Reducing Lambda Terms with Traversals (preprint)

February 28, 2018

William Blum

Microsoft Research, One Microsoft Way, Redmond, WA, 98052, USA

Abstract

We introduce a method to evaluate untyped lambda terms by combining the theory of traversals, a term-tree traversing technique inspired from Game Semantics [12, 8], with judicious use of the eta-conversion rule of the lambda calculus.

The traversal theory of the simply-typed lambda calculus relies on the eta-long transform to ensure that when traversing an application, there is a subterm representing every possible operator's argument[8, 12]. In the untyped setting, we instead exhibit the missing operand via ad-hoc instantiation of the eta-expansion rule, which allows the traversal to proceed as if the operand existed in the original term. This gives rise to a more generic concept of traversals for lambda terms. A notable improvement, in addition to handling untyped terms, is that no preliminary transformation is required: the original unaltered lambda term is traversed.

We show that by bounding the non-determinism of the traversal rule for free variables, one can effectively compute a set of traversals characterizing the paths in the tree representation of the beta-normal form, when it exists. This yields an evaluation algorithm for untyped lambda-terms. We prove correctness by showing that traversals implement *leftmost linear reduction*, a generalization of the *head linear reduction* of Danos *et al.* [10, 9].

Contents

1. Background and overview

Traversals were originally introduced in the context of higher-order recursion schemes [12] as tree generators. They were then extended to the more general setting of simply-typed languages with higher-order free variables, such as the simply-typed lambda calculus (STLC) [8]. In the typed setting, the traversal theory has a well understood relationship with Game Semantics: traversals are in bijection with the plays of the game denotation of a term in which all the internal moves are revealed. Further, the *core projections* of traversals are in bijection with the *standard* game denotation, in which internal moves are hidden (Theorem 3.7). This correspondence gives rise to a method for reducing beta-redexes of a simply-typed lambda term that does not involve the traditional capture-avoiding substitution [10, 8, 5, 7].

In this paper we (i) generalize game-semantic traversals of the simply-typed lambda calculus[8] to the untyped lambda calculus (ULC), which we call *imaginery travserals*; (ii) infer a method to evaluate untyped lambda terms based on enumeration of their traversals; (iii) formalize the connection between traversals and the notion of *head linear reduction*.

Key idea behind imaginary traversals: In the absence of types, eta-long expansion is not feasible as it would yield an infinite term. Imaginary traversals overcome this problem by performing eta-expansion *on-demand*. Traversing an untyped lambda term using the original traversal rules can lead the traversal to get stuck. This happens when the operator sub-term of an application has an

insufficient number of operands. In such situation, there is no operand node in the term tree that the traversal can jump to. In the term $(\lambda f.f)(\lambda y.y)$, for example, an STLC traversal would get stuck when trying to resolve the "y argument" of the variable occurrence f. In contrasts, an imaginary traversal just jumps to a fictitious fresh node z that would exist if the term were eta-expanded as $\lambda z.(\lambda f.f)(\lambda y.y)z$. This trick allows the traversal to proceed.

For terms having a beta-normal form, normalization can then be implemented by traversal enumeration. Although the set of imaginary traversals can be infinite, it is sufficient to consider sufficiently enough traversals to cover all paths of the tree representation of the beta-normal form. We demonstrated how such set is effectively computable by restricting the non-determinism of the free-variable rule. In other words, only a finite number of on-demand eta-expansions is required to evaluate the term. We implemented the normalization procedure in the HOG software [4]. The last section provides some examples of terms normalized with it.

Finally, to prove correctness of the evaluation procedure, we show that traversals essentially implement a recursive version of *head linear reduction*, a non-standard reduction strategy based on an alternative notion of redex where individual variable occurrences are matched to their corresponding arguments and where substitution occurs one variable occurrence at a time.[1]

1.1. Related work

1.1.1. Game-semantics Traversal of STLC

In [8] we extended the theory of traversals to the simply-typed lambda calculus by introducing the new traversal rule (IVar) to traverse higher-order free variables. We formalized the correspondence between the theory of traversals and Game semantics by establishing a bijection between traversals and the game denotation of a term. This technical proof relies on several concepts [8]:

> **Free variables** The introduction of the traversal rule (IVar) modeling free variables of the lambda calculus. Such rule is not needed in the original presentation of traversals because higher-order recursion schemes are necessarily closed terms of ground type where terminals are all of order 1 at most[12].

> **Interaction Game Semantics** Traversals do not immediately correspond to the standard game denotation of a term. Instead they correspond to a more verbose variant of the innocent game denotation that preserves all the internal moves played while composing the sub-strategy denoting its subterms; as opposed to the standard innocent game denotation which hides all internal moves;

> **Traversal core** The correspondence makes use of various operations on justified sequences and traversals. The *projection with respect to the tree root*, also called "core of a traversal", plays a key role in the proof.

This correspondence with Game Semantics yields a method to reduce beta-redexes in eta-long simply-typed lambda-terms. This procedure was studied in [8, 5, 4, 14] and implemented in the HOG tool[5, 8]. We recall these results in Section 3.4.

The results presented in the present paper can be viewed as a generalization of this work to the untyped case. Adapting the traversal theory to the untyped setting requires new ingredients detailed in the rest of the paper. In particular:

---

[1]This is reminiscent of the standard call-by-name strategy except that linear reduction can occur under lambda, and performs substitution one variable occurrence at a time.

- Eta-expansion is performed 'on-the-fly' at each point where the traditional STLC traversal would get stuck: when there is insufficiently many operands in an application to continue traversing the term.
- We augment the rules traversing lambda and variable nodes to support ad-hoc eta-expansion.
- We introduce the concept of "ghost" nodes: those are fictitious tree nodes that progressively appear as the traversal eta-expands the sub-terms.
- The term tree itself is not altered during eta-expansion. Ghost nodes only fictitiously appear in the traversal and are only visible in the context where they get introduced.
- We constrain the non-determinism of the rule traversing free variables by a critical quantity called the *arity threshold*, calculated in linear time from the traversal itself. This limit guarantees that the normalization procedure terminates when the beta-normal form exists.

