characterization of strong normalizability in $\overline{\lambda\tilde{\mu}}$ as typability in a certain intersection-type assignment system. The system we propose for $\overline{\lambda\tilde{\mu}}$ is obtained by adapting the system for assigning intersection types used to characterize the strongly normalizing proof-terms of λGtz, in previous work by the authors and colleagues [10, 11]. The λGtz-calculus [7] is another sequent lambda calculus, where the treatment of the $\tilde{\mu}$-operator follows the original and simpler one found in [3]: the $\tilde{\mu}$-operator is a term-substitution former, and its reduction principle triggers an ordinary term substitution.

The characterization of strong normalizability in $\overline{\lambda\tilde{\mu}}$ is proved, not by re-running the proof for λGtz, but by "lifting" the characterization in λGtz. This requires a detailed comparison of the two rewriting systems, which is of independent interest, as it highlights sensitive choice points in the design of calculi of proof terms for the sequent calculus, particularly the treatment of proof-term variables and the related substitution principles.

Finally, since it is known how to obtain bidirectional natural deduction systems isomorphic to these sequent calculi [7, 9], characterizations are obtained of the strongly normalizing proof-terms of such natural deduction systems. To the best of our knowledge, it is the first time that intersection type systems are formulated in the bidirectional style. As we will see, such combination is quite revealing.

Overview of the paper. In Section 2 we recall our departure systems λGtz and its intersection-type system. Next, in Section 3, we recall system $\overline{\lambda\tilde{\mu}}$ and introduce an intersection-type system for it, and give the immediate connection at level of typability between the two type systems. The challenge is the relationship between the reduction systems of λGtz and $\overline{\lambda\tilde{\mu}}$ and the respective notions

PPDP'17, October 9–11, 2017, Namur, Belgium

ACM ISBN 978-1-4503-5291-8/17/10…$15.00
https://doi.org/10.1145/3131851.3131867

of strong normalizability. We are led to make several adjustments to λGtz in Section 4 to establish the required equivalence. For the adjusted λGtz, one has to recheck that the original type system still characterizes strong normalizability - this is done in Section 5. Once this is completed, the characterization for $\overline{\lambda\tilde{\mu}}$ follows easily, and is proved in the same section. Section 6 recalls the bidirectional natural deduction system $\underline{\lambda\text{let}}$ [9] and introduces an intersection-type system for it which characterizes strong normalizability, profiting from the isomorphism with $\overline{\lambda\tilde{\mu}}$. Section 7 concludes.

2 BACKGROUND

In this section, we recall the sequent calculus λGtz of [7] and its characterization of strong normalizability taken from [10, 11].

2.1 The λGtz-calculus

The abstract syntax of λGtz is given by:

$$\begin{array}{lrcl} \text{Terms} & t,u,v & ::= & x \mid \lambda x.t \mid tk \\ \text{Contexts} & k & ::= & \widehat{x}.v \mid u :: k \end{array}$$

where x ranges over a denumerable set of term variables. As usual, $\lambda x.$ and $\widehat{x}.$ are considered to be binders and the sets of free variables $Fv(t)$ and $Fv(k)$ are defined accordingly. We work modulo α-conversion and adopt Barendregt's convention and assume that free and bound variables of a term are different.

One way of seeing this system is as a variant of the λ-calculus where the application constructor has been replaced by the constructor tk, which we call a *cut*. A context k has the general form

$$u_1 :: \cdots :: u_m :: \widehat{x}.v \ ,$$

for some $m \geq 0$. Here we assume the expression to be bracketed from the right, with u_1 (resp. $\widehat{x}.v$) at the surface (resp. bottom) of the expression. If $m = 0$, then tk is $t(\widehat{x}.v)$ and may be seen as an explicit substitution; otherwise, tk is an enlarged concept of function application, with t the function expression, u_i's the arguments, and $\widehat{x}.v$ the awaiting substitution. This enlarged concept of substitution gives the possibility of chaining more than one argument (when $m > 1$), and is "generalized", in the sense of [13], because it contains the refereed awaiting substitution.

The reduction rules of λGtz are the following:

$$\begin{array}{lrcl} (\beta) & (\lambda x.t)(u :: k) & \to & u(\widehat{x}.tk) \\ (\pi) & (tk)k' & \to & t(k@k') \\ (\sigma) & t(\widehat{x}.v) & \to & [t/x]v \\ (\mu) & \widehat{x}.xk & \to & k, \text{ if } x \notin k \end{array}$$

where $[u/x]t$ (or $[u/x]k$) denotes meta-substitution, and $k@k'$ is defined by $(u :: k)@k' = u :: (k@k')$ and $(\widehat{x}.v)@k' = \widehat{x}.vk'$.

Rule β generates an explicit substitution, which can be executed by rules σ. Rule π profits from the fact that contexts can be appended to simplify the function expression. The $\beta\pi\sigma$-normal forms of λGtz are given by:

$$\begin{array}{lrcl} (\beta\pi\sigma\text{-normal terms}) & t_{nf},u_{nf},v_{nf} & ::= & x \mid \lambda x.t_{nf} \\ & & \mid & x(u_{nf} :: k_{nf}) \\ (\beta\pi\sigma\text{-normal contexts}) & k_{nf} & ::= & \widehat{x}.t_{nf} \mid u_{nf} :: k_{nf} \end{array}$$

As to rule μ, suppose k is a μ-redex. If k occurs in tk, then the μ-reduction

$$t(\widehat{x}.xk') \to tk' \tag{1}$$

is a σ-reduction as well. If k occurs in $u :: k$, then the μ-reduction $u :: (\widehat{x}.xk') \to u :: k'$ removes an unnecessary break in the chain of arguments.

The system λGtz comes with a type system for assigning simple types [7]. Logically, this type system is a sequent calculus for intuitionistic implicational logic, and the expressions are proof-expressions for this proof system: x corresponds to the axiom, $\lambda x.t$ and $u :: k$ correspond respectively to right and left introduction of implication, tk corresponds to cut, and $\widehat{x}.v$ corresponds to the inference that selects or activates a formula on the antecedent of the sequent (while deactivation is captured by the cut xk). Moreover, while rule μ corresponds to the elimination of a redundant sequence of deactivation followed by activation of the same formula, the other reduction rules correspond to cut-elimination rules: β is the main step, with the cut-formula principal in both premises, whereas σ (resp. π) reduces a cut whose cut-formula is not principal in the right (resp. left) premise.

We do not give more details about this type system, because a generalization of it for assigning intersection types is recalled next.

2.2 Intersection types

The abstract syntax of *intersection types* is given by:

$$A,B \ ::= \ p \mid A \to B \mid A \cap B$$

where p ranges over a denumerable set of type variables. We assume that intersection types are commutative $A \cap B \equiv B \cap A$, associative $(A \cap B) \cap C \equiv A \cap (B \cap C)$ and idempotent $A \cap A \equiv A$. Furthermore, we identify $A \to (B \cap C) \equiv (A \to B) \cap (A \to C)$.

The intersection type system for λGtz is given in Fig. 1. In these rules, $\cap A_i = A_1 \cap \cdots \cap A_n$, for some $n \geq 1$. There are two kinds of sequents (type assignments) $\Gamma \vdash t : A$ for typing terms and $\Gamma; B \vdash k : A$ for typing contexts. The sequents $\Gamma \vdash t : A$ and $\Gamma; B \vdash k : A$ are *derivable* (in λGtz) if they are derivable in the system of Fig. 1. The term t is *typable* if, for some A, $\Gamma \vdash t : A$ is derivable.

Proposition 2.1 (Generation lemma).

(1) $\Gamma \vdash x : A$ *iff* $x : \cap A_i \in \Gamma$ *and* $A \equiv A_i$ *for some* $i \in \{1,\ldots,n\}$.
(2) $\Gamma \vdash \lambda x.t : A$ *iff* $A \equiv B \to C$ *and* $\Gamma, x : B \vdash t : C$.
(3) $\Gamma; B \vdash \widehat{x}.t : A$ *iff* $\Gamma, x : B \vdash t : A$.
(4) $\Gamma \vdash tk : A$ *iff there exists* $B \equiv \cap B_i$ *and* $\Gamma; B \vdash k : A$ *and* $\Gamma \vdash t : B_i$ *for all* $i \in \{1,\ldots,n\}$.
(5) $\Gamma; B \vdash t :: k : A$ *iff* $B \equiv \cap C_i \to D$ *and* $\Gamma; D \vdash k : A$ *and* $\Gamma \vdash t : C_i$ *for all* $i \in \{1,\ldots,n\}$.

Proof. Easy since the system in Fig. 1 is syntax-directed. □

The rest of the section deals with the main properties of this intersection type system for λGtz, proven in [10, 11], that are relevant for the new results presented in this paper.

Theorem 2.2 (Subject Reduction). *If* $\Gamma \vdash t : A$ *and* $t \to t'$, *then* $\Gamma \vdash t' : A$.

Proposition 2.3 (Typability of normal forms). *$\beta\pi\sigma$-normal forms of λGtz calculus are typable in the system of Fig. 1.Hence so are $\beta\pi\sigma\mu$-normal forms.*

Figure 1: Intersection types for the λGtz-calculus

$$\frac{\exists i \in \{1,\cdots,n\}\, A = A_i}{\Gamma, x : \cap A_i \vdash x : A}\ (Ax) \qquad \frac{\Gamma \vdash t : A_i,\ \forall i \in \{1,\cdots,n\} \qquad \Gamma; \cap A_i \vdash k : B}{\Gamma \vdash tk : B}\ (Cut) \qquad \frac{\Gamma, x : A \vdash v : B}{\Gamma; A \vdash \widehat{x}.v : B}\ (Sel)$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ (\to_R) \qquad \frac{\Gamma \vdash t : A_i,\ \ \forall i \in \{1,\cdots,n\} \qquad \Gamma; B \vdash k : C}{\Gamma; \cap A_i \to B \vdash t :: k : C}\ (\to_L)$$

Figure 2: Typing rules of the $\overline{\lambda\tilde{\mu}}$-calculus

$$Ax\ \frac{}{\Gamma | A \vdash [] : A} \qquad Cut\ \frac{\Gamma \vdash t : A \quad \Gamma | A \vdash k : B}{\Gamma \vdash tk : B}$$

$$L\supset\ \frac{\Gamma \vdash u : A \quad \Gamma | B \vdash k : C}{\Gamma | A \supset B \vdash u :: k : C} \qquad R\supset\ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \supset B}$$

$$Pass\ \frac{\Gamma, x : A | A \vdash k : B}{\Gamma, x : A \vdash x\hat{}k : B} \qquad Act\ \frac{\Gamma, x : A \vdash v : B}{\Gamma | A \vdash \tilde{\mu}x.v : B}$$

Theorem 2.4 (Characterization of SN in λGtz). *A term is strongly normalising in λGtz if and only if it is typable in the system given in Fig. 1.*

3 THE CALCULUS WITH CO-CONTROL

In this section, we first recall the system $\overline{\lambda\tilde{\mu}}$ from [9] and then we propose an intersection type system for it.

