# Call-by-name linguistic side effects

Oleg Kiselyov
FNMOC
oleg@pobox.com

## Abstract

We propose a typed call-by-name $\lambda$-calculus with shift, reset and strict functions and describe its linguistic applications, improving on the previous continuation-based analyses of quantification, binding, raised and in-situ *wh*-questions, binding in *wh*-questions, and superiority. The evaluation order is not fixed left-to-right: rather, it is determined by the demand for values exerted by reset and strict functions. Since functions can take general, effectful terms, our analyses need no thunks and similar type raising.

The main improvement of the present analyses is in typing: assigning types both to terms and contexts and building types using connectives with clear logical interpretation. Types abstractly interpret operational semantics, and thus concisely describe all the effects that could occur in the evaluation of a term. Our main result is that both typing and call-by-name are necessary to correctly predict superiority and binding in *wh*-questions with topicalization, without resorting to thunking or type raising and thus maintaining the uniformity of the analyses.

We have implemented the calculus including the type checking and mechanically verified all the analyses.

## 1. Introduction

Delimited continuations proved useful in linguistics [13] and computer science [4, 5]. Most programming languages and calculi with delimited continuations are call-by-value (CBV), where the argument of an application must be fully evaluated before the application can be reduced. Recently [6] the attention turned to call-by-name (CBN) calculi with delimited control, which support substitutions of general, even effectful terms.

We describe a novel call-by-name calculus with delimited control that most closely corresponds to the familiar CBV shift/reset calculi [1, 3] and embeds CBV with the help of strict functions. Our calculus is typed, in the spirit of [9]. The types provide an abstract view of term's evaluation and are built from connectives of clear logical meaning. We apply the calculus as the logical metalanguage to express the whole range of linguistic analyses described in Shan [14]. The analyses deal with quantification, binding, raised and in-situ *wh*-questions, binding in *wh*-questions, and superiority. We are able to reproduce the analyses, without resorting to thunks or postulating left-to-right overall evaluation order. Our analyses are marked by typing. We treat types as a syntax-semantic interface: Well-typed terms, representing utterances in our metalanguage, assuredly evaluate to denotations and so only well-typed terms could have meaning. Our main result is that *typed* CBN calculus correctly predicts superiority (§3.2) and binding in *wh*-questions with topicalization (§3.1) without resorting to thunking or type raising. An individual is *always* represented by a term of the type $\mathrm{e}$.

Abiding by Karttunen [8]'s exhortation for "the formalization and computational implementation of linguistic theories" we have implemented calculus' evaluation and type-checking relations. We used the implementation to mechanically verify all the analyses in the present paper. The complete code, which includes more analyses, is available at `http://okmij.org/ftp/Computation/gengo/`.

| | |
|---|---|
| Primitive Constants | $D ::= \mathrm{john} \mid \mathrm{mary} \mid \mathrm{see} \mid \mathrm{tall} \mid \mathrm{mother}$ |
| Constants | $C ::= D \mid C \wedge C \mid c \mid \forall_c \mid \partial_c$ |
| Terms | $E, F ::= x \mid C \mid \lambda x.\, E \mid FE \mid E \curlywedge F$ |
| Transitions | $(\lambda x.\, E)F \rightsquigarrow E\{x \mapsto F\} \quad (\beta^*)$ |
| | $C_1 \curlywedge C_2 \rightsquigarrow C_1 \wedge C_2 \quad (\delta)$ |

**Figure 1.** Basic calculus: $\lambda$-calculus with constants

Our work is in the tradition of dynamic semantics, summed lucidly by Moschovakis [10]: the sense of an expression is the algorithm that allows one to compute the denotation of the expression. Specifically, we follow the 'variable-free dynamic semantics' of [11, 14]. We use our calculus as a logical metalanguage to represent utterances in some abstract form as terms. We define an evaluation procedure to reduce the terms to logical formulas representing denotations. The reduction of a term may fail to produce a formula, which we take as an indication that the corresponding utterance is ungrammatical. We introduce a type system to delineate a set of terms whose reduction never fails to produce denotation. Being well-typed is thus an alternative criterion of grammaticality. Typeability, unlike reducibility, has an advantage of being applicable to separate phrases rather than the whole sentences.

In the next section, we motivate our calculus in two steps, starting from the untyped $\lambda$-calculus. We add a form of delimited control, which causes us to define contexts and distinguish evaluation orders, CBV and CBN. For comparison, we describe the familiar CBV calculus with delimited control, emphasizing thunking that is required for the analyses of quantification and raised *wh*-questions. Our CBN calculus is presented in §3 (dynamic semantics) and §3.1 (static semantics: types). The latter section re-analyzes quantification and raised *wh*-questions without resorting to thunking while maintaining correct predictions. Superiority is analyzed in §3.2. We then discuss the related work and conclude.

## 2. Building up CBN with delimited control

We introduce here the basic calculus, to be extended in the later sections. We draw distinctions (constants vs. terms, evaluation orders) that are not commonly made, are seemingly contrived and irrelevant in simple cases. It is only when we come to effects that the significance of the distinctions emerges.

Our basic calculus is $\lambda$-calculus with constants, which, until §3.1, we regard as untyped. The calculus, Fig. 1, consists of two languages, of *constants* and of *terms*, with a defined reduction relation on terms. The syntax of the calculus is specified by the BNF grammar with non-terminals denoted by the capital letters $C$, $D$, etc.; we may use subscripts to distinguish syntactic elements of the same category. We use $x$, $y$, $z$, and $k$ to refer to variables, with a countably infinite supply. Terms are identified modulo $\alpha$-

conversion. We write $E\{x \mapsto F\}$ for a capture-avoiding substitution of $F$ for $x$ in $E$.

Constants, referring to elements in the appropriate denotational domain, are built from primitive constants $D$, possibly connected by $\wedge$. We also introduce constant parameters $c$, too with a countably infinite supply, and the binding forms $\forall_c$ and $\partial_c$ so that, e.g., $\forall_c \wedge C$ represents $C$ with free occurrences of $c$ universally quantified. We shall stipulate that each new appearance of $\forall_c$ or $\partial_c$ correspond to a unique, 'fresh' $c$. This round-about approach of building quantified predicate logic formulas significantly simplifies the presentation. Essentially the same approach has been previously used in [12, Sec 5]. All constants are terms and serve, in this capacity, as lexical elements. Other terms are familiar variables, applications, abstractions, as well as $E \curlywedge F$. The latter can be explained by stressing that $\mathrm{tall} \curlywedge \mathrm{john}$ is a term that reduces to the constant $\mathrm{tall} \wedge \mathrm{john}$. The difference is akin to that between Scheme's `(cons 1 2)` (which is an expression) and `'(1 . 2)` (which is a literal constant). We take the application and the infix operations $\curlywedge$ and $\wedge$ to be left-associative and so we will write $(xy)z$, $(\mathrm{see} \wedge \mathrm{john}) \wedge \mathrm{mary}$, etc. without parentheses.