Conveniently, when restricted to simply-typed terms, the two notion of traversal coincide (Proposition 3.5).

The absence of η-long transform of the term is a notable difference with the traversals for recursion scheme [12] or the simply-typed lambda calculus [8]. It simplifies the presentation as the traversed tree is a straight abstract syntax representation of the original, unmodified, lambda term. Similarly, the evaluation procedure from Algorithm C produces the traditional beta-normal form rather than its beta-*eta-long* normal form. Furthermore, we prove soundness by establishing a connection with head-linear reduction rather than appealing to Game Semantics.

1.1.2. Head linear reduction

Danos *et al.* established the first connection between Game Semantics and the concept of *head linear reduction*[10, 9]. Despite this connection, their work does not explicitly describe how to use head linear reduction to fully evaluate a lambda term. The *pointer abstract machine* introduced in the same paper, for instance, yields the "*quasi-head normal form (qhn)*" of the term, not the normal form itself. The resulting *qhn* still needs to be first reduced to head normal form (using head reduction) and then normalized using standard normal reduction, which must terminate by the λ-calculus theory.

The traversals discussed in this paper, just like the traversals of Berezun-Jones [2], essentially implement a recursive variation of the head linear reduction strategy on the lambda term to find its normal form. A traversal starts with a depth-first search for what Danos-Reigner call the *head occurrence* of the *hoc redex*. Traversals implement linear substitution by "jumping" to the node in the tree that represents the term to be substituted for the head occurrence. After a jump, a traversal just continues its descent for the next head occurrence, as if the first head occurrence in the tree had been replaced by the redex argument. A property of head-linear reduction is that all the terms appearing in the reduction sequence consist of sub-terms of the original term. This property means that there is no need for the traditional notions of environments or closures to reduce beta-redexes. Such mechanisms are instead replaced by a system of pointers referencing subterms of the original term.

In this paper, we generalize head linear reduction to *leftmost linear reduction*, a strategy that recursively applies head linear reduction; just like normal reduction from the lambda calculus literature which proceeds by recursive application of head reduction. We show how the *imaginary traversals* effectively implement leftmost linear reduction, which proves soundness of the traversal normalization procedure.

Berezun and Jones introduced the first notion of traversals for the *untyped* lambda calculus (ULC) [2]. Although they took inspiration from the Game Semantic traversals of [12, 5], their definition is more operational in nature and does not directly relate to the game denotation of a term. Starting from an operational semantics of the untyped lambda calculus they derive a normalization method that, very much like the traversal of [12, 8], proceeds by traversing some tree representation of the term.

The tree representation used in Berezun-Jones' traversals is a direct abstract syntax tree representation of the lambda-term, whereas in the present paper we use a more compressed form where consecutive lambdas and applications are merged into a single bulk node. Unlike the original definition of traversals for STLC, where eta-long transformation is performed prior to traversing a term, neither Berezun-Jones' traversals nor imaginary traversals require any prior syntactic transformation of the input term.

The Berezun-Jones traversals distinctively rely on two justification pointers: each 'token' of the traversal can have one *binding pointer* as well as one *control pointer*. They also involve the use of a 'flag' boolean parameter associated with every token of the traversal. In contrast to Berezun-Jones' traversals, imaginary traversals necessitate a single justification pointer per node occurrence.

Another notable difference is that the normalization algorithm of Berezun-Jones requires a single traversal of the term. Our normalization algorithm, on the other hand, produces one traversal for every branch in the term tree f the beta-normal form. Each branching point corresponds to some occurrence of a variable in the normal form, and each branch corresponds to one operand of the variable. This non-deterministic branching may conceivably be eliminated by adding auxiliary pointers to allow imaginary traversals to backtrack: after the first operand of an application is evaluated, the traversal would return to the operator variable and start exploring the remaining operands.

1.2. Rest of the paper

In the remaining of this paper we introduce basic definition from the traversal theory and introduce the notion of *imaginary traversals* for the untyped lambda calculus. We briefly discuss the correspondence with game models of the Untyped Lambda Calculus [11] and show how they yield an algorithm to evaluate untyped lambda terms. We illustrate the evaluation algorithm on some examples, and finally prove correctness by establishing a correspondence with head-linear reduction.

2. Terms, trees and justified sequences

2.1. Notations

Sequences: For any alphabet Σ we write Σ^* to denote the set of all (possibly infinite) sequences over elements of Σ. We use the abbreviated syntax $\overline{x}$ for the sequence of elements $x_1 \ldots x_n$ for some $n \geq 0$.

Labelled trees: Given a set of labels $\mathcal{L}$, we use expressions of the form '$l\langle t_1, \ldots, t_m \rangle$' for $m \geq 0$ to denote a $\mathcal{L}$-labelled tree with root labelled $l \in \mathcal{L}$. If $m = 0$ then the tree is a leaf otherwise it's a node labelled l with m ordered children trees t_1, $\ldots$, t_m. We will also use the formal definition of a labelled-tree given by a partial function $\mathbb{N}^* \to \mathcal{L}$ mapping directed tree-paths to labels in $\mathcal{L}$, where a directed path is given by the sequence of child indices to follow to reach a given node from the tree root (represented by the empty sequence).

2.2. Untyped lambda calculus

We consider the set of terms Λ of the untyped lambda calculus constructed from the grammar $\Lambda := x \mid (\Lambda\ \Lambda) \mid \lambda x.\Lambda$ where x ranges over a countable set of variable names $\mathcal{V}$. For conciseness we abbreviate consecutive lambda abstractions when writing lambda terms, so that $\lambda x_1 \ldots \lambda x_n.U$ for some $n \geq 0$ and term U will be written $\lambda x_1 \ldots x_n.U$; or just $\lambda \overline{x}.U$ if $\overline{x} = x_1 \ldots x_n$. We write $M \equiv N$ to denote that M and N are equal up to variable renaming.