3.1 The $\overline{\lambda\tilde{\mu}}$-calculus

The abstract syntax of $\overline{\lambda\tilde{\mu}}$ is given by the following grammar:

$$\begin{array}{rrcl} \text{(Terms)} & t, u, v & ::= & \lambda x.t \mid x\hat{}k \mid tk \\ \text{(Generalized vectors)} & k & ::= & [] \mid \tilde{\mu}x.v \mid u :: k \end{array}$$

This is a syntax of proof-expressions for the sequent calculus in Fig. 2, which is a proof system for intuitionistic implicational logic. The system handles two kinds of sequents: $\Gamma \vdash t : A$ and $\Gamma | A \vdash k : B$. The distinguished formula A in the latter is not exactly a "stoup" or a focused formula, because the operator $\tilde{\mu}x.t$ may select an arbitrary formula from the context Γ. The construction $x\hat{}k$ comes from Herbelin's $\overline{\lambda}$-calculus [12], but here it forms a pair with $\tilde{\mu}x.t$: logically, these are an activation/passification pair, in the style of the $\lambda\mu$-calculus, but acting on the l.h.s. of sequents.

The other reading of the system in Fig. 2 is as a system for assigning simple types to expressions that come, not from some anonymous syntax, but rather from an interesting variant of λ-calculus: $\overline{\lambda\tilde{\mu}}$ is a formal, relaxed, vector notation for λ-terms with first class co-control [9]. Let us see what this means.

In the vector notation $\overline{\lambda\tilde{\mu}}$ one has three possible forms for terms, namely $\lambda x.t$, $x\hat{}k$ and tk, while the original, informal, vector notation consists of the forms $\lambda x.M$, $x\vec{N}$, and $(\lambda x.M)N\vec{N}$. The third form of the former notation is more general, and this explains the qualification "relaxed". In $\overline{\lambda\tilde{\mu}}$, a variable x is not a term, and the third form cannot result from substitution of t for x in the second form. An immediate question is: what does a variable in $\overline{\lambda\tilde{\mu}}$ stand for. In addition, $\overline{\lambda\tilde{\mu}}$ has primitive syntax for the vectors themselves: in their inductive definition one finds the empty vector [] and the vector constructor $u :: k$, but also another base case, the co-control operator $\tilde{\mu}x.v$.

Figure 3: Reduction rules of $\overline{\lambda\tilde{\mu}}$

$$\begin{array}{lrcl} (\beta) & (\lambda x.t)(u :: k) & \to & (u(\tilde{\mu}x.t))k \\ (\tilde{\mu}) & \mathcal{H}[\tilde{\mu}x.t] & \to & [\mathcal{H}/x]t \\ (\epsilon) & t[] & \to & t \\ (\pi_1) & (x\hat{}k)k' & \to & x\hat{}(k@k') \\ (\pi_2) & (tk)k' & \to & t(k@k') \end{array}$$

The notation $\tilde{\mu}x.t$ comes from $\overline{\lambda}\mu\tilde{\mu}$ [3]; but here, contrary to what happens in $\overline{\lambda}\mu\tilde{\mu}$, the reduction rule that defines the behavior of $\tilde{\mu}$ does not trigger a term-substitution, it triggers a kind of "structural substitution" as found in the $\lambda\mu$-calculus, whose actual parameter is a certain context captured in the reduction step.

The notion of context that $\tilde{\mu}$ captures is this:

$$\mathcal{H} ::= x\hat{}[\cdot] \mid t([\cdot]) \mid \mathcal{H}[u :: [\cdot]] \tag{2}$$

These expressions are called *co-continuations*. Informally, a co-continuation is a context with one of the forms

$$x\hat{}(u_1 :: \cdots :: u_m :: [\cdot]) \quad \text{or} \quad t(u_1 :: \cdots :: u_m :: [\cdot])\ .$$

The operator $u :: k$ is "right associative", so the hole $[\cdot]$ is under a chain of arguments, but at "tail" position, while in a continuation $[\cdot]N_1 \cdots N_m$ of the λ-calculus or $\lambda\mu$-calculus the hole is in "head" position. The co-control rule $\tilde{\mu}$ triggers *co-continuation substitution* $[\mathcal{H}/x]\_$, in whose definition the only non-routine case is:

$$[\mathcal{H}/x](x\hat{}k) = \mathcal{H}[k'] \text{ with } k' = [\mathcal{H}/x]k\ . \tag{3}$$

This equation says that variables in $\overline{\lambda\tilde{\mu}}$ stand for co-continuations, and that the context k in $x\hat{}k$ will be filled in the hole of $\mathcal{H}$, if $\mathcal{H}$ happens to be substituted for x.

The reduction rules of $\overline{\lambda\tilde{\mu}}$ are in Fig. 3. Let $\pi := \pi_1 \cup \pi_2$. The π_i-rules employ concatenation of generalized vectors $k@k'$, defined by the obvious equations $[]@k' = k'$ and $(u :: k)@k' = u :: (k@k')$, together with $(\tilde{\mu}x.t)@k' = \tilde{\mu}x.tk'$. Rule $\tilde{\mu}$ can be partitioned into three cases:

$$\begin{array}{lrcl} (\rho) & y\hat{}(\tilde{\mu}x.t) & \to & [y\hat{}[\cdot]/x]t \\ (\sigma) & u(\tilde{\mu}x.t) & \to & [u([\cdot])/x]t \\ (\tau) & \mathcal{H}[u :: \tilde{\mu}x.t] & \to & [\mathcal{H}[u :: [\cdot]]/x]t \end{array}$$

Rule $\tilde{\mu}$ eliminates all occurrences of the $\tilde{\mu}$-operator. The remaining rules eliminate all occurrences of cuts tk. So the $\beta\tilde{\mu}\epsilon\pi$-normal forms correspond to a well-known representation of β-normal λ-terms, namely cut-free $\overline{\lambda}$-terms [12]. Hence, the reduction rules of $\overline{\lambda\tilde{\mu}}$ not only perform cut-elimination, but also focalization [15], as the normal forms live in the focused fragment LJT.

Here are some computational intuitions about the reduction rules. The $\pi\epsilon$-normal forms are the terms with the forms $\lambda x.t$, $x\hat{}k$ and $(\lambda x.t)(u :: k)$: these are in correspondence with the three forms of the informal vector notation for λ-terms - so the mentioned rules eliminate the "relaxed" aspect of the vector notation. We are disregarding for a moment that the vectors k may start from an occurrence of the co-control operator. Rule $\tilde{\mu}$ eliminates such possibility, so the $\tilde{\mu}\pi\epsilon$-normal are a formal vector notation in perfect correspondence with the informal one: we may see such formal notation as living inside $\overline{\lambda}$-calculus: in fact they constitute the fragment of $\overline{\lambda}$ proved isomorphic to the ordinary λ-calculus in [6]. Finally, $\beta\tilde{\mu}\pi\epsilon$-normal forms are a formal vector notation for the β-normal λ-terms, only allowing the forms $\lambda x.t$ and $x\hat{}k$.

Even if we start from a $\tilde{\mu}\pi\epsilon$-normal form, reduction in $\overline{\lambda\tilde{\mu}}$ is far from being confined to proceed as in the ordinary λ-calculus. For instance, reduction is "agnostic" [9] to the call-by-value vs call-by-name dilemma [3]: if we start from $(\lambda x.t)(u :: k)$ and u is another cut $t'k'$, then a β-step produces $(u(\tilde{\mu}x.t))k$ and now we may proceed with a σ-step, executing the explicit substitution $u(\tilde{\mu}x.t)$ (the call-by-name option), or perform a series of π-steps (the call-by-value option), to obtain $t'(k'@(\tilde{\mu}x.tk))$, where the function expression t' of cut u is now the function expression of the whole expression.

To finish this subsection, we stress the parallel between the μ-operator of $\lambda\mu$ and the $\tilde{\mu}$-operator of $\overline{\lambda\tilde{\mu}}$. In $\lambda\mu$ there are three reduction rules for the μ-operator: the main rule, which reduces $\lambda\mu$-terms of the form $C[(\mu a.M)N]$, with $C = [\cdot]N_1 \cdots N_m$; the renaming rule; and the η-rule, which reads $\mu a.[a]M \to M$, if $a \notin M$.

- The redex for the main rule of $\lambda\mu$ is $(\mu a.M)N$ filled in the hole of continuation C, while the particular case τ of reduction rule $\tilde{\mu}$ has redexes consisting of $u :: (\tilde{\mu}x.t)$ filled in the hole of a co-continuation $\mathcal{H}$.
- The particular case ρ of reduction rules $\tilde{\mu}$ is a renaming rule because the particular case $[y\hat{}[\cdot]/x]t$ of context substitution is very much like a substitution operation that renames variables, since the critical case of its definition reads

$$[y\hat{}[\cdot]/x](x\hat{}k) = y\hat{}k' \text{ with } k' = [y\hat{}[\cdot]/x]k \ .$$

- From reduction rule $\tilde{\mu}$ we derive the η-rule for $\tilde{\mu}$, *i.e.*

$$\mathcal{H}[\tilde{\mu}x.x\hat{}k] \to \mathcal{H}[k] \tag{4}$$

with $x \notin k$.[1] In fact, if $\mathcal{H}$ has the first (resp. second, third) form in the inductive definition (2), then (4) corresponds to a particular case of rule ρ (resp. σ, τ).

Further computational intuitions about $\overline{\lambda\tilde{\mu}}$ are obtained from formal comparisons with other calculi, particularly from the isomorphism with a specific natural deduction system, obtained in [9] and recalled later in Section 6.

[1] This is in the same spirit of λGtz's rule named μ!

Figure 4: Maps between $\overline{\lambda\tilde{\mu}}$ and λGtz

$$(\cdot)^* : \overline{\lambda\tilde{\mu}} \to \lambda\text{Gtz}$$

$$\begin{array}{rcl} (x\hat{}k)^* & = & xk^* \\ (\tilde{\mu}x.v)^* & = & \widehat{x}.v^* \\ []^* & = & \widehat{x}.x \end{array}$$

$$(\cdot)^+ : \lambda\text{Gtz} \to \overline{\lambda\tilde{\mu}}$$

$$\begin{array}{rcl} x^+ & = & x\hat{}[] \\ (tk)^+ & = & \begin{cases} t^+k^+ & \text{if } t \text{ is not a var.} \\ x\hat{}k^+ & \text{if } t = x \end{cases} \\ (\widehat{x}.v)^+ & = & \begin{cases} \tilde{\mu}x.v^+ & \text{if } v \neq x \\ [] & \text{if } v = x \end{cases} \end{array}$$

3.2 Intersection types for the $\overline{\lambda\tilde{\mu}}$-calculus

The intersection type assignment system we propose for $\overline{\lambda\tilde{\mu}}$ is "derived" from that of λGtz given before in Fig 1, having in mind how to map $\overline{\lambda\tilde{\mu}}$ into λGtz. Such a map, together with a map in the opposite direction, is given in Fig. 4, where only the non-homomorphic clauses are shown. Actually, map $(\cdot)^*$ injects $\overline{\lambda\tilde{\mu}}$ into λGtz:

Lemma 3.1. *For all $t \in \overline{\lambda\tilde{\mu}}$, $(t^*)^+ = t$.*

Proof. Proved together with $(k^*)^+ =^* k$, for $k \in \overline{\lambda\tilde{\mu}}$. The proof is a simultaneous induction on t and k, and the only piece of reasoning needed is that v^* is never a variable - hence $((tk)^*)^+ = (t^*)^+(k^*)^+$ and $((\tilde{\mu}x.v)^*)^+ = \tilde{\mu}x.(v^*)^+$. □

The intersection type assignment system we propose for $\overline{\lambda\tilde{\mu}}$ is given in Fig. 5. Again, in these rules, $\cap A_i = A_1 \cap \cdots \cap A_n$, for some $n \geq 1$. As with the typing system for λGtz, this is a syntax-directed typing system, which ensures the usual generation lemmas.

