Untyped polarized calculi

XAVIER MONTILLET

We revisit the polarized L calculus, an abstract-machine-like calculus with mixed evaluation order (i.e. call-by-name and call-by-value) and pattern-matches, and its relation to the λ-calculus. We then show that it is a more symmetric syntax for Call-By-Push-Value. We also introduce a dynamically typed / bi-typed variant of this calculus which completely eliminates clashes (i.e. pattern-matching failures) without relying on any form of typing judgments, and illustrate its usefulness in the study of extensions of the untyped λ-calculus with constructors.

ACM Reference Format:
Xavier Montillet. 2020. Untyped polarized calculi. 1, 1 (May 2020), 15 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

INTRODUCTION

History

The λ-calculus is a well-known abstraction used to study programming languages. It has two main evaluation strategies: *call-by-name* (CBN) evaluates subprograms only when they are observed / used, while *call-by-value* (CBV) evaluates subprograms when they are constructed. Each strategy has its own advantage: CBN ensures that no unnecessary computations are done, while CBV ensures that no computations are duplicated. Somewhat surprisingly, the study of CBV turned out to be more involved than that of CBN, for example requiring computation monads [12, 13] to build models. Some properties of CBN, given by Barendregt in 1984 [1], have yet to be adapted to CBV. *Call-by-push-value* (CBPV) [10, 11] decomposes Moggi's computation monad as an adjunction, subsumes both CBV and CBN and sheds some light on the interactions and differences of both strategies.

Another direction the λ-calculus has evolved in is the computational interpretation of classical logic, with the continuation-passing style translation and the $\lambda\mu$-calculus [16]. This eventually led to the $\bar{\lambda}\mu\tilde{\mu}$-calculus [3], which instead of having natural deduction as type system, has the sequent calculus. An interesting property of $\bar{\lambda}\mu\tilde{\mu}$ is that it resembles both the λ-calculus and the Krivine abstract machine [9], allowing to speak of both the equational theory and the operational semantics. It also sheds more light on the relationship between CBN and CBV: the full calculus is not confluent because of the Lafont critical pair [8]

$$c^1\left[\tilde{\mu}x\,.\,c^2\middle/\alpha\right] \lhd \left\langle \mu\alpha\,.\,c^1 \mathbin{||} \tilde{\mu}x\,.\,c^2 \right\rangle \rhd c^2\left[\mu\alpha\,.\,c^1\middle/x\right]$$

where $\mu\alpha\,.\,c^1$ represents "the result of running the computation c^1" and $\tilde{\mu}x\,.\,c^2$ represents the context $\mathsf{let}\ x\ =\ \square\ \mathsf{in}\ c^2$, so that the critical pair can be reformulated (if we restrict ourselves to the intuitionistic fragment) as

$$\mathsf{let}\ x\ =\ \underline{T^1}\ \mathsf{in}\ T^2 \lhd \underline{\mathsf{let}\ x\ =\ T^1\ \mathsf{in}\ T^2} \rhd \underline{T^2\left[T^1\middle/x\right]}$$

(where the underlined subterm is the one that the machine is currently trying to evaluate). This is exactly the distinction between CBV (where we want to evaluate T^1 before substituting it), and CBN (where we substitute it immediately). Since CBV is syntactically dual to CBN in $\bar{\lambda}\mu\tilde{\mu}$, the additional difficulty in the study of CBV can be understood as coming from the restriction to the intuitionistic fragment (as illustrated in Figures A.1 and A.2) which breaks this symmetry.

Surprisingly, those two lines of work (CBPV and $\bar{\lambda}\mu\tilde{\mu}$) lead to very similar calculi (especially if one looks at the abstract machine of CBPV), and both can be combined into a polarized sequent calculus LJ_p^η [2], an intuitionistic variant of (a syntax for) Danos, Joinet and Schellinx's LK_p^η [4]. The main difference between (the abstract machine of) CBPV and LJ_p^η is the same as that of the Krivine abstract machine and the CBN fragment of $\bar{\lambda}\mu\tilde{\mu}$: Subcomputations are also represented by subcommands / subconfigurations, so that the "abstract machine style" evaluation is no longer restricted to the top-level. The difference between $\bar{\lambda}\mu\tilde{\mu}$ and LJ_p^η is that instead of begin restricted to a single evaluation strategy (which is necessary in $\bar{\lambda}\mu\tilde{\mu}$ to preserve confluence), both are available, and commands are annotated by a polarity $+$ (for CBV) or $-$ (for CBN) to denote the current evaluation strategy, which removes the Lafont critical pair. The type system also changes from classical logic to intuitionistic logic with explicitly-polarised connectives.

In this article, we use a slight variation of LJ_p^η which we will call $\mathrm{L_p}$, the main difference being that the calculus is untyped but well-polarized. This calculus inherits many of the advantages of $\bar{\lambda}\mu\tilde{\mu}$: it is abstract-machine-like so that weak head evaluation is just top-level reduction; commuting conversions are derivable and give rise to a confluent reduction; the classical (as in classical logic) binder μ is available and the full calculus exhibits a perfect symmetry between CBN and CBV; it is easy to restrict to the intuitionistic fragment, and the way in which this breaks the symmetry gives some insight into why CBV is harder than CBN; applicative contexts can be represented by stacks and plugging a term in an applicative context can therefore be seen a substituting a stack for a stack variable. It also inherits many of the advantages of CBPV: It subsumes CBN and CBV and allows mixing both evaluation strategies; it has nice models; and natural η-conversion laws. The additional restriction to well-polarized terms restricts the possible shapes of *clashes* (i.e. pattern-matching failures). It also makes the "dynamically typed" variant (in which pattern-matches match over all constructors) clashless.

Author's address: Xavier Montillet.

XXXX-XXXX/2020/5-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Goals

Given a calculi, one has two choices of syntax: a λ-calculus-like / natural-deduction-like syntax or an abstract-machine-like / sequent-like syntax. Both choices are equivalent in terms of what they represent, and it is easy to translate terms from one to the other. However, for most, if not all, uses, the abstract-machine-like syntax will make everything (definitions, proofs, getting intuition, ...) easier. The cost of using an abstract-machine-like syntax is unfortunately still very high: One has to step out of the well-known syntax of the λ-calculus, therefore making results more difficult to understand by many, and one often does not have the space to describe everything in both variants of the calculus[1]. The main goals of this article are:

- To provide a self-contained introduction to abstract-machine-like calculi, by showing all the steps involved in transforming a λ-calculus-like into an abstract-machine-like syntax;
- To provide a self-contained description of $\mathrm{L_p}$, its equivalent λ-calculus-like syntax λ_{p}, and its link with well known calculi (call-by-name and call-by-value λ-calculi, Call-by-push-value, ...);
- To convince the reader that the abstract-machine-like syntax indeed makes (nearly) everything (definitions, proofs, getting intuition, ...) easier;
- To put forward and motivate the use of dynamically typed / bi-typed calculi for the study of untyped programs.

The main technical contributions of this article is the introduction of the $\lambda_{\mathrm{p}}^{\to\&\Uparrow\otimes\oplus\Downarrow}$ calculus and the description of its relation with $\mathrm{L_p}^{\to\&\Uparrow\otimes\oplus\Downarrow}$ and Call-by-push-value, and hence of the relation between $\mathrm{L_p}^{\to\&\Uparrow\otimes\oplus\Downarrow}$ and Call-by-push-value. Minor technical contributions include: The concise description of the intuitionistic fragment $\mathrm{L_p}^{\to\&\Uparrow\otimes\oplus\Downarrow}$, a syntactic description of the direct-style embedding of CBV in CBN with downshifts.

Outline

In Section 1, we introduce a pure polarized calculus $\lambda_{\mathrm{p}}^{\to}$ and embed the call-by-name and call-by-value λ-calculi in it. In Section 2, we extend $\lambda_{\mathrm{p}}^{\to}$ with datatypes, yielding $\lambda_{\mathrm{p}}^{\to\&\Uparrow\otimes\oplus\Downarrow}$ and describe its relation to Call-by-push-value. In Section 3, we describe the progressive transformation of a λ-calculus-like syntax into an abstract-machine-like syntax, and give an abstract-machine-like syntax to $\lambda_{\mathrm{p}}^{\to\&\Uparrow\otimes\oplus\Downarrow}$: $\mathrm{L_p}^{\to\&\Uparrow\otimes\oplus\Downarrow}$. In Section 4, we look at solvability and η-conversion in $\mathrm{L_p}^{\to\&\Uparrow\otimes\oplus\Downarrow}$, showcasing its advantages.

Conventions and notations

In this article, we will describe many calculi, and will use the same conventions for all of them.

Calculi. We write $T[V/x]$ for the capture-avoiding substitution of the free occurrences of x by V in T. We also use contexts $\mathbb{K}$, i.e. expressions (terms, values, ...) with a hole $\square$. We also write $\mathbb{T}$ for a term with a hole, $\mathbb{V}$ for a value with a hole, ...). We write $\mathbb{K}\llbracket T\rrbracket$ for the result of plugging T in $\mathbb{K}$, i.e. the result of the *non*-capture-avoiding substitution of the unique occurrence of $\square$ by T in $\mathbb{K}$.

Similar constructions in different calculi will be differentiated by adding a symbol: n for call-by-name, v for call-by-value, p for polarized (or + and − when the polarized calculus contains two variants). We also differentiate between λ-calculus-like calculi, whose terms T, values V, and the rest are denoted by uppercase letters, and abstract-machine-like calculi whose terms t, values v, and the rest are denoted by lowercase letters.

The translation of a term (or value, or ...) into another calculi will be denoted by the term underlined, with a subscript specifying the target calculus. For example $\underline{T_{\mathrm{n}}}_{\mathrm{v}}$ is the translation of call-by-name λ-term T_{n} into the call-by-value λ-calculus.

Reductions. We use three reductions: The top-level reduction $>$ is used to factor the definitions of the two other reductions. The operational reduction $\triangleright$ is the one that defines the operational semantics of the calculus, and can be defined as the closure or the top-level reduction $>$ under a chosen set of contexts, called operational contexts and denoted by $\mathbb{O}$. For all the calculi in this paper, the operational reduction $\triangleright$ is deterministic (i.e. $T_1 \triangleleft T \triangleright T_2$ implies $T_1 = T_2$). The strong reduction $\to$ defines the (oriented) equational theory, and is defined as the closure of the top-level reduction $>$ under all contexts (i.e. it can reduce anywhere). Since it is both simple and long, its definition will remain implicit for all calculi.

We write $\rightsquigarrow$ for an arbitrary reduction (i.e. an arbitrary binary relation whose domain and codomain are equal). Given a reduction $\rightsquigarrow$, we write $\rightsquigarrow^+$ for its transitive closure and $\rightsquigarrow^*$ for its reflexive transitive closure. We say that T $\rightsquigarrow$-reduces to T', written $T \rightsquigarrow T'$, when $\left(T, T'\right) \in \rightsquigarrow$. Relations will sometimes be used as predicate in which case the second argument is to be understood as existentially quantified (e.g. $T \rightsquigarrow$ means that there exists T' such that $T \rightsquigarrow T'$) unless the relation is striked in which case it should be understood as universally quantified (e.g. $T \not\rightsquigarrow$ means that for all T', $T \not\rightsquigarrow T'$, in other words there exists no T' such that $T \rightsquigarrow T'$). We will say that T is $\rightsquigarrow$-reducible if $T \rightsquigarrow$ and $\rightsquigarrow$-normal otherwise. We will say that T' is a $\rightsquigarrow$-normal form of T if $T \rightsquigarrow^* T' \not\rightsquigarrow$, and that T has an $\rightsquigarrow$-normal form if such a T' exists. If $\rightsquigarrow$ is deterministic, we will say that T $\rightsquigarrow$-converges if it has a normal form, and that it diverges otherwise.

1 PURE λ-CALCULI

1.1 Pure call-by-name λ-calculus

We recall the pure call-by-name λ-calculus, we which we will call $\lambda_{\mathrm{n}}^{\to}$, in Figure 1.1. When compared with the usual presentation, there are a few slight differences. First, in order to differentiate it from the other calculi that will be introduced, we added n everywhere and have $T_{\mathrm{n}} @^{\mathrm{n}} V_{\mathrm{n}}$ instead of $T_{\mathrm{n}} V_{\mathrm{n}}$ in the formal syntax for the application of T_{n} to V_{n}. We will still write $T_{\mathrm{n}} V_{\mathrm{n}}$ for $T_{\mathrm{n}} @^{\mathrm{n}} V_{\mathrm{n}}$ (and $T_{\mathrm{n}} V_{\mathrm{n}} W_{\mathrm{n}}$ for $(T_{\mathrm{n}} V_{\mathrm{n}}) W_{\mathrm{n}}$) when the calculus in which the application takes place is clear. Secondly, since we are in a call-by-name calculus, there is no distinction between terms T_{n} and values V_{n}, and both can be used interchangeably. We will nevertheless name V_{n} any term that will be substituted for a variable to keep the naming convention similar to that of the call-by-value calculi. Thirdly, we added let-expressions $\mathsf{let}\ x^{\mathrm{n}} = V_{\mathrm{n}}\ \mathsf{in}\ T_{\mathrm{n}}$, even though they behave exactly like $(\lambda x^{\mathrm{n}}.\, T_{\mathrm{n}}) V_{\mathrm{n}}$,

---

[1] This leads some authors to resigning themselves to doing everything in the λ-calculus-like syntax, even though the intuition comes from the abstract-machine-like syntax [17]. (One can search "sequent" to find mentions of the abstract-machine-like syntax)

Terms / values:

$$T_{\mathrm{n}}, U_{\mathrm{n}}, V_{\mathrm{n}}, W_{\mathrm{n}} \quad ::= \quad x^{\mathrm{n}} \mid \lambda x^{\mathrm{n}}.\, T_{\mathrm{n}} \mid T_{\mathrm{n}} \mathbin{@^{\mathrm{n}}} V_{\mathrm{n}} \mid \mathsf{let}\ x^{\mathrm{n}} = V_{\mathrm{n}}\ \mathsf{in}\ T_{\mathrm{n}}$$

(a) Syntax

$$\begin{array}{lcl} (\lambda x^{\mathrm{n}}.\, T_{\mathrm{n}}) \mathbin{@^{\mathrm{n}}} V_{\mathrm{n}} & \succ & T_{\mathrm{n}}[V_{\mathrm{n}}/x^{\mathrm{n}}] \\ \mathsf{let}\ x^{\mathrm{n}} = V_{\mathrm{n}}\ \mathsf{in}\ T_{\mathrm{n}} & \succ & T_{\mathrm{n}}[V_{\mathrm{n}}/x^{\mathrm{n}}] \end{array}$$

(b) Top-level reduction

Operational contexts:

$$\mathbb{O}_{\mathrm{n}} \quad ::= \quad \square \mid \mathbb{O}_{\mathrm{n}} \mathbin{@^{\mathrm{n}}} V_{\mathrm{n}} \qquad\qquad \frac{T_{\mathrm{n}} \succ T'_{\mathrm{n}}}{\mathbb{O}_{\mathrm{n}}\boxed{T_{\mathrm{n}}} \triangleright \mathbb{O}_{\mathrm{n}}\boxed{T_{\mathrm{n}}}}$$

(c) Operational reduction

Fig. 1.1. Pure call-by-name λ-calculus: $\lambda_{\mathrm{n}}^{\rightarrow}$

Values:

$$V_{\mathrm{v}}, W_{\mathrm{v}} \quad ::= \quad x^{\mathrm{v}} \mid \lambda x^{\mathrm{v}}.\, T_{\mathrm{v}}$$

Terms:

$$T_{\mathrm{v}}, U_{\mathrm{v}} \quad ::= \quad \mathsf{val}^{\mathrm{v}}(V_{\mathrm{v}}) \mid T_{\mathrm{v}} \mathbin{@^{\mathrm{v}}} V_{\mathrm{v}} \mid \mathsf{let}\ x^{\mathrm{v}} = T_{\mathrm{v}}\ \mathsf{in}\ U_{\mathrm{v}}$$

(a) Syntax

$$\begin{array}{lcl} (\lambda x^{\mathrm{v}}.\, T_{\mathrm{v}}) \mathbin{@^{\mathrm{v}}} V_{\mathrm{v}} & \succ & T_{\mathrm{v}}[V_{\mathrm{v}}/x^{\mathrm{v}}] \\ \mathsf{let}\ x^{\mathrm{v}} = V_{\mathrm{v}}\ \mathsf{in}\ T_{\mathrm{v}} & \succ & T_{\mathrm{v}}[V_{\mathrm{v}}/x^{\mathrm{v}}] \end{array}$$

(b) Top-level reduction

Operational contexts:

$$\mathbb{O}_{\mathrm{v}} \quad ::= \quad \square \mid \mathbb{O}_{\mathrm{v}} \mathbin{@^{\mathrm{v}}} V_{\mathrm{v}} \mid \mathsf{let}\ x^{\mathrm{v}} = \mathbb{O}_{\mathrm{v}}\ \mathsf{in}\ U_{\mathrm{v}} \qquad\qquad \frac{T_{\mathrm{v}} \succ T'_{\mathrm{v}}}{\mathbb{O}_{\mathrm{v}}\boxed{T_{\mathrm{v}}} \triangleright \mathbb{O}_{\mathrm{v}}\boxed{T_{\mathrm{v}}}}$$

(c) Operational reduction

Fig. 1.2. Pure call-by-value λ-calculus: $\lambda_{\mathrm{v}}^{\rightarrow}$

because their translations into other calculi will be simpler than that of $(\lambda x^{\mathrm{n}}.\, T_{\mathrm{n}})V_{\mathrm{n}}$.

1.2 Pure call-by-value λ-calculus

We recall the pure call-by-value λ-calculus, which we will call $\lambda_{\mathrm{v}}^{\rightarrow}$, in Figure 1.2. We again added v everywhere, have $T_{\mathrm{v}} \mathbin{@^{\mathrm{v}}} V_{\mathrm{v}}$ instead of $T_{\mathrm{v}}V_{\mathrm{v}}$, and added let-expressions. We also made the inclusion of values into terms explicit: The value V_{v} seen as a term is $\mathsf{val}^{\mathrm{v}}(V_{\mathrm{v}})$, and not just V_{v}. Since the context-free grammar remains non-ambiguous without it, we will leave this conversion implicit most of the time, for example writing $\lambda x^{\mathrm{v}}.\, x^{\mathrm{v}}$ for the identity instead of $\lambda x^{\mathrm{v}}.\, \mathsf{val}^{\mathrm{v}}(x^{\mathrm{v}})$. This will however be useful when translating from $\lambda_{\mathrm{v}}^{\rightarrow}$ to another language as we can translate V_{v} and $\mathsf{val}^{\mathrm{v}}(V_{\mathrm{v}})$ differently (as is done in Figure 1.6).