In the remaining of the document we use uppercase letters M, N, T, U, V for lambda terms and lowercase x, y, z to denote variables. Terms will all have variable names ranging in some fixed set $\mathcal{V}$ of identifiers.

2.3. Computation tree and enabling relation

Given an untyped lambda term M we define its **computation tree** as the abstract syntax tree representation of the term where (i) consecutive lambda abstractions are merged into a single 'bulk' node labelled by the list of bound variables and have a single child; (ii) consecutive applications are represented by a single node labelled @ whose first child represents the operator, and subsequent children represents the operands. This definition is similar to that of STLC [12, 8] with the notable difference that we do not eta-long expand the term prior to constructing the computation tree.

Definition 2.1 (Computation tree). Let M be an untyped lambda term with variable names in $\mathcal{V}$.

- The set of labels $\mathcal{L} = \{@\} \cup \mathcal{V} \cup \{\lambda x_1 \ldots x_n \mid x_1, \ldots, x_n \in \mathcal{V}, n \in \mathbb{N}\}$

- The **computation tree** $CT(M)$ is a $\mathcal{L}$-labelled tree defined inductively on the syntax of M:

$$
\begin{aligned}
CT(\lambda \overline{x}.z s_1 \ldots s_m) &= \lambda \overline{x} \,\langle\, z \,\langle CT(s_1), \ldots, CT(s_m) \rangle \rangle \\
&\quad \text{where } m \geq 0,\ z \in \mathcal{V}, \\
CT(\lambda \overline{x}.(\lambda y.t) s_1 \ldots s_m) &= \lambda \overline{x} \,\langle\, @ \,\langle CT(\lambda y.t), CT(s_1), \ldots, CT(s_m) \rangle \rangle \\
&\quad \text{where } m \geq 1,\ y \in \mathcal{V}.
\end{aligned}
$$

- We write $\mathcal{N}(M)$ to denote the set of nodes of $CT(M)$, or just $\mathcal{N}$ if the term is clear from context; $\mathcal{N}_{\mathsf{var}}$ for the set of variable nodes; $\mathcal{N}_\lambda$ for lambda nodes, and $\mathcal{N}_@$ for application nodes.

- For any lambda node $\alpha \in \mathcal{N}_\lambda$ we write $\mathsf{Ch}(\alpha)$ to denote its unique child node (either an @ or variable node).

Note that any lambda term can indeed be written in one of the two forms above. In particular, applicative terms are handled by the case $n = 0$ of the form $\lambda.N$ for some term N where 'λ' is referred to as a "dummy lambda". This compact tree representation helps maintain alternation between lambda nodes (at odd level, counting from 1 onwards) and variable nodes (at even level).

We define the **enabling relation** $\vdash$ on tree nodes $\mathcal{N}$ as the relation mapping (i) each lambda node to all the variable nodes that it binds (ii) the root of the tree to all the free variable nodes (iii) variable and application node to each one of their child node. We write $m \vdash_k \alpha$ to indicate that a lambda node α is the kth child of a variable or application node m for some $k \geq 0$; and $\alpha \vdash_k m$ to indicate that variable node m is labelled by the kth variable bound by α for $k \geq 1$.

We define **hereditarily enabling** as the reflexive transitive closure of the enabling relation, and we say that node m **hereditarily enables** n written $m \vdash^* n$.

A tree node is **external** if it is hereditarily enabled by the root, otherwise it is **internal**. We write $\mathcal{N}^{\mathsf{ext}}$ for the set of external nodes, that is the image of the tree root by the reflexive transitive closure of the enabling relation. Internal nodes are therefore precisely the nodes hereditarily justified by application nodes @. We write $\mathcal{N}^{\mathsf{int}}$ to denote the set of internal nodes.

We define the **arity** of a node n, written $|n|$ as follows: the arity of a lambda node $\lambda x_1 \cdots x_k$ for $k \geq 0$ is defined as k; the arity of a variable node x is given by the number of its children in the computation tree; the arity of an @-node is the number of its children nodes minus 1 (*i.e.*, the number of operands in the application).

2.4. Ghost nodes

The nodes in $\mathcal{N}$ correspond to tokens in the syntax representation of the lambda term; we call them **structural nodes**. We complement the computation tree with two additional infinite sets of nodes representing fictitious term tokens: (i) the set θ of *ghost variable nodes*, (ii) the set $\lambda\!\!\lambda$ of *ghost lambda nodes*. Together, nodes in $\lambda\!\!\lambda$ and θ are called the **ghost nodes**. We write $\tilde{\mathcal{N}}$ for the extended set of nodes $\mathcal{N} + \theta + \lambda\!\!\lambda$; $\mathcal{N}_\lambda^{\lambda\!\!\lambda}$ as a shorthand for $\mathcal{N}_\lambda + \lambda\!\!\lambda$ and $\mathcal{N}_{\mathsf{var}}^\theta$ as a shorthand for $\mathcal{N}_{\mathsf{var}} + \theta$. Ghost lambda nodes and ghost variables all have the same unique label $\lambda\!\!\lambda$ and θ respectively. We will thus not distinguish between them and will just use the identifiers θ and $\lambda\!\!\lambda$ as placeholders for some ghost variable node or ghost lambda node respectively.

Formally, we simultaneously define ghost nodes and the extension of the enabling relation $\vdash$ to ghost nodes as follows: (i) for every (ghost) variable or application node m and for all $k > |m|$ there is a ghost lambda node $\lambda\!\!\lambda$ such that $m \vdash_k \lambda\!\!\lambda$; (ii) for every (ghost) lambda node α and $k > |\alpha|$ there is a ghost variable node θ such that $\alpha \vdash_k \theta$. Thus, variable and ghost lambda nodes are uniquely defined by their enabler node (possibly themselves ghost nodes) and associated label $k \geq 1$. The sets θ and $\lambda\!\!\lambda$ are therefore uniquely determined by the pair $(\mathcal{N}, \vdash)$.

By convention ghost variables and lambda nodes are assigned arity 0.