Let $t \in \overline{\lambda\tilde{\mu}}$. We say $\Gamma \vdash t : A$ is *derivable* (in $\overline{\lambda\tilde{\mu}}$) if this sequent is derivable in the system of Fig. 5. We say t is *typable* if, for some A, $\Gamma \vdash t : A$ is derivable.

Example 3.2. In λ-calculus the term $\lambda x.xx$, which is a normal form, is not typable by simple types since self-application is not typable with simple types, in turn it is typable by intersection types $\vdash \lambda x.xx : ((A \to B) \cap A) \to B$. The λGtz-term $u_2 := \lambda x.x(x :: \widehat{y}.y)$, also a normal form in λGtz, is not typable by simple types, however it is typable with the same intersection type, as is easily seen. The same is true of the $\overline{\lambda\tilde{\mu}}$-term $t := \lambda x.x\hat{}(x\hat{}[] :: [])$:

$$\dfrac{\dfrac{\dfrac{}{x : (A \to B) \cap A | A \vdash [] : A}(Ax)}{x : (A \to B) \cap A \vdash x\hat{}[] : A}(Pass) \quad \dfrac{}{x : (A \to B) \cap A | B \vdash [] : B}(Ax)}{\dfrac{x : (A \to B) \cap A | A \to B \vdash (x\hat{}[] :: []) : B}{x : (A \to B) \cap A \vdash x\hat{}(x\hat{}[] :: []) : B}(Pass)}(\to L)$$
$$\dfrac{}{\vdash \lambda x.x\hat{}(x\hat{}[] :: []) : ((A \to B) \cap A) \to B}(\to R)$$

Notice $t^* = \lambda x.x((x(\widehat{y}.y)) :: \widehat{y}.y) =: u_1$ and $u_1 \to_\sigma u_2$ and $(u_1)^+ = (u_2)^+ = t$. We will see this is a rare case where $(\cdot)^+$ collapses

Figure 5: Intersection types for the $\overline{\lambda\tilde{\mu}}$-calculus

$$\frac{\exists i \in \{1,\cdots,n\}\ A = A_i}{\Gamma| \cap A_i \vdash [] : A}\ (Ax) \qquad \frac{\Gamma \vdash t : A_i,\ \forall i \in \{1,\cdots,n\} \qquad \Gamma| \cap A_i \vdash k : B}{\Gamma \vdash tk : B}\ (Cut)$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ (\to_R) \qquad \frac{\Gamma \vdash t : A_i,\ \ \forall i \in \{1,\cdots,n\} \qquad \Gamma|B \vdash k : C}{\Gamma| \cap A_i \to B \vdash t :: k : C}\ (\to_L)$$

$$\frac{\Gamma, x : \cap A_i| \cap A_j \vdash k : B \quad \forall j\, \exists i \in \{1,\cdots,n\},\ A_j = A_i}{\Gamma, x : \cap A_i \vdash x\hat{}k : B}\ (Pass) \qquad \frac{\Gamma, x : A \vdash v : B}{\Gamma|A \vdash \tilde{\mu}x.v : B}\ (Act)$$

reduction steps. Moreover, $F(u_1) = F(u_2) = \lambda x.xx$, where F is a map to λ-calculus to be studied below.

Proposition 3.3 (Generation lemma).

(1) $\Gamma|B \vdash [] : A$ *iff* $B \equiv \cap A_i$ *and* $A \equiv A_i$ *for some* $i \in \{1,\ldots,n\}$.
(2) $\Gamma \vdash \lambda x.t : A$ *iff* $A \equiv B \to C$ *and* $\Gamma, x : B \vdash t : C$.
(3) $\Gamma \vdash tk : A$ *iff there exists* $B \equiv \cap B_i$ *and* $\Gamma; B \vdash k : A$ *and* $\Gamma \vdash t : B_i$ *for all* $i \in \{1,\ldots,n\}$.
(4) $\Gamma, x : B \vdash x\hat{}k : A$ *iff* $B \equiv \cap C_i$ *and* $\Gamma, x : B| \cap C_j \vdash k : A$, *where for each* j *there is* $i \in \{1,\ldots,n\}$ *s.t.* $C_j \equiv C_i$.
(5) $\Gamma|B \vdash t :: k : A$ *iff* $B \equiv \cap C_i \to D$ *and* $\Gamma|D \vdash k : A$ *and* $\Gamma \vdash t : B_i$ *for all* $i \in \{1,\ldots,n\}$.
(6) $\Gamma|B \vdash \tilde{\mu}x.t : A$ *iff* $\Gamma, x : B \vdash t : A$.

Proof. Easy since the system in Fig. 5 is syntax-directed. □

In designing the typing system for $\overline{\lambda\tilde{\mu}}$, the only interesting rules are those for typing $x\hat{}k$ and []: we obtained them by inverting a typing derivation of xk and $\widehat{x}.x$ in λGtz. This ensured the "only if" direction in the following result:

Theorem 3.4. *Let* $t \in \overline{\lambda\tilde{\mu}}$. t *is typable iff* t^* *is typable.*

Proof. The "only if" statement follows from this fact: (i) if $\Gamma \vdash t : A$ is derivable in $\overline{\lambda\tilde{\mu}}$ then $\Gamma \vdash t^* : A$ is derivable in λGtz, and (ii) if $\Gamma|A \vdash k : B$ is derivable in $\overline{\lambda\tilde{\mu}}$ then $\Gamma; A \vdash k^* : B$ is derivable in λGtz. The proof is a simultaneous induction on t and k. The generation lemma of $\overline{\lambda\tilde{\mu}}$ (Prop. 3.3) is used to analyze the given derivations of $\Gamma \vdash t : A$ or $\Gamma|A \vdash k : B$. As said, the two interesting cases are $t = x\hat{}k$ and $k = []$.

Case $t = x\hat{}k$. Suppose $\Gamma, x : \cap A_i \vdash x\hat{}k : B$ in $\overline{\lambda\tilde{\mu}}$. Then in $\overline{\lambda\tilde{\mu}}$, $\Gamma, x : \cap A_i| \cap A_j \vdash k : B$, where each A_j is an A_i - and this allows us to derive in λGtz, by rule Ax, $\Gamma, x : \cap A_i \vdash x : A_j$ for each j . The IH gives $\Gamma, x : \cap A_i| \cap A_j \vdash k^* : B$ in λGtz. We obtain $\Gamma, x : \cap A_i \vdash xk^* : B$ in λGtz, as required, by one application of Cut.

Case $k = []$. Suppose $\Gamma| \cap A_i \vdash [] : A$ in $\overline{\lambda\tilde{\mu}}$ with $A = A_i$ for some $i \in \{1,\cdots,n\}$. We want $\Gamma| \cap A_i \vdash \widehat{x}.x : A$ in λGtz, and this is obtained by an application of Ax followed by an application of Sel.

The "if" statement follows from two facts. The first is Lemma 3.1. The second is: if $\Gamma \vdash t : A$ is derivable in λGtz then $\Gamma \vdash t^+ : A$ is derivable in $\overline{\lambda\tilde{\mu}}$, and (ii) if $\Gamma|A \vdash k : B$ is derivable in λGtz then $\Gamma; A \vdash k^+ : B$ is derivable in $\overline{\lambda\tilde{\mu}}$. The proof is a simultaneous induction using the generation lemma of λGtz (Prop. 2.1). □

4 λGTZ REVISITED

We have analyzed the relationship between $\overline{\lambda\tilde{\mu}}$ and λGtz w.r.t. typing. Now we want to do the same w.r.t. reduction. It turns out that, in order to get satisfactory results, one needs to make several adjustments in the reduction rules of λGtz, namely:

(1) Change the definition of the rule β.
(2) Refine the definition of $k@k'$, which causes an indirect change in the definition of rule π.
(3) Add a new reduction rule named τ.
(4) Define rule μ as a relation on terms.

We call λGtz$'$ the variant of λGtz with these modifications; we refer to the definitions of λGtz as the *original* or *native* ones, while we call the definitions of λGtz$'$ the *revised* ones. Notice the syntax of expressions is not changed. So the typing system of Fig. 1 is not changed either, nor are changed the notions of derivable sequent or typable term. Let us see the four adjustments one by one.

First adjustment. The rule β now reads

$$(\lambda x.t)(u :: k) \to (u(\widehat{x}.t))k \ .$$

The contractum reduces by π to $u(\widehat{x}.tk)$, and this is the contractum of the original definition. In this paper, we prefer not to incorporate in the definition of β this reduction step, as this is the style of β in $\overline{\lambda\tilde{\mu}}$.

Second adjustment. The definition of $k@k'$ is refined in the case of $k = \widehat{x}.v$. If $v \neq x$, $(\widehat{x}.v)@k'$ is defined (as before) to be $\widehat{x}.vk'$; but now we put $(\widehat{x}.x)@k' = k'$. So, in this second case, we are incorporating in the definition the reduction step $\widehat{x}.xk' \to_\mu k'$.

Third adjustment. For the definition of rule τ, we need the concepts of co-continuation and co-continuation substitution in λGtz. In the previous papers on λGtz, no such concepts were introduced. So here we first provide the "native" definitions of co-continuation $\mathcal{H}$, substitution $[\mathcal{H}/x]\_$, and rule τ, thus completing the definition of λGtz. Afterwards "revised" definitions of substitution and τ pertaining to λGtz$'$ will be given.

Informally, a co-continuation in λGtz is a context of the form

$$t(u_1 :: \cdots :: u_m[\cdot])$$

that is, a cut with a hole in the right end, hidden below a chain of applications of the operator ::, expecting a k to form a cut. Formally, these contexts are generated by:

$$\mathcal{H} \ ::= \ t([\cdot]) \,|\, \mathcal{H}[u :: [\cdot]] \tag{5}$$

Figure 6: The revised definition of co-continuation substitution in λGtz$'$

$$\begin{array}{rcl} [\mathcal{H}/x]x & = & \left\{ \begin{array}{ll} t & \text{if } \mathcal{H} = t([\cdot]) \\ \mathcal{H}[\widehat{x}.x] & \text{otherwise} \end{array} \right. \\ [\mathcal{H}/x](tk) & = & \left\{ \begin{array}{ll} ([\mathcal{H}/x]t)([\mathcal{H}/x]k) & \text{if } t \neq x \\ \mathcal{H}[[\mathcal{H}/x]k] & \text{if } t = x \end{array} \right. \end{array}$$

With this inductive definition, $\mathcal{H}[k]$, denoting the cut resulting from filling k in the hole of $\mathcal{H}$, can be defined by recursion on $\mathcal{H}$ as follows:

$$\begin{array}{rcl} t([\cdot])[k] & = & tk \\ \mathcal{H}[u :: [\cdot]][k] & = & \mathcal{H}[u :: k] \end{array}$$

A cut $t(u_1 :: \cdots u_m :: \widehat{x}.v)$ can be written as $\mathcal{H}[k]$ in $m+1$ different ways. Pattern-matching a cut with $\mathcal{H}[k]$ is a way of extracting the component k hidden at the right end of the cut.