The following two terms contain no primitive constants and are taken to be phonetically silent by default:

(1) $\acute{\epsilon} \stackrel{\text{def}}{=} \lambda x.\, \lambda y.\, x \curlywedge y \quad \grave{\epsilon} \stackrel{\text{def}}{=} \lambda x.\, \lambda y.\, y \curlywedge x$

Terms can be evaluated (or, reduced), by applying the transitions $(\beta^*)$ and $(\delta)$ whenever and wherever[1] they apply. For example, the term $\grave{\epsilon}\,\mathrm{mary}(\acute{\epsilon}\,\mathrm{see}\,\mathrm{john})$[2], which in the full form is $(\lambda x.\, \lambda y.\, y \curlywedge x)\,\mathrm{mary}((\lambda x.\, \lambda y.\, x \curlywedge y)\,\mathrm{see}\,\mathrm{john})$ can be reduced by (1) performing applications with mary then with see and john; the transition $(\delta)$ now applies giving $\mathrm{see} \wedge \mathrm{john}$, another $(\beta^*)$ and then $(\delta)$ transitions give $\mathrm{see} \wedge \mathrm{john} \wedge \mathrm{mary}$. We can perform reductions in a different way, again, starting with the application to mary and then substituting the whole term $((\lambda x.\, \lambda y.\, x \curlywedge y)\,\mathrm{see}\,\mathrm{john})$ for $y$. The result is the same. Terms are meant to be representations of utterances, their evaluation being a map to denotations. So far, our map is mere reshuffling.

### 2.1 The interrogation effect

We add a new form of terms in our calculus, who, with the intended meaning of a *wh*-question word, so that the term $\mathrm{tall} \curlywedge \mathrm{who}$ could evaluate to $\partial_c \curlywedge (\mathrm{tall} \curlywedge c)$ and eventually to $\partial_c \wedge (\mathrm{tall} \wedge c)$. Replacing the constant parameter $c$ in the body of $\partial_c$ with various individuals in the semantic domain, we obtain (truth-valued) denotations $\mathrm{tall} \wedge \mathrm{mary}$, $\mathrm{tall} \wedge \mathrm{john}$, etc. If we regard $\partial_c$ to be a binding form we may then call $\partial_c \wedge (\mathrm{tall} \wedge c)$ a characteristic function of the set of tall individuals – which is regarded as the denotation of a question [7]. In a more interesting example, we wish $\grave{\epsilon}\,\mathrm{mary}(\acute{\epsilon}\,\mathrm{see}\,\mathrm{who})$ to evaluate to $\partial_c \curlywedge (\grave{\epsilon}\,\mathrm{mary}(\acute{\epsilon}\,\mathrm{see}\,c))$ – that is, who being replaced with a constant parameter bound "on the outside." The first problem thus is to define the "outside" of a term. The form who raises the second problem, seen in $\grave{\epsilon}\,\mathrm{who}(\acute{\epsilon}\,\mathrm{see}\,\mathrm{who})$, representing the in-situ question "Who saw whom?" If we select the leftmost who for reduction first, the result will be $\partial_{c_1} \wedge (\partial_{c_2} \wedge (\mathrm{see} \wedge c_2 \wedge c_1))$. If we first select the rightmost who for reduction, the result will be $\partial_{c_2} \wedge (\partial_{c_1} \wedge (\mathrm{see} \wedge c_2 \wedge c_1))$. The evaluation order does matter now.

The ambiguity caused by the evaluation order can be taken to be a model of the ambiguity in the original phrase: an ambiguous utterance is described by one term, which evaluates to different denotations depending on the evaluation order. We wish however the evaluation of our terms be deterministic and so more easily analyzable. Thus we map an ambiguous utterance to two terms, each of which evaluates (or fails to evaluate at all) to only one denotation.

To make the evaluation deterministic, we should no longer apply the transitions whenever and wherever; rather, they become *context-sensitive*: a transition applies only if it occurs in the "right" context. The notion of context also solves the previous problem of defining the "outside" of the term.

[1] This phrase assumes congruence rules for transitions, left implicit in Figure 1. We rectify the omission in the following sections.

[2] A reader may notice that after disregarding punctuation and Greek characters the term reads like an uninflected English phrase.

Terms $\quad E, F ::= V \mid FE \mid E \curlywedge F \mid Q \,\$\, E \mid \Downarrow\!\!k.\, E$

Values $\quad V ::= C \mid x \mid \lambda x.\, E$

Coterms $\quad Q ::= k \mid \# \mid E, Q \mid Q; V \mid E,_c Q \mid Q;_c V$

Term equalities

$Q \,\$\, FE = E, Q \,\$\, F \qquad Q \,\$\, VE = Q; V \,\$\, E$

$Q \,\$\, F \curlywedge E = E,_c Q \,\$\, F \qquad Q \,\$\, V \curlywedge E = Q;_c V \,\$\, E$

$\# \,\$\, V = V$

Transitions

$Q_1 \,\$ \cdots \$\, Q_n \,\$\, (\lambda x.\, E)V \rightsquigarrow Q_1 \,\$ \cdots \$\, Q_n \,\$\, E\{x \mapsto V\}$

$Q_1 \,\$ \cdots \$\, Q_n \,\$\, C_1 \curlywedge C_2 \rightsquigarrow Q_1 \,\$ \cdots \$\, Q_n \,\$\, C_1 \wedge C_2$

$Q_1 \,\$ \cdots \$\, Q_n \,\$\, Q \,\$\, \Downarrow\!\!k.\, E \rightsquigarrow Q_1 \,\$ \cdots \$\, Q_n \,\$\, \# \,\$\, E\{k \mapsto Q\}$

**Figure 2.** The $\lambda_v^{sr}$-calculus: syntax and operational semantics. The constants $C$ are defined in Fig. 1

### 2.2 Call-by-value contexts

The most common contexts are left-to-right call-by-value. In computer science, the left-to-right evaluation order (i.e., evaluating the left-hand side of an application before the right-hand side) is mere convention. The choice is not arbitrary in linguistics however: Shan [13, 14] has argued that the left-to-right evaluation order is the only linguistically meaningful. To define the left-to-right CBV contexts, Fig. 2, we first distinguish terms of a certain form (viz., constants, variables, and abstractions) and call them *values*. Values are not further reducible.

Context are commonly introduced as a "term with a hole." We employ an equivalent presentation, which more easily generalizes to other evaluation orders, following the lead of [9]. We introduce *co-terms*: $\#$ (called 'top') and the others built using the comma and the semi-colon connectives, Fig. 2. Term equalities, which may be applied in either direction and at any time, define equivalence classes of terms. If it were not for the restrictions on values in some equalities, the co-term connectives could be called left- and right-adjuncts of the two term connectives: application and $\curlywedge$. Given the original term of the form[3] $\# \,\$\, E$ we 'rotate' (or, 'focus') it using the term equalities into the form where one of the transitions can be performed. Applying the equalities again, the result can be converted to a value or to a form where another transition applies. It could be that a term is neither a value nor convertible to a form suitable for transition, e.g., $\# \,\$\, \mathrm{mary} \curlywedge \lambda x.\, x$. In that case, we *get stuck* – and take the failure of term reductions to yield a value to be the indication the corresponding utterance ungrammatical.

Instead of the term who our calculus defines a more general version $\Downarrow\!\!k.\, E$ (called 'shift'). Like $\lambda x.\, E$, it is a binding form, binding a (co-)variable $k$ in the body $E$. We then define who:

$$\mathrm{who} \stackrel{\text{def}}{=} \Downarrow\!\!k.\, \partial_c \curlywedge (k \,\$\, c) \quad c \text{ is fresh}$$

Our earlier example from §2.1 of a term representing the in-situ question "Who saw whom?" is reduced as follows:

[3] We assume #\$ 'prefixed' to any non-value term that is not already of the form $Q \$ E$, following the convention that the whole program is 'delimited'; see [13] for discussion.