We also restricted the application so that the argument has to be a value, i.e. the application is $T_{\mathrm{v}}V_{\mathrm{v}}$ and not $T_{\mathrm{v}}U_{\mathrm{v}}$. Note that there are 4 possibilities ($T_{\mathrm{v}}U_{\mathrm{v}}$, $V_{\mathrm{v}}U_{\mathrm{v}}$, $T_{\mathrm{v}}W_{\mathrm{v}}$ or $V_{\mathrm{v}}W_{\mathrm{v}}$) and that those are all equivalent in terms of expressiveness because we can let-expand terms: $\mathsf{let}\ x^{\mathrm{v}} = T_{\mathrm{v}}\ \mathsf{in}\ \mathsf{let}\ y^{\mathrm{v}} = U_{\mathrm{v}}\ \mathsf{in}\ x^{\mathrm{v}}y^{\mathrm{v}}$ simulates $T_{\mathrm{v}}U_{\mathrm{v}}$ with left-to-right evaluation (i.e. evaluation of T_{v} before U_{v}) and $\mathsf{let}\ y^{\mathrm{v}} = U_{\mathrm{v}}\ \mathsf{in}\ \mathsf{let}\ x^{\mathrm{v}} = T_{\mathrm{v}}\ \mathsf{in}\ x^{\mathrm{v}}y^{\mathrm{v}}$ simulates $T_{\mathrm{v}}U_{\mathrm{v}}$ with right-to-left evaluation (i.e. evaluation of U_{v} before T_{v}). There are two reasons for our choice of not allowing terms as arguments:

First, the calculi in which we will embed $\lambda_{\mathrm{v}}^{\rightarrow}$ naturally restricts the argument to being a value, and allowing terms as arguments would therefore make the embeddings more complex: The translation of $T_{\mathrm{v}}U_{\mathrm{v}}$ or $V_{\mathrm{v}}U_{\mathrm{v}}$ would have to contain the let-expansion. In other words, we would be describing the composition of let-expansion (i.e. the translation from $\lambda_{\mathrm{v}}^{\rightarrow}$ with $T_{\mathrm{v}}U_{\mathrm{v}}$ to $\lambda_{\mathrm{v}}^{\rightarrow}$ with $T_{\mathrm{v}}V_{\mathrm{v}}$) with the translation from $\lambda_{\mathrm{v}}^{\rightarrow}$ with $T_{\mathrm{v}}V_{\mathrm{v}}$ to the other calculus.

Secondly, it condenses the difference between call-by-name and call-by-value to a single spot: let-expressions. If $T_{\mathrm{v}}U_{\mathrm{v}}$ or $V_{\mathrm{v}}U_{\mathrm{v}}$ are allowed, then the reductions for the application also differ between call-by-value and call-by-name.

1.3 Relative expressiveness of call-by-name and call-by-value

The fundamental distinction between call-by-name and call-by-value is how let-expressions are reduced, as shown below. In call-by-name a let-expression $\mathsf{let}\ x = T\ \mathsf{in}\ U$ is immediately reduced to $U[T/x]$ because any T is a value, whereas in call-by-value the term T is first reduced until is reaches a value W (and if it never does, i.e. T diverges, then so does $\mathsf{let}\ x = T\ \mathsf{in}\ U$) and only then does the substitution happen.

$$\begin{array}{lcl lcl} \mathsf{let}\ x^{\mathrm{n}} = T_{\mathrm{n}}\ \mathsf{in}\ U_{\mathrm{n}} & = & \mathsf{let}\ x^{\mathrm{n}} = V_{\mathrm{n}}\ \mathsf{in}\ U_{\mathrm{n}} & \triangleright & U_{\mathrm{n}}[V_{\mathrm{n}}/x^{\mathrm{n}}] \\ \mathsf{let}\ x^{\mathrm{v}} = T_{\mathrm{v}}\ \mathsf{in}\ U_{\mathrm{v}} & \triangleright^{*} & \mathsf{let}\ x^{\mathrm{v}} = W_{\mathrm{v}}\ \mathsf{in}\ U_{\mathrm{v}} & \triangleright & T_{\mathrm{n}}[W_{\mathrm{n}}/x^{\mathrm{n}}] \end{array}$$

With that in mind, we now look at how $\lambda_{\mathrm{n}}^{\rightarrow}$ and $\lambda_{\mathrm{v}}^{\rightarrow}$ can be embedded in each other in direct style (i.e. not in continuation-passing style). In section 1.3.1, we give an embedding of $\lambda_{\mathrm{v}}^{\rightarrow}$ into a slight extension of $\lambda_{\mathrm{n}}^{\rightarrow}$ called $\lambda_{\mathrm{n}}^{\rightarrow\Downarrow}$, and in section 1.3.2, we give an embedding of $\lambda_{\mathrm{n}}^{\rightarrow}$ into a slight extension of $\lambda_{\mathrm{v}}^{\rightarrow}$ called $\lambda_{\mathrm{v}}^{\rightarrow\Uparrow}$. Since there is a translation from $\lambda_{\mathrm{v}}^{\rightarrow\Uparrow}$ to $\lambda_{\mathrm{v}}^{\rightarrow}$, we could have embedded $\lambda_{\mathrm{n}}^{\rightarrow}$ into $\lambda_{\mathrm{v}}^{\rightarrow}$ directly, but introducing $\lambda_{\mathrm{v}}^{\rightarrow\Uparrow}$ makes the translation easier to understand, and the duality between CBN and CBV more apparent.

1.3.1 *Embedding call-by-name in call-by-value.*

The extension of $\lambda_{\mathrm{v}}^{\rightarrow}$, called $\lambda_{\mathrm{v}}^{\rightarrow\Uparrow}$, is defined in Figure 1.3. Given a computation T_{v}, we add $\mathsf{freeze}^{\mathrm{v}}(T_{\mathrm{v}})$ which represents the computation T_{v} paused: $\mathsf{freeze}^{\mathrm{v}}(T_{\mathrm{v}}) \not\triangleright$. The computation can later be resumed: $\mathsf{unfreeze}^{\mathrm{v}}(\mathsf{freeze}^{\mathrm{v}}(T_{\mathrm{v}})) \triangleright T_{\mathrm{v}}$. Since $\mathsf{freeze}^{\mathrm{v}}(T_{\mathrm{v}})$ is a value, we can now pass "paused" computations to functions, and let these functions resume the computation if needed by means of the $\mathsf{unfreeze}^{\mathrm{v}}$ construction. In a typed calculus, $\mathsf{freeze}^{\mathrm{v}}$ would be the constructor of a type $\Uparrow A$ called upshift, and $\mathsf{unfreeze}^{\mathrm{v}}$ its destructor, as shown in Figure 1.3. Both $\mathsf{freeze}^{\mathrm{v}}$ and $\mathsf{unfreeze}^{\mathrm{v}}$ can actually be encoded in $\lambda_{\mathrm{v}}^{\rightarrow}$ so that there is a translation $\lambda_{\mathrm{v}}^{\rightarrow\Uparrow} \rightarrow \lambda_{\mathrm{v}}^{\rightarrow}$. The idea is that we can take $\mathsf{freeze}^{\mathrm{v}}(T_{\mathrm{v}}) = \lambda x^{\mathrm{v}}.\, T_{\mathrm{v}}$ and $\mathsf{unfreeze}^{\mathrm{v}}(V_{\mathrm{v}}) = V_{\mathrm{v}}W_{\mathrm{v}}$ where x^{v} is an arbitrary fresh variable, and W_{v} an arbitrary value. The reduction $\mathsf{unfreeze}^{\mathrm{v}}(\mathsf{freeze}^{\mathrm{v}}(T_{\mathrm{v}})) \triangleright T_{\mathrm{v}}$ then becomes $(\lambda\_^{\mathrm{v}}.\, T_{\mathrm{v}})W_{\mathrm{v}} \triangleright T_{\mathrm{v}}$. In programming languages that have a unit type with a unique inhabitant $()^{\mathrm{v}}$, it is common to take $\mathsf{freeze}^{\mathrm{v}}(T_{\mathrm{v}}) = \lambda()^{\mathrm{v}}.\, T_{\mathrm{v}}$ and $\mathsf{unfreeze}^{\mathrm{v}}(V_{\mathrm{v}}) = V_{\mathrm{v}}()^{\mathrm{v}}$ which work exactly the same except with two additional advantages: There are no arbitrary choices for the variable x^{v} and the value W_{v}, and the fact that x^{v} is not free in T_{v} is

$$
\begin{array}{lll}
\text{Values:} & & \\
\quad V_{\mathrm{v}}, W_{\mathrm{v}} & ::= & \cdots \mid \mathsf{freeze}^{\mathrm{v}}\,(T_{\mathrm{v}}) \\
\text{Terms:} & & \\
\quad T_{\mathrm{v}}, U_{\mathrm{v}} & ::= & \cdots \mid \mathsf{unfreeze}^{\mathrm{v}}\,(V_{\mathrm{v}})
\end{array}
$$

(a) Syntax

$$\mathsf{unfreeze}^{\mathrm{v}}\,(\,\mathsf{freeze}^{\mathrm{v}}\,(T_{\mathrm{v}})) \quad \succ \quad T_{\mathrm{v}}$$

(b) Top-level reduction

$$\frac{\Gamma \vdash T_{\mathrm{v}} : A_{\mathrm{v}}}{\Gamma \vdash \mathsf{freeze}^{\mathrm{v}}\,(T_{\mathrm{v}}) : \Uparrow A_{\mathrm{v}}} \qquad \frac{\Gamma \vdash V_{\mathrm{v}} : \Uparrow A_{\mathrm{v}}}{\Gamma \vdash \mathsf{unfreeze}^{\mathrm{v}}\,(V_{\mathrm{v}}) : A_{\mathrm{v}}}$$

(c) Typing

Fig. 1.3. Call-by-value λ-calculus with upshift: $\lambda_{\mathrm{v}}^{\rightarrow\Uparrow}$

$$
\begin{array}{rcl}
\underline{\cdot}_{\mathrm{v}} : T_{\mathrm{n}} & \rightarrow & T_{\mathrm{v}} \\
\underline{x^{\mathrm{n}}}_{\mathrm{v}} & \overset{\mathrm{def}}{=} & \mathsf{unfreeze}^{\mathrm{v}}\,(x^{\mathrm{v}}) \\
\underline{\lambda x^{\mathrm{n}}.\, T_{\mathrm{n}}}_{\mathrm{v}} & \overset{\mathrm{def}}{=} & \lambda x^{\mathrm{v}}.\, \underline{T_{\mathrm{n}}}_{\mathrm{v}} \\
\underline{T_{\mathrm{n}} \mathbin{@^{\mathrm{n}}} V_{\mathrm{n}}}_{\mathrm{v}} & \overset{\mathrm{def}}{=} & \underline{T_{\mathrm{n}}}_{\mathrm{v}} \mathbin{@^{\mathrm{v}}} \mathsf{freeze}^{\mathrm{v}}\,(\underline{V_{\mathrm{n}}}_{\mathrm{v}}) \\
\underline{\mathsf{let}\ x^{\mathrm{n}} = V_{\mathrm{n}}\ \mathsf{in}\ U_{\mathrm{n}}}_{\mathrm{v}} & \overset{\mathrm{def}}{=} & \mathsf{let}\ x^{\mathrm{v}} = \mathsf{freeze}^{\mathrm{v}}\,(\underline{V_{\mathrm{n}}}_{\mathrm{v}})\ \mathsf{in}\ \underline{U_{\mathrm{n}}}_{\mathrm{v}}
\end{array}
$$

Fig. 1.4. Embedding of $\lambda_{\mathrm{n}}^{\rightarrow}$ into $\lambda_{\mathrm{v}}^{\rightarrow\Uparrow}$

easier to see. In terms of types, this means that we can encode $\Uparrow A$ as $\Uparrow A = \mathsf{unit} \rightarrow A$.

The embedding $\lambda_{\mathrm{n}}^{\rightarrow} \hookrightarrow \lambda_{\mathrm{v}}^{\rightarrow\Uparrow}$ is described in Figure 1.4. The idea is that we wrap every term T_{v} to make it a value $\mathsf{freeze}^{\mathrm{v}}\,(T_{\mathrm{v}})$ if it is meant to be substituted for a variable, and then use $\mathsf{unfreeze}^{\mathrm{v}}$ on variables to restart the computations after the substitution.

1.3.2 Embedding call-by-value in call-by-name. The extension of $\lambda_{\mathrm{n}}^{\rightarrow}$, called $\lambda_{\mathrm{v}}^{\rightarrow\Downarrow}$, is described in Figure 1.3. The idea is that in $\lambda_{\mathrm{n}}^{\rightarrow}$ there is no way of distinguishing a value $\lambda x^{\mathrm{n}}.\, T_{\mathrm{n}}$ from an arbitrary term U_{n} because two η-convertible terms can not be distinguished (internally) and $U_{\mathrm{n}} =_{\eta} \lambda x^{\mathrm{n}}.\, U_{\mathrm{n}} x^{\mathrm{n}}$. We therefore add a way to "mark" a term T_{n} by placing it under $\mathsf{box}^{\mathrm{n}}$: $\mathsf{box}^{\mathrm{n}}\,(T_{\mathrm{n}})$. We also add a match $\mathsf{match}\ T_{\mathrm{n}}\ \mathsf{with}\,[\,\mathsf{box}^{\mathrm{n}}\,(x^{\mathrm{n}}).\, U_{\mathrm{n}}]$ that forces the evaluation of T_{n} until it reaches a marked term $\mathsf{box}^{\mathrm{n}}\,(V_{\mathrm{n}})$. In a typed calculus, $\mathsf{box}^{\mathrm{n}}$ would be the constructor of a type $\Downarrow A$ called downshift, and $\mathsf{match}\ T_{\mathrm{n}}\ \mathsf{with}\,[\,\mathsf{box}^{\mathrm{n}}\,(x^{\mathrm{n}}).\, U_{\mathrm{n}}]$ its associated pattern-match, as shown in Figure 1.5.

Note that the pattern-match allows to define a destructor $\mathsf{unbox}^{\mathrm{n}}\,(T_{\mathrm{n}}) \overset{\mathrm{def}}{=} \mathsf{match}\ T_{\mathrm{n}}\ \mathsf{with}\,[\,\mathsf{box}^{\mathrm{n}}\,(x^{\mathrm{n}}).\, x^{\mathrm{n}}]$, with the expected induced reduction $\mathsf{unbox}^{\mathrm{n}}\,(\,\mathsf{box}^{\mathrm{n}}\,(T_{\mathrm{n}})) \triangleright T_{\mathrm{n}}$. The destructor, however, does not allow to define the pattern-match. Indeed, one could try to define the pattern-match $\mathsf{match}\ T_{\mathrm{n}}\ \mathsf{with}\,[\,\mathsf{box}^{\mathrm{n}}\,(x^{\mathrm{n}}).\, U_{\mathrm{n}}]$ as $\mathsf{let}\ x^{\mathrm{n}} = \mathsf{unbox}^{\mathrm{n}}\,(T_{\mathrm{n}})\ \mathsf{in}\ U_{\mathrm{n}}$ but since this is a call-by-name let-expression, it will immediately reduce to $U_{\mathrm{n}}[\mathsf{unbox}^{\mathrm{n}}\,(T_{\mathrm{n}})/x^{\mathrm{n}}]$ while the match would first reduce T_{n} until it reaches a $\mathsf{box}^{\mathrm{n}}$. Note however that in a call-by-value calculus, the pattern-match could be expressed using the destructor because $\mathsf{let}\ x^{\mathrm{v}} = \mathsf{unbox}^{\mathrm{v}}\,(T_{\mathrm{v}})\ \mathsf{in}\ U_{\mathrm{v}}$ would also start by reducing T_{v} as expected. In a way, the pattern-match is inherently call-by-value, which is why adding it to the call-by-name calculus will allow us to embed call-by-value in direct style.

$$
\begin{array}{lll}
\text{Terms / values:} & & \\
\quad T_{\mathrm{n}}, U_{\mathrm{n}}, V_{\mathrm{n}}, W_{\mathrm{n}} & ::= & \cdots \mid \mathsf{box}^{\mathrm{n}}\,(V_{\mathrm{n}}) \mid \mathsf{match}\ T_{\mathrm{n}}\ \mathsf{with}\,[\,\mathsf{box}^{\mathrm{n}}\,(x^{\mathrm{n}}).\, U_{\mathrm{n}}]
\end{array}
$$

(a) Syntax

$$\mathsf{match}\ \mathsf{box}^{\mathrm{n}}\,(V_{\mathrm{n}})\ \mathsf{with}\,[\,\mathsf{box}^{\mathrm{n}}\,(x^{\mathrm{n}}).\, U_{\mathrm{n}}] \quad \succ \quad U_{\mathrm{n}}[V_{\mathrm{n}}/x^{\mathrm{n}}]$$

(b) Top-level reduction

$$
\begin{array}{lll}
\text{Operational contexts:} & & \\
\quad \mathbb{O}_{\mathrm{v}} & ::= & \cdots \mid \mathsf{match}\ \mathbb{O}_{\mathrm{v}}\ \mathsf{with}\,[\,\mathsf{box}^{\mathrm{n}}\,(x^{\mathrm{n}}).\, U_{\mathrm{n}}]
\end{array}
$$

(c) Operational reduction

$$\frac{\Gamma \vdash T_{\mathrm{n}} : A_{\mathrm{n}}}{\Gamma \vdash \mathsf{box}^{\mathrm{n}}\,(T_{\mathrm{n}}) : \Downarrow A_{\mathrm{n}}} \qquad \frac{\Gamma \vdash T_{\mathrm{n}} : \Downarrow A_{\mathrm{n}} \qquad \Gamma, x^{\mathrm{n}} : A_{\mathrm{n}} \vdash U_{\mathrm{n}} : B_{\mathrm{n}}}{\Gamma \vdash \mathsf{match}\ T_{\mathrm{n}}\ \mathsf{with}\,[\,\mathsf{box}^{\mathrm{n}}\,(x^{\mathrm{n}}).\, U_{\mathrm{n}}] : B_{\mathrm{n}}}$$

(d) Typing

Fig. 1.5. Call-by-name λ-calculus with downshift: $\lambda_{\mathrm{n}}^{\rightarrow\Downarrow}$

This $\mathsf{box}^{\mathrm{n}}$ operator is not really common in programming languages but some other constructors are, including pairs. Let us imagine that we add pairs $(V_{\mathrm{n}} \otimes^{\mathrm{n}} W_{\mathrm{n}})$ of type $A_{\mathrm{n}} \otimes B_{\mathrm{n}}$ to the calculus, and the corresponding match $\mathsf{match}\ T_{\mathrm{n}}\ \mathsf{with}\,[(x^{\mathrm{n}} \otimes^{\mathrm{n}} y^{\mathrm{n}}).\, U_{\mathrm{n}}]$ with the reduction $\mathsf{match}\ (V_{\mathrm{n}} \otimes^{\mathrm{n}} W_{\mathrm{n}})\ \mathsf{with}\,[(x^{\mathrm{n}} \otimes^{\mathrm{n}} y^{\mathrm{n}}).\, U_{\mathrm{n}}] \triangleright U_{\mathrm{n}}[V_{\mathrm{n}}/x^{\mathrm{n}}, W_{\mathrm{n}}/y^{\mathrm{n}}]$. The constructor $\mathsf{box}^{\mathrm{n}}\,(T_{\mathrm{n}})$ can then be encoded as $(T_{\mathrm{n}} \otimes^{\mathrm{n}} V_{\mathrm{n}})$ where V_{n} is an arbitrary term, and the match $\mathsf{match}\ T_{\mathrm{n}}\ \mathsf{with}\,[\,\mathsf{box}^{\mathrm{n}}\,(x^{\mathrm{n}}).\, U_{\mathrm{n}}]$ by $\mathsf{match}\ T_{\mathrm{n}}\ \mathsf{with}\,[(x^{\mathrm{n}} \otimes^{\mathrm{n}} y^{\mathrm{n}}).\, U_{\mathrm{n}}]$ with y^{n} fresh. Just like when encoding $\mathsf{freeze}^{\mathrm{v}}\,(T_{\mathrm{v}})$ as $\lambda()^{\mathrm{v}}.\, T_{\mathrm{v}}$ instead of $\lambda x^{\mathrm{v}}.\, T_{\mathrm{v}}$, the intended behavior becomes more apparent by replacing unused variables and values by $()^{\mathrm{n}}$, so that $\mathsf{box}^{\mathrm{n}}\,(T_{\mathrm{n}})$ becomes $(T_{\mathrm{n}} \otimes^{\mathrm{n}} ()^{\mathrm{n}})$ and $\mathsf{match}\ T_{\mathrm{n}}\ \mathsf{with}\,[\,\mathsf{box}^{\mathrm{n}}\,(x^{\mathrm{n}}).\, U_{\mathrm{n}}]$ becomes $\mathsf{match}\ T_{\mathrm{n}}\ \mathsf{with}\,[(x^{\mathrm{n}} \otimes^{\mathrm{n}} ()^{\mathrm{n}}).\, U_{\mathrm{n}}]$. In a typed calculus, this would correspond to encoding $\Downarrow A_{\mathrm{n}}$ as $\Downarrow A_{\mathrm{n}} = A_{\mathrm{n}} \otimes \mathsf{unit}$.