In $\overline{\lambda\tilde{\mu}}$, variables stand for co-continuations, as seen from equation (3) in the definition of $[\mathcal{H}/x]\_$. In λGtz, variables are, and stand for, terms; and we only have the ordinary substitution $[t/x]\_$. We could define a substitution operation in λGtz analogous to $[\mathcal{H}/x]\_$ in $\overline{\lambda\tilde{\mu}}$, with critical clause

$$[\mathcal{H}/x]x = \mathcal{H}[\widehat{x}.x] \ ,$$

but this would be artificial. This operation can be defined in terms of ordinary substitution: $[\mathcal{H}/x]e := [\mathcal{H}[\widehat{y}.y]/x]e$, for $e = v, k$. This is possible because $\mathcal{H}[\widehat{x}.x]$ is a term.

With this in place, the native rule τ is defined to be:

$$\mathcal{H}[u :: \widehat{x}.v] \to [\mathcal{H}[u :: \widehat{y}.y]/x]v \text{ , if } v \neq x.$$

The proviso is needed, otherwise redex and contractum would be the same term in the case $v = x$.

We now move to the revised concepts. These depend on incorporating in the definition of $[\mathcal{H}/x]\_$ the following reductions, that can be observed in λGtz:

$$[t([\cdot])/x]x = t(\widehat{x}.x) \to_\sigma t \tag{6}$$

$$\begin{array}{rcll} [\mathcal{H}/x](xk) & = & (\mathcal{H}[\widehat{x}.x])([\mathcal{H}/x]k) & \\ & \to_\pi & \mathcal{H}[(\widehat{x}.x)@([\mathcal{H}/x]k)] & (7) \\ & \to_\mu & \mathcal{H}[[\mathcal{H}/x]k] & \end{array}$$

The π-reduction in (7) is a particular case of

$$(\mathcal{H}[k])k' \to_\pi \mathcal{H}[k@k'] \ , \tag{8}$$

and this is easily proved by induction on $\mathcal{H}$.

The revised definition of $[\mathcal{H}/x]v$ and $[\mathcal{H}/x]k$ is by simultaneous recursion on v and k, where all clauses are homomorphic, except those given in Fig. 6.

The revised concept of co-continuation substitution generalizes ordinary substitution:

Lemma 4.1. *In λGtz$'$, $[t([\cdot])/x]e = [t/x]e$, for $e = v, k$.*

Proof. By simultaneous induction on v and k. □

Given the discussion above (recall (6) and (7)) about the reduction steps built in the revised concept of co-continuation substitution, the following is expected.

Lemma 4.2. *In λGtz$'$, $[\mathcal{H}[u :: \widehat{y}.y]/x]e \to^*_{\pi\mu} [\mathcal{H}[u :: [\cdot]]/x]e$, for $e = v, k$.*

Proof. By simultaneous induction on v and k. □

The revised rule τ is defined to be:

$$\mathcal{H}[u :: \widehat{x}.v] \to [\mathcal{H}[u :: [\cdot]]/x]v \text{ , if } v \neq x.$$

Here of course we employ the revised concept of substitution.

Lemma 4.2 implies that $t \to_\tau t'$ in λGtz$'$ can be decomposed into a "native" τ-reduction step followed by a $\pi\mu$-reduction.

Fourth adjustment. Rule μ now reads

$$\mathcal{H}[\widehat{x}.xk] \to \mathcal{H}[k] \ . \tag{9}$$

Notice that the original definition of μ is a relation on contexts, from which we derive this new one by compatible closure. So we are adopting now a slightly less general version of the rule.

Comments. In all, we touched all reduction rules of λGtz but σ, and added the new τ. Actually, once τ is added, rule μ, in the revised form (9), becomes derivable, very much like what happens in $\overline{\lambda\tilde{\mu}}$ with the η-rule for $\tilde{\mu}$. To see this, consider the two possible cases of $\mathcal{H}$ in grammar (5). If $\mathcal{H}$ has the first form, then reduction (9) is a particular case of σ (recall (1)); else $\mathcal{H} = \mathcal{H}'[u :: [\cdot]]$ and reduction (9) is a particular case of τ:

$$\begin{array}{rcll} \mathcal{H}[\widehat{x}.xk] & = & \mathcal{H}'[u :: (\widehat{x}.xk)] & \\ & \to_\tau & [\mathcal{H}/x](xk) & \\ & = & \mathcal{H}[[\mathcal{H}/x]k] & \text{(by def. in Fig. 6)} \\ & = & \mathcal{H}[k] & \text{(since } x \notin k) \end{array}$$

The reduction (8) still holds in λGtz$'$. Conversely, every π-reduction step has this form; indeed, rule π can be given as the union of the following two rules:

$$\begin{array}{lrcl} (\pi_1) & (\mathcal{H}[\widehat{x}.v])k & \to & \mathcal{H}[\widehat{x}.vk], \ v \neq x \\ (\pi_0) & (\mathcal{H}[\widehat{x}.x])k & \to & \mathcal{H}[k] \end{array}$$

In Lemma 4.2, π can be restricted to π_0 (recall calculation (7)). So $t \to_\tau t'$ in λGtz$'$ can be decomposed into a "native" τ-reduction step followed by a $\pi_0\mu$-reduction.

We will need a further split:

$$\begin{array}{lrcl} (\pi_{11}) & (\mathcal{H}[\widehat{x}.v])k & \to & \mathcal{H}[\widehat{x}.vk], \ v \neq x, \ k \neq \widehat{y}.y \\ (\pi_{10}) & (\mathcal{H}[\widehat{x}.v])(\widehat{y}.y) & \to & \mathcal{H}[\widehat{x}.v(\widehat{y}.y)], \ v \neq x \end{array}$$

Regarding σ, it is useful to single out these particular cases:

$$\begin{array}{lrcll} (\epsilon_1) & t(\widehat{x}.x) & \to & t & (t \text{ not a var.}) \\ (\epsilon_0) & y(\widehat{x}.x) & \to & y & \end{array}$$

We let $\epsilon := \epsilon_0 \cup \epsilon_1$.

From now on, when we refer to the reduction rules we mean the revised one, unless we explicitly say we mean otherwise. In particular, from now on we are interested in the strong normalizability of λGtz-terms as determined by the new rules.

Comparison of the two rewrite systems. We now study the simulation properties of $(\cdot)^*$ and $(\cdot)^+$ which justify the changes made to λGtz. Let us start with $(\cdot)^*$. The first thing to do is to extend this map to co-continuations:

$$\begin{array}{rcl} (x\hat{}[\cdot])^* & = & x([\cdot]) \\ (t([\cdot]))^* & = & t^*([\cdot]) \\ (\mathcal{H}[u : [\cdot]])^* & = & \mathcal{H}^*[u^* :: [\cdot]] \end{array}$$

From this definition it follows easily, by induction on $\mathcal{H}$, that

$$(\mathcal{H}[k])^* = \mathcal{H}^*[k^*] \ . \tag{10}$$

Lemma 4.3. *For all $t, v, k_1, k_2, \mathcal{H} \in \overline{\lambda\tilde{\mu}}$:*

(1) $(k_1@k_2)^* = k_1^*@k_2^*$.
(2) $[\mathcal{H}^*/x]v^* = ([\mathcal{H}/x]v)^*$.

Proposition 4.4 (Simulation). *Let $t_1, t_2 \in \overline{\lambda\tilde{\mu}}$.*

(1) *If $t_1 \to_\beta t_2$, then $t_1^* \to_\beta t_2^*$.*
(2) *If $t_1 \to_{\tilde{\mu}} t_2$, then $t_1^* \to_{\sigma\tau} t_2^*$.*
(3) *If $t_1 \to_\pi t_2$, then $t_1^* \to_\pi t_2^*$.*
(4) *If $t_1 \to_\epsilon t_2$, then $t_1^* \to_{\epsilon_1} t_2^*$.*

Proof. The proof consists of 4 proofs by induction on $t_1 \to_R t_2$, for $R = \beta, \tilde{\mu}, \pi, \epsilon$. However, we show all the base cases first, and then argue the inductive cases uniformly.

Cases β and ϵ. Straightforward.

Case π. Easy, using the first item of Lemma 4.3.

Case $\tilde{\mu}$: $t_1 = \mathcal{H}[\tilde{\mu}x.v] \to [\mathcal{H}/x]v = t_2$. We have three sub-cases.

First subcase: $\mathcal{H} = t([\cdot])$. Then

$$\begin{array}{rcll} t_1^* & = & t^*(\widehat{x}.v^*) & \text{(by defs.)} \\ & \to_\sigma & [t^*/x]v^* & \\ & = & [t^*([\cdot])/x]v^* & \text{(by Lemma 4.1)} \\ & = & [(t([\cdot]))^*/x]v^* & \\ & = & ([t([\cdot])/x]v)^* & \text{(by Lemma 4.3)} \\ & = & t_2^* & \end{array}$$

Second subcase: $\mathcal{H} = x\hat{}([\cdot])$. Analogous to the previous subcase.

Third subcase: $\mathcal{H} = \mathcal{H}_0[u :: [\cdot]]$. Then

$$\begin{array}{rcll} t_1^* & = & \mathcal{H}_0^*[u^* :: \widehat{x}.v^*] & \text{(by defs. and (10))} \\ & \to_\tau & [\mathcal{H}_0^*[u^* :: [\cdot]]/x]v^* & (v^* \text{ is never a var.}) \\ & = & [\mathcal{H}^*/x]v^* & \text{(by def. of } \mathcal{H}^*) \\ & = & ([\mathcal{H}/x]v)^* & \text{(by Lemma 4.3)} \\ & = & t_2^* & \end{array}$$

□

Proposition 4.4 fully confirms the idea that $(\cdot)^*$ injects $\overline{\lambda\tilde{\mu}}$ into $\lambda\text{Gtz}'$: each reduction step in the source is simulated by a reduction step in the target. There is even a reasonable correspondence between reduction rules.

Next we study simulation in the opposite direction, and things will not be as smooth. Again, the first thing to do is to extend map $(\cdot)^+$ to co-continuations:

$$\begin{array}{rcl} (x([\cdot]))^+ & = & x\hat{}([\cdot]) \\ (t([\cdot]))^+ & = & t^+([\cdot]) \quad t \text{ not a var.} \\ (\mathcal{H}[u : [\cdot]])^+ & = & \mathcal{H}^+[u^+ :: [\cdot]] \end{array}$$

From this definition it follows easily, by induction on $\mathcal{H}$, that

$$(\mathcal{H}[k])^+ = \mathcal{H}^+[k^+] \ . \tag{11}$$

In the next result we employ notation $\to_R^=$ to denote the reflexive closure of $\to_R$.