(2) $\# \$ \grave{\epsilon}\,\mathrm{who}(\acute{\epsilon}\,\mathrm{see\,who}) = (\acute{\epsilon}\,\mathrm{see\,who}), \# \$ \grave{\epsilon}\,\mathrm{who}$
$= (\acute{\epsilon}\,\mathrm{see\,who}), \#; \grave{\epsilon} \$ \mathrm{who}$
$\rightsquigarrow \# \$ \partial_{c_1} \curlywedge ((\acute{\epsilon}\,\mathrm{see\,who}), \#; \grave{\epsilon} \$ c_1)$
$= \# \$ \partial_{c_1} \curlywedge ((\acute{\epsilon}\,\mathrm{see\,who}), \# \$ \grave{\epsilon}\, c_1)$
$= \# \$ \partial_{c_1} \curlywedge (\# \$ \grave{\epsilon}\, c_1(\acute{\epsilon}\,\mathrm{see\,who}))$
$= \#;_c \partial_{c_1} \$ \# \$ \grave{\epsilon}\, c_1(\acute{\epsilon}\,\mathrm{see\,who})$
$= \#;_c \partial_{c_1} \$ (\acute{\epsilon}\,\mathrm{see\,who}), \# \$ (\lambda x.\, \lambda y.\, y \curlywedge x) c_1$
$\rightsquigarrow \#;_c \partial_{c_1} \$ (\acute{\epsilon}\,\mathrm{see\,who}), \# \$ \lambda y.\, y \curlywedge c_1$
$= \#;_c \partial_{c_1} \$ \# \$ (\lambda y.\, y \curlywedge c_1)(\acute{\epsilon}\,\mathrm{see\,who})$
$= \#;_c \partial_{c_1} \$ \#; (\lambda y.\, y \curlywedge c_1) \$ \acute{\epsilon}\,\mathrm{see\,who}$
$= \#;_c \partial_{c_1} \$ \mathrm{who}, (\#; (\lambda y.\, y \curlywedge c_1)) \$ (\lambda x.\, \lambda y.\, x \curlywedge y)\,\mathrm{see}$
$\rightsquigarrow \#;_c \partial_{c_1} \$ \mathrm{who}, (\#; (\lambda y.\, y \curlywedge c_1)) \$ \lambda y.\, \mathrm{see} \curlywedge y$
$= \#;_c \partial_{c_1} \$ (\#; (\lambda y.\, y \curlywedge c_1)); (\lambda y.\, \mathrm{see} \curlywedge y) \$ \mathrm{who}$
$\rightsquigarrow \#;_c \partial_{c_1} \$ \# \$ \partial_{c_2} \curlywedge ((\#; (\lambda y.\, y \curlywedge c_1)); (\lambda y.\, \mathrm{see} \curlywedge y) \$ c_2)$
$= \#;_c \partial_{c_1} \$ \#;_c \partial_{c_2} \$ \#; (\lambda y.\, y \curlywedge c_1) \$ (\lambda y.\, \mathrm{see} \curlywedge y) c_2$
$\rightsquigarrow \#;_c \partial_{c_1} \$ \#;_c \partial_{c_2} \$ \#; (\lambda y.\, y \curlywedge c_1) \$ \mathrm{see} \curlywedge c_2$
$\rightsquigarrow \#;_c \partial_{c_1} \$ \#;_c \partial_{c_2} \$ \#; (\lambda y.\, y \curlywedge c_1) \$ \mathrm{see} \wedge c_2$
$= \#;_c \partial_{c_1} \$ \#;_c \partial_{c_2} \$ \# \$ (\lambda y.\, y \curlywedge c_1)(\mathrm{see} \wedge c_2)$
$\rightsquigarrow \#;_c \partial_{c_1} \$ \#;_c \partial_{c_2} \$ \# \$ (\mathrm{see} \wedge c_2) \curlywedge c_1$
$\rightsquigarrow \#;_c \partial_{c_1} \$ \#;_c \partial_{c_2} \$ \# \$ \mathrm{see} \wedge c_2 \wedge c_1$
$= \#;_c \partial_{c_1} \$ \#;_c \partial_{c_2} \$ \mathrm{see} \wedge c_2 \wedge c_1$
$= \#;_c \partial_{c_1} \$ \# \$ \partial_{c_2} \curlywedge (\mathrm{see} \wedge c_2 \wedge c_1)$
$\rightsquigarrow \#;_c \partial_{c_1} \$ \# \$ \partial_{c_2} \wedge (\mathrm{see} \wedge c_2 \wedge c_1)$
$= \# \$ \partial_{c_1} \curlywedge (\partial_{c_2} \wedge (\mathrm{see} \wedge c_2 \wedge c_1))$
$\rightsquigarrow \# \$ \partial_{c_1} \wedge (\partial_{c_2} \wedge (\mathrm{see} \wedge c_2 \wedge c_1))$
$= \partial_{c_1} \wedge (\partial_{c_2} \wedge (\mathrm{see} \wedge c_2 \wedge c_1))$

There is no longer any ambiguity of reductions; in particular, we cannot move $\grave{\epsilon}\,\mathrm{who}$ of the original term into the context because the corresponding equality requires the moved term be a value, which $\grave{\epsilon}\,\mathrm{who}$ is not. The complete reduction sequence above well demonstrates the interleaving of transitions with applications of the equalities. In this approach we need no (implicit or explicit) congruence rules. Performing reductions by hand is quite tedious; the accompanying source code includes the interpreter to reduce terms automatically. We used the interpreter to verify all the examples. For that reason and to save space we shall in the following abbreviate reduction sequences, writing $\rightsquigarrow^*$ for a sequence of transitions interspersed with an arbitrary number of equalities.

A useful variant of who is the term $\iota$ (called 'input')

(3) $\iota \stackrel{\mathrm{def}}{=} \Uparrow k.\, \lambda x.\, k \$ x$

which helps, for example, to represent gapped clauses like "Mary sees _". In the first approximation, we define the silent element _ (called 'trace') [14] as $\iota$

(4) $\_ \stackrel{\mathrm{def}}{=} \iota \quad \check{\epsilon} \stackrel{\mathrm{def}}{=} \lambda x.\, \lambda y.\, yx$

so that the gapped clause has the following representation and reduction:

(5) $\# \$ \grave{\epsilon}\,\mathrm{mary}(\acute{\epsilon}\,\mathrm{see}\,\iota) \quad \rightsquigarrow^* \lambda x.\, \# \$ \mathrm{see} \curlywedge x \curlywedge \mathrm{mary}$

For the full sentence (6) we have

(6) John, Mary sees _

(7) $\# \$ \check{\epsilon}\,\mathrm{john}(\# \$ \grave{\epsilon}\,\mathrm{mary}(\acute{\epsilon}\,\mathrm{see}\,\_))$
$\rightsquigarrow \# \$ (\lambda y.\, y\,\mathrm{john})(\# \$ \grave{\epsilon}\,\mathrm{mary}(\acute{\epsilon}\,\mathrm{see}\,\iota))$
$\rightsquigarrow^* \mathrm{see} \wedge \mathrm{john} \wedge \mathrm{mary}$

where $\check{\epsilon}$ is just the reverse application. The original term in (7) had two occurrences of #$ – one is on the very left, where we are used to seeing it. The other occurrence is inside the term. That occurrence was 'outside' the gapped clause, which is now embedded in the full sentence. Thus the binding of the variable replacing $\iota$ should occur not necessarily on the outside of the whole term. Rather, we wish the binding inserted at the (clausal or other such) boundary. Our calculus is capable of such *delimited* control effect: the action of $\iota$ spreads to the closest dynamic occurrence of a delimiter, which is #$. By convention, the form that places the control delimiter is called *reset*.