The embedding $\lambda_{\mathrm{n}}^{\rightarrow} \hookrightarrow \lambda_{\mathrm{v}}^{\rightarrow\Uparrow}$ is described in Figure 1.6. The idea is to translate values as expected with the $\underset{\cdots}{\cdot}_{\mathrm{n}}$ part of the translation, and then use $\mathsf{box}^{\mathrm{n}}$ to mark values, i.e. we translate $\mathsf{val}^{\mathrm{v}}$ by $\mathsf{box}^{\mathrm{n}}$. We then extract the actual value when applying it or substituting it for a variable.

One way to think of this translation in the well-typed fragment is that $\mathsf{box}^{\mathrm{n}}$ and its pattern-match provide a runnable monad [5] as explained in [14, 15]. A computation of type A is represented as an element of $\mathrm{M}\,A = \Downarrow A$, and the monad M has an extra operation $\mathrm{run} : \mathrm{M}\,A \rightarrow A$ that runs the computation, in addition to the usual ones: $\mathrm{return} : A \rightarrow \mathrm{M}\,A$ and $\mathrm{bind} : \mathrm{M}\,A \rightarrow (A \rightarrow \mathrm{M}\,B) \rightarrow \mathrm{M}\,B$. Here, return is $\mathsf{box}^{\mathrm{n}}$, $\mathrm{bind}(T_{\mathrm{n}}, U_{\mathrm{n}})$ is $\mathsf{match}\ T_{\mathrm{n}}\ \mathsf{with}\,[\,\mathsf{box}^{\mathrm{n}}\,(x^{\mathrm{n}}).\, U_{\mathrm{n}} x^{\mathrm{n}}]$ and run is $\mathsf{unbox}^{\mathrm{n}}$. This translation is dual to the one done to encode CBN in CBV.

1.4 Pure polarized λ-calculus

1.4.1 Syntax.

$$
\begin{array}{rcl}
\underset{\cdots}{\cdot}_{\mathrm{n}} : V_{\mathrm{v}} & \to & T_{\mathrm{n}} \\
\underset{\cdots}{x^{\mathrm{v}}}_{\mathrm{n}} & \overset{\mathrm{def}}{=} & x^{\mathrm{n}} \\
\underset{\cdots\cdots}{\lambda x^{\mathrm{v}}.\,T_{\mathrm{v}}}_{\mathrm{n}} & \overset{\mathrm{def}}{=} & \lambda x^{\mathrm{n}}.\,\underline{T_{\mathrm{v}}}_{\mathrm{n}} \\
\\
\underline{\cdot}_{\mathrm{n}} : V_{\mathrm{v}} & \to & T_{\mathrm{n}} \\
\underline{\mathsf{val}^{\mathrm{v}}\,(V_{\mathrm{v}})}_{\mathrm{n}} & \overset{\mathrm{def}}{=} & \mathsf{box}^{\mathrm{n}}\,(\underset{\cdots}{V_{\mathrm{v}}}_{\mathrm{n}}) \\
\underline{T_{\mathrm{v}}\;@^{\mathrm{v}}\;V_{\mathrm{v}}}_{\mathrm{n}} & \overset{\mathrm{def}}{=} & \mathsf{unbox}^{\mathrm{n}}\,(\underline{T_{\mathrm{v}}}_{\mathrm{n}})\;@^{\mathrm{n}}\;\underset{\cdots}{V_{\mathrm{v}}}_{\mathrm{n}} \\
\underline{\mathsf{let}\;x^{\mathrm{v}} = T_{\mathrm{v}}\;\mathsf{in}\;U_{\mathrm{v}}}_{\mathrm{n}} & \overset{\mathrm{def}}{=} & \mathsf{match}\;\underline{T_{\mathrm{v}}}_{\mathrm{n}}\;\mathsf{with}\,[\,\mathsf{box}^{\mathrm{n}}\,(x^{\mathrm{n}}).\,\underline{U_{\mathrm{v}}}_{\mathrm{n}}\,]
\end{array}
$$

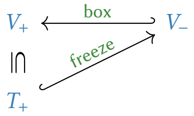
Fig. 1.6. Embedding of $\lambda_{\mathrm{v}}^{\to}$ into $\lambda_{\mathrm{n}}^{\to\Downarrow}$

We now introduce a pure polarized calculus $\lambda_{\mathrm{p}}^{\to\Uparrow\Downarrow}$ described in Figure 1.7. Just like call-by-name was annotated with n and call-by-value with v, we annotate most constructors by either p, if there is only one variant of this construction in the calculus, or $+$ and $-$ if there are two variants. When it does not lead to ambiguity, we will remove the p. In this calculus, there are 3 syntactical categories: positive values V_+, positive terms T_+, and negative values / terms T_-. Values are the terms that can be substituted for variables, so that a negative variable x^- can be substituted by any negative term T_- because the same term is also a value V_-, but a positive variable x^+ can only be substituted by a positive value V_+ (in this pure case, this means either another variable y^+ or $\mathsf{box}\,(V_-)$, but in general it can also include, for example, booleans true and false, and positive pairs $(V \otimes W)$). The distinction between the two polarities $+$ and $-$ is that the positive polarity $+$ represents call-by-value while a negative polarity $-$ represents call-by-name. The distinction is best seen on let-expressions: $\mathsf{let}^{\varepsilon}\;x^- = T_-\;\mathsf{in}\;U_{\varepsilon}$ will immediately substitute T_- for x^- (because any negative term T_- is also a negative value V_-), while $\mathsf{let}^{\varepsilon}\;x^+ = T_+\;\mathsf{in}\;U_{\varepsilon}$ will start by reducing T_+ to a value W_+ and then substitute that value for x^+:

$$
\begin{array}{lclcl}
\mathsf{let}^{\varepsilon}\;x^- = T_-\;\mathsf{in}\;U_{\varepsilon} & = & \mathsf{let}^{\varepsilon}\;x^- = V_-\;\mathsf{in}\;U_{\varepsilon} & \triangleright & U_{\varepsilon}[V_-/x^-] \\
\mathsf{let}^{\varepsilon}\;x^+ = T_+\;\mathsf{in}\;U_{\varepsilon} & \triangleright^* & \mathsf{let}^{\varepsilon}\;x^+ = W_+\;\mathsf{in}\;U_{\varepsilon} & \triangleright & U_{\varepsilon}[V_+/x^+]
\end{array}
$$

Note that the polarity ε_1 in $\mathsf{let}^{\varepsilon_1}\;x^{\varepsilon_2} = T_{\varepsilon_2}\;\mathsf{in}\;U_{\varepsilon_1}$ or $\mathsf{match}^{\varepsilon_1}\;T_+\;\mathsf{with}\,[\,\mathsf{box}\,(x^+).\,U_{\varepsilon_1}\,]$ is only here to remind us whether we are building a positive term U_+ (i.e. $\varepsilon_1 = +$) or negative term U_- (i.e. $\varepsilon_1 = -$). Since it does not matter for the reduction, and the grammar would still be unambiguous without it, it could be removed. We nevertheless keep it because it makes knowing if a term is positive or negative very easy, whereas without it, one may have to look deep into the term to know. For example $\mathsf{let}\;x^+ = V_+\;\mathsf{in}\;\mathsf{let}\;y^- = W_-\;\mathsf{in}\;T_{\varepsilon}$ is a term of polarity ε but one has to read the whole term before realizing it, whereas with our notation it is immediately clear that $\mathsf{let}^{\varepsilon}\;x^+ = V_+\;\mathsf{in}\;\mathsf{let}^{\varepsilon}\;y^- = W_-\;\mathsf{in}\;T_{\varepsilon}$ is a term of polarity ε. The polarity ε_2 on the variable x^{ε_2} however impacts the reduction as shown above.

1.4.2 Shifts. In order to go from one polarity to the other, one uses shifts, as described in Figure 1.8: $\mathsf{box}^{\mathrm{p}}\,(V_-)$ is a positive value, and $\mathsf{freeze}^{\mathrm{p}}\,(T_+)$ is a negative value. Both can be inverted: $\mathsf{unbox}^{\mathrm{p}}\,(\,\mathsf{box}^{\mathrm{p}}\,(V_-)\,) \triangleright V_-$ and $\mathsf{unfreeze}^{\mathrm{p}}\,(\,\mathsf{freeze}^{\mathrm{p}}\,(T_+)\,) \triangleright T_+$. Common names for box / unbox include wrap / unwrap [14] and thunk / force [10], and common names for freeze / $\mathsf{unfreeze}$ include delay / force [14] and return [10] (for freeze, and $\mathsf{unfreeze}$ is not present there). To remember which shift goes in which direction, one can notice that freeze goes from positive to negative, so that one can think of polarities as temperatures, and box goes the other way. The intuitions about $\mathsf{box}^{\mathrm{n}}$ and $\mathsf{freeze}^{\mathrm{v}}$ given in section 1.3 also apply to $\mathsf{box}^{\mathrm{p}}$ and $\mathsf{freeze}^{\mathrm{p}}$: We can think of $\mathsf{box}^{\mathrm{p}}$ as being a pattern-match-able constructor, of $\mathsf{freeze}^{\mathrm{p}}\,(T_+)$ as being $\lambda()^+.\,T_+$, and of $\mathsf{unfreeze}^{\mathrm{p}}\,(V_-)$ as being $V_-\,()^+$. Note however that functions $\lambda x^+.\,T_-$ have a negative body T_- so that $\mathsf{freeze}^{\mathrm{p}}$ is not expressible with functions (because we would need functions $\lambda x^+.\,T_+$ with a positive body T_+).



$$
\begin{tikzcd}
V_+ \arrow[d, phantom, sloped, "\subseteq"] & V_- \arrow[l, hook', "\mathsf{box}"'] \\
T_+ \arrow[ur, hook, "\mathsf{freeze}"] &
\end{tikzcd}
$$

Fig. 1.8. Shifts

In fact, functions with a positive body $\lambda x^+.\,T_+$ will be encoded as $\lambda x^+.\,\mathsf{freeze}^{\mathrm{p}}\,(T_+)$. More generally, we can encode functions $\lambda x^{\varepsilon_1}.\,T_{\varepsilon_2}$ that take an argument of arbitrary polarity ε_1, and returns a term of arbitrary polarity ε_2, and the corresponding application $T_-\;@^{\varepsilon_1,\varepsilon_2}\;V_{\varepsilon_1}$ so that $(\lambda x^{\varepsilon_1}.\,T_{\varepsilon_2})\;@^{\varepsilon_1,\varepsilon_2}\;V_{\varepsilon_1} \triangleright^+ T_{\varepsilon_2}[V_{\varepsilon_1}/x^{\varepsilon_1}]$. Some encodings are given in Figure 1.9. In the typed variant of the calculus, these encodings would correspond to using whatever shift is needed to make the domain positive and the codomain negative: $A_- \to B_-$ becomes $(\Downarrow A_-) \to B_-$, $A_+ \to B_+$ becomes $A_+ \to (\Uparrow B_+)$ and $A_- \to B_+$ becomes $(\Downarrow A_-) \to (\Uparrow B_+)$. We give two encodings for $\lambda x^-.\,T_+$ because we see no reason to prefer one over the other since the only difference is the order in which they remove the two shifts of $(\Downarrow A_-) \to (\Uparrow B_+)$. Those are not the only possible encodings, but are the simplest ones.

1.4.3 Embedding call-by-name and call-by-value. When trying to embed a calculus into a polarized calculus such as $\lambda_{\mathrm{p}}^{\to\Uparrow\Downarrow}$, the first choice that one has to make is the polarity of the translations of terms, values and variables. It is often a good idea to use the same polarity for variables and values, so that $T[V/x]$ can be translated to $T_{\varepsilon_1}[V_{\varepsilon_2}/x^{\varepsilon_2}]$. The polarities of terms and values however should be chosen to match the source calculus as closely as possible, without necessarily being the same (and indeed we will see in section 2.2 that in call-by-push-value, values are positive and terms are negative).

An embedding of $\lambda_{\mathrm{n}}^{\to}$ into $\lambda_{\mathrm{p}}^{\to\Uparrow\Downarrow}$ (or even into $\lambda_{\mathrm{p}}^{\to\Downarrow}$ since we use neither $\mathsf{freeze}^{\mathrm{p}}$ nor $\mathsf{unfreeze}^{\mathrm{p}}$) is described in Figure 1.10: Terms T_{n} are sent to negative terms T_-, with functions $\lambda x^{\mathrm{n}}.\,T_{\mathrm{n}}$ being sent to the encoding of $\lambda x^-.\,T_-$ described in Figure 1.9, and let-expressions $\mathsf{let}\;x^{\mathrm{n}} = T_{\mathrm{n}}\;\mathsf{in}\;U_{\mathrm{n}}$ being sent to $\mathsf{let}^-\;x^- = T_-\;\mathsf{in}\;U_-$. In terms of types this corresponds to call-by-name types being sent to negative types, with $\underline{A_{\mathrm{n}} \to B_{\mathrm{n}}}_{\mathrm{p}} \overset{\mathrm{def}}{=} (\Downarrow \underline{A_{\mathrm{n}}}_{\mathrm{p}}) \to \underline{B_{\mathrm{n}}}_{\mathrm{p}}$.

An embedding of $\lambda_{\mathrm{v}}^{\to}$ into $\lambda_{\mathrm{p}}^{\to\Uparrow\Downarrow}$ is described in Figure 1.11: Terms T_{v} are sent to positive terms T_+ and values V_{v} to positive values V_+, with functions $\lambda x^{\mathrm{v}}.\,T_{\mathrm{v}}$ being sent to the encoding of $\lambda x^+.\,T_+$ described in Figure 1.9 wrapped in $\mathsf{box}^{\mathrm{p}}$ to make them positive, and let-expressions $\mathsf{let}\;x^{\mathrm{v}} = T_{\mathrm{v}}\;\mathsf{in}\;U_{\mathrm{v}}$ being sent to $\mathsf{let}^+\;x^+ = T_+\;\mathsf{in}\;U_+$. In terms of types, this corresponds to call-by-value types being sent to positive types, with $\underline{A_{\mathrm{v}} \to B_{\mathrm{v}}}_{\mathrm{p}} \overset{\mathrm{def}}{=} \Downarrow(\underline{A_{\mathrm{v}}}_{\mathrm{p}} \to (\Uparrow \underline{B_{\mathrm{v}}}_{\mathrm{p}}))$.

Positive values:

$$\begin{array}{rcl} V_+, W_+ & ::= & x^+ \\ & \mid & \mathrm{box}^{p}\,(V_-) \end{array}$$

Negative values / terms:

$$\begin{array}{rcl} V_-, W_-, T_-, U_- & ::= & x^- \mid \mathrm{let}^-\, x^+ = T_+ \;\mathrm{in}\; U_- \mid \mathrm{let}^-\, x^- = V_- \;\mathrm{in}\; U_- \\ & \mid & \lambda x^+ .\, T_- \mid T_- \mathbin{@^{p}} V_+ \\ & \mid & \mathrm{freeze}^{p}\,(T_+) \\ & \mid & \mathrm{match}^-\, T_+ \;\mathrm{with}\,[\,\mathrm{box}^{p}\,(x^-).\, U_-] \end{array}$$

Positive terms:

$$\begin{array}{rcl} T_+, U_+ & ::= & \mathrm{val}^{p}\,(V_+) \mid \mathrm{let}^+\, x^+ = T_+ \;\mathrm{in}\; U_+ \mid \mathrm{let}^+\, x^- = V_- \;\mathrm{in}\; U_+ \\ & \mid & \mathrm{unfreeze}^{p}\,(T_-) \\ & \mid & \mathrm{match}^+\, T_+ \;\mathrm{with}\,[\,\mathrm{box}^{p}\,(x^-).\, U_+] \end{array}$$

Polarities:

$$\begin{array}{rcl} \varepsilon & ::= & + \mid - \end{array}$$

Notations:

$$\mathrm{unbox}^{p}\,(V_+) \overset{\mathrm{ntn}}{=} \mathrm{match}^-\, V_+ \;\mathrm{with}\,[\,\mathrm{box}^{p}\,(x^-).\, x^-]$$

(a) Syntax of the pure polarized λ-calculus $\lambda_{p}^{\to\Uparrow\Downarrow}$

$$\begin{array}{rcl} \mathrm{let}^{\varepsilon_1}\, x^{\varepsilon_2} = V_{\varepsilon_2} \;\mathrm{in}\; T_{\varepsilon_1} & \succ_{\mathrm{let}} & T_{\varepsilon_1}[V_{\varepsilon_2}/x^{\varepsilon_2}] \\ (\lambda x^+ .\, T_-)V_+ & \succ_{\to} & T_-[V_+/x^+] \\ \mathrm{unfreeze}^{p}\,(\,\mathrm{freeze}^{p}\,(T_+)) & \succ_{\Uparrow} & T_+ \\ \mathrm{match}^{\varepsilon}\,\mathrm{box}^{p}\,(V_-)\;\mathrm{with}\,[\,\mathrm{box}^{p}\,(x^-).\, T_\varepsilon] & \succ_{\Downarrow} & T_\varepsilon[V_-/x^-] \end{array}$$

(b) Top-level reduction

$$\mathrm{unbox}^{p}\,(\,\mathrm{box}^{p}\,(V_-)) \succ V_-$$

(c) Induced top-level reductions

Negative operational contexts:

$$\begin{array}{rcl} \mathbb{O}_- & ::= & \square_- \\ & \mid & \mathbb{O}_- \mathbin{@^{p}} V_+ \end{array}$$

Positive operational contexts:

$$\begin{array}{rcl} \mathbb{O}_+ & ::= & \square_+ \mid \mathrm{let}^{\varepsilon}\, x^+ = \mathbb{O}_+ \;\mathrm{in}\; U_\varepsilon \\ & \mid & \mathrm{unfreeze}^{p}\,(\mathbb{O}_-) \\ & \mid & \mathrm{match}^{\varepsilon}\,\mathbb{O}_+ \;\mathrm{with}\,[\,\mathrm{box}^{p}\,(x^-).\, U_\varepsilon] \end{array}$$

Notation:

$\mathbb{O}_{\varepsilon_1 \rightsquigarrow \varepsilon_2}$ for $\mathbb{O}_{\varepsilon_2}$ such that the hole it contains is $\square_{\varepsilon_1}$

$$\frac{T_{\varepsilon_1} \succ T'_{\varepsilon_1}}{\mathbb{O}_{\varepsilon_1 \rightsquigarrow \varepsilon_2}\boxed{T_{\varepsilon_1}} \rhd \mathbb{O}_{\varepsilon_1 \rightsquigarrow \varepsilon_2}\boxed{T'_{\varepsilon_1}}}$$

(d) Operational contexts and reduction

Fig. 1.7. Pure polarized λ-calculus $\lambda_{p}^{\to\Uparrow\Downarrow}$

$$\begin{array}{lcl} \lambda x^+ .\, T_- & \overset{\mathrm{ntn}}{=} & \lambda x^+ .\, T_- \\ \lambda x^- .\, T_- & \overset{\mathrm{ntn}}{=} & \lambda y^+ .\, \mathrm{match}^-\, y^+ \;\mathrm{with}\,[\,\mathrm{box}\,(x^-).\, T_-] \\ \lambda x^+ .\, T_+ & \overset{\mathrm{ntn}}{=} & \lambda x^+ .\, \mathrm{freeze}\,(T_+) \\ \lambda x^- .\, T_+ & \overset{\mathrm{ntn}}{=} & \lambda y^+ .\, \mathrm{match}^-\, y^+ \;\mathrm{with}\,[\,\mathrm{box}\,(x^-).\, \mathrm{freeze}\,(T_+)] \\ \lambda x^- .\, T_+ & \overset{\mathrm{ntn}}{=} & \lambda y^+ .\, \mathrm{freeze}\,(\,\mathrm{match}^+\, y^+ \;\mathrm{with}\,[\,\mathrm{box}\,(x^-).\, T_-]) \end{array}$$

$$\begin{array}{lcl} T_- \mathbin{@^{+,-}} V_+ & \overset{\mathrm{ntn}}{=} & T_- V_+ \\ T_- \mathbin{@^{-,-}} V_- & \overset{\mathrm{ntn}}{=} & T_-\, \mathrm{box}\,(V_-) \\ T_- \mathbin{@^{+,+}} V_+ & \overset{\mathrm{ntn}}{=} & \mathrm{unfreeze}\,(T_- V_+) \\ T_- \mathbin{@^{-,+}} V_- & \overset{\mathrm{ntn}}{=} & \mathrm{unfreeze}\,(T_-\, \mathrm{box}\,(V_-)) \\ T_- \mathbin{@^{-,+}} V_- & \overset{\mathrm{ntn}}{=} & \mathrm{unfreeze}\,(T_-\, \mathrm{box}\,(V_-)) \end{array}$$

Fig. 1.9. Encoding functions $\lambda x^{\varepsilon_1} .\, T_{\varepsilon_2}$ in $\lambda_{p}^{\to\Uparrow\Downarrow}$

$$\begin{array}{rcl} \underline{\cdot}_{p} : T_{n} & \to & T_- \\ \underline{x^{n}}_{p} & \overset{\mathrm{def}}{=} & x^- \\ \underline{\lambda x^{n} .\, T_{n}}_{p} & \overset{\mathrm{def}}{=} & \lambda y^+ .\, \mathrm{match}^-\, y^+ \;\mathrm{with}\,[\,\mathrm{box}^{p}\,(x^-).\, \underline{T_{n}}_{p}] \\ \underline{T_{n} \mathbin{@^{n}} V_{n}}_{p} & \overset{\mathrm{def}}{=} & \underline{T_{n}}_{p} \mathbin{@^{p}} \mathrm{box}^{p}\,(\underline{V_{n}}_{p}) \\ \underline{\mathrm{let}\; x^{n} = V_{n} \;\mathrm{in}\; U_{n}}_{p} & \overset{\mathrm{def}}{=} & \mathrm{let}^-\, x^- = \underline{V_{n}}_{p} \;\mathrm{in}\; \underline{U_{n}}_{p} \end{array}$$

Fig. 1.10. An embedding of $\lambda_{n}^{\to}$ into $\lambda_{p}^{\to\Uparrow\Downarrow}$

2 λ-CALCULI WITH DATATYPES

2.1 Polarized λ-calculus with datatypes

We now extend the syntax of $\lambda_{p}^{\to\Uparrow\Downarrow}$ which yields $\lambda_{p}^{\to\&\Uparrow\otimes\oplus\Downarrow}$ as described in Figure 2.1. The new supscripts are the names of the type constructors that correspond to the expressions we added to the calculus. We already had functions $\lambda x^+ .\, T_- : A_+ \to B_-$, upshifts $\mathrm{freeze}^{p}\,(T_+) : \Uparrow A_+$ and downshifts $\mathrm{box}^{p}\,(V_-) : \Downarrow A_-$. We now add positive / strict pairs $(V_+ \otimes^{p} W_+) : A_+ \otimes B_+$; sums $\iota_i{}^{p}\,(V_+) : A_+ \oplus B_+$; and negative / lazy pairs $(V_- \&^{p} W_-) : A_- \& B_-$.

$$\begin{array}{rcl} \underset{\cdots}{\cdot}_{p} : V_{v} & \to & V_+ \\ \underset{\cdots}{x^{v}}_{p} & \overset{\mathrm{def}}{=} & x^+ \\ \underset{\cdots}{\lambda x^{v} .\, T_{v}}_{p} & \overset{\mathrm{def}}{=} & \mathrm{box}^{p}\,(\lambda x^+ .\, \mathrm{freeze}^{p}\,(\underline{T_{v}}_{p})) \\ \\ \underline{\cdot}_{p} : T_{v} & \to & T_+ \\ \underline{\mathrm{val}^{v}\,(V_{v})}_{p} & \overset{\mathrm{def}}{=} & \underset{\cdots}{V_{v}}_{p} \\ \underline{T_{v} \mathbin{@^{v}} V_{v}}_{p} & \overset{\mathrm{def}}{=} & \mathrm{unfreeze}^{p}\,(\,\mathrm{unbox}^{p}\,(\underline{T_{v}}_{p}) \mathbin{@^{p}} \underset{\cdots}{V_{v}}_{p}) \\ \underline{\mathrm{let}\; x^{v} = T_{v} \;\mathrm{in}\; U_{v}}_{p} & \overset{\mathrm{def}}{=} & \mathrm{let}^-\, x^+ = \underline{T_{v}}_{p} \;\mathrm{in}\; \underline{U_{v}}_{p} \end{array}$$

Fig. 1.11. An embedding of $\lambda_{v}^{\to}$ into $\lambda_{p}^{\to\Uparrow\Downarrow}$

Negative term are lazy, i.e. they will evaluate only when they are used, while positive terms are eager and will evaluate as soon are they are built. This is the distinction between a positive pair $(V_+ \otimes W_+)$ and a negative pair $(V_- \& W_-)$: Both components of the pair $(V_+ \otimes W_+)$ are already evaluated at the construction of the pair, while the components of the pair $(V_- \& W_-)$ will only be evaluated if

Positive values:

$$\begin{array}{rcl} V_+, W_+ & ::= & x^+ \\ & \mid & (V_+ \otimes^{\mathrm{p}} W_+) \\ & \mid & \iota_1{}^{\mathrm{p}}(V_+) \mid \iota_2{}^{\mathrm{p}}(V_+) \\ & \mid & \mathsf{box}^{\mathrm{p}}(V_-) \end{array}$$

Negative values / terms:

$$\begin{array}{rcl} V_-, W_-, T_-, U_- & ::= & x^- \mid \mathsf{let}^- \, x^+ = T_+ \,\mathsf{in}\, U_- \mid \mathsf{let}^- \, x^- = T_- \,\mathsf{in}\, U_- \\ & \mid & \lambda x^+ . T_- \mid T_- \,@^{\mathrm{p}}\, V_+ \\ & \mid & (T_- \,\&^{\mathrm{p}}\, U_-) \mid \pi_1^{\mathrm{p}}(T_-) \mid \pi_2^{\mathrm{p}}(T_-) \\ & \mid & \mathsf{freeze}^{\mathrm{p}}(T_+) \\ & \mid & \mathsf{match}^- \, T_+ \,\mathsf{with}\, [(x^+ \otimes^{\mathrm{p}} y^+). U_-] \\ & \mid & \mathsf{match}^- \, T_+ \,\mathsf{with}\, [\, \iota_1{}^{\mathrm{p}}(x_1{}^+). U_-^1 \mid \iota_2{}^{\mathrm{p}}(x_2{}^+). U_-^2] \\ & \mid & \mathsf{match}^- \, T_+ \,\mathsf{with}\, [\, \mathsf{box}^{\mathrm{p}}(x^-). U_-] \end{array}$$

Positive terms:

$$\begin{array}{rcl} T_+, U_+ & ::= & V_+ \mid \mathsf{let}^+ \, x^+ = T_+ \,\mathsf{in}\, U_+ \mid \mathsf{let}^+ \, x^- = T_- \,\mathsf{in}\, U_+ \\ & \mid & \mathsf{unfreeze}^{\mathrm{p}}(T_-) \\ & \mid & \mathsf{match}^+ \, T_+ \,\mathsf{with}\, [(x^+ \otimes^{\mathrm{p}} y^+). U_+] \\ & \mid & \mathsf{match}^+ \, T_+ \,\mathsf{with}\, [\, \iota_1{}^{\mathrm{p}}(x_1{}^+). U_+^1 \mid \iota_2{}^{\mathrm{p}}(x_2{}^+). U_+^2] \\ & \mid & \mathsf{match}^+ \, T_+ \,\mathsf{with}\, [\, \mathsf{box}^{\mathrm{p}}(x^-). U_+] \end{array}$$

Polarities:

$$\begin{array}{rcl} \varepsilon & ::= & + \mid - \end{array}$$

Indices:

$$\begin{array}{rcl} i & ::= & 1 \mid 2 \end{array}$$

(a) Syntax

$$\begin{array}{rcl} \mathsf{match}^{\varepsilon}\, (V_+ \otimes^{\mathrm{p}} W_+) \,\mathsf{with}\, [(x^+ \otimes^{\mathrm{p}} y^+). M_\varepsilon] & \succ & M_\varepsilon[V_+/x^+, W_+/x^+] \\ \mathsf{match}^{\varepsilon}\, \iota_i{}^{\mathrm{p}}(V_+) \,\mathsf{with}\, [\, \iota_1{}^{\mathrm{p}}(x_1{}^+). U_\varepsilon^1 \mid \iota_2{}^{\mathrm{p}}(x_2{}^+). U_\varepsilon^2] & \succ & U_\varepsilon^i[V_+/x_i{}^+] \\ \mathsf{match}^{\varepsilon}\, \mathsf{box}^{\mathrm{p}}(V_-) \,\mathsf{with}\, [\, \mathsf{box}^{\mathrm{p}}(x^-). T_\varepsilon] & \succ & T_\varepsilon[V_-/x^-] \\ \mathsf{let}^{\varepsilon_1}\, x^{\varepsilon_2} = V_{\varepsilon_2} \,\mathsf{in}\, T_{\varepsilon_1} & \succ & T_{\varepsilon_1}[V_{\varepsilon_2}/x^{\varepsilon_2}] \\ (\lambda x^+ . T_-) V_+ & \succ & T_-[V_+/x^+] \\ \pi_i^{\mathrm{p}}((T_-^1 \,\&^{\mathrm{p}}\, T_-^2)) & \succ & T_-^i \\ \mathsf{unfreeze}^{\mathrm{p}}(\mathsf{freeze}^{\mathrm{p}}(T_+)) & \succ & T_+ \end{array}$$

(c) Top-level reduction

Negative operational contexts:

$$\begin{array}{rcl} \mathbb{O}_- & ::= & \square_- \\ & \mid & \mathbb{O}_- \,@^{\mathrm{p}}\, V_+ \\ & \mid & \pi_1^{\mathrm{p}}(\mathbb{O}_-) \mid \pi_2^{\mathrm{p}}(\mathbb{O}_-) \end{array}$$

Positive operational contexts:

$$\begin{array}{rcl} \mathbb{O}_+ & ::= & \square_+ \mid \mathsf{let}^{\varepsilon}\, x^+ = \mathbb{O}_+ \,\mathsf{in}\, U_\varepsilon \\ & \mid & \mathsf{unfreeze}^{\mathrm{p}}(\mathbb{O}_-) \\ & \mid & \mathsf{match}^{\varepsilon}\, \mathbb{O}_+ \,\mathsf{with}\, [(x^+ \otimes^{\mathrm{p}} y^+). U_\varepsilon] \\ & \mid & \mathsf{match}^{\varepsilon}\, \mathbb{O}_+ \,\mathsf{with}\, [\, \mathsf{box}^{\mathrm{p}}(x^-). U_\varepsilon] \\ & \mid & \mathsf{match}^{\varepsilon}\, \mathbb{O}_+ \,\mathsf{with}\, [\, \iota_1{}^{\mathrm{p}}(x_1{}^+). U_\varepsilon^1 \mid \iota_2{}^{\mathrm{p}}(x_2{}^+). U_\varepsilon^2] \end{array}$$

Notation:

$\mathbb{O}_{\varepsilon_1 \rightsquigarrow \varepsilon_2}$ for $\mathbb{O}_{\varepsilon_2}$ such that the hole it contains is $\square_{\varepsilon_1}$

$$\frac{T_{\varepsilon_1} \succ T'_{\varepsilon_1}}{\mathbb{O}_{\varepsilon_1 \rightsquigarrow \varepsilon_2}\boxed{T_{\varepsilon_1}} \triangleright \mathbb{O}_{\varepsilon_1 \rightsquigarrow \varepsilon_2}\boxed{T_{\varepsilon_1}}}$$

(d) Operational contexts and reduction

Positive types:

$$\begin{array}{rcl} A_+, B_+ & ::= & A_+ \otimes B_+ \mid A_+ \oplus B_+ \mid \Downarrow A_- \end{array}$$

Positive types:

$$\begin{array}{rcl} A_-, B_- & ::= & A_+ \to B_- \mid A_- \,\&\, B_- \mid \Uparrow A_+ \end{array}$$

(b) Types of the polarized λ-calculus $\lambda_{\mathrm{p}}^{\to\&\Uparrow\otimes\oplus\Downarrow}$

Fig. 2.1. Polarized λ-calculus $\lambda_{\mathrm{p}}^{\to\&\Uparrow\otimes\oplus\Downarrow}$

a projection applied to it $\pi_i((V_- \,\&\, W_-))$ is evaluated. It is common to allow positive constructors to take terms as arguments instead of values, for example allowing $(T_+ \otimes U_+)$. This however means that one has to add many more operational contexts, and pick some arbitrary evaluation order (left-to-right or right-to-left). Instead, we prefer to not allow $(T_+ \otimes U_+)$ in the formal syntax and see it as a notation for $\mathsf{let}^+ \, x^+ = T_+ \,\mathsf{in}\, \mathsf{let}^+ \, y^+ = U_+ \,\mathsf{in}\, (x^+ \otimes y^+)$ (for the left-to-right variant), or $\mathsf{let}^+ \, y^+ = U_+ \,\mathsf{in}\, \mathsf{let}^+ \, x^+ = T_+ \,\mathsf{in}\, (x^+ \otimes y^+)$ (for the right-to-left variant).

Of course, we have ways of delaying or forcing evaluation: shifts. Using them, we could encode each pair using the other one as seen in Figure 2.2. This encoding is valid when one only considers evaluation, but not when one considers η-conversion.

2.2 Call-by-push-value

Call-by-push-value (CBPV) [11] is a well-known calculus that subsumes both call-by-name and call-by-value. In this section, we describe its relation to $\lambda_{\mathrm{p}}^{\to\&\Uparrow\otimes\oplus\Downarrow}$.

In Figure 2.3, we recall the syntax of $\lambda_{\mathrm{p}}^{\to\&\Uparrow\otimes\oplus\Downarrow}$ (which was given in Figure 2.1) on the left, and of CBPV (figure 2 of [11]) on the right (ignoring complex values for now). Terms and values that correspond to each other are placed on the same line, and differences are highlighted. There are a few minor differences when compared with figure 2 of [11]: We only have binary sum and negative pairs, we write $(V_{\mathrm{pv}}, W_{\mathrm{pv}})^{\mathrm{pv}}$ for a pair instead of $\langle V, W \rangle$, and we add pv everywhere. Through the translation described in Figure 2.4, values of CBPV V_{pv} correspond to positive values V_+, and terms of CBPV T_{pv} correspond to negative terms. For shifts, $\mathsf{thunk}^{\mathrm{pv}}(T_{\mathrm{pv}})$ corresponds to $\mathsf{box}^{\mathrm{p}}(T_-)$ (and its inverse $\mathsf{force}^{\mathrm{pv}}(V_{\mathrm{pv}})$ to $\mathsf{unfreeze}^{\mathrm{p}}(V_+) \overset{\mathrm{ntn}}{=} \mathsf{match}^- \, V_+ \,\mathsf{with}\, [\, \mathsf{box}^{\mathrm{p}}(x^-). x^-]$), and $\mathsf{return}^{\mathrm{pv}}(V_{\mathrm{pv}})$ corresponds to

$$\begin{array}{rcl}
A_+ \otimes B_+ & \rightsquigarrow & \Downarrow((\Uparrow A_+) \,\&\, (\Uparrow B_+)) \\
(V_+ \otimes W_+) & \rightsquigarrow & \mathrm{box}\,((\mathrm{freeze}\,(V_+) \,\&\, \mathrm{freeze}\,(W_+))) \\
\mathrm{match}\ T_+\ \mathrm{with}\,[(x^+ \otimes y^+).\,U_\varepsilon] & \rightsquigarrow & \mathrm{match}\ T_+\ \mathrm{with}\,[\,\mathrm{box}\,(z^-).\ \mathrm{let}\ x^+ = \mathrm{unfreeze}\,(\pi_1\,(z^-))\ \mathrm{in}\ \mathrm{let}\ y^+ = \mathrm{unfreeze}\,(\pi_2\,(z^-))\ \mathrm{in}\ U_\varepsilon] \\
\\
A_- \,\&\, B_- & \rightsquigarrow & \Uparrow((\Downarrow A_-) \otimes (\Downarrow B_-)) \\
(V_- \,\&\, W_-) & \rightsquigarrow & \mathrm{freeze}\,((\mathrm{box}\,(V_-) \otimes \mathrm{box}\,(W_-))) \\
\pi_i\,(V_-) & \rightsquigarrow & \mathrm{match}\ \mathrm{unfreeze}\,(V_-)\ \mathrm{with}\,[(x_1{}^+ \otimes x_2{}^+).\ \mathrm{unbox}\,(x_i{}^+)]
\end{array}$$

Fig. 2.2. Mutual expressiveness of positive and negative pairs

$\mathrm{freeze}^{\mathrm{p}}\,(T_+)$. The "inverse" of $T_{\mathrm{pv}}\ \mathrm{to}\ x^{\mathrm{pv}}.\,U_{\mathrm{pv}}$ of $\mathrm{return}^{\mathrm{pv}}\,(V_{\mathrm{pv}})$ corresponds to $\mathrm{let}^-\ x^+ = \mathrm{unfreeze}^{\mathrm{p}}\,(T_-)\ \mathrm{in}\ U_-$.