2.5. Justified sequence of nodes

A **justified sequence** is defined a sequence of (extended) nodes from the computation tree where every occurrence n in the sequence, except the first one, has an associated link–the "justi-fication pointer"–pointing to some previous node occurrence j in the sequence–its "justifier"–with an associated "link label" $l \geq 0$, such that the justifier node $\vdash$-enables the source node with the corresponding label. That is: $j \vdash_l n$. We represent justified sequences as a sequence of labelled node occurrences with back-pointing arrows representing justification pointers: $s = \cdots \overset{\overset{l}{\frown}}{j} \ldots n$. For readability, we sometime just indicate the link label in exponent of the justified occurrence, or even omit the justification pointers altogether. Note that the enabling relation is *statically* induced by the structure of the tree whereas the justification relation is defined on node *occurrences* in one specific justified sequence. The set $\mathcal{J}(M)$ of justified sequences over M consists of all justified sequences over the extended set of nodes $\tilde{\mathcal{N}}$.

Name-free representation: We define the **structure** of a justified sequence as the sequence obtained by discarding node labels while preserving node types. It is encoded as a sequence of triples $\{\lambda, \mathsf{Var}, @\} \times \mathbb{N} \times \mathbb{N}$ where the first component indicates the type of each occurrence (lambda bunch, variable, application); the second component indicates the link distance, and the last component gives the link label (0 for no pointer). Two sequences s and u over possibly two distinct terms are **equivalent**, written $s \equiv u$ just if they have the same structure: the types of the underlying nodes in the two sequences match pairwise (either two variable/@ nodes or two lambda nodes) and have the

same justification pointers. Two equivalent sequences are not necessarily equal since they may have different variable names. We say that two subsets $J_1 \subseteq \mathcal{J}(M_1)$ and $J_2 \subseteq \mathcal{J}(M_2)$ are isomorphic, written $J_1 \cong J_2$, if there exists a structure-preserving bijection between J_1 and J_2. That is there exists a bijection $\phi : J_1 \longrightarrow J_2$ such that for any $j \in J$, $j \equiv \phi(j)$.

Quadruplet sequence: We define the set of **quadruplet sequences** over variables in $\mathcal{V}$ as O_4^* where $O_4 = \mathbb{N}^* \times \mathcal{V}^* \times \mathbb{N} \times \mathbb{N}$. The first component represents a node path; the second is a sequence of variable names representing *pending lambdas*; the third represents a link distance and the last is a link label.

The 'pending lambdas' component represents a sequence of variable names meant to be appended to the bulk of variables already bound in a lambda node. Although not used when encoding raw justified sequences of M, it will be useful when defining transformation on justified sequences that involve modifications to the lambda abstractions.

Definition 2.2 (Quadruplet encoding). We encode a justified sequence of nodes over a term M as a sequence of quadruplets in O_4^* obtained by applying the following mapping occurrence-wise:
1. (Node) The first component is the path to an extended node of the computation tree that uniquely defines the type and label of the node occurrence.
2. (Pending lambdas) The second component is set to the empty list ϵ.
3. (Link distance) The third component gives the distance (number of occurrences in the sequence) between the encoded occurrence and its justifier.
4. (Link label) The fourth component gives the justification link label.

We adopt the quadruplet encoding throughout the rest of the paper and thus assume $\mathcal{J}(M)$ to be a (strict) subset of O_4^*.

When typesetting a sequence from $\mathcal{J}(M)$, we represent occurrences by their associated node labels, and we write the *pending lambdas* in exponent. For example: $t = \cdots @ \cdot x \cdot \lambda \overline{x}^{[y_1 \ldots y_n]}$ where $\lambda \overline{x}$ is the label of the tree node associated with that occurrence, and $y_1 \ldots y_n$ represent *pending lambdas* for this occurrence. We omit the exponent if the list of pending lambdas is empty.

Example 2.1. Let $M = \lambda x.(\lambda yz.z)u$. The set of structural paths of the computation tree of M

is given by $\Sigma = \{\epsilon, 0, 00, 000, 01, 010\}$. The justified sequence $t = \lambda x \cdot @ \cdot \lambda yz \cdot z \cdot \lambda\!\lambda \cdot \theta$ has quadruplet encoding $(\epsilon, \epsilon, 0, 0), (0, @, 0, 0), (00, \epsilon, 1, 0), (000, \epsilon, 1, 2), (02, \epsilon, 3, 2), (02, \epsilon, 5, 2)$ whereas the quadruplets $(\epsilon, g, 0, 0), (0, @, 0, 0), (00, h\ k, 1, 0), (000, \epsilon, 1, 2), (02, \epsilon, 3, 2), (02, \epsilon, 5, 2)$ encode the jus-

tified sequence $\lambda x^{[g]} \cdot @ \cdot \lambda yz^{[h\ k]} \cdot z \cdot \lambda\!\lambda \cdot \theta$.

Two justified sequences are considered **equal** if they have the same structure and same variable names after appending each pending lambda list to the bound variables in the corresponding lambdas. Formally, given two terms M and N we define the relation $Eq_{M,N} \subseteq O_4(N) \times O_4(M)$ as:

$$(n, \overline{p}, d, k) \ Eq_{M,N} \ (n', \overline{p'}, d', k') \iff \begin{cases} n \text{ and } n' \text{ have the same type} \\ d = d' \\ k = k' \\ \text{If } n = \lambda \overline{x} \text{ and } n' = \lambda \overline{y} \text{ then } \overline{x} \cdot \overline{p} = \overline{y} \cdot \overline{p'}. \end{cases}$$

The reflexive closure of $Eq^*_{M,N}$ defines an equivalence relation on the disjoint union $O_4(M)+O_4(N)$. By extension the relation $Eq^*_{M,N}$ defines an equivalence relation on the disjoint union $\mathcal{J}(M)+\mathcal{J}(N)$ where two sequences are in relation if their element-wise occurrences are.

In the remaining of the article we will consider occurrences and justified sequences equal up to this equivalence relation. For all variable names $\overline{x}$ and $\overline{y}$ the two notations $\lambda\overline{x}^{[\overline{y}]}$ and $\lambda\overline{xy}$ will denote the same occurrence and will be used interchangeably.