One may think that we can easily define $reset$ as in (8)

(8) $reset \stackrel{\mathrm{def}}{=} \lambda x.\, \# \$ x$

(7') $\# \$ \check{\epsilon}\,\mathrm{john}(reset(\grave{\epsilon}\,\mathrm{mary}(\acute{\epsilon}\,\mathrm{see}\,\_)))$
$\rightsquigarrow \# \$ (\lambda y.\, y\,\mathrm{john})(reset(\grave{\epsilon}\,\mathrm{mary}(\acute{\epsilon}\,\mathrm{see}\,\iota)))$
$= \#; (\lambda y.\, y\,\mathrm{john}); reset \$ \grave{\epsilon}\,\mathrm{mary}(\acute{\epsilon}\,\mathrm{see}\,\iota)$
$\rightsquigarrow^* \# \$ \lambda x.\, \#; (\lambda y.\, y\,\mathrm{john}); reset \$ \mathrm{see} \curlywedge x \curlywedge \mathrm{mary}$
$= \lambda x.\, \# \$ (\lambda y.\, y\,\mathrm{john})(reset(\mathrm{see} \curlywedge x \curlywedge \mathrm{mary}))$

Hence if we try to use $reset$ (8) in place of #$ in (7) we get quite a different, and undesired result, (7'), because the argument of $reset$ is evaluated before the application of $reset$. The delimiter remains hidden in the body of $reset$ and fails to delimit the action of $\iota$. We shall see below another instance of this problem of 'argument being evaluated too early'.

We introduce another effect $\rho V$, called 'output'

$\rho \stackrel{\mathrm{def}}{=} \lambda x.\, \Uparrow k.(k \$ x)x \qquad \mathrm{he} \stackrel{\mathrm{def}}{=} \iota$

with the intended meaning of 'marking' the referent of a pronoun, he. The latter is modeled by $\iota$. Informally $\rho V$ is replaced by $V$ at the same time providing $V$ as the input for the closest $\iota$; see [14] for the extensive discussion. Thus the phrase "John's mother saw him" is represented by the term $\grave{\epsilon}(\grave{\epsilon}(\rho\,\mathrm{john})\,\mathrm{mother})(\acute{\epsilon}\,\mathrm{see\,he})$. The term evaluates to the expected denotation $\mathrm{see} \wedge \mathrm{john} \wedge (\mathrm{mother} \wedge \mathrm{john})$. As is [14], we take mother to be an element of the denotational domain mapping individuals to their mothers.

Let us now consider binding in combination with (raised) *wh*-questions, borrowing the examples (9) and (11) from [14]. The straightforward combination of the approaches described earlier gives the terms (10) and (12),

(9) $\mathrm{Who}_i$ _ saw $\mathrm{his}_i$ mother

(10) $\check{\epsilon}(\rho\,\mathrm{who})(\# \$ \grave{\epsilon}\,\_(\acute{\epsilon}\,\mathrm{see}(\grave{\epsilon}\,\mathrm{he\,mother})))$

(11) *$\mathrm{Who}_i$ did $\mathrm{his}_i$ mother see _

(12) $\check{\epsilon}(\rho\,\mathrm{who})(\# \$ \grave{\epsilon}(\grave{\epsilon}\,\mathrm{he\,mother})(\acute{\epsilon}\,\mathrm{see}\,\_))$

which both evaluate to denotations of questions. We thus failed to rule out (11).

The solution to the problems of $reset$ representation (7') and over-generation (11) was proposed by Shan [14]. He introduced a singleton type () and thunks $\lambda().\, E$. We then write our earlier examples as

($4_1$) $\_ \stackrel{\mathrm{def}}{=} \iota\ ()$

($7_1$) $\# \$ \check{\epsilon}(\lambda().\,\mathrm{john})(\# \$ \grave{\epsilon}\,\mathrm{mary}(\acute{\epsilon}\,\mathrm{see}\,\_))$

($10_1$) $\check{\epsilon}(\lambda().(\rho\,\mathrm{who}))(\# \$ \grave{\epsilon}\,\_(\acute{\epsilon}\,\mathrm{see}(\grave{\epsilon}\,\mathrm{he\,mother})))$

($12_1$) $\check{\epsilon}(\lambda().(\rho\,\mathrm{who}))(\# \$ \grave{\epsilon}(\grave{\epsilon}\,\mathrm{he\,mother})(\acute{\epsilon}\,\mathrm{see}\,\_))$

so that ($7_1$) and ($10_1$) give respectively the same results as (7) and (10), whereas ($12_1$) gets stuck as desired, correctly ruling out the phrase (11). Alas, the new definition of trace, ($4_1$), makes us rewrite (7), using $\lambda().\,\mathrm{john}$ rather than just john, see ($7_1$). An individual thus is represented sometimes by the corresponding constant (of the type e, as we will see), sometimes by the thunkified constant of, albeit isomorphic, but still different type $() \to \mathrm{e}$. Uniformity of the analyses suffers.

## 3. Call-by-name calculus

Our development culminates in a CBN delimited-control calculus presented in Fig. 3. We have two forms of functions: strict $\lambda^! u.\, E$ (binding *strict variables* $u$) and general $\lambda x.\, E$. Strict functions are familiar from CBV calculi; they can only be applied to values (because, for example, they do semantic operations on values such as lookup, concatenation, etc). In particular, $\acute{\epsilon}$, $\grave{\epsilon}$ and $\rho$ are now defined as strict:

(13) $\acute{\epsilon} \stackrel{\mathrm{def}}{=} \lambda^! u.\, \lambda^! v.\, u \curlywedge v \quad \grave{\epsilon} \stackrel{\mathrm{def}}{=} \lambda^! u.\, \lambda^! v.\, v \curlywedge u$
$\rho \stackrel{\mathrm{def}}{=} \lambda^! u.\, \Uparrow k.(k \$ u)u$

From the point of view of strict functions, our language is (or, embeds) the earlier $\lambda_v^{sr}$. General functions however – which can

Terms $E, F ::= V \mid x \mid FE \mid E \curlywedge F \mid Q \,\$\, E \mid \Downarrow k : S.\, E$

Values $V ::= C \mid u \mid \lambda x{:}T.\, E \mid W$

Strict Values $W ::= \lambda^{!} u{:}U.\, E$

Coterms $Q ::= k \mid \# \mid E, Q \mid Q ;_{!} W \mid E ,_{c} Q \mid Q ;_{c} V$

Term equalities

$Q \,\$\, FE = E, Q \,\$\, F \qquad Q \,\$\, WE = Q ;_{!} W \,\$\, E$

$Q \,\$\, F \curlywedge E = E ,_{c} Q \,\$\, F \qquad Q \,\$\, V \curlywedge E = Q ;_{c} V \,\$\, E$

$\# \,\$\, V = V$

Transitions

$Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, (\lambda x.\, E) F \rightsquigarrow Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, E\{x \mapsto F\}$

$Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, (\lambda^{!} x.\, E) V \rightsquigarrow Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, E\{x \mapsto V\}$

$Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, C_1 \curlywedge C_2 \rightsquigarrow Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, C_1 \wedge C_2$

$Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, Q \,\$\, \Downarrow k.\, E \rightsquigarrow Q_1 \,\$\, \cdots \,\$\, Q_n \,\$\, \# \,\$\, E\{k \mapsto Q\}$

**Figure 3.** The $\lambda_n^{sr}$-calculus: syntax and operational semantics

substitute in an arbitrary term[4], even $\iota$ – add expressiveness. The general functions and the corresponding transition are the *only* difference of the present CBN calculus from the CBV one in §2.2.