The main difference between the two calculi is that $\lambda_{\mathrm{p}}^{\rightarrow\&\Uparrow\otimes\oplus\Downarrow}$ has positive terms while CBPV does not. The fact that one could want to add more "values" to CBPV is acknowledged in [11], and leads to the introduction of complex values (figure 12 of [11]) which can be used anywhere a value could be used. Complex values are values that can be built using let-expressions and pattern-matches on other values. Examples include the first projection of a value, $\mathrm{pm}\ x^{\mathrm{pv}}\ \mathrm{as}\,[(y^{\mathrm{pv}}, z^{\mathrm{pv}})^{\mathrm{pv}}.\ y^{\mathrm{pv}}]$, the result of swapping both components of a pair $\mathrm{pm}\ x^{\mathrm{pv}}\ \mathrm{as}\,[(y^{\mathrm{pv}}, z^{\mathrm{pv}})^{\mathrm{pv}}.\ (z^{\mathrm{pv}}, y^{\mathrm{pv}})^{\mathrm{pv}}]$. We give a syntax for a subset of complex values in Figure 2.3, and one can see that they correspond to a subset of positive terms. Complex values in [11] also allow let-expressions and pattern-matches deep in the value, for example $(x^{\mathrm{pv}}, \mathrm{let}\ V_{\mathrm{pv}}\ \mathrm{be}\ y^{\mathrm{pv}}.\ W_{\mathrm{pv}})^{\mathrm{pv}}$. Here, to make the resemblance with our positive terms more striking, we prefer to disallow this (which is why our syntax does not cover all complex values) and think of $(x^{\mathrm{pv}}, \mathrm{let}\ V_{\mathrm{pv}}\ \mathrm{be}\ y^{\mathrm{pv}}.\ W_{\mathrm{pv}})^{\mathrm{pv}}$ as being a notation for $\mathrm{let}\ V_{\mathrm{pv}}\ \mathrm{be}\ y^{\mathrm{pv}}.\ (x^{\mathrm{pv}}, W_{\mathrm{pv}})^{\mathrm{pv}}$, just like $(x^+ \otimes^{\mathrm{p}} \mathrm{let}^+\ y^+ = V_+\ \mathrm{in}\ W_+)$ is a notation for $\mathrm{let}^+\ y^+ = V_+\ \mathrm{in}\ (x^+ \otimes^{\mathrm{p}} y^+)$.

Adding complex values has no effect on what computations can be expressed, which is stated in proposition 14 of [11], and proven using a translation from CBPV with complex values to CBPV without complex values described in figure 13 of [11]. This translation sends computations to computations, and complex values to computations that reduce to $\mathrm{return}^{\mathrm{pv}}\,(V_{\mathrm{pv}})$. In our calculus, this corresponds to sending negative terms to negative terms, and positive terms to negative terms that reduce to $\mathrm{freeze}^{\mathrm{p}}\,(V_+)$ as follows: x^+ is sent to $\mathrm{freeze}^{\mathrm{p}}\,(x^+)$, $\mathrm{let}^+\ x^+ = T_+\ \mathrm{in}\ U_+$ to $\mathrm{let}^-\ x^+ = \mathrm{unfreeze}^{\mathrm{p}}\,(\underline{T_+})\ \mathrm{in}\ \underline{U_+}$, $\mathrm{match}^+\ T_+\ \mathrm{with}\,[(x^+ \otimes^{\mathrm{p}} y^+).\ U_+]$ to $\mathrm{match}^-\ \mathrm{unfreeze}^{\mathrm{p}}\,(\underline{T_+})\ \mathrm{with}\,[(x^+ \otimes^{\mathrm{p}} y^+).\ \underline{U_+}]$, and $\mathrm{unfreeze}^{\mathrm{p}}\,(T_-)$ to $\mathrm{let}^-\ x^+ = \mathrm{unfreeze}^{\mathrm{p}}\,(\underline{T_-})\ \mathrm{in}\ \mathrm{freeze}^{\mathrm{p}}\,(x^+)$. Note that in a well-typed, strongly-normalizing, effect-free[2], and closed setting, complex values reduce to (non-complex) values, and justifying that they have no effect on the expressiveness of the calculus is therefore much easier.

Since we can completely remove positive terms, the reader may wonder why we have them in the first place. There are two reasons. First, just like for complex values, they correspond to terms we would like to write, and being able to write them directly is more satisfying than having to encode them. Secondly, it allows to have $\mathrm{unfreeze}^{\mathrm{p}}\,(T_-)$ instead of $T_{\mathrm{pv}}\ \mathrm{to}\ x^{\mathrm{pv}}.\,U_{\mathrm{pv}}$, which we believe to be slightly more primitive, and makes the corresponding $\mathrm{L}_{\mathrm{p}}^{\rightarrow\&\Uparrow\otimes\oplus\Downarrow}$ calculus (that we will introduce in Section 3) perfectly symmetric.

The last remaining difference between the two calculi is that CBPV has no negative variables. This is a minor difference and there is a translation $\underline{\cdot}$ from $\lambda_{\mathrm{p}}^{\rightarrow\&\Uparrow\otimes\oplus\Downarrow}$ to itself without negative variables that sends x^- to $\mathrm{unbox}^{\mathrm{p}}\,(x^+)$, $\mathrm{let}^\varepsilon\ x^- = V_-\ \mathrm{in}\ U_\varepsilon$ to $\mathrm{let}^\varepsilon\ x^+ = \mathrm{box}^{\mathrm{p}}\,(V_-)\ \mathrm{in}\ \underline{U_\varepsilon}$ and $\mathrm{match}^\varepsilon\ T_+\ \mathrm{with}\,[\,\mathrm{box}^{\mathrm{p}}\,(x^-).\ U_\varepsilon]$ to $\mathrm{let}^\varepsilon\ y^+ = T_+\ \mathrm{in}\ \underline{U_\varepsilon}$. Similarly, one could introduce computation variables X^{pv} in CBPV, encode them as $\mathrm{force}^{\mathrm{pv}}\,(x^{\mathrm{pv}})$, and their associated let-expressions $\mathrm{let}\ T_{\mathrm{pv}}\ \mathrm{be}\ X^{\mathrm{pv}}.\ U_{\mathrm{pv}}$ as $\mathrm{let}\ \mathrm{thunk}^{\mathrm{pv}}\,(T_{\mathrm{pv}})\ \mathrm{be}\ x^{\mathrm{pv}}.\ U_{\mathrm{pv}}$.

3 ABSTRACT MACHINE CALCULI

3.1 Abstract machines

Calculi presented via a natural-deduction syntax and whose reductions are defined through operational contexts tend to hide some parts of the evaluation of real-word programming languages. Two examples are the search for the position (in the term representing the program) of the next redex to reduce according to the operational reduction, and the propagation of substitutions. Abstract machines more closely model how those are done in real-world programming languages: An abstract machine will typically "remember" where it is in the term, and "move" towards the next redex, and some abstract machines have environments and closures instead of substitutions.

In this article, we will only introduce abstract machines of the first kind. The remainder of this section takes place in the call-by-name λ-calculus $\lambda_{\mathrm{n}}^{\rightarrow}$, and we will drop the n sup/subscripts. Note that after the reduction $\mathbb{O}^1\boxed{\mathbb{O}^2\boxed{(\lambda x.\,T)V}} \triangleright \mathbb{O}^1\boxed{\mathbb{O}^2\boxed{T[V/x]}}$ (where $\mathbb{O}_2 \neq \square$), the next reduction step can not involve $\mathbb{O}^1$, so that starting to search for the next redex from the top of the term would be inefficient. A concrete example is $(((II)V^1)\ldots)V^k$ where $I = \lambda x.\,x$. Using the definition of the head reduction of Figure 1.1, we see that the only way to infer that $(((II)V^1)\ldots)V^k$ is reducible is to first infer that $(((II)V^1)\ldots)V^{k-1}$ is and so on until we get to II which indeed is reducible. It therefore takes a linear (in k) amount of time to infer that $(((II)V^1)\ldots)V^k \triangleright ((IV^1)\ldots)V^k$. We then have to start over: To infer that $((IV^1)\ldots)V^k$ is reducible, we need to infer that $((IV^1)\ldots)V^{k-1}$ is and so on until we get to IV^1. It again takes a linear amount of time to infer that $((IV^1)\ldots)V^k \triangleright (V^1\ldots)V^k$: Starting to look for the next redex to reduce from the top of the term at each step is inefficient. In order to make this more efficient, one can remember which term one was looking at by writing $\mathbb{O}\boxed{\underline{T}}$ for the term $\mathbb{O}\boxed{T}$ where the machine is currently looking at the subterm T. When encountering an application, the machine moves to the left

---

[2] Effects are consequences of evaluating a term other than the result, for example printing or storing a value in a mutable variable.

Positive values:

$$\begin{array}{rcl} V_+, W_+ & ::= & x^+ \\ & \mid & (V_+ \otimes^{\mathrm{p}} W_+) \\ & \mid & \iota_1{}^{\mathrm{p}}\,(V_+) \mid \iota_2{}^{\mathrm{p}}\,(V_+) \\ & \mid & \mathrm{box}^{\mathrm{p}}\,(V_-) \end{array}$$

Negative values / terms:

$$\begin{array}{rcl} V_-, W_-, T_-, U_- & ::= & x^- \mid \mathrm{let}^-\, x^+ = T_+ \;\mathrm{in}\; U_- \mid \mathrm{let}^-\, x^- = T_- \;\mathrm{in}\; U_- \\ & \mid & \lambda x^+ .\, T_- \mid T_- \mathbin{@^{\mathrm{p}}} V_+ \\ & \mid & (T_- \mathbin{\&^{\mathrm{p}}} U_-) \mid \pi_1^{\mathrm{p}}\,(T_-) \mid \pi_2^{\mathrm{p}}\,(T_-) \\ & \mid & \mathrm{freeze}^{\mathrm{p}}\,(T_+) \\ & \mid & \mathrm{match}^-\, T_+ \;\mathrm{with}\; [(x^+ \otimes^{\mathrm{p}} y^+).\, U_-] \\ & \mid & \mathrm{match}^-\, T_+ \;\mathrm{with}\; [\,\iota_1{}^{\mathrm{p}}\,(x_1{}^+).\, U_-^1 \mid \iota_2{}^{\mathrm{p}}\,(x_2{}^+).\, U_-^2] \\ & \mid & \mathrm{match}^-\, T_+ \;\mathrm{with}\; [\,\mathrm{box}^{\mathrm{p}}\,(x^-).\, U_-] \end{array}$$

Positive terms:

$$\begin{array}{rcl} T_+, U_+ & ::= & \mathrm{val}^{\mathrm{p}}\,(V_+) \mid \mathrm{let}^+\, x^+ = T_+ \;\mathrm{in}\; U_+ \mid \mathrm{let}^+\, x^- = T_- \;\mathrm{in}\; U_+ \\ & \mid & \mathrm{unfreeze}^{\mathrm{p}}\,(T_-) \\ & \mid & \mathrm{match}^+\, T_+ \;\mathrm{with}\; \left[(x^+ \otimes^{\mathrm{p}} y^+).\, U_+\right] \\ & \mid & \mathrm{match}^+\, T_+ \;\mathrm{with}\; \left[\,\iota_1{}^{\mathrm{p}}\,(x_1{}^+).\, U_+^1 \mid \iota_2{}^{\mathrm{p}}\,(x_2{}^+).\, U_+^2\right] \\ & \mid & \mathrm{match}^+\, T_+ \;\mathrm{with}\; [\,\mathrm{box}^{\mathrm{p}}\,(x^-).\, U_+] \end{array}$$

Values:

$$\begin{array}{rcl} V_{\mathrm{pv}}, W_{\mathrm{pv}} & ::= & x^{\mathrm{pv}} \\ & \mid & (V_{\mathrm{pv}}, W_{\mathrm{pv}})^{\mathrm{pv}} \\ & \mid & (1, V_{\mathrm{pv}})^{\mathrm{pv}} \mid (2, V_{\mathrm{pv}})^{\mathrm{pv}} \\ & \mid & \mathrm{thunk}^{\mathrm{pv}}\,(T_{\mathrm{pv}}) \end{array}$$

Terms / computations:

$$\begin{array}{rcl} T_{\mathrm{pv}}, U_{\mathrm{pv}} & ::= & \mathrm{let}\; V_{\mathrm{pv}} \;\mathrm{be}\; x^{\mathrm{pv}}.\, T_{\mathrm{pv}} \\ & \mid & \lambda x^{\mathrm{pv}}.\, T_{\mathrm{pv}} \mid V_{\mathrm{pv}}\,`T_{\mathrm{pv}} \\ & \mid & \lambda^{\mathrm{pv}}\left[1.T_{\mathrm{pv}}^1 \mid 2.T_{\mathrm{pv}}^2\right] \mid 1`T_{\mathrm{pv}} \mid 2`T_{\mathrm{pv}} \\ & \mid & \mathrm{return}^{\mathrm{pv}}\,(V_{\mathrm{pv}}) \mid T_{\mathrm{pv}} \;\mathrm{to}\; x^{\mathrm{pv}}.\, U_{\mathrm{pv}} \\ & \mid & \mathrm{pm}\; V_{\mathrm{pv}} \;\mathrm{as}\; \left[(x^{\mathrm{pv}}, y^{\mathrm{pv}})^{\mathrm{pv}}.\, T_{\mathrm{pv}}\right] \\ & \mid & \mathrm{pm}\; V_{\mathrm{pv}} \;\mathrm{as}\; \left[(1, x_1{}^{\mathrm{pv}})^{\mathrm{pv}}.\, T_{\mathrm{pv}}^1 \mid (2, x_2{}^{\mathrm{pv}})^{\mathrm{pv}}.\, T_{\mathrm{pv}}^2\right] \\ & \mid & \mathrm{force}^{\mathrm{pv}}\,(V_{\mathrm{pv}}) \end{array}$$

Chosen complex values:

$$\begin{array}{rcl} V_{\mathrm{pv}}^c, W_{\mathrm{pv}}^c & ::= & V_{\mathrm{pv}} \mid \mathrm{let}\; V_{\mathrm{pv}}^c \;\mathrm{be}\; x^{\mathrm{pv}}.\, W_{\mathrm{pv}}^c \\ & & \\ & \mid & \mathrm{pm}\; V_{\mathrm{pv}}^c \;\mathrm{as}\; \left[(x^{\mathrm{pv}}, y^{\mathrm{pv}})^{\mathrm{pv}}.\, W_{\mathrm{pv}}^c\right] \\ & \mid & \mathrm{pm}\; V_{\mathrm{pv}}^c \;\mathrm{as}\; \left[(1, x_1{}^{\mathrm{pv}})^{\mathrm{pv}}.\, W_{\mathrm{pv}}^{c,1} \mid (2, x_2{}^{\mathrm{pv}})^{\mathrm{pv}}.\, W_{\mathrm{pv}}^{c,2}\right] \end{array}$$

Fig. 2.3. Correspondence between the polarized λ-calculus $\lambda_{\mathrm{p}}^{\to\&\Uparrow\otimes\oplus\Downarrow}$ (left) and CBPV$^{\to\&\Uparrow\otimes\oplus\Downarrow}$ (right, from figures 2 and 12 of [11])

$$\begin{array}{rcl} \underline{\cdot}_{\mathrm{p}} : V_{\mathrm{pv}} & \to & V_+ \\ \underline{x^{\mathrm{pv}}}_{\mathrm{p}} & \stackrel{\mathrm{def}}{=} & x^+ \\ \underline{(V_{\mathrm{pv}}, W_{\mathrm{pv}})^{\mathrm{pv}}}_{\mathrm{p}} & \stackrel{\mathrm{def}}{=} & (\underline{V_{\mathrm{pv}}}_{\mathrm{p}} \otimes^{\mathrm{p}} \underline{W_{\mathrm{pv}}}_{\mathrm{p}}) \\ \underline{(i, V_{\mathrm{pv}})^{\mathrm{pv}}}_{\mathrm{p}} & \stackrel{\mathrm{def}}{=} & \iota_i{}^{\mathrm{p}}\,(\underline{V_{\mathrm{pv}}}_{\mathrm{p}}) \\ \underline{\mathrm{thunk}^{\mathrm{pv}}\,(T_{\mathrm{pv}})}_{\mathrm{p}} & \stackrel{\mathrm{def}}{=} & \mathrm{box}^{\mathrm{p}}\,(\underline{T_{\mathrm{pv}}}_{\mathrm{p}}) \\ & & \\ \underline{\cdot}_{\mathrm{p}} : T_{\mathrm{pv}} & \to & T_- \\ \underline{\mathrm{pm}\; V_{\mathrm{pv}} \;\mathrm{as}\; \left[(x^{\mathrm{pv}}, y^{\mathrm{pv}})^{\mathrm{pv}}.\, T_{\mathrm{pv}}\right]}_{\mathrm{p}} & \stackrel{\mathrm{def}}{=} & \mathrm{match}^-\, \underline{V_{\mathrm{pv}}}_{\mathrm{p}} \;\mathrm{with}\; \left[(x^+ \otimes^{\mathrm{p}} y^+).\, \underline{T_{\mathrm{pv}}}_{\mathrm{p}}\right] \\ \underline{\mathrm{pm}\; V_{\mathrm{pv}} \;\mathrm{as}\; \left[(1, x_1{}^{\mathrm{pv}})^{\mathrm{pv}}.\, T_{\mathrm{pv}}^1 \mid (2, x_2{}^{\mathrm{pv}})^{\mathrm{pv}}.\, T_{\mathrm{pv}}^2\right]}_{\mathrm{p}} & \stackrel{\mathrm{def}}{=} & \mathrm{match}^-\, \underline{V_{\mathrm{pv}}}_{\mathrm{p}} \;\mathrm{with}\; \left[\,\iota_1{}^{\mathrm{p}}\,(x_1{}^+).\, \underline{T_{\mathrm{pv}}^1}_{\mathrm{p}} \mid \iota_2{}^{\mathrm{p}}\,(x_2{}^+).\, \underline{T_{\mathrm{pv}}^2}_{\mathrm{p}}\right] \\ \underline{\mathrm{force}^{\mathrm{pv}}\,(V_{\mathrm{pv}})}_{\mathrm{p}} & \stackrel{\mathrm{def}}{=} & \mathrm{unbox}^{\mathrm{p}}\,(\underset{\cdots}{V_{\mathrm{pv}}}_{\mathrm{p}}) \\ \underline{\mathrm{let}\; V_{\mathrm{pv}} \;\mathrm{be}\; x^{\mathrm{pv}}.\, T_{\mathrm{pv}}}_{\mathrm{p}} & \stackrel{\mathrm{def}}{=} & \mathrm{let}^-\, x^+ = \underset{\cdots}{V_{\mathrm{pv}}}_{\mathrm{p}} \;\mathrm{in}\; \underline{T_{\mathrm{pv}}}_{\mathrm{p}} \\ \underline{\lambda x^{\mathrm{pv}}.\, T_{\mathrm{pv}}}_{\mathrm{p}} & \stackrel{\mathrm{def}}{=} & \lambda x^+ .\, \underline{T_{\mathrm{pv}}}_{\mathrm{p}} \\ \underline{V_{\mathrm{pv}}\,`T_{\mathrm{pv}}}_{\mathrm{p}} & \stackrel{\mathrm{def}}{=} & \underline{T_{\mathrm{pv}}}_{\mathrm{p}} \mathbin{@^{\mathrm{p}}} \underset{\cdots}{V_{\mathrm{pv}}}_{\mathrm{p}} \\ \underline{\lambda^{\mathrm{pv}}\left[1.T_{\mathrm{pv}}^1 \mid 2.T_{\mathrm{pv}}^2\right]}_{\mathrm{p}} & \stackrel{\mathrm{def}}{=} & (\underline{T_{\mathrm{pv}}^1}_{\mathrm{p}} \mathbin{\&^{\mathrm{p}}} \underline{T_{\mathrm{pv}}^2}_{\mathrm{p}}) \\ \underline{i`T_{\mathrm{pv}}}_{\mathrm{p}} & \stackrel{\mathrm{def}}{=} & \pi_i^{\mathrm{p}}\,(\underline{T_{\mathrm{pv}}}_{\mathrm{p}}) \\ \underline{\mathrm{return}^{\mathrm{pv}}\,(V_{\mathrm{pv}})}_{\mathrm{p}} & \stackrel{\mathrm{def}}{=} & \mathrm{freeze}^{\mathrm{p}}\,(\underset{\cdots}{V_{\mathrm{pv}}}_{\mathrm{p}}) \\ \underline{T_{\mathrm{pv}} \;\mathrm{to}\; x^{\mathrm{pv}}.\, U_{\mathrm{pv}}}_{\mathrm{p}} & \stackrel{\mathrm{def}}{=} & \mathrm{let}^-\, x^+ = \mathrm{unfreeze}^{\mathrm{p}}\,(\underline{T_{\mathrm{pv}}}_{\mathrm{p}}) \;\mathrm{in}\; \underline{U_{\mathrm{pv}}}_{\mathrm{p}} \end{array}$$