We call a justified sequence **canonical** just if all the *pending lambda* components are empty. Observe that given a fixed term M, the encoding of a canonical justified sequence is uniquely determined by its structure and reciprocally: discarding the label information yields the structure; reciprocally the labels can be uniquely reconstructed from the structure of the sequence and the tree enabling relation (a chain of justification pointers in the sequence corresponds to a path in the tree).

Example 2.2. We have $\lambda\overleftarrow{xy} \cdot x \equiv \lambda\overleftarrow{zy} \cdot z$. But we also have $\lambda x\overleftarrow{yz}\overparen{wt} \cdot x \equiv \lambda\overleftarrow{x} \cdot x$ since they have same structure $(\lambda, 0) \cdot (\mathsf{Var}, 1)$.

We will make use of non-canonical justified sequences when studying justified sequences in the context of β-reduction. In particular the *pending lambda* component will allow us to define two useful operations:

- Lambda merging: where abstractions are merged in a combined bunch lambda node;

- Node materialization: where a ghost node materializes into an occurrence of a bound or free variable. This occurs when a ghost variable points to a lambda node $\lambda\overline{x}^{[\overline{y}]}$ with label i strictly greater than the arity of $|x|$ but smaller than $|x| + |y|$.

A justified sequence verifies the **alternation condition** if the first node is a lambda node and subsequent nodes occurrences alternate between (i) a variable or application node (ii) a lambda node.

We say that an occurrence of a node in a justified sequence is **hereditarily justified by some other occurrence** if recursively following justification pointers starting from the first occurrence in the sequence leads to the second one. Because justification pointers must honor the enabling relation $\vdash$ induced by the term structure, if a node occurrence n is hereditarily justified by some occurrence of a node $m \in \mathcal{N}$ then n is necessarily hereditarily enabled by m. Further if m occurs only once in the justified sequence then the occurrences hereditarily justified by m are precisely the occurrences of nodes that are hereditarily *enabled* by m.

For any justified sequence t we write t^ω to denote the last occurrence in t. The notion of sequence prefix naturally extends to justified sequences. For any occurrence n in t we write $t_{\leq n}$ for the prefix subsequence of t ending at n, and $t_{<n}$ for the prefix ending at the occurrence immediately preceding n (or the empty sequence if n is the first occurrence in t). We say that t is an **extension** of justified sequence u if u is a strict prefix of t sharing the same justification pointers.

We use the standard operations borrowed from Game Semantics on justified sequences [1].

Definition 2.3 (Projection)**.** Let s be a justified sequence of nodes and n be an occurrence in s.

- The **projection of** s with respect to n, written $s \upharpoonright n$, is the subsequence of s obtained by keeping only nodes that are hereditarily justified by n in s;

- Let A be a subset of nodes in $\mathcal{N}$, we write $s \upharpoonright A$ to denote the subsequence of s obtained by keeping only occurrences of nodes that belong to A;

- We will consider the subsequence $s \upharpoonright \mathcal{N}^{\text{ext}}$ consisting of external nodes of s. Observe that if $r_1, \ldots r_n$, $n \geq 1$ are the occurrences of the root in s then it is also given by the projection of s with respect to those occurrences: $s \upharpoonright \mathcal{N}^{\text{ext}} = s \upharpoonright r_1 \upharpoonright \ldots \upharpoonright r_n$.

The P-view of a justified sequence is the sub-sequence obtained by reading the sequence backwards and following the justification pointer every other node: (i) if the last node read is a variable node then follow its justification pointer (*i.e.*, skip all the occurrences lying underneath) (ii) if the node is a lambda node then move to the preceding node. Formally:

Definition 2.4 (Views). The P-view $\ulcorner s \urcorner$ of a justified sequence s is defined recursively by:

$$
\begin{aligned}
\ulcorner \epsilon \urcorner &= \epsilon \\
\ulcorner s \cdot n \urcorner &= \ulcorner s \urcorner \cdot n && \text{if } n \text{ is a variable or @ node;} \\
\ulcorner s \cdot \overset{\frown}{m \cdots n} \urcorner &= \ulcorner s \urcorner \cdot \overset{\frown}{m \cdot n} && \text{if } n \text{ is a lambda node;} \\
\ulcorner s \cdot n \urcorner &= n && \text{if } n \text{ is a lambda node with no pointer.}
\end{aligned}
$$

The O-view, denoted $\llcorner s \lrcorner$, is the defined dually:

$$
\begin{aligned}
\llcorner \epsilon \lrcorner &= \epsilon \\
\llcorner s \cdot n \lrcorner &= \llcorner s \lrcorner \cdot n && \text{if } n \text{ is an } \lambda\text{-node;} \\
\llcorner s \cdot \overset{\frown}{m \cdots \cdots n} \lrcorner &= \llcorner s \lrcorner \cdot \overset{\frown}{m \cdot n} && \text{if } n \text{ is a variable node;} \\
\llcorner s \cdot n \lrcorner &= n && \text{if } n \text{ is an @ node.}
\end{aligned}
$$

Given two node occurrences occurrence n and m in a justified sequence s, we say that n is **visible at** m just if n occurs in the P-view $\ulcorner s_{\leq m} \urcorner$.

2.6. Justified paths of the term tree

We will consider paths of a term tree as a set of justified sequences. For any term M we define the **set of justified paths** $\mathcal{P}aths(M)$ as the set of justified sequences in $\mathcal{J}(M)$ whose underlying sequences of nodes are the structural paths (*i.e.*, path without ghosts) in the labelled-tree $CT(M)$ with associated justification pointers induced by the enabling relation as follows. Occurrence of bound variable nodes are justified by their binder node with link label determined by the variable index in the binding node; free variables are justified by the root (the first node in the sequence) with label index determined by the free variable index; and lambda nodes are justified by their parent (the immediate predecessor in the sequence) with label index given by the child index. This notion is well defined because a variable binder necessarily occurs in the path from it to the root.