Lemma 4.5. *For all $v, k_1, k_2, \mathcal{H} \in \lambda\text{Gtz}'$:*

(1) $k_1^+@k_2^+ \to_{\pi_1}^= (k_1@k_2)^+$.
(2) $[\mathcal{H}^+/x]v^+ \to_\epsilon^* ([\mathcal{H}/x]v)^+$.

In the following result, we opted to include an item for μ, because its direct prove gives a result slightly more precise than the one that is obtained by the fact that $\mu \subset \sigma \cup \tau$.

Proposition 4.6 (Simulation). *Let $t_1, t_2 \in \lambda\text{Gtz}'$.*

(1) *If $t_1 \to_\beta t_2$, then $t_1^+ \to_{\beta\tilde{\mu}\pi}^+ t_2^+$.*
(2) *If $t_1 \to_\pi t_2$, then $t_1^+ \to_\pi^+ t_2^+$.*
(3) *If $t_1 \to_\mu t_2$, then $t_1^+ \to_{\tilde{\mu}}^+ t_2^+$.*
(4) *If $t_1 \to_\tau t_2$, then $t_1^+ \to_{\tilde{\mu}\epsilon}^+ t_2^+$.*
(5) *If $t_1 \to_\sigma t_2$, then:*
   (a) *$t_1^+ = t_2^+$, if $t_1 \to_{\epsilon_0} t_2$;*
   (b) *$t_1^+ \to_\epsilon t_2^+$, if $t_1 \to_{\epsilon_1} t_2$;*
   (c) *$t_1^+ \to_{\tilde{\mu}\epsilon}^+ t_2^+$, otherwise.*

Proof. The proof consists of 5 proofs by induction on $t_1 \to_R t_2$, for $R = \beta, \pi, \mu, \tau, \sigma$. We show the base cases.

Case β: $t_1 = (\lambda x.t)(u :: k) \to (u(\widehat{x}.t))k = t_2$. Then

$$t_1^+ = (\lambda x.t^+)(u^+ :: k^+) \to_\beta (u^+(\tilde{\mu}x.t^+))k^+ =: v \ .$$

Now there are 4 subcases:

- u not a variable and $t \neq x$. Then $v = t_2^+$.
- u not a variable and $t = x$. Then

$$v = (u^+(\tilde{\mu}x.x\hat{}[]))k^+ \to_{\tilde{\mu}} (u^+[])k^+ = t_2^+ \ .$$

- $u = y$ and $t \neq x$. Then

$$v = ((y\hat{}[])(\tilde{\mu}x.t^+))k^+ \to_\pi (y\hat{}(\tilde{\mu}x.t^+))k^+ = t_2^+ \ .$$

- $u = y$ and $t = x$.

$$v = ((y\hat{}[])(\tilde{\mu}x.x\hat{}[]))k^+ \to_\pi (y\hat{}(\tilde{\mu}x.x\hat{}[]))k^+ \to_{\tilde{\mu}} (y\hat{}[])k^+ = t_2^+ \ .$$

Case τ. $t_1 = \mathcal{H}[u :: \widehat{x}.v] \to [\mathcal{H}[u :: [\cdot]]/x]v = t_2$, if $v \neq x$. Let $\mathcal{H}_0 = \mathcal{H}[u :: [\cdot]]$.

$$\begin{array}{rcll} t_1^+ & = & \mathcal{H}^+[u^+ :: (\tilde{\mu}x.v^+)] & \text{(by def. and (11) and } v \neq x) \\ & = & \mathcal{H}_0^+[\tilde{\mu}x.v^+] & \text{(by def.)} \\ & \to_{\tilde{\mu}} & [\mathcal{H}_0^+/x]v^+ & \\ & \to_\epsilon^* & ([\mathcal{H}_0/x]v)^+ & \text{(by item 2 of Lemma 4.5)} \\ & = & t_2^+ & \end{array}$$

Case σ. There are 4 subcases.

Case ϵ_0: $t_1 = y(\widehat{x}.x) \to y = t_2$. Then $t_1^+ = y\hat{}[] = t_2^+$.

Case ϵ_1: $t_1 = t(\widehat{x}.x) \to t = t_2$, with t not a variable. Then $t_1^+ = t^+[] \to_\epsilon t^+ = t_2^+$.

Third subcase: $t_1 = t(\widehat{x}.v) \to [t/x]v = t_2$, with t not a variable and $v \neq x$.

$$\begin{array}{rcll} t_1^+ & = & t^+(\tilde{\mu}x.v^+) & \text{(by def.)} \\ & \to_{\tilde{\mu}} & [t^+([\cdot])/x]v^+ & \\ & = & [t([\cdot])^+/x]v^+ & \text{(by def.)} \\ & \to_\epsilon^* & ([t([\cdot])/x]v)^+ & \text{(by Lemma 4.5)} \\ & = & ([t/x]v)^+ & \text{(by Lemma 4.1)} \end{array}$$

Fourth subcase: $t_1 = y(\widehat{x}.v) \to [y/x]v = t_2$, with $v \neq x$. Analogous to the previous subcase.

Case π. Easy, using item 1 of Lemma 4.5.

Case μ: $t_1 = \mathcal{H}[\widehat{x}.xk] \to \mathcal{H}[k] = t_2$, if $x \notin k$.

$$\begin{array}{lcll} t_1^+ & = & \mathcal{H}^+[\tilde{\mu}x.x\hat{}k^+] & \text{(by def. and (11))} \\ & \to_{\tilde{\mu}} & [\mathcal{H}^+/x](x\hat{}k^+) & \\ & = & \mathcal{H}^+[[\mathcal{H}^+/x]k^+] & \text{(by def.)} \\ & = & \mathcal{H}^+[k^+] & \text{(since } x \notin k^+\text{)} \\ & = & t_2^+ & \text{(by (11))} \end{array}$$

□

Except for the collapse of ϵ_0-reduction steps by $(\cdot)^+$, both Propositions 4.4 and 4.6 give strict simulations, in the sense that each reduction step in the source calculus is mapped to at least one reduction step in the target calculus. That is why these propositions are useful to prove the next theorem.

Theorem 4.7. *Let $t \in \overline{\lambda\tilde{\mu}}$. t is $\beta\tilde{\mu}\epsilon\pi$-SN iff t^* is $\beta\sigma\pi\mu\tau$-SN in λGtz$'$.*

Proof. The "if" statement follows immediately from Proposition 4.4. As to the "only if" statement, suppose t^* is not $\beta\sigma\pi\mu\tau$-SN, and let ρ be an infinite reduction sequence staring from t^*. Since $\to_{\epsilon_0}$ is obviously terminating, ρ contains infinitely many R-reduction steps, with $R \neq \epsilon_0$. From Proposition 4.6, we conclude that $(t^*)^+$ has an infinite reduction sequence, that is, $(t^*)^+$ is not $\beta\tilde{\mu}\epsilon\pi$-SN. But $(t^*)^+ = t$, by Lemma 3.1. □

5 STRONG NORMALIZATION

This section has 3 subsections. In the first two, we continue the revisiting of λGtz and prove that the typing system of Fig. 1 also characterizes strong normalizability in λGtz$'$, by adjusting the proof given in [10, 11]. From this we get the characterization of strong normalizability in $\overline{\lambda\tilde{\mu}}$ in the third subsection.

5.1 Typability implies SN

We define a map $F : \lambda\text{Gtz}' \to \lambda$. More precisely, we define $F(t)$, for $t \in \lambda$Gtz$'$, and define $F'(N, k)$, for $k \in \lambda$Gtz$'$, given a λ-term N, by simultaneous recursion on t and k, as follows:

$$\begin{array}{rcl} F(x) & = & x \\ F(\lambda x.t) & = & \lambda x.F(t) \\ F(tk) & = & F'(F(t), k) \\ F'(N, u :: k) & = & F'(NF(u), k) \\ F'(N, \widehat{x}.t) & = & \begin{cases} (\lambda x.F(t))N & \text{if } t \neq x \\ N & \text{otherwise} \end{cases} \end{array}$$

The case analysis in the last clause is a novelty relatively to map F studied before in [10, 11], without which the statement relative to τ in Prop. 5.2 below does not work.

F is readily extended to co-continuation by:

$$F(t([\cdot])) = F(t) \qquad F(\mathcal{H}[u :: [\cdot]]) = F(\mathcal{H})F(u) \ . \tag{12}$$

The following is easily proved by induction on $\mathcal{H}$:

$$F(\mathcal{H}[\widehat{x}.t]) = \begin{cases} (\lambda x.F(t))F(\mathcal{H}) & \text{if } t \neq x \\ F(\mathcal{H}) & \text{otherwise} \end{cases} \tag{13}$$

We consider the λ-calculus equipped with β and π, where rule π is the union of the two following rules:

$$\begin{array}{llcl} (\pi_1) & (\lambda x.M)NP & \to & (\lambda x.MP)N \\ (\pi_2) & M((\lambda x.P)N) & \to & (\lambda x.MP)N \end{array}$$

Lemma 5.1. *Let $M, N, \vec{Q}$ be in λ and t, u, k, k' be in λGtz.*

1. *$F'((\lambda x.M)N\vec{Q}, k) \to_\pi^+ (\lambda x.F'(M\vec{Q}, k))N$, provided $\vec{Q}$ is not empty or $k \neq \widehat{y}.y$.*
2. *$F([u/x]t) = [F(u)/x]F(t)$.*
3. *$(\lambda x.F'(x\vec{Q}, k))N \to_\beta F'(N\vec{Q}, k)$, if $x \notin FV(k) \cup FV(\vec{Q})$.*

Proof. There is a similar lemma in [11], but the provisos in item 1 here are new. Let us see the proof of this item. It is by induction on k. Let n be the length of $\vec{Q}$.