Fig. 2.4. Translation from CBPV$^{\to\&\Uparrow\otimes\oplus\Downarrow}$ to $\lambda_{\mathrm{p}}^{\to\&\Uparrow\otimes\oplus\Downarrow}$

part of the application $\mathbb{O}\boxed{\underbrace{TV}}$ $\rhd_{\mathrm{m}}$ $\mathbb{O}\boxed{\underbrace{T}\,V}$, and when it finally reaches a λ-abstraction, it reduces $\mathbb{O}\boxed{\underbrace{(\lambda x.\,T)}\,V}$ $\rhd_{\mathrm{r}}$ $\mathbb{O}\boxed{\underbrace{T[V/x]}}$, and then keeps going down (if T is an application) or reducing and going up (if T is a λ-abstraction). Note that the "move" reductions $\rhd_{\mathrm{m}}$ are invisible in the original calculus, while the "reduce" reduction $\rhd_{\mathrm{r}}$ correspond exactly to reductions in the original calculus. An example reduction is given in the left column of Figure 3.1. The two reduction steps of $((((II)V^1)V^2)\ldots)V^k$ described above would yield the following reduction in the abstract machine (where the second search for the

$$\begin{array}{cc}
\underbracket{((\lambda x.\lambda y.xy)I)I} & \langle((\lambda x.\lambda y.xy)I)I \mid \star\rangle \\
\triangledown_{\mathrm{m}} & \triangledown_{\mathrm{m}} \\
\underbracket{((\lambda x.\lambda y.xy)I)}\,I & \langle((\lambda x.\lambda y.xy)I \mid I\bullet\star\rangle \\
\triangledown_{\mathrm{m}} & \triangledown_{\mathrm{m}} \\
(\underbracket{(\lambda x.\lambda y.xy)}\,I)I & \langle\lambda x.\lambda y.xy \mid I\bullet I\bullet\star\rangle \\
\triangledown_{\mathrm{r}} & \triangledown_{\mathrm{r}} \\
\underbracket{(\lambda y.Iy)}\,I & \langle\lambda y.Iy \mid I\bullet\star\rangle \\
\triangledown_{\mathrm{r}} & \triangledown_{\mathrm{r}} \\
\underbracket{II} & \langle II \mid \star\rangle \\
\triangledown_{\mathrm{m}} & \triangledown_{\mathrm{m}} \\
\underbracket{I}\,I & \langle I \mid I\bullet\star\rangle \\
\triangledown_{\mathrm{r}} & \triangledown_{\mathrm{r}} \\
I & \langle I \mid \star\rangle
\end{array}$$

Fig. 3.1. Example reduction in an abstract machine

next redex to reduce is immediate):

$$\begin{array}{lll}
\underbracket{((((II)V^1)V^2)\dots)V^k} & \triangleright^{k+1}_{\mathrm{m}} & ((((\underbracket{I}\,I)V^1)V^2)\dots)V^k \\
 & \triangleright_{\mathrm{r}} & (((\underbracket{I}\,V^1)V^2)\dots)V^k \\
 & \triangleright_{\mathrm{r}} & ((\underbracket{V^1}\,V^2)\dots)V^k
\end{array}$$

Instead of $\mathbb{O}\boxed{\underbracket{T}}$ it is common to write $\langle T \mid \mathbb{O}\rangle$, which is often called a configuration / command of the abstract machine. With this notation, the reductions become $\langle TV \mid \mathbb{O}\rangle \triangleright_{\mathrm{m}} \langle T \mid \mathbb{O}\boxed{\square V}\rangle$ and $\langle \lambda x.T \mid \mathbb{O}\boxed{\square V}\rangle \triangleright_{\mathrm{r}} \langle T[V/x] \mid \mathbb{O}\rangle$. Notice that contexts are used in an inside-out fashion: The first part of the context the abstract machine looks at is the innermost part. This leads to the "inside-out" syntax for contexts: We write $V\bullet\mathbb{O}$ for $\mathbb{O}\boxed{\square V}$ and $\star$ for $\square$, so that $((\square V^1)\dots)V^k = ((\square\boxed{\square V^k})\dots)\boxed{\square V^1}$ is written $V^1\bullet(\dots\bullet(V^k\bullet\star))$ where the arguments appear in the order in which they will be (possibly) needed by the computation. With this syntax, the reductions become

$$\begin{array}{rcl}
\langle TV \mid \mathbb{O}\rangle & \triangleright_{\mathrm{m}} & \langle T \mid V\bullet\mathbb{O}\rangle \\
\langle \lambda x.T \mid V\bullet\mathbb{O}\rangle & \triangleright_{\mathrm{r}} & \langle T[V/x] \mid \mathbb{O}\rangle
\end{array}$$

If we replay the reduction of $(\lambda x.\lambda y.xy)II$, the right column of Figure 3.1.

One way of thinking of the reduction in the calculus is that $\mathbb{O}\boxed{T} \triangleright \mathbb{O}\boxed{T'}$ (where $T \succ T'$, i.e. $\mathbb{O}$ is chosen maximal in the decomposition) if and only if $\underbracket{\mathbb{O}\boxed{T}} \triangleright^{*}_{\mathrm{m}} \mathbb{O}\boxed{\underbracket{T}} \triangleright_{\mathrm{r}} \mathbb{O}\boxed{\underbracket{T'}} \triangleleft^{*}_{m} \underbracket{\mathbb{O}\boxed{T'}}$: We move downwards until we reach something we can reduce, then reduce it, and move upwards until we reach the top of the term. The "search for the next redex" happening only once can then be seen simplifying the reduction using the fact that $\triangleright_{\mathrm{m}}$ is deterministic (i.e $T^1 \triangleleft_m T \triangleright_{\mathrm{m}} T^2$ implies $T^1 = T^2$) as shown in Figure 3.2.

3.2 Abstract machine calculi

As we have seen above, the $\triangleright$ reduction of the abstract machine is more precise than the one of the original calculus: The $\triangleright_{\mathrm{m}}$ moves that were invisible in the calculus are now visible. Having a calculus plus an abstract machine leads to duplication of some lemmas, and requires some other lemmas relating the two variants of many operations (substitutions, reductions, ...) and properties (termination, closedness, ...). Fortunately, we can combine the advantages of both the calculus (including being suited to reason about the equational theory), and the abstract machine (including being able to more precisely model evaluation) by representing subterms by subcommands, which we will denote by c. Instead of moving the focus marker $\underbracket{\cdot}$, the reduction steps $\triangleright_{\mathrm{r}}$ now simply removes it since there is already another one waiting. In other words, the reduction $\triangleright_r$ is now $\mathbb{O}\boxed{\underbracket{(\lambda x.T)}\,V} \triangleright_{\mathrm{r}} \mathbb{O}\boxed{T[V/x]}$ (instead of $\mathbb{O}\boxed{\underbracket{T[V/x]}}$) because T already has a focused subterm. With subterms being represented represented as subcommands, we can define $\to_m$ and $\to_r$ by taking the contextual closures of $\triangleright_m$ and $\triangleright_r$. For example, $(\lambda x.xV)I$ will be represented by $\underbracket{(\lambda x.\underbracket{xV})I}$ and reduce as follows:

$$\begin{array}{ccc}
\underbracket{(\lambda x.\underbracket{xV})(\lambda y.\underbracket{y})} & \to_m & \underbracket{(\lambda x.\underbracket{x}\,V)(\lambda y.\underbracket{y})} \\
\triangledown_{\mathrm{m}} & & \triangledown_{\mathrm{m}} \\
\underbracket{(\lambda x.\underbracket{xV})}\,(\lambda y.\underbracket{y}) & \to_m & \underbracket{(\lambda x.\underbracket{x}\,V)}\,(\lambda y.\underbracket{y}) \\
\triangledown_{\mathrm{r}} & & \triangledown_{\mathrm{r}} \\
\underbracket{(\lambda y.\underbracket{y})V} & \triangleright_{\mathrm{m}} & \underbracket{(\lambda y.\underbracket{y})}\,V \triangleright_{\mathrm{r}} \underbracket{V}
\end{array}$$

In a more abstract-machine-like syntax, this would correspond to the following:

$$\begin{array}{ccc}
\langle(\lambda x.\langle xV \mid \star\rangle)(\lambda y.\langle y \mid \star\rangle) \mid \star\rangle & \to_m & \langle(\lambda x.\langle x \mid V\bullet\star\rangle)(\lambda y.\langle y \mid \star\rangle) \mid \star\rangle \\
\triangledown_{\mathrm{m}} & & \triangledown_{\mathrm{m}} \\
\langle(\lambda x.\langle xV \mid \star\rangle) \mid (\lambda y.\langle y \mid \star\rangle)\bullet\star\rangle & \to_m & \langle(\lambda x.\langle x \mid V\bullet\star\rangle) \mid (\lambda y.\langle y \mid \star\rangle)\bullet\star\rangle \\
\triangledown_{\mathrm{r}} & & \triangledown_{\mathrm{r}} \\
\langle(\lambda y.\langle y \mid \star\rangle)V \mid \star\rangle & \triangleright_{\mathrm{m}} & \langle\lambda y.\langle y \mid \star\rangle \mid V\bullet\star\rangle \triangleright_{\mathrm{r}} \langle V \mid \star\rangle
\end{array}$$

Notice that during a $\triangleright_{\mathrm{r}}$ step, the operational contexts are concatenated:

$$\mathbb{O}^1\boxed{\underbracket{(\lambda x.\mathbb{O}^2\boxed{\underbracket{T}})}\,V} \triangleright_{\mathrm{r}} \mathbb{O}^1\boxed{(\mathbb{O}^2\boxed{\underbracket{T}})[V/x]} = ((\mathbb{O}^1\boxed{\mathbb{O}^2})\boxed{\underbracket{T}})[V/x]$$

where the concatenation $\mathbb{O}^1\boxed{\mathbb{O}^2}$ of two contexts is the non-capture-avoiding substitution of $\square$ by $\mathbb{O}^2$ in $\mathbb{O}^1$, i.e. for $\mathbb{O}^1 = \square V^1\dots V^k$ and $\mathbb{O}^2 = \square W^1\dots W^l$, we have $\mathbb{O}^1\boxed{\mathbb{O}^2} = \square W^1\dots W^k V^1\dots V^l$ and the reduction above becomes:

$$\begin{array}{c}
\underbracket{(\lambda x.\underbracket{T}\,W^1\dots W^l)}\,V^0\dots V^k \\
\triangledown_{\mathrm{r}} \\
(\underbracket{T[V^0/x]}\,W^1[V^0/x]\dots W^l[V^0/x])V^1\dots V^k
\end{array}$$

In an abstract-machine-like calculus this reduction would be written:

$$\begin{array}{c}
\langle\lambda x.\langle T \mid W^1\bullet\dots\bullet W^l\bullet\star\rangle \mid V^0\bullet\dots\bullet V^k\bullet\star\rangle \\
\triangledown_{\mathrm{r}} \\
\langle T[V^0/x] \mid W^1[V^0/x]\bullet\dots\bullet W^l[V^0/x]\bullet V^1\bullet\dots\bullet V^k\bullet\star\rangle
\end{array}$$

Notice that if we were to think of $\star$ as a variable, then we could write the following for the reduced command:

$$\langle T \mid W^1\bullet\dots\bullet W^l\bullet\star\rangle[V^0/x, V^1\bullet\dots\bullet V^k\bullet\star/\star]$$

This observation leads to using the syntax $\mu\langle(x\bullet\star).c\rangle$ instead of $\lambda x.c$, so that the reduction $\triangleright_{\mathrm{r}}$ becomes:

$$\langle\mu\langle(x\bullet\star).c\rangle \parallel V\bullet S\rangle \triangleright_{\mathrm{r}} c[V/x, S/\star]$$

The notation $\mu\langle(x\bullet\star).c\rangle$ for $\lambda x.c$ can be understood as stating that $\lambda x.c$ pattern-matches the context. Similarly, a negative pair $(T^1 \,\&\, T^2)$ will be written $\mu\langle(\pi_1\bullet\star).c^1 \mid (\pi_2\bullet\star).c^2\rangle$ with the

$$\begin{array}{llllll}
\underbracket{\mathbb{O}^1\boxed{T}} \rhd^*_\mathrm{m} \mathbb{O}^1\boxed{\underbracket{T}} \rhd_\mathrm{r} \mathbb{O}^1\boxed{\underbracket{T'}} & \lhd^*_m \underbracket{\mathbb{O}^1\boxed{T'}} = \underbracket{\mathbb{O}^1\boxed{\mathbb{O}^2\boxed{U}}} \rhd^*_\mathrm{m} & \mathbb{O}^1\boxed{\underbracket{\mathbb{O}^2\boxed{U}}} \rhd^*_\mathrm{m} & \mathbb{O}^1\boxed{\mathbb{O}^2\boxed{\underbracket{U}}} \rhd_\mathrm{r} \mathbb{O}^1\boxed{\boxed{\mathbb{O}^2\boxed{\underbracket{U'}}}} \lhd^*_m \underbracket{\mathbb{O}^1\boxed{\mathbb{O}^2\boxed{U'}}} \\
\underbracket{\mathbb{O}^1\boxed{T}} \rhd^*_\mathrm{m} \mathbb{O}^1\boxed{\underbracket{T}} \rhd_\mathrm{r} \mathbb{O}^1\boxed{\underbracket{T'}} & \qquad = & \mathbb{O}^1\boxed{\underbracket{\mathbb{O}^2\boxed{U}}} \rhd^*_\mathrm{m} & \mathbb{O}^1\boxed{\mathbb{O}^2\boxed{\underbracket{U}}} \rhd_\mathrm{r} \mathbb{O}^1\boxed{\boxed{\mathbb{O}^2\boxed{\underbracket{U'}}}} \lhd^*_m \underbracket{\mathbb{O}^1\boxed{\mathbb{O}^2\boxed{U'}}}
\end{array}$$

Fig. 3.2. Simplifying reductions in an abstract machine

intuition being again that $(T^1 \& T^2)$ pattern matches the context just above it, and then goes to T^1 or T^2 depending on which projection it sees. With this intuition that terms "look at the operational context they are evaluated in", we add $\mu\star.\,c$ with the reduction rule $\langle\mu\star.\,c \parallel S\rangle \rhd c[S/\star]$. This allows to define the following constructions as notations: $TV \overset{\text{ntn}}{=} \mu\star.\langle T \parallel V \bullet \star\rangle$ and $\pi_i\,(T) \overset{\text{ntn}}{=} \mu\star.\langle T \parallel \pi_i \bullet \star\rangle$. The calculi $\mathrm{L}^{\to}_{\mathrm{n,i}}$ and $\mathrm{L}^{\to}_{\mathrm{v,i}}$, which are abstract-machine-like syntaxes for $\lambda^{\to}_{\mathrm{n}}$ and $\lambda^{\to}_{\mathrm{v}}$ respectively are described in Figures A.1 and A.2. The $\mathrm{L}_\mathrm{p}^{\to\&\Uparrow\otimes\oplus\Downarrow}$ calculus is described in Figure A.3 alongside a new description of the syntax of $\lambda_\mathrm{p}^{\to\&\Uparrow\otimes\oplus\Downarrow}$, with the same layout to show similarities.

The last remaining step is to generalize $\mu\star.\,c$ to $\mu\alpha.\,c$, i.e. allow several stack variables. The idea is that the typing system of $\mathrm{L}_\mathrm{p}^{\to\&\Uparrow\otimes\oplus\Downarrow}$ is the sequent calculus, and that in a sequent $A_1 \wedge\cdots\wedge A_n \vdash B_1 \vee\cdots\vee B_m$, value variables x correspond to the hypothesis, i.e. $x_i : A_i$, and stack variables correspond to conclusions, i.e. $\alpha_i : B_i$. The λ-calculus is intuitionistic so that we only needed one stack variable, named $\star$, which corresponded to the unique conclusion of the intuitionistic sequents. Since we had two polarities, we needed to prevent having both $\star^+$ and $\star^-$ free at the same time, hence the cumbersome syntax described in Figure A.3. The syntax with several stack variables is much nicer, as show in Figure 3.3.

It is clear that $\mathrm{L}_\mathrm{p,i}^{\to\&\Uparrow\otimes\oplus\Downarrow}$ is a subcalculus of $\mathrm{L}_\mathrm{p}^{\to\&\Uparrow\otimes\oplus\Downarrow}$, but the description of $\mathrm{L}_\mathrm{p,i}^{\to\&\Uparrow\otimes\oplus\Downarrow}$ is cumbersome and therefore prefer to define it directly as a subcalculus of $\mathrm{L}_\mathrm{p}^{\to\&\Uparrow\otimes\oplus\Downarrow}$. To do so, we define the set number of occurrences of a variable x^ε or α^ε as follows: $|x^{\varepsilon_1}|_{y^{\varepsilon_2}} = \{1\}$ if $x^{\varepsilon_1} = y^{\varepsilon_2}$ and $\{0\}$ otherwise, and similarly for other variables. For binders, if the variable is bound then it is $\{0\}$: $|\tilde\mu[(x^+, y^+).\,c]|_{x^+} = \{0\}$, and otherwise, it is the *set* sum of the result for each subcommand: if $y^\varepsilon \neq {x_i}^+$ then $\left|\tilde\mu[\,\iota_1\,({x_1}^+).\,c^1 \mid \iota_2\,({x_2}^+).\,c^2]\right|_{y^\varepsilon} = \left\{k + l : k \in \left|c^1\right|_{y^\varepsilon} \wedge l \in \left|c^2\right|_{y^\varepsilon}\right\}$. For constructors, we also take the set sum of the different components: $|v_+ \bullet s_-|_{y^\varepsilon} = \left\{k + l : k \in |v_+|_{y^\varepsilon} \wedge l \in |s_-|_{y^\varepsilon}\right\}$. We say that a free variable a is used linearly in c if $|c|_a \subseteq \{1\}$. The intuitionistic part of the calculus is exactly the part where all stack variables are used linearly in any command they are free in.