The calculus is typed and of Church-style: all binders, $\lambda$ and $\Downarrow$, are annotated with types, Fig. 4. We disregard the types and type annotations for now since they are irrelevant for evaluation.

We go back to the simpler expression for $\_$ as just $\iota$, see (4). The term (10) representing (9) now evaluates as follows

$$
\begin{aligned}
(10_2)\ & \# \,\$\, \check{\epsilon}(\rho\,\mathrm{who})(\mathit{reset}(\grave{\epsilon}\,\_(\acute{\epsilon}\,\mathrm{see}(\grave{\epsilon}\,\mathrm{he}\,\mathrm{mother})))) \\
&= (\mathit{reset}(\grave{\epsilon}\,\_(\acute{\epsilon}\,\mathrm{see}(\grave{\epsilon}\,\mathrm{he}\,\mathrm{mother})))), \# \,\$\, (\lambda x.\, \lambda y.\, yx)(\rho\,\mathrm{who}) \\
&\rightsquigarrow (\mathit{reset}(\grave{\epsilon}\,\_(\acute{\epsilon}\,\mathrm{see}(\grave{\epsilon}\,\mathrm{he}\,\mathrm{mother})))), \# \,\$\, \lambda y.\, y(\rho\,\mathrm{who}) \\
&= \# \,\$\, (\lambda y.\, y(\rho\,\mathrm{who}))(\mathit{reset}(\grave{\epsilon}\,\_(\acute{\epsilon}\,\mathrm{see}(\grave{\epsilon}\,\mathrm{he}\,\mathrm{mother})))) \\
&\rightsquigarrow \# \,\$\, (\mathit{reset}(\grave{\epsilon}\,\_(\acute{\epsilon}\,\mathrm{see}(\grave{\epsilon}\,\mathrm{he}\,\mathrm{mother}))))(\rho\,\mathrm{who}) \\
&= (\rho\,\mathrm{who}), \# \,\$\, (\lambda x.\, \# \,\$\, x)(\grave{\epsilon}\,\_(\acute{\epsilon}\,\mathrm{see}(\grave{\epsilon}\,\mathrm{he}\,\mathrm{mother}))) \\
&\rightsquigarrow (\rho\,\mathrm{who}), \# \,\$\, \# \,\$\, (\grave{\epsilon}\,\_(\acute{\epsilon}\,\mathrm{see}(\grave{\epsilon}\,\mathrm{he}\,\mathrm{mother}))) \\
&= (\rho\,\mathrm{who}), \# \,\$\, ((\acute{\epsilon}\,\mathrm{see}(\grave{\epsilon}\,\mathrm{he}\,\mathrm{mother})), \#); \grave{\epsilon} \,\$\, \_ \\
&\rightsquigarrow^{*} (\rho\,\mathrm{who}), \# \,\$\, \lambda x.\, \# \,\$\, \grave{\epsilon}\, x(\acute{\epsilon}\,\mathrm{see}(\grave{\epsilon}\,\mathrm{he}\,\mathrm{mother})) \\
&= \# \,\$\, (\lambda x.\, \# \,\$\, \grave{\epsilon}\, x(\acute{\epsilon}\,\mathrm{see}(\grave{\epsilon}\,\mathrm{he}\,\mathrm{mother})))(\rho\,\mathrm{who}) \\
&\rightsquigarrow \# \,\$\, \# \,\$\, \grave{\epsilon}(\rho\,\mathrm{who})(\acute{\epsilon}\,\mathrm{see}(\grave{\epsilon}\,\mathrm{he}\,\mathrm{mother}))
\end{aligned}
$$

and eventually yields $\partial_c \wedge (\mathrm{see} \wedge (\mathrm{mother} \wedge c) \wedge c)$ – which is the correct denotation for (9). We note several instances of substituting the effectful term $(\rho\,\mathrm{who})$ in the body of functions (viz., in the third and the last lines of the reduction sequence). We also note the appearance of *reset* as defined in (8). Unlike the CBV calculus, that straightforward definition now works as intended.

### 3.1 Types

The type system is presented in Figure 4. The system is similar to the one introduced in [9] for a CBV calculus with so-called 'dynamic' delimited control; we refer to the latter paper for more detailed description. The type system distinguishes pure terms, whose evaluation incurs no effect, i.e., includes no $\Downarrow$-transitions – in *any* context and in *any* environment binding terms' free variables, if any. Such terms are given pure types. The types $S \downarrow U$ of terms that may have an effect (at least in some environments) and the types of co-terms $U_1 \uparrow U_2$ can be regarded 'arrow' types (similar to the types of functions), only built from *implications* of a different sort; see [9] for more discussion. A notable difference from the latter's type system is that the range (i.e., the consequent) of these new implications is always a pure type. The domain of a co-type is also always pure: according to the operational semantics, plugging an expression into a context demands the evaluation of the expression. A reified captured context can be a general rather than a strict function however.

[4] In our presentation of the calculus, we therefore distinguish regular variables $x$, which can be substituted by any term, from strict variables $u$, which can only be substituted by values.

Types $T ::= U \mid S \downarrow U$

Pure types $U ::= U \rightharpoonup T \mid T \to T \mid B$

Base types $B ::= \mathrm{t} \mid \mathrm{e} \mid B \rightharpoonup B$

Cotypes $S ::= U \uparrow U$

$$
\frac{\begin{array}{c}[u:U] \\ \vdots \\ E : T\end{array}}{\lambda^{!} u{:}U.\, E : U \rightharpoonup T}\,\lambda^{!}
\qquad
\frac{\begin{array}{c}[x:T_1] \\ \vdots \\ E : T_2\end{array}}{\lambda x{:}T_1.\, E : T_1 \to T_2}\,\lambda
$$

$$
\frac{\begin{array}{c}[k:S] \\ \vdots \\ \# \,\$\, E : U\end{array}}{\Downarrow k : S.\, E : S \downarrow U}\,\Downarrow
\qquad
\frac{\begin{array}{c}[u:U_1] \\ \vdots \\ Q \,\$\, u : U_2\end{array}}{Q : U_1 \uparrow U_2}\,\uparrow\mathrm{I}
$$

$$
\frac{E_1 : (B_2 \rightharpoonup B) \quad E_2 : B_2}{E_1 \curlywedge E_2 : B}\,\rightharpoonup\mathrm{E}
$$

$$
\frac{F : U_1 \rightharpoonup T \quad E : U_2 \quad U_2 \le U_1}{FE : T}\,\rightharpoonup\mathrm{E}
$$

$$
\frac{F : U_1 \rightharpoonup T_1 \quad E : U_2 \uparrow U_I \downarrow U_R \qquad \begin{array}{c}[u : U_2 \quad k : U_I \uparrow U_R] \\ \vdots \\ k \,\dot{\varsigma}\, Fu : T\end{array}}{FE : T}\,\rightharpoonup\mathrm{E}_1
$$

$$
\frac{F : T_1 \to T \quad E : T_2 \quad T_2 \le T_1}{FE : T}\,\to\mathrm{E}
$$

$$
\frac{F : U_1 \uparrow U_I \downarrow U_R \quad E : T_2 \qquad \begin{array}{c}[u : U_1 \quad k : U_I \uparrow U_R] \\ \vdots \\ k \,\dot{\varsigma}\, uE : T\end{array}}{FE : T}\,\to\mathrm{E}_1
$$

$$
\frac{Q : U_1 \uparrow U \quad E : U_2 \quad U_2 \le U_1}{Q \,\$\, E : U}\,\uparrow\mathrm{E}
$$

$$
\frac{Q : S_1 \quad E : S_2 \downarrow U \quad S_1 \le S_2}{Q \,\$\, E : U}\,\downarrow\mathrm{E}
\qquad
\frac{Q : U_I \uparrow U_R \quad E : U}{Q \,\dot{\varsigma}\, E : U \uparrow U_I \downarrow U_R}\,\dot{\varsigma}_U
$$

$$
\frac{Q : U_I \uparrow U_R \quad E : S \downarrow U_2 \quad U_2 \le U_I}{Q \,\dot{\varsigma}\, E : S \downarrow U_R}\,\dot{\varsigma}_T
$$