Property 2.1 (Path characterization). *(i) An untyped term M is uniquely determined by the subset of maximal justified paths of $\mathcal{P}aths$. (ii) Further M is uniquely determined, up to α-conversion, by the* structure *(i.e., node types and pointers) of the maximal justified paths in $\mathcal{P}aths$.*

Proof. (i) Computation trees are in one-to-one correspondence with the standard tree representation of a lambda term. Further, because $\mathcal{P}aths(M)$ is prefix-closed it is uniquely determined by its maximal elements. (ii) Variables names are uniquely determined by the justification pointers and their associated label index. $\qed$

Example 2.3. $\mathcal{P}aths((\lambda x.xy)(\lambda z.z))$ is the prefix closure of $\{\ \lambda \overset{0}{@} \overset{1}{\lambda x} \overset{1}{x} \lambda\, y,\ \lambda @ \overset{1}{\lambda z}\, \overset{1}{z}\ \}$.

10

Algorithm A shows how to reconstruct the tree from a set of justified sequences that is isomorphic to the justified paths of the term. Note how ghost variables get materialized by assigning real variable names. Observe as well that variable nodes with same binder and same link label are assigned the same variable name. This assignment is sound since by Property 2.1 justification pointers in justified paths uniquely determine variables names.

Definition 2.5 (Induced labelled-tree). Let $\mathcal{P}$ be a set of justified sequences of M that is isomorphic to justified paths $\mathcal{P}aths(M_2)$ of some other term M_2. Then $\mathcal{P}$ induces the labelled-tree (P, L) defined as follows:

- Directed paths $P \subseteq \mathbb{N}^*$ are *generated* from justified sequences in $\mathcal{P}$: variable occurrences have child index 1; the child index of lambda-node occurrences is given by their justification label.
- Labels are $\mathcal{L} = \tilde{\mathcal{N}} \times P \times \mathbb{N}$.
- Label function $L : P \to \mathcal{L}$. For $p \neq \epsilon$, let $s = s' \cdot n$ be any sequence in $\mathcal{P}$ generating path p. If n has not justifier then $L(p) = (n, p, 0)$. Otherwise $L(p) = (n, b, i)$ where b is the prefix of p corresponding to n's justifier and i is the label of n's justification pointer.

It is well-defined because sequences in $\mathcal{P}$ correspond to actual paths in some other computation tree, therefore for any two sequences s_1, s_2 generating a given path p, their last occurrence must have be the same node and have the same justifier.

Algorithm A Reconstructing a term tree from justified paths.

Input: A set $\mathcal{P}$ of justified sequences of a term M verifying $\mathcal{P} \equiv \mathcal{P}aths(N)$ for some other term N.

Output: Computation tree of N modulo α-conversion.

  Let (P, L) be the labelled-tree induced by $\mathcal{P}$.

  for all node $n \in P$ (by depth-first enumeration) **do**

    Let $(l, b, i) = L(n)$

    if $n \in \mathcal{N}_{\mathsf{var}} \cup \mathcal{N}_\lambda \cup \mathcal{N}_@$ **then**

      $L'(n) \leftarrow l$

    else if $n \in \lambda\!\!\lambda$ **then**

      Let α be a fresh identifier {Will be used as prefix to name variable bound by n}

      $id(n) \leftarrow \alpha$

      $arity \leftarrow \max\{i \mid L(v) = (\theta, n, i) \text{ for some } v \in P\}$

      $L'(n) \leftarrow \lambda\alpha_1 \ldots \alpha_{arity}$

    else if $n \in \theta$ **then**

      $\alpha \leftarrow id(b)$ {Get naming prefix associated with this binder}

      $L'(n) \leftarrow \alpha_i$ {Variable occurrences with same binder and justification label have same name.}

    end if

  end for

  return The labelled-tree (P, L').

---

3. Imaginary Traversals

3.1. Definition and properties

Definition 3.1 (ULC traversals). The set of **traversals** of an untyped lambda term M, denoted $\mathcal{T}rav(M)$, abbreviated $\mathcal{T}rav$ when the term is clear from context, is the set of justified sequences of nodes over M recursively defined by the rules of Table 1. A traversal that does not have any extension is a **maximal traversal**.

Some immediate properties that can be shown by induction on the rules:

Property 3.1. *For any traversal t*
1. *t verifies the alternation condition;*
2. *$t \upharpoonright \mathcal{N}^{\mathsf{ext}}$ is a valid justified sequence (with respect to the enabling relation $\vdash$) verifying the alternation condition;*
3. *If the last node in t is external then all the nodes in $\llcorner t \lrcorner$ are also external.*

On-the-fly expansion. The eta-expansion of M with respect to a traversal is a term obtained by eta-expanding some of the subterms of M according to the rules used to produce the traversal.

Definition 3.2 (On-the-fly eta-expansion). Let M be an untyped term.
- Let $n \in \mathcal{N}$ be a node and N denote the subterm of M rooted at n. We write $\mathsf{ETA}(M, n)$ to denote the term obtained by substituting the subterm N in M with the term $\lambda\theta.N\theta$ for some variable θ fresh in N.
- Let t be a *finite* traversal of M. The **eta-expansion of M with respect to** t, written M^t, is defined as just M if t is the empty traversal; and we define $M^{u \cdot n}$ inductively by case analysis on the rule used to traverse node n, where m denotes n's justifier in u, if it exists, with link label i:

$$
\begin{array}{llll}
(\mathsf{Root}) & M^u & & \\
(\mathsf{App}) & M^u & & \\
(\mathsf{Lam}) & M^u & & \\
(\mathsf{Lam}^{\lambda\!\!\lambda}) & M^u & &
\end{array}
\qquad
\begin{array}{llll}
(\mathsf{Var}) & M^u & \text{if } i \leq |m|; \\
 & \mathsf{ETA}(M^u, m) & \text{if } i > |m|. \\
(\mathsf{IVar}) & M^u & \text{if } i \leq |m|; \\
 & \mathsf{ETA}(M^u, m) & \text{if } i > |m|.
\end{array}
$$

Observe that the tree obtained after eta-expansion with respect to t contains the tree of M itself, therefore paths in the tree of M are also paths in the tree of M^t.