4 TOWARDS A STANDARD THEORY OF L

In this section, we revisit two important parts of the standard theory of the λ-calculus (solvability, and η-conversion) in $\mathrm{L}_\mathrm{p}^{\to\&\Uparrow\otimes\oplus\Downarrow}$. The goal is to convince the reader that studying them in an abstract-machine-like calculus makes things easier. [3]

4.1 Solvability

In the call-by-name λ-calculus, some terms without normal forms are still operationally relevant, i.e. they can be used to produce a result. For example, the Y combinator $Y = \lambda z.\,(\lambda x.\,z(xx))(\lambda x.\,z(xx))$ has no $\to$-normal form but $Y(\lambda x.\,I)$ does. One formal definition of T being solvable is the following: For any T', there exists a context $\mathbb{K}$ such that $\mathbb{K}\boxed{T} \to^* T'$, and it is not the case that for all U, $\mathbb{K}\boxed{U} \to^* T'$. The second part of the definition ensures that $\mathbb{K}$ really uses whatever is placed in the hole (which disallows, for example $\mathbb{K} = \mathsf{let}\ x = \lambda y.\,\square\ \mathsf{in}\ I$), and the first part ensures that T can be used to produce any T', and therefore in particular ones that we consider to be "results". This definition is very close[4] to the (SolC) one of [7]. There are many equivalent variations of this definition, including some that restrict the shape of contexts to ensure that the term plugged in the hole is evaluated (therefore removing the need for the second part of the definition), or choosing a special T', often I. Our favorite version is the following: A λ-terms T is solvable iff there exists a variable x, a substitution σ and an operational context $\mathbb{O}$ such that $\mathbb{O}\boxed{T[\sigma]} \rhd^* x$. Note that we changed the reduction from $\to$ to $\rhd$, but this is equivalent thanks to standardization (i.e. if $T \to^* T'$ then there exists U such that $T \rhd^* U\ (\to \setminus \rhd)^*\ T'$) and the fact that there is no U such that $U\ (\to \setminus \rhd)\ x$. We now define solvability in $\mathrm{L}_\mathrm{p}^{\to\&\Uparrow\otimes\oplus\Downarrow}$, adapting this last definition of solvability.

Definition 1. A command c is said to be solvable when there exists a substitution φ (of values and stacks) such that $c[\varphi] \rhd^* \langle x^\varepsilon \parallel \star^\varepsilon\rangle^\varepsilon$. A term t_ε is solvable when $\langle t_\varepsilon \parallel \star^\varepsilon\rangle^\varepsilon$ is, and an evaluation context e_ε is solvable when $\langle x^\varepsilon \parallel e_\varepsilon\rangle^\varepsilon$ is.

Note that all positive values are solvable: $\langle V_+ \parallel \star^+\rangle^+[\tilde\mu x^+.\,\langle y^\varepsilon \parallel \star^\varepsilon\rangle^\varepsilon/\star^+] = \langle V_+ \parallel \tilde\mu x^+.\,\langle y^\varepsilon \parallel \star^\varepsilon\rangle^\varepsilon\rangle^+ \rhd \langle y^\varepsilon \parallel \star^\varepsilon\rangle^\varepsilon$. The intuition behind the correspondence between this definition and the one in the λ-calculus is that φ is the value substitution σ extended by $\star^\varepsilon \mapsto s_\varepsilon$ with s_ε corresponding to the operational context $\mathbb{O}$. This definition is the right one:

Lemma 2. *A command c is solvable if and only if for any c', there exists $\mathbb{k}$ such that $\mathbb{k}\boxed{c} \rhd^* c'$ and it is not the case that for all d, $\mathbb{k}\boxed{d} \rhd^* c'$.*

The $\Rightarrow$ half of the proof is done by transforming φ into a context by combining contexts of the shape $\langle\mu\star^\varepsilon.\,\square \parallel s_\varepsilon\rangle^\varepsilon$ and $\langle v_\varepsilon \parallel \tilde\mu x^\varepsilon.\,\square\rangle^\varepsilon$, and taking d to be any diverging command. The $\Leftarrow$ is a bit trickier. First, we extend reductions to contexts in such a way that if $\mathbb{k} \rhd \mathbb{k}'$ then $\mathbb{k}\boxed{c} \rhd \mathbb{k}'\boxed{c}$. There are several ways of achieving this, and all resolve around how $\square[\varphi]$ is defined, the idea being that we somehow have to record the substitution on the hole so that it can later be applied to the term we plug. This can, for example, be done by adding explicit substitutions [6]. For our uses, a slightly simpler approach works: changing the syntax of contexts so that every hole $\square^\varphi$ comes with a substitution φ (and defining $\square$ as a notation for when φ is the identity), and taking $\square^\varphi[\psi] \overset{\text{def}}{=} \square^{\psi\circ\varphi}$ and

---

[3] More details can be found in TODO

[4] The only difference being that they quantify over $\to$-normal T'. Both definitions are still equivalent: If one can reach I then can reach any T by replacing $\mathbb{K}$ by $\mathbb{K}T$.

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
v_+ & ::= & x^+ & s_+, e_+ & ::= & \alpha^+ \mid \tilde{\mu}x^+ . c \\
 & \mid & (v_+, w_+) & & \mid & \tilde{\mu}[(x^+, y^+) . c] \\
 & \mid & \iota_1\,(v_+) \mid \iota_2\,(v_+) & & \mid & \tilde{\mu}[\,\iota_1\,({x_1}^+) . c^1 \mid \iota_2\,({x_2}^+) . c^2] \\
 & \mid & \{v_-\} & & \mid & \tilde{\mu}\{x^-\} . c \\
 & & & & & \\
v_-, t_- & ::= & x^- \mid \mu\alpha^- . c & s_- & ::= & \alpha^- \\
 & \mid & \mu\langle (x^+ \bullet \star^-) . c \rangle & & \mid & v_+ \bullet s_- \\
 & \mid & \mu\langle (\pi_1 \bullet \star^-) . c^1 \mid (\pi_2 \bullet \star^-) . c^2 \rangle & & \mid & \pi_1 \bullet s_- \mid \pi_2 \bullet s_- \\
 & \mid & \mu\langle \{\star^+\} . c \rangle & & \mid & \{s_+\} \\
 & & & & & \\
t_+ & ::= & \mu\alpha^+ . c & e_- & ::= & \tilde{\mu}x^- . c
\end{array}
$$

$$c ::= \langle t_- \parallel e_- \rangle^- \mid \langle t_+ \parallel e_+ \rangle^+$$

Fig. 3.3. The $\mathrm{L}_{\mathrm{p}}^{\to\&\Uparrow\otimes\oplus\Downarrow}$ calculus

$$
\begin{array}{rcl}
\underset{\cdots}{\cdot}_{\mathrm{p}} : V_{\mathrm{v}} & \to & V_- \\
\underset{\cdots}{x^{\mathrm{v}}}_{\mathrm{p}} & \overset{\mathrm{def}}{=} & x^- \\
\underset{\cdots}{\lambda x^{\mathrm{v}} . T_{\mathrm{v}}}_{\mathrm{p}} & \overset{\mathrm{def}}{=} & \lambda y^- . \operatorname{unbox}^{\mathrm{p}}\,(\operatorname{unfreeze}^{\mathrm{p}}\,(\underline{T_{\mathrm{v}}}_{\mathrm{p}})) \\
 & & \\
\underline{\cdot}_{\mathrm{p}} : T_{\mathrm{v}} & \to & T_- \\
\underline{\operatorname{val}^{\mathrm{v}}\,(V_{\mathrm{v}})}_{\mathrm{p}} & \overset{\mathrm{def}}{=} & \operatorname{freeze}^{\mathrm{p}}\,(\operatorname{box}^{\mathrm{p}}\,(\underset{\cdots}{V_{\mathrm{v}}}_{\mathrm{p}})) \\
\underline{T_{\mathrm{v}} \mathbin{@^{\mathrm{v}}} V_{\mathrm{v}}}_{\mathrm{p}} & \overset{\mathrm{def}}{=} & \operatorname{unbox}^{\mathrm{p}}\,(\operatorname{unfreeze}^{\mathrm{p}}\,(\underline{T_{\mathrm{v}}}_{\mathrm{p}})) \mathbin{@^{-,-}} \underset{\cdots}{V_{\mathrm{v}}}_{\mathrm{p}} \\
\underline{\operatorname{let} x^{\mathrm{v}} = T_{\mathrm{v}} \operatorname{in} U_{\mathrm{v}}}_{\mathrm{p}} & \overset{\mathrm{def}}{=} & \operatorname{match}^{\mathrm{p}} \operatorname{unfreeze}^{\mathrm{p}}\,(\underline{T_{\mathrm{v}}}_{\mathrm{p}}) \operatorname{with} [\operatorname{box}^{\mathrm{p}}\,(x^-) . \underline{U_{\mathrm{v}}}_{\mathrm{p}}]
\end{array}
$$

Fig. 4.1. Another translation from λ_{v} to λ_{p}

$\square^{\varphi}\boxed{t} \overset{\mathrm{def}}{=} t[\varphi]$. We also extend the definition of plugging so that it places the term in all holes (in case the original hole got duplicated by a reduction). Going back to the proof of the $\Leftarrow$ direction, by taking $c' = \langle x^{\varepsilon} \parallel \star^{\varepsilon} \rangle^{\varepsilon}$, we get that $\Bbbk\boxed{c} \triangleright^* \langle x^{\varepsilon} \parallel \star^{\varepsilon} \rangle^{\varepsilon}$ and there exists d such that we do not have $\Bbbk\boxed{d} \triangleright^* \langle x^{\varepsilon} \parallel \star^{\varepsilon} \rangle^{\varepsilon}$. We can not have $\Bbbk \triangleright^{\omega}$ because otherwise we would have $\Bbbk\boxed{c} \triangleright^{\omega}$. Let $\Bbbk'$ be the normal form of $\Bbbk$: $\Bbbk \triangleright^* \Bbbk' \ntriangleright$. We now show that we necessarily have $\Bbbk' = \square^{\varphi}$, so that we can conclude that $c[\varphi] \triangleright^* \langle x^{\varepsilon} \parallel \star^{\varepsilon} \rangle^{\varepsilon}$. The only other possible shape for $\Bbbk'$ is $\Bbbk' = \langle \mathbb{t} \parallel \mathbb{e} \rangle^{\varepsilon}$. Since it is not a redex, either it is a clash, or at least one of the two sides is a variable. If both sides are variables, i.e. $\Bbbk' = \langle x^{\varepsilon} \parallel \star^{\varepsilon} \rangle^{\varepsilon}$ (which is possible if the hole was in an erased subterm), then $\Bbbk\boxed{d} = \langle x^{\varepsilon} \parallel \star^{\varepsilon} \rangle^{\varepsilon}$ which is absurd. Otherwise, $\Bbbk'\boxed{c} \ntriangleright$ and $\Bbbk'\boxed{c} \neq \langle x^{\varepsilon} \parallel \star^{\varepsilon} \rangle^{\varepsilon}$ which is absurd. Note that replaying this argument in a natural-deduction-style calculus would be much harder: notations are less convenient as instead of having $c[\varphi]$, one would have $T[\sigma]\overrightarrow{V}$; and the case analysis on the shape of $\Bbbk'$ would be much more complicated.

A natural question at this point is: Do the embeddings presented in earlier sections preserve solvability? Note that even for embeddings that behave well with respect to the operational reduction, the strong reduction, substitutions and plugging, this question is still valid: they are not surjective, and since there are more contexts in the target, it could very well be the case that some of the extra contexts make a term operationally relevant. For the embedding of λ_{n} in λ_{p} of Figure 1.10, this does not happen: The only extra freedom that the contexts get is the ability to give as argument to functions a term that is not inside a box, but since functions match on it immediately, giving them anything else yields a clash. For the embedding of λ_{v} in λ_{p} of Figure 1.11 however, solvability is not preserved: $\lambda x^{\mathrm{v}} . \Omega_{\mathrm{v}}$ is not solvable but $\underline{\lambda x^{\mathrm{v}} . \Omega_{\mathrm{v}}}_{\mathrm{p}} = \operatorname{box}^{\mathrm{p}}\,(\lambda x^+ . \operatorname{freeze}^{\mathrm{p}}\,(\underline{\Omega_{\mathrm{v}}}_{\mathrm{p}}))$ is because it is a positive value. In fact, $\underline{T_{\mathrm{v}}}_{\mathrm{p}}$ is solvable if and only if T_{v} is potentially valuable.

The problem is that we translated $A_{\mathrm{v}} \to B_{\mathrm{v}}$ as $\Downarrow(\underline{A_{\mathrm{v}}}_{\mathrm{p}} \to \Uparrow B_{\mathrm{v}})$, i.e. a positive type. It could be tempting to send $A_{\mathrm{v}} \to B_{\mathrm{v}}$ to $\Downarrow\underline{A_{\mathrm{v}}}_{\mathrm{p}} \to \Uparrow\Downarrow\underline{B_{\mathrm{v}}}_{\mathrm{p}}$, however while $\underline{\lambda x^{\mathrm{v}} . \Omega_{\mathrm{v}}}_{\mathrm{p}} = \lambda x^+ . \operatorname{freeze}^{\mathrm{p}}\,(\operatorname{box}^{\mathrm{p}}\,(\dots))$ is no longer positive, it is still solvable. The problem is more general that just having the function in box: If there is a box in the translation of a function $\underline{\lambda x^{\mathrm{v}} . T_{\mathrm{v}}}_{\mathrm{p}}$ that is accessible without evaluating the body of the function, then there is a context that just extracts this box, and the translation of the function is therefore solvable. The solution is to send $A_{\mathrm{v}} \to B_{\mathrm{v}}$ to $\Uparrow\Downarrow(\Downarrow\underline{A_{\mathrm{v}}}_{\mathrm{p}} \to \underline{B_{\mathrm{v}}}_{\mathrm{p}})$ as shown in Figure 4.1. In this translation, we send values to fully evaluated negative terms, i.e. variables or functions, and terms to negative terms that evaluate to a term of the shape $\operatorname{freeze}^{\mathrm{p}}\,(\operatorname{box}^{\mathrm{p}}\,(V_-))$. Since it is the function that forces the evaluation of its body, and not our translation of application, no context will be able to use a function without evaluating its body. Once we $\operatorname{unfreeze}^{\mathrm{p}}$ and $\operatorname{unbox}^{\mathrm{p}}$ the result of this translation, we get what we wanted: T_{v} is solvable if and only if $\operatorname{unbox}^{\mathrm{p}}\,(\operatorname{unfreeze}^{\mathrm{p}}\,(\underline{T_{\mathrm{v}}}_{\mathrm{p}}))$ is.

Another good property of L_{p} to study solvability is that it was built with effects in mind. We conjecture that this allows to reconcile both view of solvability presented in [7]. This paper argues that operational relevance should be defined with respect to open contexts, and that stuck terms should be considered results. In the call-by-name λ-calculus this distinction does not matter since the only stuck terms are solvable. For example, $U \overset{\mathrm{def}}{=} \lambda x^{\mathrm{v}} . \operatorname{let} y^{\mathrm{v}} = x^{\mathrm{v}} I_{\mathrm{v}} \operatorname{in} \delta_{\mathrm{v}}\delta_{\mathrm{v}}$ is now considered solvable, while $\lambda x^{\mathrm{v}} . \delta_{\mathrm{v}}\delta_{\mathrm{v}}$ still is not, so that those to terms are no longer considered equivalent. Indeed, even though when applied to a closed value, both terms will reduce to $\delta_{\mathrm{v}}\delta_{\mathrm{v}}$ and therefore diverge, applying them to an open value (for example a variable z^{v}) will distinguish them: $U z^{\mathrm{v}}$ converges while $(\lambda x^{\mathrm{v}} . \delta_{\mathrm{v}}\delta_{\mathrm{v}}) z^{\mathrm{v}}$ diverges. Though the translation of Figure 4.1, both are unsolvable. However, if we add effects to the language it becomes clear they they should be distinguished. For example, we we add exceptions to the language (i.e. add $\operatorname{throw}^+\,(\text{"text"})$ to T_+ and $\operatorname{throw}^-\,(\text{"text"})$ to T_-), we could apply $\underline{U}_{\mathrm{p}}$ to $\mathsf{throw}^{-}\ (\text{"The variable x is in head position!"})$ and catch this error with the surrounding context. Note that in the classical variant of L_{p}, they can also be distinguished by applying them to $\mu\alpha^{-}.\langle x^{\varepsilon} \parallel \beta^{\varepsilon}\rangle^{\varepsilon}$.

More generally, we conjecture that most variants of solvability for pure calculi can be encoded in L_{p} extended by some effect, as it gives many possibilities for tweaking the translation to prevent unwanted observations / to allow more observations. The rest of this paragraph is to be understood as raw untested intuition, and the reader should read the following sentences as if they had "maybe" or "perhaps" inserted everywhere. Allowing variables to be effectful could be done by encoding them as negative variables, while encoding them as positive variables prevents this. Forcing something to be solvable can be done by placing it in $\mathsf{box}^{\mathrm{p}}$. The distinction between lazy / weak calculi (i.e. between those that use weak head reduction as operational reduction) and strong calculi (i.e. those that use head reduction as operational reduction) can be done by placing a $\mathsf{freeze}^{\mathrm{p}}$ under λ-abstractions that gives the context the choice between reducing the body or not *after* giving the argument.

We have extended an operational characterization of solvability in L in a to-be-resubmitted paper (link).

4.2 Dynamically typed L and η-conversion

The thing that allows to use η-conversion in the untyped λ-calculus is that everything is a function. However, once one adds other datatypes to the calculus, for example pairs, sums or boolean, the untyped calculus becomes much less well-behaved. The reason for this is that clashes, i.e. the interaction of two constructors that were not supposed to interact, appear. Examples include $\mathsf{match}\ \iota_1\ (V)\ \mathsf{with}\ [(x \otimes y).\,T]$, $\mathsf{match}\ \lambda x.\,T\ \mathsf{with}\ [(x \otimes y).\,U]$ and $\pi_1\ (\lambda x.\,T)$. These clashes considerably complicate the study of the untyped calculus, for example invalidating η-conversion: $\pi_1\ ((V \,\&\, W))$ is fine but $\pi_1\ (\lambda x.\,(V \,\&\, W)x)$ is a clash[5]. Indeed, most calculi with datatypes other than functions restrict η-conversion to typed terms.

Most dynamically typed programming languages allow to match over different constructors, even if they are of different types. For example, one can write $\mathsf{match}\ T\ \mathsf{with}\ \left[(x \otimes y).\,U^1 \mid \iota_1\ (z).\,U^2\right]$. Notice that if one replaces all the different $\tilde{\mu}$s by a big $\tilde{\mu}$ over all positive value constructors, then there can no longer be clashes in positive commands:

$$\tilde{\mu}\left[(x_1{}^{+}, y_1{}^{+}).\,c^1 \mid \iota_1\,(x_2{}^{+}).\,c^2 \mid \iota_2\,(x_3{}^{+}).\,c^3 \mid \{x^{-}\}.\,c^4\right]$$

Dually, replacing the μs by a single big μ removes clashes in negative commands:

$$\mu\left\langle (x^{+} \cdot \star^{-}).\,c^1 \mid (\pi_1 \cdot \star^{-}).\,c^2 \mid (\pi_2 \cdot \star^{-}).\,c^3 \mid \{\star^{+}\}.\,c^4\right\rangle$$

In a λ-calculus-like syntax, this corresponds to no longer having functions $\lambda x.\,T$, negative pairs $(T \,\&\, U)$ or upshifts $\mathsf{freeze}\ (T)$, but instead a combination of the three that will compute depending on how it is used. Note that with the CBPV syntax, combining functions and negative pairs looks nearly natural: $\lambda x^{\mathrm{pv}}.\,T_{\mathrm{pv}}$ combined with $\lambda^{\mathrm{pv}}\left[1.U^1_{\mathrm{pv}} \mid 2.U^2_{\mathrm{pv}}\right]$ becomes $\lambda^{\mathrm{pv}}\left[x^{\mathrm{pv}}.\,T_{\mathrm{pv}} \mid 1.U^1_{\mathrm{pv}} \mid 2.U^2_{\mathrm{pv}}\right]$.