Typing of constants

$\mathrm{john} : \mathrm{e} \quad \mathrm{mary} : \mathrm{e}$

$\mathrm{tall} : \mathrm{e} \rightharpoonup \mathrm{t} \quad \mathrm{mother} : \mathrm{e} \rightharpoonup \mathrm{e} \quad \mathrm{see} : \mathrm{e} \rightharpoonup \mathrm{e} \rightharpoonup \mathrm{t}$

$c : \mathrm{e} \quad \forall_c : \mathrm{t} \rightharpoonup \mathrm{t} \quad \partial_{cB} : B \rightharpoonup (\mathrm{e} \rightharpoonup B)$

$$
\frac{C_1 : (B_2 \rightharpoonup B) \quad C_2 : B_2}{C_1 \wedge C_2 : B}
$$

**Figure 4.** Types in the $\lambda_n^{sr}$-calculus

The other difference from [9] is yet another arrow type, $U \rightharpoonup T$, corresponding to strict functions. Strict and general functions have different types. The sub-language of constants is also typed (with so-called 'base types' and its own arrow $B \rightharpoonup B$, for which there is no introduction rule); we see the operation $\wedge$ is actually the application in the sub-language. The typing uses an auxiliary relation $Q \,\dot{\varsigma}\, E : T$ defined in the same figure. The typing rules also depend on the subtyping relation $T_1 \le T_2$ defined in Figure 5.

It may appear that the type system is missing rules, for example, to type $\#$. That co-term however is typeable using the existing rules: we note first that for any populated pure type $U$ there is a value of that type; let $V : U$ be such a value. From the equality

$$\frac{}{T \le T} \qquad \frac{U \le U_A \quad U_I \le U_R}{U \le U_A \uparrow U_I \downarrow U_R} \qquad \frac{U'_A \le U_A \quad T_R \le T'_R}{(U_A \rightharpoonup T_R) \le (U'_A \rightharpoonup T'_R)}$$

$$\frac{T'_A \le T_A \quad T_R \le T'_R}{(T_A \to T_R) \le (T'_A \to T'_R)} \qquad \frac{(U_A \rightharpoonup T_R) \le (U'_A \rightharpoonup T'_R)}{(U_A \rightharpoonup T_R) \le (U'_A \to T'_R)}$$

$$\frac{U_A \le U'_A \quad U'_I \le U_I \quad U_R \le U'_R}{(U_A \uparrow U_I \downarrow U_R) \le (U'_A \uparrow U'_I \downarrow U'_R)} \qquad \frac{U'_A \le U_A \quad U_I \le U'_I}{(U_A \uparrow U_I) \le (U'_A \uparrow U'_I)}$$

**Figure 5.** Subtyping relation for types and co-types

$\# \,\$\, V = V$ we conclude that the left-hand side has the type $U$ as well. By applying the rule $\uparrow$I we obtain $\# : U \uparrow U$. The other term equalities likewise let us derive the typing rules for other co-terms.

Our type system is sound: a typed term does not get stuck and so shall evaluate to a denotation. In particular, the type system has the subject-reduction property. The type system for our Church-style calculus is decidable: The accompanying source code presents the constructive proof, the implementation of the terminating type reconstruction algorithm in Twelf.

As just mentioned our calculus is Church style and so bound variables are annotated with types (our Twelf implementation can in some cases infer the annotations). Therefore, terms such as who, $\iota$, etc., which contain binding forms, must be type-annotated correspondingly:

$$\mathrm{who}_B \stackrel{\text{def}}{=} \Downarrow k : \mathrm{e} \uparrow B.\, \partial_{cB} \curlywedge (k \,\$\, c) \quad c \text{ is fresh}$$
$$\iota_{TS} \stackrel{\text{def}}{=} \Downarrow k : S.\, \lambda x{:}T.\, k \,\$\, x$$
$$\rho_{U_1 U_2} \stackrel{\text{def}}{=} \lambda^! u{:}U_1.\, \Downarrow k : U_1 \uparrow (U_1 \to U_2).(k \,\$\, u)u$$

The constant $\partial_{cB}$ also bears the annotation, for the type of its 'body.' The corresponding typing rule in Figure 4 ensures then that the denotation $\partial_{c\,\mathrm{t}} \wedge (\mathrm{tall} \wedge c)$ (seen in §2.1) has the type $\mathrm{e} \rightharpoonup \mathrm{t}$. The type demonstrates that the denotation is not a proposition; rather, it is a characteristic function – in our case, of a set of tall individuals, the set of the true answers to the question "Who is tall?" (see [7] for discussion).

The immediate application of types is to make correct predictions for binding in *wh*-questions, using the earlier examples (9) and (11) (repeated below for ease of reference):

(9) Who$_i$ _ saw his$_i$ mother
(10) $\check{\epsilon}(\rho\,\mathrm{who})(\#\,\$\,\grave{\epsilon}\,\_(\acute{\epsilon}\,\mathrm{see}(\grave{\epsilon}\,\mathrm{he}\,\mathrm{mother})))$
(11) *Who$_i$ did his$_i$ mother see _
(12) $\check{\epsilon}(\rho\,\mathrm{who})(\#\,\$\,\grave{\epsilon}(\grave{\epsilon}\,\mathrm{he}\,\mathrm{mother})(\acute{\epsilon}\,\mathrm{see}\,\_))$

The corresponding terms (10) and (12) both successfully evaluate in CBV if _ is defined as $\iota$, see (4). The terms also successfully evaluate in *untyped* CBN, using the same definition of _. Indeed, if both he and _ are defined to be the same $\iota$, it makes no difference whether the trace occurs before or after the pronoun, and so we cannot rule out the latter case. Here is where the types help.