Fist case: $k = \widehat{y}.v$, $v \neq y$.

$$\begin{array}{lcll} & & F'((\lambda x.M)N\vec{Q}, k) & \\ & = & (\lambda y.F(v))((\lambda x.M)N\vec{Q}) & \text{(by def.)} \\ & \to_{\pi_1}^n & (\lambda y.F(v))((\lambda x.M\vec{Q})N) & \\ & \to_{\pi_2} & (\lambda x.(\lambda y.F(v))(M\vec{Q}))N & \\ & = & (\lambda x.F'(M\vec{Q}, k))N & \text{(by def.)} \end{array}$$

Case $k = \widehat{y}.y$.

$$\begin{array}{lcll} & & F'((\lambda x.M)N\vec{Q}, k) & \\ & = & (\lambda x.M)N\vec{Q} & \text{(by def.)} \\ & \to_{\pi_1}^n & (\lambda x.M\vec{Q})N & \\ & = & (\lambda x.F'(M\vec{Q}, k))N & \text{(by def.)} \end{array}$$

Case $k = u :: k_0$.

$$\begin{array}{lcll} & & F'((\lambda x.M)N\vec{Q}, k) & \\ & = & F'((\lambda x.M)N\vec{Q}F(u), k_0) & \text{(by def.)} \\ & \to_\pi^+ & (\lambda x.F'(M\vec{Q}F(u), k_0))N & \text{(by IH)} \\ & = & (\lambda x.F'(M\vec{Q}, k))N & \text{(by def.)} \end{array}$$

□

Proposition 5.2. *Let $t, t' \in \lambda$Gtz$'$.*

1. *If $t \to_\beta t'$, then $F(t) = F(t')$.*
2. *If $t \to_\sigma t'$, then:*
   (a) *$F(t) = F(t')$, if $t \to_\epsilon t'$*
   (b) *$F(t) \to_\beta F(t')$, otherwise.*
3. *If $t \to_{\pi_0} t'$, then $F(t) = F(t')$.*
4. *If $t \to_{\pi_1} t'$, then*
   (a) *$F(t) = F(t')$, if $t \to_{\pi_{10}} t'$*
   (b) *$F(t) \to_\pi^+ F(t')$, otherwise.*
5. *If $t \to_\mu t'$, then $F(t) \to_\beta F(t')$*
6. *If $t \to_\tau t'$, then $F(t) \to_\beta^+ F(t')$.*

Proof. Each item is proved by induction on $t \to_R t'$. The base cases follow easily with the help of Lemma 5.1, except case τ, which needs a little twist: in fact, item (6) has to be proved after items (3) and (5). Let us see why. We have seen that $t \to_\tau t'$ in λGtz$'$ can be decomposed into a "native" τ-reduction step followed by a $\pi_0\mu$-reduction. Using items (3) and (5), it suffices to check a "native" τ-reduction step $t = \mathcal{H}[u :: \widehat{x}.v] \to [\mathcal{H}[u :: \widehat{y}.y]/x]v = t'$, with $v \neq x$.

$$\begin{array}{lcll} F(t) & = & (\lambda x.F(v))(F(\mathcal{H})F(y)) & \text{(by (12), (13) and } v \neq x\text{)} \\ & \to_\beta & [F(\mathcal{H})F(u)/x]F(v) & \\ & = & [F(\mathcal{H}[u :: \widehat{y}.y])/x]F(v) & \text{(by (12) and (13))} \\ & = & F(t') & \text{(item 2. in Lemma 5.1)} \end{array}$$

□

Proposition 5.3. *Let $t \in \lambda\mathrm{Gtz}'$. If $F(t)$ is $\beta\pi$-SN, then t is $\beta\sigma\pi\mu\tau$-SN.*

Proof. Follows from Prop. 5.2 plus termination of R-reduction, where $R = \beta\epsilon\pi_0\pi_{10}$. Termination of R-reduction is proved in two steps. First, π_{10}-reduction is terminating. Let $n(t)$ be a measure witnessing this fact. Second, a R-reduction step on t decreases the measure $(l(t), m(t), n(t))$, where $l(t)$ (resp. $m(t)$) is the number of occurrences of λ (resp. $\widehat{x}.x$) in t, and these tuples are ordered lexicographically. □

As the typing system for the λ-calculus we take system named $\mathcal{D}$ in [14]. In this system we have the ordinary rules for assigning simple types, plus the introduction and the two elimination rules for $\cap$. We say a λ-term is *typable* if it is typable in this system.

Proposition 5.4. *If $t \in \lambda\mathrm{Gtz}'$ is typable then $F(t)$ is typable.*

Proof. It suffices to prove the soundness of F: (i) if $\Gamma \vdash t : A$ in $\lambda\mathrm{Gtz}'$, then $\Gamma \vdash F(t) : A$ in $\mathcal{D}$; (ii) if $\Gamma; A \vdash k : B$ in $\lambda\mathrm{Gtz}'$ and $\Gamma \vdash N : A$ in $\mathcal{D}$, then $\Gamma \vdash F'(N,k) : B$ in $\mathcal{D}$. The proof is a simultaneous induction on t and k. This is Prop. 16 in [10], except that now the definition of F has the case separation in the definition of $F'(N, \widehat{x}.t)$. So it remains to check the case $k = \widehat{x}.x$. From $\Gamma; A \vdash \widehat{x}.x : B$ in $\lambda\mathrm{Gtz}'$ we get $A = \cap A_i$ and $B = A_j$ for some $j \in \{1, \cdots, n\}$. Since $F'(N, \widehat{x}.x) = N$, we want $\Gamma \vdash N : B$ in $\mathcal{D}$. But this follows from the given $\Gamma \vdash N : A$ by a sequence of applications of the elimination rules for $\cap$. □

Now we collect the strong normalization result for $\lambda\mathrm{Gtz}'$:

Proposition 5.5. *If $t \in \lambda\mathrm{Gtz}'$ is typable, then t is $\beta\sigma\pi\mu\tau$-SN.*

Proof. Suppose t is typable. By Prop. 5.4, $F(t)$ is typable. Hence $F(t)$ is β-SN (a classical result found in [14, 18]). By the theorem in [8], $F(t)$ is $\beta\pi$-SN. By Prop. 5.3, t is $\beta\sigma\pi\mu\tau$-SN. □

5.2 SN implies typability

In order to prove the completeness, i.e., that all strongly normalizing terms in $\overline{\lambda\tilde{\mu}}$ are typable in the intersection type system given in Fig. 5, we will first show that this property holds for $\lambda\mathrm{Gtz}'$ with typability defined by the intersection types system in Fig. 1. We apply the method established in [10] and just stress the differences, which are very few. First we prove that all normal forms in $\lambda\mathrm{Gtz}'$ are typable and then we prove subject expansion at root position.

The *$\beta\sigma\pi$-normal forms* of $\lambda\mathrm{Gtz}'$ are exactly the same as those of $\lambda\mathrm{Gtz}$ (given before in subsection 2.1), because the rule σ is the same in both systems, and although we changed rules β and π in $\lambda\mathrm{Gtz}'$, the notion of $\beta\pi$-redex was not changed.

Lemma 5.6 (Typability of normal forms). *$\beta\sigma\pi$-normal forms of $\lambda\mathrm{Gtz}'$ are typable in the system of Fig. 1. Hence so are $\beta\sigma\pi\mu\tau$-normal forms.*

Proof. The $\beta\sigma\pi$-normal forms of $\lambda\mathrm{Gtz}'$ and $\lambda\mathrm{Gtz}$ are the same, and typability is defined by the same typing system, so the result follows by Prop. 2.3. □

Lemma 5.7 (Inverse substitution lemma).

(1) *Let $\Gamma \vdash [t/x]v : A$ and let t be typable. Then there is a basis Γ' and a type $B \equiv \cap B_i$, such that $\Gamma', x : \cap B_i \vdash v : A$ and $\Gamma' \vdash t : B_i$ for all i.*
(2) *Let $\Gamma; C \vdash [t/x]k : A$ and let t be typable. Then there is a basis Γ' and a type $B \equiv \cap B_i$, such that $\Gamma', x : \cap B_i; C \vdash k : A$ and $\Gamma' \vdash t : B_i$ for all i.*

Proof. Along the lines of the proofs in [10] and [11]. □

Lemma 5.8 (Inverse append lemma). *If $\Gamma; B \vdash k@k' : A$, then there is a type $C \equiv \cap C_i$ s.t. $\Gamma; B \vdash k : C_i$, for all i, and $\Gamma; \cap C_i \vdash k' : A$.*

Proof. By induction on the structure of k. We consider the new case when $k \equiv \widehat{x}.x$, i.e., $(\widehat{x}.x)@k' = k'$. Let $\Gamma; B \vdash k' : A$. The following is an axiom $\Gamma, x : B \vdash x : B$, hence $\Gamma; B \vdash \widehat{x}.x : B$. Therefore the required $C \equiv C_i \equiv B$ and $i \in \{1\}$. □

Proposition 5.9 (Subject expansion at root position). *Let $R = \beta\sigma\pi$. If $t \in \lambda\mathrm{Gtz}'$ and $t \to_R t'$, where t is a contracted redex and t' is typable, then t is typable.*

Proof. We check the case of the new β rule. Suppose $\Gamma \vdash (u(\widehat{x}.t))k : A$ and show $\Gamma \vdash (\lambda x.t)(u :: k) : A$. By Generation Lemma (Prop. 2.1.4) there is a $B \equiv \cap B_j$ such that $\Gamma; \cap B_j \vdash k : A$ and $\Gamma \vdash u(\widehat{x}.t) : B_j$ for all j. Then again by Generation Lemma (Prop. 2.1.4) there is a $C \equiv \cap C_i$ such that $\Gamma; \cap C_i \vdash \widehat{x}.t : B_j$ and $\Gamma \vdash u : C_i$ for all i. Further, this implies, by Generation Lemma (Prop. 2.1.3) that $\Gamma, x : \cap C_i \vdash t : B_j$, hence $\Gamma \vdash \lambda x.t : \cap C_i \to B_j$, for all j. Then we have

$$\dfrac{\Gamma \vdash \lambda x.t : \cap C_i \to B_j,\ \forall j \qquad \dfrac{\Gamma \vdash u : C_i,\ \forall i \qquad \Gamma; \cap B_j \vdash k : A}{\Gamma; \cap C_i \to \cap B_j \vdash u :: k : A}(\to L)}{\Gamma \vdash (\lambda x.t)(u :: k) : A}(Cut)$$

taking into account that $\cap C_i \to \cap B_j \equiv \cap(\cap C_i \to B_j)$, due to the assumed equivalence on types.

The rest of the proof is along the lines of the proofs in [10] and [11] and relies on Lemmas 5.7 and 5.8. □

Proposition 5.10. *If $t \in \lambda\mathrm{Gtz}'$ is $\beta\sigma\pi\mu\tau$-SN, then t is typable.*

Proof. As in [10, 11], by induction on the length of the longest reduction path out of a strongly normalising $t \in \lambda\mathrm{Gtz}'$, with a subinduction on the size of t, using Lemma 5.6 and Prop. 5.9. □

Theorem 5.11 (Characterization of SN in $\lambda\mathrm{Gtz}'$). *Let $t \in \lambda\mathrm{Gtz}'$. t is $\beta\sigma\pi\mu\tau$-SN iff t is typable.*

Proof. By Prop. 5.5 and 5.10. □

5.3 SN in $\overline{\lambda\tilde{\mu}}$.

Everything is in place for the main result of this paper.