In addition to making clashes disappear, this makes η-conversion valid again: η-expanding in $\left\langle \mu\left\langle (\pi_1 \cdot \star^{-}).\,c^1 \mid (\pi_2 \cdot \star^{-}).\,c^2\right\rangle \parallel \pi_1 \cdot \star^{-}\right\rangle^{-}$ yielded $\left\langle \mu\left\langle (x^{+} \cdot \star^{-}).\,\left\langle \mu\left\langle (\pi_1 \cdot \star^{-}).\,c^1 \mid (\pi_2 \cdot \star^{-}).\,c^2\right\rangle \parallel x^{+} \cdot \star^{-}\right\rangle^{-}\right\rangle \parallel \pi_1 \cdot \star^{-}\right\rangle^{-}$ which is a clash, but with the η-expansion of the dynamically-typed calculus, all possible stacks are handled so we no longer risk creating clashes!

---

[5] For this specific case of the interaction between functions and lazy pairs, it has been shown [18] that one can safely make constructors that should not interact just cross each other, i.e. $\pi_1\ (\lambda x.\,T) \rightsquigarrow \lambda x.\,\pi_1\ (T)$. However, while this reduction is interesting because it allows to prove that adding pairs leads to a conservative extension, it is unlikely that this is a reduction that we want in our operational semantics, and we are not aware of any similar results for other datatypes.

REFERENCES

[1] Hendrik Pieter Barendregt. 1985. *The lambda calculus - its syntax and semantics.* Studies in logic and the foundations of mathematics, Vol. 103. North-Holland.

[2] Pierre-Louis Curien, Marcelo Fiore, and Guillaume Munch-Maccagnoni. 2016. A Theory of Effects and Resources: Adjunction Models and Polarised Calculi. In *Proc. POPL.* https://doi.org/10.1145/2837614.2837652

[3] Pierre-Louis Curien and Hugo 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] Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. 1997. A new deconstructive logic: linear logic. *Journal of Symbolic Logic* 62, 3 (1997), 755–807. https://doi.org/10.2307/2275572

[5] Martin Erwig and Deling Ren. 2004. Monadification of Functional Programs. *Sci. Comput. Program.* 52, 1–3 (Aug. 2004), 101–129. https://doi.org/10.1016/j.scico.2004.03.004

[6] Murdoch J. Gabbay and Stéphane Lengrand. 2009. The lambda-context calculus (extended version). *Information and Computation* 207, 12 (2009), 1369 – 1400. https://doi.org/10.1016/j.ic.2009.06.004

[7] Á. García-Pérez and P. Nogueira. 2016. No solvable lambda-value term left behind. *Logical Methods in Computer Science* Volume 12, Issue 2 (June 2016). https://doi.org/10.2168/LMCS-12(2:12)2016

[8] Jean-Yves Girard, Paul Taylor, and Yves Lafont. 1989. *Proofs and Types.* Cambridge University Press, USA.

[9] Jean-Louis Krivine. 2007. A call-by-name lambda-calculus machine. *Higher Order Symbolic Computation* 20 (2007), 199–207. https://hal.archives-ouvertes.fr/hal-00154508

[10] Paul Blain Levy. 2004. *Call-By-Push-Value: A Functional/Imperative Synthesis.* Semantics Structures in Computation, Vol. 2. Springer.

[11] Paul Blain Levy. 2006. Call-by-push-value: Decomposing call-by-value and call-by-name. *Higher-Order and Symbolic Computation* 19, 4 (Dec. 2006), 377–414. https://doi.org/10.1007/s10990-006-0480-6

[12] Eugenio Moggi. 1989. Computational Lambda-Calculus and Monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989.* IEEE Computer Society, 14–23. https://doi.org/10.1109/LICS.1989.39155

[13] Eugenio Moggi. 1991. Notions of Computation and Monads. *Inf. Comput.* 93, 1 (1991), 55–92. https://doi.org/10.1016/0890-5401(91)90052-4

[14] Guillaume Munch-Maccagnoni. 2013. *Syntax and Models of a non-Associative Composition of Programs and Proofs.* Ph.D. Dissertation. Univ. Paris Diderot.

[15] Guillaume Munch-Maccagnoni. 2014. Models of a Non-Associative Composition. In *Proceedings of the 17th International Conference on Foundations of Software Science and Computation Structures (FoSSaCs) (Lecture Notes in Computer Science, Vol. 8412)*, Anca Muscholl (Ed.). Springer Heidelberg, 397–412.

[16] Michel Parigot. 1992. $\lambda\mu$-Calculus: An algorithmic interpretation of classical natural deduction. In *Logic Programming and Automated Reasoning*, Andrei Voronkov (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 190–201.

[17] Gabriel Scherer. 2017. Deciding Equivalence with Sums and the Empty Type. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages* (Paris, France) *(POPL 2017).* Association for Computing Machinery, New York, NY, USA, 374–386. https://doi.org/10.1145/3009837.3009901

[18] Kristian Stoevring. 2006. Extending the Extensional Lambda Calculus with Surjective Pairing is Conservative. *Logical Methods in Computer Science* Volume 2, Issue 2 (March 2006). https://doi.org/10.2168/LMCS-2(2:1)2006

Terms / values:

$$\begin{array}{lcl} t_{\mathrm{n}}, u_{\mathrm{n}}, v_{\mathrm{n}}, w_{\mathrm{n}} & ::= & x^{\mathrm{n}} \mid \mu\star^{\mathrm{n}}.\, c_{\mathrm{n}} \\ & \mid & \mu\langle(x^{\mathrm{n}} \bullet \star^{\mathrm{n}}).\, c_{\mathrm{n}}\rangle \end{array}$$

Stacks:

$$\begin{array}{lcl} s_{\mathrm{n}} & ::= & \star^{\mathrm{n}} \\ & \mid & v_{\mathrm{n}} \bullet s_{\mathrm{n}} \end{array}$$

Commands:

$$c_{\mathrm{n}} \; ::= \; \langle t_{\mathrm{n}} \parallel e_{\mathrm{n}} \rangle^{\mathrm{n}}$$

Evaluation contexts:

$$e_{\mathrm{n}} \; ::= \; s_{\mathrm{n}} \mid \tilde{\mu} x^{\mathrm{n}}.\, c_{\mathrm{n}}$$

(a) Syntax

$$\begin{array}{rcl} \langle \mu\star^{\mathrm{n}}.\, c_{\mathrm{n}} \parallel s_{\mathrm{n}} \rangle^{\mathrm{n}} & \triangleright & c_{\mathrm{n}}[s_{\mathrm{n}}/\star^{\mathrm{n}}] \\ \langle v_{\mathrm{n}} \parallel \tilde{\mu} x^{\mathrm{n}}.\, c_{\mathrm{n}} \rangle^{\mathrm{n}} & \triangleright & c_{\mathrm{n}}[v_{\mathrm{n}}/x^{\mathrm{n}}] \\ \langle \mu\langle(x^{\mathrm{n}} \bullet \star^{\mathrm{n}}).\, c_{\mathrm{n}}\rangle \parallel v_{\mathrm{n}} \bullet s_{\mathrm{n}} \rangle^{\mathrm{n}} & \triangleright & c_{\mathrm{n}}[v_{\mathrm{n}}/x^{\mathrm{n}}, s_{\mathrm{n}}/\star^{\mathrm{n}}] \end{array}$$

(b) Operational reduction

Fig. A.1. Pure call-by-name L-calculus: $\mathrm{L}_{\mathrm{n}}^{\rightarrow}$

Values:

$$\begin{array}{lcl} v_{\mathrm{v}}, w_{\mathrm{v}} & ::= & x^{\mathrm{v}} \\ & \mid & \mu\langle(x^{\mathrm{v}} \bullet \star^{\mathrm{v}}).\, c_{\mathrm{v}}\rangle \end{array}$$

Stacks / evaluation contexts:

$$\begin{array}{lcl} s_{\mathrm{v}}, e_{\mathrm{v}} & ::= & \star^{\mathrm{v}} \\ & \mid & v_{\mathrm{v}} \bullet s_{\mathrm{v}} \end{array}$$

Terms:

$$t_{\mathrm{n}}, u_{\mathrm{n}} \; ::= \; v_{\mathrm{v}} \mid \mu\star^{\mathrm{n}}.\, c_{\mathrm{n}}$$

Commands:

$$c_{\mathrm{v}} \; ::= \; \langle t_{\mathrm{v}} \parallel e_{\mathrm{v}} \rangle^{\mathrm{v}}$$

(a) Syntax

$$\begin{array}{rcl} \langle \mu\star^{\mathrm{v}}.\, c_{\mathrm{v}} \parallel s_{\mathrm{v}} \rangle^{\mathrm{v}} & \triangleright & c_{\mathrm{v}}[s_{\mathrm{v}}/\star^{\mathrm{v}}] \\ \langle v_{\mathrm{v}} \parallel \tilde{\mu} x^{\mathrm{v}}.\, c_{\mathrm{v}} \rangle^{\mathrm{n}} & \triangleright & c_{\mathrm{v}}[v_{\mathrm{v}}/x^{\mathrm{v}}] \\ \langle \mu\langle(x^{\mathrm{v}} \bullet \star^{\mathrm{v}}).\, c_{\mathrm{v}}\rangle \parallel v_{\mathrm{v}} \bullet s_{\mathrm{v}} \rangle^{\mathrm{n}} & \triangleright & c_{\mathrm{v}}[v_{\mathrm{v}}/x^{\mathrm{v}}, s_{\mathrm{v}}/\star^{\mathrm{v}}] \end{array}$$

(b) Operational reduction

Fig. A.2. Pure call-by-value L-calculus: $\mathrm{L}_{\mathrm{v}}^{\rightarrow}$

A APPENDIX

$$\begin{array}{rcl}
V_{+} & ::= & x^{+} \\
 & \mid & (V_{+} \otimes^{\mathrm{p}} W_{+}) \\
 & \mid & \iota_{1}{}^{\mathrm{p}}\,(V_{+}) \mid \iota_{2}{}^{\mathrm{p}}\,(V_{+}) \\
 & \mid & \mathrm{box}^{\mathrm{p}}\,(V_{-}) \\
 & & \\
V_{-}, T_{-} & ::= & x^{-} \mid E_{-\rightsquigarrow-}\boxed{T_{-}} \mid E_{+\rightsquigarrow-}\boxed{T_{+}} \\
 & \mid & \lambda x^{+}.\, T_{-} \\
 & \mid & (T_{-}^{1} \mathbin{\&^{\mathrm{p}}} T_{-}^{2}) \\
 & \mid & \mathrm{freeze}^{\mathrm{p}}\,(T_{+}) \\
 & & \\
T_{+}, U_{+} & ::= & E_{-\rightsquigarrow+}\boxed{T_{-}} \mid E_{+\rightsquigarrow+}\boxed{T_{+}}
\end{array}$$

$$\begin{array}{rcl}
S_{+\rightsquigarrow+}, E_{+\rightsquigarrow+} & ::= & \square_{+} \\
 & \mid & \mathrm{match}^{+}\ \square_{+}\ \mathrm{with}\,[(x^{+} \otimes^{\mathrm{p}} y^{+}).\, T_{+}] \\
 & \mid & \mathrm{match}^{+}\ \square_{+}\ \mathrm{with}\,[\,\iota_{1}{}^{\mathrm{p}}\,(x_{1}{}^{+}).\, T_{+}^{1} \mid \iota_{2}{}^{\mathrm{p}}\,(x_{2}{}^{+}).\, T_{+}^{2}] \\
 & \mid & \mathrm{match}^{+}\ \square_{+}\ \mathrm{with}\,[\,\mathrm{box}^{\mathrm{p}}\,(x^{-}).\, T_{+}] \\
 & \mid & \mathrm{let}^{+}\ x^{+} = \square_{+}\ \mathrm{in}\ N_{+} \\
 & & \\
S_{-\rightsquigarrow+} & ::= & \\
 & \mid & S_{-\rightsquigarrow+}\boxed{\square_{-}\, V_{+}} \\
 & \mid & S_{-\rightsquigarrow+}\boxed{\pi_{1}^{\mathrm{p}}\,(\,\square_{-}\,)} \mid S_{-\rightsquigarrow+}\boxed{\pi_{2}^{\mathrm{p}}\,(\,\square_{-}\,)} \\
 & \mid & S_{+\rightsquigarrow+}\boxed{\mathrm{unfreeze}^{\mathrm{p}}\,(\,\square_{-}\,)} \\
 & & \\
E_{-\rightsquigarrow+} & ::= & \mathrm{let}^{+}\ x^{-} = \square_{-}\ \mathrm{in}\ U_{+}
\end{array}$$

$$\begin{array}{rcl}
S_{+\rightsquigarrow-}, E_{+\rightsquigarrow-} & ::= & \\
 & \mid & \mathrm{match}^{-}\ \square_{+}\ \mathrm{with}\,[(x^{+} \otimes^{\mathrm{p}} y^{+}).\, T_{-}] \\
 & \mid & \mathrm{match}^{-}\ \square_{+}\ \mathrm{with}\,[\,\iota_{1}{}^{\mathrm{p}}\,(x_{1}{}^{+}).\, T_{-}^{1} \mid \iota_{2}{}^{\mathrm{p}}\,(x_{2}{}^{+}).\, T_{-}^{2}] \\
 & \mid & \mathrm{match}^{-}\ \square_{+}\ \mathrm{with}\,[\,\mathrm{box}^{\mathrm{p}}\,(x^{-}).\, T_{-}] \\
 & \mid & \mathrm{let}^{-}\ x^{+} = \square_{+}\ \mathrm{in}\ T_{-} \\
 & & \\
S_{-\rightsquigarrow-} & ::= & \square_{-} \\
 & \mid & S_{-\rightsquigarrow-}\boxed{\square_{-}\, V_{+}} \\
 & \mid & S_{-\rightsquigarrow-}\boxed{\pi_{1}^{\mathrm{p}}\,(\,\square_{-}\,)} \mid S_{-\rightsquigarrow-}\boxed{\pi_{2}^{\mathrm{p}}\,(\,\square_{-}\,)} \\
 & \mid & S_{+\rightsquigarrow-}\boxed{\mathrm{unfreeze}^{\mathrm{p}}\,(\,\square_{-}\,)} \\
 & & \\
E_{-\rightsquigarrow-} & ::= & \mathrm{let}^{-}\ x^{-} = \square_{-}\ \mathrm{in}\ U_{-}
\end{array}$$

(a) The $\lambda_{\mathrm{p}}^{\rightarrow\&\Uparrow\otimes\oplus\Downarrow}$ calculus

$$\begin{array}{rcl}
v_{+} & ::= & x^{+} \\
 & \mid & (v_{+}, w_{+}) \\
 & \mid & \iota_{1}\,(v_{+}) \mid \iota_{2}\,(v_{+}) \\
 & \mid & \{v_{-}\} \\
 & & \\
v_{-}, t_{-} & ::= & x^{-} \mid \mu\star^{-}.\, c_{\rightsquigarrow-} \\
 & \mid & \mu\langle(x^{+} \bullet \star^{-}).\, c_{\rightsquigarrow-}\rangle \\
 & \mid & \mu\langle(\pi_{1} \bullet \star^{-}).\, c_{\rightsquigarrow-}^{1} \mid (\pi_{2} \bullet \star^{-}).\, c_{\rightsquigarrow-}^{2}\rangle \\
 & \mid & \mu\langle\{\star^{+}\}.\, c_{\rightsquigarrow+}\rangle \\
 & & \\
t_{+} & ::= & \mu\star^{+}.\, c_{\rightsquigarrow+}
\end{array}$$

$$\begin{array}{rcl}
s_{+\rightsquigarrow+}, e_{+\rightsquigarrow+} & ::= & \star^{+} \mid \tilde{\mu}x^{+}.\, c_{\rightsquigarrow+} \\
 & \mid & \tilde{\mu}[(x^{+}, y^{+}).\, c_{\rightsquigarrow+}] \\
 & \mid & \tilde{\mu}[\,\iota_{1}\,(x_{1}{}^{+}).\, c_{\rightsquigarrow+}^{1} \mid \iota_{2}\,(x_{2}{}^{+}).\, c_{\rightsquigarrow+}^{2}] \\
 & \mid & \tilde{\mu}\{x^{-}\}.\, c_{\rightsquigarrow+} \\
 & & \\
s_{-\rightsquigarrow+} & ::= & \\
 & \mid & v_{+} \bullet s_{-\rightsquigarrow+} \\
 & \mid & \pi_{1} \bullet s_{-\rightsquigarrow+} \mid \pi_{2} \bullet s_{-\rightsquigarrow+} \\
 & \mid & \{s_{+\rightsquigarrow+}\} \\
 & & \\
e_{-\rightsquigarrow+} & ::= & \tilde{\mu}x^{-}.\, c_{\rightsquigarrow+} \\
 & & \\
c_{\rightsquigarrow+} & ::= & \langle t_{-} \parallel e_{-\rightsquigarrow+}\rangle^{-} \mid \langle t_{+} \parallel e_{+\rightsquigarrow+}\rangle^{+}
\end{array}$$

$$\begin{array}{rcl}
s_{+\rightsquigarrow-}, e_{+\rightsquigarrow-} & ::= & \tilde{\mu}x^{+}.\, c_{\rightsquigarrow-} \\
 & \mid & \tilde{\mu}[(x^{+}, y^{+}).\, c_{\rightsquigarrow-}] \\
 & \mid & \tilde{\mu}[\,\iota_{1}\,(x_{1}{}^{+}).\, c_{\rightsquigarrow-}^{1} \mid \iota_{2}\,(x_{2}{}^{+}).\, c_{\rightsquigarrow-}^{2}] \\
 & \mid & \tilde{\mu}\{x^{-}\}.\, c_{\rightsquigarrow-} \\
 & & \\
s_{-\rightsquigarrow-} & ::= & \star^{-} \\
 & \mid & v_{+} \bullet s_{-\rightsquigarrow-} \\
 & \mid & \pi_{1} \bullet s_{-\rightsquigarrow-} \mid \pi_{2} \bullet s_{-\rightsquigarrow-} \\
 & \mid & \{s_{+\rightsquigarrow-}\} \\
 & & \\
e_{-\rightsquigarrow-} & ::= & \tilde{\mu}x^{-}.\, c_{\rightsquigarrow-} \\
 & & \\
c_{\rightsquigarrow-} & ::= & \langle t_{-} \parallel e_{-\rightsquigarrow-}\rangle^{-} \mid \langle t_{-} \parallel e_{-\rightsquigarrow-}\rangle^{-}
\end{array}$$

(b) The $\mathrm{L}_{\mathrm{p}}^{\rightarrow\&\Uparrow\otimes\oplus\Downarrow}$ calculus

Fig. A.3. The $\lambda_{\mathrm{p}}^{\rightarrow\&\Uparrow\otimes\oplus\Downarrow}$ and $\mathrm{L}_{\mathrm{p}}^{\rightarrow\&\Uparrow\otimes\oplus\Downarrow}$ calculi