Example 3.1. Take $M = (\lambda u.u\ (y_1\ u))(\lambda v.v\ y_2)$. Its computation tree is shown in Ex. 5.3.

A valid traversal is $t = \lambda\ @\ \lambda u\ u\ \lambda v\ v\ \lambda\ y_1\ \lambda\ u\ \lambda v\ v\ \lambda\!\!\lambda^1\ \theta^1\ \lambda\!\!\lambda^1\ \theta^1\ \lambda\ y_2$. The eta-expansion of M with respect to t is $M^t = (\lambda u.u\ (y_1\ (\lambda\alpha.u(\lambda\beta.\alpha\beta))))(\lambda v.v\ y_2)$.

Path-View correspondence. We now generalize a known result from the theory of traversals for higher-order grammars [12] and simply-typed lambda terms [8, Proposition 4.29] to the untyped setting. The game-semantic concept of 'Proponent view' from definition 2.4 corresponds to the concept of 'tree path' in the following sense:

Imaginary traversals of λ-calculus

PROGRAM – Structural rules

(Root) The singleton sequence r is in $\mathcal{T}rav$ where r is the root of the tree.

(App) If $t \cdot @$ is a traversal then so is $t \cdot \overset{\frown{0}}{@ \cdot \alpha}$ where $\alpha \in \mathcal{N}_\lambda$ is @'s 0th child λ-node.

(Lam) If $t \cdot \alpha$ is a traversal where $\alpha \in \mathcal{N}_\lambda$ then so is $t \cdot \alpha \cdot n$ where n denotes α's unique child. Furthermore:

    (Lam$^@$) If n is an @-node then it has no justifier,

    (Lam$^{\mathsf{var}}$) If n is a free variable node then it points to the only occurrence of the root in $\ulcorner t \cdot \alpha \urcorner$. Otherwise it points to the only occurrence of its binder in $\ulcorner t \cdot \alpha \urcorner$.

(Lam$^{\lambda\!\!\lambda}$) If $t \cdot \alpha \cdot \overset{\frown{i}}{n \cdot \ldots \cdot \lambda\!\!\lambda} \in \mathcal{T}rav$ for some prefix t, $\alpha \in \mathcal{N}_\lambda^{\lambda\!\!\lambda}$ and $n \in \mathcal{N}_{\mathsf{var}}^\theta$ then

$$t \cdot \overset{\overset{\textstyle |\alpha| + i - |n|}{\frown{i}}}{\alpha \cdot n \cdot \ldots \cdot \lambda\!\!\lambda} \cdot \theta \in \mathcal{T}rav$$

PROGRAM – Copy-cat rules

(Var) If $t \cdot m \cdot \overset{\frown{i}}{\alpha \ldots n} \in \mathcal{T}rav$ for $i > 0$, $n \in \mathcal{N}_{\mathsf{var}}^\theta$ hereditarily justified by an @-node; $m \in \mathcal{N}_{\mathsf{var}}^\theta \cup \mathcal{N}_@$; and $\alpha \in \mathcal{N}_\lambda^{\lambda\!\!\lambda}$ then:

$$t \cdot \overset{\overset{\textstyle i}{\frown{i}}}{m \cdot \alpha \ldots n} \cdot \beta \in \mathcal{T}rav$$

    Concrete $i \le |m|$: $\beta \in \mathcal{N}_\lambda$ is the ith child of m;

    Eta-expanded $i > |m|$: β is a ghost lambda node $\lambda\!\!\lambda$.

DATA – Input-variable rules

(IVar) If $t \cdot n$ is a traversal where $n \in \mathcal{N}_{\mathsf{var}}^\theta$ is hereditarily justified by the root. For every node $m \in \mathcal{N}_{\mathsf{var}}^\theta$ occurring in $\llcorner t \cdot n \lrcorner$ and every $i \ge 1$ we have $t \cdot n \cdot \alpha \in \mathcal{T}rav$ with α pointing to m with label i, where:

    Concrete $i \le |m|$: $\alpha \in \mathcal{N}_\lambda$ is the ith child of m;

    Eta-expanded $i > |m|$: α is a ghost lambda node $\lambda\!\!\lambda$.

Table 1: Imaginary traversals $\mathcal{T}rav$ of the untyped lambda calculus.

Proposition 3.2 (Path-View correspondence for ULC)**.** *Let M be an untyped term and t be a finite traversal of M then $\ulcorner t \urcorner$ is a path (with associated justification pointers) in the computation tree of the eta-expansion of M with respect to t. Thus, if t does not contain ghost nodes then $\ulcorner t \urcorner$ is a structural path in the computation tree of M.*

Proof. Proven by induction on the traversal rules of t. $\square$
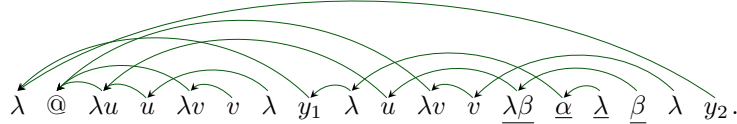
Property 3.3. *The traversal rules are well-defined.*

Proof. (i) Rule (Lam): By Proposition 3.2, the P-view is a path in the tree of the eta-expansion of M with respect to t. But the last node of t is a structural node therefore, since the eta-expanded tree contains the tree of M, this path is necessarily also the path to t^ω in the tree of M (and thus $\ulcorner t \urcorner$ does not contain any ghost node, even though t itself may contain ghost nodes). Consequently, if the last node is a variable node, its enabler necessarily occurs exactly once in the P-view.