Theorem 5.12 (Characterization of SN in $\overline{\lambda\tilde{\mu}}$). *Let $t \in \overline{\lambda\tilde{\mu}}$. t is $\beta\tilde{\mu}\pi\epsilon$-SN iff t is typable.*

Proof.

t is $\beta\sigma\pi\mu\tau$-SN	iff	t^* is $\beta\sigma\pi\mu\tau$-SN	(by Thm. 4.7)
	iff	t^* is typable	(by Thm. 5.11)
	iff	t is typable	(by Thm. 3.4)

□

Figure 7: Typing rules of $\underline{\lambda\text{let}}$

$$\frac{}{\Gamma, x : A \rhd x : A}\ Hyp \qquad \frac{\Gamma \rhd H : A \quad \Gamma, x : A \vdash P : B}{\Gamma \vdash \mathsf{let}\, x := H \,\mathsf{in}\, P : B}\ Let$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \supset B}\ Intro \qquad \frac{\Gamma \rhd H : A \supset B \quad \Gamma \vdash N : A}{\Gamma \rhd HN : B}\ Elim$$

$$\frac{\Gamma \rhd H : A}{\Gamma \vdash \mathsf{app}(H) : A}\ WCoercion \qquad \frac{\Gamma \vdash M : A}{\Gamma \rhd \mathsf{hd}(M) : A}\ SCoercion$$

Figure 8: Reduction rules of $\underline{\lambda\text{let}}$

(beta)	$\mathsf{hd}(\lambda x.M)N$	$\to$	$\mathsf{hd}(\mathsf{let}\, x := \mathsf{hd}(N) \,\mathsf{in}\, M)$
(let)	$\mathsf{let}\, x := H \,\mathsf{in}\, P$	$\to$	$[H/x]P$
(triv)	$\mathsf{app}(\mathsf{hd}(M))$	$\to$	M
(head_1)	$\mathsf{hd}(\mathsf{app}(H))$	$\to$	H
(head_2)	$\mathcal{K}[\mathsf{hd}(\mathsf{let}\, x := H \,\mathsf{in}\, P)]$	$\to$	$\mathsf{let}\, x := H \,\mathsf{in}\, \mathcal{K}[\mathsf{hd}(P)]$

6 NATURAL DEDUCTION

When first introduced in [9], $\overline{\lambda\tilde{\mu}}$ was accompanied by an isomorphic, bidirectional, natural deduction system, called $\underline{\lambda\text{let}}$. In this section, we recall the latter system and profit from the mentioned isomorphism, transferring the characterization of strong normalizability from $\overline{\lambda\tilde{\mu}}$ to $\underline{\lambda\text{let}}$. In [9], the isomorphic natural deduction system helped to clarify what co-control means, by translating the sequent calculus syntax to the more familiar format of natural deduction. $\underline{\lambda\text{let}}$ is a kind of "computational" λ-calculus (in the sense of Moggi) agnostic w.r.t call-by-name vs call-by-value [3, 9]. Building a bidirectional intersection-type assignment system for such a calculus is a novelty.

6.1 The $\underline{\lambda\text{let}}$-calculus

The proof-expressions of the calculus are given by:

(Terms)	M, N, P	::=	$\lambda x.M \mid \mathsf{app}(H) \mid \mathsf{let}\, x := H \,\mathsf{in}\, P$
(Heads)	H	::=	$x \mid \mathsf{hd}(M) \mid HN$

Notice that variables x and applications HN are not terms.

The fist way of understanding this syntax is as a syntax of proof-expressions for the bidirectional natural deduction system in Fig. 7. The system handles two kinds of sequents: $\Gamma \vdash M : A$ and $\Gamma \rhd H : A$. Four rules are standard, with the appropriate kind of sequent determined by the kind of expression being typed. The remaining two rules switch the kind of sequent, and are called *coercions*. However, despite the superficial impression, the two coercions are quite different, one being called weak and the other strong. The first difference is that only the strong coercion breaks the sub-formula property - see [9] for details.

The reduction rules of $\underline{\lambda\text{let}}$ are in Fig. 8. We single out two particular cases of let: ren, when $H = x$; and sub, when $H = \mathsf{hd}(M)$.

Figure 9: Map $\Theta : \overline{\lambda\tilde{\mu}} \longrightarrow \underline{\lambda\text{let}}$

$$\begin{array}{rcl} \Theta(\lambda x.t) & = & \lambda x.\Theta t \\ \Theta(x\hat{}k) & = & \Theta(x, k) \\ \Theta(tk) & = & \Theta(hd(\Theta t), k) \\ \Theta(H, []) & = & \mathsf{app}(H) \\ \Theta(H, \tilde{\mu}x.t) & = & \mathsf{let}\, x := H \,\mathsf{in}\, \Theta t \\ \Theta(H, u :: k) & = & \Theta(H\Theta u, k) \end{array}$$

We put $\mathsf{t} := \mathsf{let} \backslash (\mathsf{ren} \cup \mathsf{sub})$. Let $\mathsf{head} := \mathsf{head}_1 \cup \mathsf{head}_2$. Notice that rules beta and head_1 are relations on heads.

A second way of understanding the syntax of $\underline{\lambda\text{let}}$ is as bidirectional "computational" λ-calculus. Rule beta generates a let-expression, which can be reduced away by the separate rule let. Notice that let is ready to fire with any H, i.e. any expression the formal parameter x can stand for (no need to wait for a value to be computed); in addition, let triggers ordinary substitution $[H/x]P$.

Rule head_2 employs certain contexts that we call *continuations*, generated by the grammar:

$$\mathcal{K} ::= \mathsf{app}([\cdot]) \mid \mathsf{let}\, x := [\cdot] \,\mathsf{in}\, P \mid \mathcal{K}[[\cdot]N] \tag{14}$$

So a continuation is a context with one of the forms

$$\mathsf{app}([\cdot]N_1 \cdots N_m) \qquad \mathsf{let}\, x := [\cdot]N_1 \cdots N_m \,\mathsf{in}\, P$$

Rule head_2 is a general form of the assoc-rule for let-expressions. Notice $\mathcal{K}[\mathsf{hd}(\mathsf{app}(H))] \to \mathcal{K}[H]$ is a head_1 reduction.

Every closed non-abstraction term P can be written in a unique way as $\mathcal{K}[\mathsf{hd}(M)]$ - let us call $\mathsf{hd}(M)$ the *singled-out* head. If this M is not an abstraction, P is a head-redex, whose reduction produces another closed non-abstraction with simpler singled-out head. Otherwise P contains a β-redex, if $\mathcal{K}$ has the third form in (14); or P is a triv-redex (resp. let-redex), if $\mathcal{K}$ has the first (resp. second) form in (14) - and then the abstraction is returned (resp. substituted) by the reduction of redex P. It follows that rules triv and head_1 have rather different roles, despite looking similar because both erase a double coercion. See [9] for more on this.

The normal forms w.r.t. all reduction rules are given by:

$$M ::= \lambda x.M \mid \mathsf{app}(H) \qquad\qquad H ::= x \mid HN$$

That is, these normal forms are characterized by the absence of occurrences of lets and hd(_). Lets are eliminated by let whereas all the other rules concur to eliminate the coercion hd(_).

The isomorphism. See Fig. 9 for the map $\Theta : \overline{\lambda\tilde{\mu}} \longrightarrow \underline{\lambda\text{let}}$. There is actually a function $\Theta : \overline{\lambda\tilde{\mu}} - Terms \longrightarrow \underline{\lambda\text{let}} - Terms$, together with an auxiliary function $\Theta : \underline{\lambda\text{let}} - Heads \times \overline{\lambda\tilde{\mu}} - Vectors \longrightarrow \underline{\lambda\text{let}} - Terms$. Let $\Theta(t) = M$, $\Theta(u_i) = N_i$ and $\Theta(v) = P$. The idea is to map, say, $t(u_1 :: u_2 :: \tilde{\mu}x.v)$ to $\mathsf{let}\, x := hd(M)N_1N_2 \,\mathsf{in}\, P$, and $x\hat{}(u_1 :: u_2 :: [])$ to $\mathsf{app}(xN_1N_2)$: left-introductions are replaced by applications, inverting the associativity of non-abstractions.

Recall from [9] that Θ is an isomorphism between $\overline{\lambda\tilde{\mu}}$ and $\underline{\lambda\text{let}}$. More precisely, Θ is a bijection between the set of $\overline{\lambda\tilde{\mu}}$-terms and the set of $\underline{\lambda\text{let}}$-terms (whose inverse Ψ is shown in Fig. 10). Maps Θ and Ψ are sound, that is, the implications presented as "rules" in Fig. 11 are true, where the typing relations are those of Figs. 2 and 7. Moreover, Θ is an isomorphism of reduciton relations:

Figure 10: Map $\Psi : \underline{\lambda\mathsf{let}} \longrightarrow \overline{\lambda\tilde{\mu}}$

$$\begin{array}{rcl}
\Psi(\lambda x.M) & = & \lambda x.\Psi M \\
\Psi(\mathsf{app}(H)) & = & \Psi(H,[]) \\
\Psi(\mathsf{let}\, x := H \,\mathsf{in}\, P) & = & \Psi(H, \tilde{\mu}x.\Psi P) \\
\Psi(x,k) & = & x\hat{}k \\
\Psi(\mathsf{hd}(M),k) & = & (\Psi M)k \\
\Psi(HN,k) & = & \Psi(H,(\Psi N) :: k)
\end{array}$$

Figure 11: Soundness of Θ and Ψ

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \Theta t : A} \qquad \frac{\Gamma \rhd H : A \quad \Gamma | A \vdash k : B}{\Gamma \vdash \Theta(H,k) : B}$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \Psi M : A} \qquad \frac{\Gamma \rhd H : A \quad \Gamma | A \vdash k : B}{\Gamma \vdash \Psi(H,k) : B}$$

Theorem 6.1 (Isomorphism of reduction [9]). *Let R be rule β (resp. $\tilde{\mu}$, ϵ, π) of $\overline{\lambda\tilde{\mu}}$, and let R' be rule* beta *(resp.* let, triv, head*) of* $\underline{\lambda\mathsf{let}}$. *Then, $t \to_R t'$ in $\overline{\lambda\tilde{\mu}}$ iff $\Theta t \to_R \Theta t'$ in $\underline{\lambda\mathsf{let}}$.*

In particular, the co-control rule $\tilde{\mu}$ of $\overline{\lambda\tilde{\mu}}$ is isomorphic to the let-elimination rule let of $\underline{\lambda\mathsf{let}}$.

6.2 Characterization of SN

The intersection type system for $\underline{\lambda\mathsf{let}}$ is given in Fig. 12. The same definitions and conventions regarding intersection types adopted for $\overline{\lambda\tilde{\mu}}$ apply here. In these rules, $\cap A_i = A_1 \cap \cdots \cap A_n$, for some $n \geq 1$. If we imposed $n = 1$, we would recover the system of Fig. 7 for assigning simple types.

Notice how the coercion rules are now an elimination rule and an introduction rule for $\cap$. But, precisely because they are coercion rules, they involve a change in the kind of sequent - let us say a *change of directionality*.

On the other hand, we might consider an elimination rule and an introduction rule for $\cap$ without change of directionality:

$$\frac{\Gamma \rhd H : \cap A_i \quad \exists i \in \{1,\cdots,n\} A = A_i}{\Gamma \rhd H : A}\ \cap E$$

$$\frac{\Gamma \vdash M : A_i,\ \forall i \in \{1,\cdots,n\}}{\Gamma \vdash M : \cap A_i}\ \cap I$$

If these rules were primitive, we could take rules *Hyp* and *Elim* in the form they have in the system of Fig. 7 for assigning simple types - let us denote the latter forms as *SHyp* and *SElim* just for the sake of the present discussion. The claim for *Elim* is quite obvious:

$$\frac{\Gamma \rhd H : \cap A_i \supset B \qquad \dfrac{\Gamma \vdash N : A_i,\ \forall i \in \{1,\cdots,n\}}{\Gamma \vdash N : \cap A_i}\ \cap I}{\Gamma \rhd HN : B}\ SElim$$

As to *Hyp*, suppose the premise of the rule in Fig. 12 holds. Then we can rearrange $\cap A_i$ as $B_1 \cap \cdots \cap B_m$ so that $\cap A_i = \cap B_k$ and, for some $k \in \{1,\cdots,m\}$, $\cap A_j = B_k$. Then, we can consider the following derivation, where the "inferences" named *Idem* just repeat the sequent.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, x : \cap A_i \rhd x : \cap A_i}\ SHyp}{\Gamma, x : \cap B_k \rhd x : \cap B_k}\ Idem}{\Gamma, x : \cap B_k \rhd x : \cap A_j}\ \cap E}{\Gamma, x : \cap A_i \vdash x : \cap A_j}\ Idem$$

In other words, rules $\cap E$ and $\cap I$ are built in the rules *Hyp* and *Elim* of Fig. 12. But, fortunately, $\cap E$ and $\cap I$ are not primitive, as they would spoil syntax-directedness.