Although he and _ are indeed the same modulo type annotations, the annotations set them apart:

(14) $\mathrm{he}_S \stackrel{\text{def}}{=} \iota_{eS} \quad \_{}_{TS} \stackrel{\text{def}}{=} \iota_{TS}$

Both terms have the type of the same form: $S \downarrow (T \to U)$. In he, however, $T$ is fixed to be the pure type e. This type assignment is expected: informally, a pronoun expects an individual as 'input'. Trace has no restrictions on the type $T$. With these annotations, (10) is well-typed but (12) cannot be typed. Indeed, the subterm of the latter $(\#\,\$\,\grave{\epsilon}(\grave{\epsilon}\,\mathrm{he}\,\mathrm{mother})(\acute{\epsilon}\,\mathrm{see}\,\_))$, being equal to $(\mathrm{mother}, (((\acute{\epsilon}\,\mathrm{see}\,\_), \#); \grave{\epsilon}))\,;\grave{\epsilon}\,\$\,\mathrm{he}$, has the type $\mathrm{e} \to U$ for some $U$ according to the rule $\downarrow$E of Fig. 4. The subterm cannot therefore be applied to $(\rho\,\mathrm{who})$ because the latter is an effectful term and cannot have the pure type e. Thus we make the correct predictions for binding in *wh*-questions without resorting to thunks and the accompanying type raising, preserving the uniformity of the analyses.

For illustration, here is the term (10) with all type annotations:

$$\#\,\$\,\check{\epsilon}(\rho_{\mathrm{e}\,\mathrm{t}}\,\mathrm{who}_{\mathrm{t}})(\mathit{reset}(\grave{\epsilon}\,\_{}_{TS}(\acute{\epsilon}\,\mathrm{see}(\grave{\epsilon}\,\mathrm{he}_{\mathrm{e}\uparrow\mathrm{t}}\,\mathrm{mother}))))$$
where $S = \mathrm{e}\uparrow(\mathrm{e}\to\mathrm{t}) \quad T = S \downarrow (\mathrm{e} \rightharpoonup \mathrm{t})$

The reconstructed type of the whole term is $\mathrm{e} \rightharpoonup \mathrm{t}$, the type of questions. The type annotation on _ demonstrates that trace may accept an effectful term such as $\rho$ who.

### 3.2 Superiority

We now apply the typed CBN calculus to the analysis of superiority. This application turns out quite straightforward and analogous to the analyses of binding in *wh*-questions in the previous section. Once again we borrow our examples, (15) and (17), from [14]. These phrases are represented as terms (16) and (18) in our calculus.

(15) Who _ saw who?
(16) $\#\,\$\,\check{\epsilon}\,\mathrm{who}(\mathit{reset}(\grave{\epsilon}\,\_(\acute{\epsilon}\,\mathrm{see}\,\mathrm{who})))$
(17) *Who did who see _?
(18) $\#\,\$\,\check{\epsilon}\,\mathrm{who}(\mathit{reset}(\grave{\epsilon}\,\mathrm{who}(\acute{\epsilon}\,\mathrm{see}\,\_)))$

The term (16) evaluates to $\partial_{c_1} \wedge (\partial_{c_2} \wedge (\mathrm{see} \wedge c_2 \wedge c_1))$, which is the expected denotation. This result is anticipated by the fact that (16) is well-typed. Here is that term with the explicit type annotations:

$$\#\,\$\,\check{\epsilon}\,\mathrm{who}_{\mathrm{e}\rightharpoonup\mathrm{t}}(\mathit{reset}(\grave{\epsilon}\,\_{}_{TS}(\acute{\epsilon}\,\mathrm{see}\,\mathrm{who}_{\mathrm{t}})))$$
where $S = \mathrm{e}\uparrow(\mathrm{e}\rightharpoonup\mathrm{t}) \quad T = S \downarrow (\mathrm{e} \rightharpoonup \mathrm{e} \rightharpoonup \mathrm{t})$

The reconstructed type of the whole term is $\mathrm{e} \rightharpoonup \mathrm{e} \rightharpoonup \mathrm{t}$, the type of denotations of double questions. The types are quite informative: one sees at a glance that the left-most who is evaluated before the right-most one. The right-most who is replaced by a variable of the type e bound outside of the term of the type t. The left-most who is too replaced by a variable of the type e, bound outside of the term of the type $\mathrm{e} \rightharpoonup \mathrm{t}$ – that is, the latter binding is outermost.

In contrast, (18) cannot be typed no matter which annotations we may assign to who and trace. In fact, even the simpler (20), corresponding to (19), cannot be typed.

(19) *Mary, who see _?
(20) $\#\,\$\,\check{\epsilon}\,\mathrm{mary}(\mathit{reset}(\grave{\epsilon}\,\mathrm{who}(\acute{\epsilon}\,\mathrm{see}\,\_)))$

The problem lies with the subterm $\grave{\epsilon}\,\mathrm{who}_B(\acute{\epsilon}\,\mathrm{see}\,\_{}_{TS})$, for which we attempt the detailed derivation below. To keep the derivation as general as possible, we fix no concrete type for $B$ and $T$; as we shall see, the typing of $\acute{\epsilon}\,\mathrm{see}\,\_{}_{TS}$ requires $S$ be $\mathrm{e}\uparrow U_1$ for some $U_1$.

$\mathrm{who}_B : \mathrm{e}\uparrow B \downarrow (\mathrm{e} \rightharpoonup B)$ by $\Downarrow$
$\grave{\epsilon} \equiv (\lambda^! u{:}\mathrm{e}.\, \lambda^! v{:}\mathrm{e} \rightharpoonup \mathrm{t}.\, v \curlywedge u) : \mathrm{e} \rightharpoonup (\mathrm{e} \rightharpoonup \mathrm{t}) \rightharpoonup \mathrm{t}$ by $\lambda^!$.
$(k : B \uparrow (\mathrm{e} \rightharpoonup B)) \,\dot{;}\, \grave{\epsilon}(u : \mathrm{e}) : ((\mathrm{e} \rightharpoonup \mathrm{t}) \rightharpoonup \mathrm{t}) \uparrow B \downarrow (\mathrm{e} \rightharpoonup B)$
by $\rightharpoonup$E, $\dot{;}_U$
$\grave{\epsilon}\,\mathrm{who}_B : ((\mathrm{e} \rightharpoonup \mathrm{t}) \rightharpoonup \mathrm{t}) \uparrow B \downarrow (\mathrm{e} \rightharpoonup B)$ by $\rightharpoonup\mathrm{E}_1$
$\acute{\epsilon} \equiv \lambda^! u{:}\mathrm{e} \rightharpoonup \mathrm{e} \rightharpoonup \mathrm{t}.\, \lambda^! v{:}\mathrm{e}.\, u \curlywedge v : (\mathrm{e} \rightharpoonup \mathrm{e} \rightharpoonup \mathrm{t}) \rightharpoonup \mathrm{e} \rightharpoonup (\mathrm{e} \rightharpoonup \mathrm{t})$
$\acute{\epsilon}\,\mathrm{see} : \mathrm{e} \rightharpoonup (\mathrm{e} \rightharpoonup \mathrm{t})$ by $\rightharpoonup$E
$\_{}_{T(\mathrm{e}\uparrow U_1)} : (\mathrm{e}\uparrow U_1) \downarrow (T \to U)$ by $\Downarrow$
$(k : U_1 \uparrow (T \to U)) \,\dot{;}\, (\acute{\epsilon}\,\mathrm{see})(u : \mathrm{e}) : (\mathrm{e} \rightharpoonup \mathrm{t}) \uparrow U_1 \downarrow (T \to U)$
$\acute{\epsilon}\,\mathrm{see}\,\_{}_{T(\mathrm{e}\uparrow U_1)} : (\mathrm{e} \rightharpoonup \mathrm{t}) \uparrow U_1 \downarrow (T \to U)$ by $\rightharpoonup\mathrm{E}_1$
$(\grave{\epsilon}\,\mathrm{who}_B)(\acute{\epsilon}\,\mathrm{see}\,\_{}_{TS}) : T_R$ by $\rightharpoonup\mathrm{E}_1$
**if** $(k : B \uparrow (\mathrm{e} \rightharpoonup B)) \,\dot{;}\, ((u : (\mathrm{e} \rightharpoonup \mathrm{t}) \rightharpoonup \mathrm{t})(\acute{\epsilon}\,\mathrm{see}\,\_{}_{TS})) : T_R$
$(u : (\mathrm{e} \rightharpoonup \mathrm{t}) \rightharpoonup \mathrm{t})(\acute{\epsilon}\,\mathrm{see}\,\_{}_{T(\mathrm{e}\uparrow U_1)}) : \mathrm{t}\uparrow U_1 \downarrow (T \to U)$ by $\rightharpoonup\mathrm{E}_1$
$(k : B \uparrow (\mathrm{e} \rightharpoonup B)) \,\dot{;}\, (E : \mathrm{t}\uparrow U_1 \downarrow (T \to U)) : \mathrm{t}\uparrow U_1 \downarrow (\mathrm{e} \rightharpoonup B)$
by $\dot{;}_T$ **if** $(T \to U) \le B$

The last line of the derivation includes the subtyping condition $(T \to U) \le B$, which came from the side-condition of the rule $\dot{;}_T$. Figure 5 demonstrates that this subtyping relation cannot hold no matter what $T$, $U$ and $B$ are. The subterm $\grave{\epsilon}\,\mathrm{who}_B(\acute{\epsilon}\,\mathrm{see}\,\_{}_{TS})$ and hence (20) and (18) are untypeable.

We observe the other benefit of typing: ruling out the phrases like (17) and (19) on the basis of the failure to type a single subterm. If a subterm cannot be typed, the whole term cannot be typed. In contrast, in a calculus with (control) effects the failure to reduce a closed subterm on its own does not imply at all that a sentence with that subterm gets stuck.

## 4. Related Work

Herbelin and Ghilezan [6] too developed a series of CBN calculi with delimited control. Our approaches have markedly different motivations and hence arrive at different results. The calculi of [6] are motivated by the desire to closely represent abstract machines for delimited control and their components such as meta-continuations; to facilitate investigations of computational duality and classical reasoning. We, in contrast, aim at operational and axiomatic formulation of delimited control, considering meta-continuations a part of implementation, which we abstract over. We also specifically aim to make the CBN calculus as close as possible to the familiar CBV calculus with delimited control. All $\lambda_v^{sr}$ examples in the paper can be evaluated in $\lambda_n^{sr}$ exactly as they are – often yielding the same results despite different transition sequences. Our calculus also easily generalizes to dynamic delimited continuations. Because of subtyping, we can type more terms compared to [6]; we have used subtyping extensively in our analyses.

One straightforward way of building CBN calculi with delimited control is by emulating CBN in a CBV calculus with thunks. The type system of the CBV calculus with delimited control (described in [1, 3]) then naturally extends to the resulting CBN calculus. That emulation is a global term transformation affecting all parts of a term; the individuals will no longer be denoted by terms of the type $\mathrm{e}$ but of the different (albeit iso-morphic) type $() \to \mathrm{e}$. This procedure is reminiscent of type raising, which we wished to escape from using the metalanguage with delimited control. Our type system is also quite distinct from that of [1, 3] in that our types are built out of connectives with the meaning of logical implication (see [9] for more discussion).

Shan [14] has been the inspiration and the template for the present work. Our calculi differ – CBV in [14] vs. CBN here – and so do the analyses. Another difference is the explicit use of types in the present work. We are also more explicit in denotations of questions, differentiating genuine questions (a constant of the type $\mathrm{e} \to \mathrm{t}$, for example) from a seemingly similar gapped clause (which is a term of the type $\mathrm{e} \to \mathrm{t}$).

We distinguish our CBN calculus of delimited control from the Lambek-Grishin calculus of [2]. We define delimited control directly and operationally, without appealing to CBN CPS denotations. Mainly, our calculus, like that of [9], is a substructural logic with neither negation nor multiple conclusions, and thus intuitionistic in character.

## 5. Conclusions

We have presented the typed CBN calculus of delimited control and showed that types and CBN are both necessary for correct prediction of superiority and binding in *wh*-questions with topicalization without resorting to thunking. We thus maintain the uniformity of the analyses: individuals are denoted by the terms that have the type just $\mathrm{e}$ in all the cases, with no need for raising. Linguistics turns out to offer the first interesting application of the typed CBN shift/reset.

The immediate future work is to analyse intensional phrases, in particular, coordination including de-dicto phrases. It is interesting to more formally relate de-re/de-dicto and CBV/CBN distinctions.

**Acknowledgments**

This paper could not have been written without numerous extensive discussions with Chung-chieh Shan and without his encouragement. I am deeply indebted to him for his explanations, advice, and support. I thank Rui Otake and the anonymous reviewers for many helpful comments.

## References

[1] Asai, Kenichi, and Yukiyoshi Kameyama. 2007. Polymorphic delimited continuations. In *APLAS*, vol. 4807 of *LNCS*, 239–254.

[2] Bernardi, Raffaella, and Michael Moortgat. 2007. Continuation semantics for symmetric categorial grammar. In *WoLLIC*, vol. 4576 of *LNCS*, 53–71. Springer.

[3] Danvy, Olivier, and Andrzej Filinski. 1989. A functional abstraction of typed contexts. Tech. Rep. 89/12, DIKU. `http://www.daimi.au.dk/~danvy/Papers/fatc.ps.gz`.

[4] ———. 1990. Abstracting control. In *Proc. conf. Lisp & funct. prog.*, 151–160.

[5] Filinski, Andrzej. 1994. Representing monads. In *POPL*, 446–457.

[6] Herbelin, Hugo, and Silvia Ghilezan. 2008. An approach to call-by-name delimited continuations. In *POPL*, 383–394.

[7] Karttunen, Lauri. 1977. Syntax and Semantics of Questions. *Linguistics and Philosophy* 1(1):3–44.

[8] ———. 2006. The insufficiency of paper-and-pencil linguistics: the case of Finnish prosody. In *Intelligent linguistic architectures: Variations on themes*, ed. Ronald M. Kaplan, Miriam Butt, Mary Dalrymple, and Tracy Holloway King, 287–300. CSLI Publications, Stanford, California.

[9] Kiselyov, Oleg, and Chung-chieh Shan. 2007. A substructural type system for delimited continuations. In *TLCA*, vol. 4583 of *LNCS*, 223–239. Springer.

[10] Moschovakis, Yiannis. 1994. Sense and Denotation as Algorithm and Value. In *Logic colloquium '90*, ed. Jouko Väänänen and Juha Oikkonen, 382–396.

[11] Shan, Chung-chieh. 2001. A variable-free dynamic semantics. In *Proc. 13th Amsterdam Colloquium*, ed. Robert van Rooy and Martin Stokhof, 204–209. ArXiv.org:cs/0205027.

[12] ———. 2004. Delimited continuations in natural language: Quantification and polarity sensitivity. In *CW'04: Proceedings of the 4th ACM SIGPLAN continuations workshop*, ed. Hayo Thielecke, 55–64. Tech. Rep. CSR-04-1, School of Computer Science, University of Birmingham.

[13] ———. 2005. Linguistic side effects. Ph.D. thesis, Harvard U.

[14] ———. 2007. Linguistic side effects. In *Direct compositionality*, ed. Chris Barker and Pauline Jacobson, 132–163. New York: Oxford University Press.