Maps Θ and Ψ are also sound w.r.t. the systems assigning intersection types:

Proposition 6.2. *The "rules" of Fig. 11 are true, where the typing relations are those of Figs. 5 and 12.*

Proof. The upper (resp. lower) pair of rules is proved by simultaneous induction on t and k (resp. M and H). □

Theorem 6.3 (Characterization of SN in $\underline{\lambda\mathsf{let}}$). *Let $M \in \underline{\lambda\mathsf{let}}$. M is* beta, let, triv, head*-SN iff M is typable in the system of Fig. 12.*

Proof.

	M is typable in the system of Fig. 12	
iff	$\Psi(M)$ is typable in the system of Fig. 5	(Prop. 6.2)
iff	$\Psi(M)$ is SN	(Thm. 5.12)
iff	M is SN	(Thm. 6.1)

□

So this result follows from soundness of Θ and Ψ (Prop. 6.2), isomorphism (Thm. 6.1), and characterization of strong normalizability in the sequent calculus side (Thm. 5.12). This pattern could be applied to other systems. One could take from [7] the natural deduction system isomorphic to λGtz, extend it with intersection types, and transfer to it, through the isomorphism, the characterization of strong normalizability in λGtz of [10, 11]; or introduce the variant of such natural deduction system which is isomorphic to $\lambda\mathrm{Gtz}'$, extend it with intersection types, and transfer to it, through the isomorphism, the characterization of strong normalizability in $\lambda\mathrm{Gtz}'$ contained in Thm. 5.11.

7 CONCLUSIONS

We have studied the sequent λ-calculus with co-control $\overline{\lambda\tilde{\mu}}$, introduced in [9], which is a λ-calculus of proof-terms for an intuitionistic sequent calculus where the management of formulas on the left hand side of typing judgments is "dual" to the management of formulas on the right hand side of the typing judgments in Parigot's $\lambda\mu$-calculus, known to be a λ-calculus with a control operator. We have set up and analyzed the relationship between $\overline{\lambda\tilde{\mu}}$ and λGtz, another calculus for intuitionistic sequent calculus, given in [7]. The complete characterization of strong normalization in $\overline{\lambda\tilde{\mu}}$ is obtained by means of intersection types, from a similar characterization known before for λGtz [10, 11]. Further, we have considered the bidirectional natural deduction systems isomorphic to these sequent calculi [7, 9], and built for them bidirectional intersection type systems giving characterizations of the strongly normalizing proof-terms of these natural deduction systems.

The $\overline{\lambda\tilde{\mu}}$-calculus was introduced as an improvement over the $\overline{\lambda}$-calculus [12] (as it is not confined to a focused fragment) and

Figure 12: Intersection types for the $\underline{\lambda\text{let}}$-calculus

$$\frac{\forall j \exists i \in \{1,\cdots,n\}\, A_j = A_i}{\Gamma, x : \cap A_i \rhd x : \cap A_j}\ Hyp \qquad \frac{\Gamma \rhd H : A \quad \Gamma, x : A \vdash P : B}{\Gamma \vdash \mathsf{let}\, x := H \,\mathsf{in}\, P : B}\ Let \qquad \frac{\Gamma \rhd H : \cap A_i \quad \exists i \in \{1,\cdots,n\} A = A_i}{\Gamma \vdash \mathsf{app}(H) : A}\ WCoercion$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \supset B}\ Intro \qquad \frac{\Gamma \rhd H : \cap A_i \supset B \quad \Gamma \vdash N : A_i,\ \forall i \in \{1,\cdots,n\}}{\Gamma \rhd HN : B}\ Elim \qquad \frac{\Gamma \vdash M : A_i,\ \forall i \in \{1,\cdots,n\}}{\Gamma \rhd \mathsf{hd}(M) : \cap A_i}\ SCoercion$$

over systems like λGtz, which simplify the task of understanding sequent calculus by considering proof variables as term variables. In this way, the present paper completes the papers [10, 11], in their purpose of showing the applicability of intersection types to the sequent calculus, because it does the same with an improved system.

This supposed improvement is analyzed rigorously, because, as said, the present paper contains a detailed comparison of the two designs λGtz and $\overline{\lambda\tilde{\mu}}$ of sequent calculus. This comparison is interesting on its own, as it highlights many choice points in the design of a sequent lambda-calculus (treatment of proof variables and related substitution principles, definition of the reduction rules); in particular, it reveals the alternatives that exist when defining co-continuation substitution in the setting with proof variables as terms variables.

On the other hand, relatively to papers [10, 11], the present paper opens a new path, as it explores the combination of intersection type systems and bidirectional natural deduction, a combination that turned out to be illuminating. There is a range of treatments of $\cap$ in intersection type systems: in one end we have a system like $\mathcal{D}$, where $\cap$ is treated as a connective, with its primitive rules of introduction and elimination, which destroy syntax-directedness; in the opposite end we have systems like those studied in this paper, enjoying syntax-directedness, where the rules for $\cap$ are built in the other rules. The bidirectional natural deduction system we proposed has primitive rules for $\cap$ without losing syntax-directedness: the point is that those primitive rules are also coercion rules that change directionality, and hence are witnessed in the proof-term.

Further work is planned to extend the methods presented in this paper to characterize strong normalization in proof-term calculi for multi-conclusion sequent intuitionistic calculus proposed by Maehara [16], which is used to translate intuitionistic reasoning into classical one. Another line, but still related to multi-conclusion systems, is to move from $\overline{\lambda\tilde{\mu}}$ to a classical system where the treatment of l.h.s. formulas of $\overline{\lambda\tilde{\mu}}$ and the treatment of r.h.s. formulas of $\lambda\mu$ coexist and are dual to each other. Initial steps in this direction are in [9].

Acknowledgments. The authors would like to thank the anonymous referees for their valuable comments and helpful suggestions.

This work was partly supported by FCT—Fundação para a Ciência e a Tecnologia, within the project UID-MAT-00013/2013; by COST Action CA15123 - The European research network on types for programming and verification (EUTypes) via STSM; and by the Ministry of Education, Science and Technological Development, Serbia, under the projects ON174026 and III44006.

REFERENCES

[1] Henk P. Barendregt, Wil Dekkers, and Richard Statman. 2013. *Lambda Calculus with Types.* Cambridge University Press.
[2] Mario Coppo and Mariangiola Dezani. 1978. A new type assignment for lambda terms. *Archiv für Mathematische Logik* 19 (1978), 139–156.
[3] Pierre-Louis Curien and Herbelin Herbelin. 2000. The duality of computation. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada, September 18-21, 2000 (SIGPLAN Notices 35(9))*. ACM, 233–243. https://doi.org/10.1145/351240.351262
[4] Daniel J. Dougherty, Silvia Ghilezan, and Pierre Lescanne. 2004. Characterizing strong normalization in a language with control operators. In *Sixth ACM SIGPLAN Conference on Principles and Practice of Declarative Programming PPDP'04, Verona, Italy.* ACM Press, 155–166.
[5] Daniel J. Dougherty, Silvia Ghilezan, and Pierre Lescanne. 2008. Characterizing strong normalization in the Curien-Herbelin symmetric lambda calculus: extending the Coppo-Dezani heritage. *Theoretical Computer Science* 398 (2008), 114–128.
[6] José Espírito Santo. 2000. Revisiting the correspondence between cut-elimination and normalisation. In *Proceedings of ICALP'2000 (Lecture Notes in Computer Science)*, Vol. 1853. Springer-Verlag, 600–611.
[7] José Espírito Santo. 2009. The λ-calculus and the unity of structural proof theory. *Theory of Computing Systems* 45 (2009), 963–994.
[8] José Espírito Santo. 2011. A note on preservation of strong normalisation in the λ-calculus. *Theor. Comput. Sci.* 412, 11 (2011), 1027–1032.
[9] José Espírito Santo. 2015. Curry-Howard for Sequent Calculus at Last!. In *13th International Conference on Typed Lambda Calculi and Applications, TLCA 2015, July 1-3, 2015, Warsaw, Poland (LIPIcs)*, Thorsten Altenkirch (Ed.), Vol. 38. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 165–179.
[10] José Espírito Santo, Silvia Ghilezan, and Jelena Ivetic. 2008. Characterising Strongly Normalising Intuitionistic Sequent Terms. In *Types for Proofs and Programs, International Conference, TYPES 2007, Cividale del Friuli, Italy, May 2-5, 2007, Revised Selected Papers (Lecture Notes in Computer Science)*, Marino Miculan, Ivan Scagnetto, and Furio Honsell (Eds.), Vol. 4941. Springer, 85–99.
[11] José Espírito Santo, Jelena Ivetic, and Silvia Likavec. 2012. Characterising Strongly Normalising Intuitionistic Terms. *Fundam. Inform.* 121, 1-4 (2012), 83–120.
[12] Hugo Herbelin. 1995. A λ-calculus structure isomorphic to a Gentzen-style sequent calculus structure. In *Proceedings of CSL'94 (Lecture Notes in Computer Science)*, L. Pacholski and J. Tiuryn (Eds.), Vol. 933. Springer-Verlag, 61–75.
[13] Felix Joachimski and Ralph Matthes. 2000. Standardization and Confluence for a Lambda Calculus with Generalized Applications. In *Proceedings of RTA 2000 (Lecture Notes in Computer Science)*, Vol. 1833. Springer, 141–155.
[14] Jean-Louis Krivine. 1990. *Lambda-calcul, types et modèles.* Masson, Paris.
[15] Chuck Liang and Dale Miller. 2009. Focusing and Polarization in Linear, Intuitionistic, and Classical Logic. *Theoretical Computer Science* 410 (2009), 4747–4768.
[16] Shoji Maehara. 1954. Eine Darstellung über der Intuitionistichen Logik in der Klassischen. *Nagoya Mathematical Journal* 7 (1954), 45–64.
[17] Michel Parigot. 1992. Lambda-Mu-Calculus: An Algorithmic Interpretation of Classical Natural Deduction. In *Logic Programming and Automated Reasoning, International Conference LPAR'92, St. Petersburg, Russia, July 15-20, 1992, Proceedings.* 190–201.
[18] Simona Ronchi Della Rocca. 1988. Principal Type Scheme and Unification for Intersection Type Discipline. *Theoretical Computer Science* 59 (1988), 181–209.