(ii) Rule (Var): In the concrete sub-case, m is necessarily itself a structural node since $|m| \geq i > 0$, and its ith child exists in $\mathcal{N}$ since m as arity greater than i. $\square$

Proposition 3.4. *Let M be an untyped term and t be a finite traversal of M. There exists a one-to-one function $\eta_t : \tilde{\mathcal{N}}(M) \longrightarrow \tilde{\mathcal{N}}(M^t)$ such that its implicit element-wise pointer-preserving extension $\eta_t : \mathcal{J}(M) \longrightarrow \mathcal{J}(M^t)$ verifies the properties:*
  (i) For any $u \in \mathcal{T}rav(M)$, $\eta_t(u)$ is a traversal of M^t.
  (ii) If v traverses M^t then there exists a traversal u of M (with possibly ghost nodes) such that $v = \eta_t(u)$.
  (iii) The restriction of η_t to $\mathcal{T}rav(M)$ defines a strong bijection with $\mathcal{T}rav(M^t)$ (i.e., a bijection preserving the traversal structure, but not necessarily labels).

Example 3.2. Continuing with the previous example: the following traversal obtained from t is a valid traversal of M^t where the occurrences corresponding to ghosts in t are underlined:

$$\lambda \quad @ \quad \lambda u \quad u \quad \lambda v \quad v \quad \lambda \quad y_1 \quad \lambda \quad u \quad \lambda v \quad v \quad \underline{\lambda\beta} \quad \underline{\alpha} \quad \underline{\lambda} \quad \underline{\beta} \quad \lambda \quad y_2 \, .$$

Traversal core. We now generalize the notion of *traversal core* from [8] to imaginary traversals. In the typed setting, the *core* is defined as the subsequence of a traversal consisting of external nodes only. In the untyped setting, we complement this filtering with a relabelling operation on lambda nodes.

We consider sequences of variable names in $\mathcal{V}^*$ as stacks equipped with the standard stack operations: for any sequence of variables $\overline{x} = x_1 \ldots x_n$, $n, j \geq 0$, the 'pop' operation pop_j removes the first j elements of the sequence: $pop_j(x_1 \ldots x_n) = x_{j+1} \ldots x_n$ if $j < n$, and $pop_j(x_1 \ldots x_n) = \epsilon$ otherwise. And for any sequence $\overline{y} = y_1 \ldots y_m$, $m \geq 0$ we write $\overline{xy}$ for $x_1 \ldots x_n y_1 \ldots y_n$, the stack obtained after pushing the variables $\overline{x}$ (read backwards) onto $\overline{y}$.

Definition 3.3 (Core projection)**.** Let $\overline{y} \in \mathcal{V}^*$ denote a stack of variable names. We define the partial function $\pi_{\overline{y}} \colon \mathcal{J}(M) \longrightarrow \mathcal{J}(M)$ on the set of justified sequences by induction on the sequence.

Let $t \cdot n \in \mathcal{J}(M)$ denote a justified sequence for some prefix subsequence t and last occurrence n.

$$
\pi_{\overline{y}}(t \cdot n) = \begin{cases}
\pi_{\epsilon}(t) \cdot n & \text{if } n \in \mathcal{N}_{\mathsf{var}}^{\theta} \cap \mathcal{N}^{\mathsf{ext}} \\
\pi_{pop_{|n|}(\overline{y})}(t) & \text{if } n \in (\mathcal{N}_{\mathsf{var}}^{\theta} \cap \mathcal{N}^{\mathsf{int}}) \cup \mathcal{N}_{@} \\
\pi_{\epsilon}(t) \cdot \lambda\overline{xy} & \text{if } n \in \mathcal{N}_{\lambda} \cap \mathcal{N}^{\mathsf{ext}}, n \text{ labelled } \lambda\overline{x} \\
\pi_{\overline{xy}}(t) & \text{if } n \in \mathcal{N}_{\lambda} \cap \mathcal{N}^{\mathsf{int}}, n \text{ labelled } \lambda\overline{x} \\
\pi_{\epsilon}(t) \cdot \lambda\overline{y} & \text{if } n \in \lambda\!\!\lambda \cap \mathcal{N}^{\mathsf{ext}} \\
\pi_{\overline{y}}(t) & \text{if } n \in \lambda\!\!\lambda \cap \mathcal{N}^{\mathsf{int}} .
\end{cases}
$$

The above definition is pointer-preserving, so that the justifiers in $\pi(t \cdot n)$ are defined to be the occurrences corresponding to the original justifiers in $t \cdot n$.

Note that if t is a traversal, the second last case is not needed since ghost lambdas are necessarily internal by Prop 3.1.
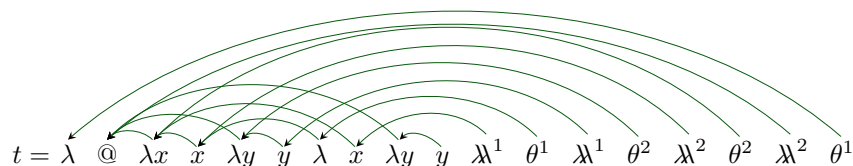
We call $\pi_{\epsilon}(t)$ the **core** of t which we abbreviate as $\pi(t)$. The P-view of the core $\ulcorner\pi(t)\urcorner$ is called the **core P-view** of t.

The suffix parameter $\overline{y}$ in the definition of π, is used as an accumulator for the list of variables forming the abstractions $\lambda\overline{y}$ that eventually gets prepended to the last external lambda nodes. We call it the **stack of pending lambdas**. Hence, *core projection* can be defined more succinctly as the sequence obtained by removing internal nodes and prepending to each external lambda node's label, the stack of *pending lambdas* between that point to the next external node.

Since the core of a justified sequence only contains external nodes it follows that π is idempotent.

Example 3.3. Take the term $\lambda w.(\lambda xy.x)z$ and consider traversal $t = \lambda w \cdot @ \cdot \lambda xy \cdot x \cdot \lambda \cdot z$. Then $\pi(t) = \lambda wy \cdot z$.

Example 3.4. Take $(\lambda x.xx)(\lambda y.y)$ with traversal t represented below. Projecting nodes hereditarily justified by the root gives $\overset{\frown}{\lambda \quad \theta^1}$ while the core projection gives $\pi(t) = \overset{\frown}{\lambda y \quad y}$.

$$t = \lambda \quad @ \quad \lambda x \quad x \quad \lambda y \quad y \quad \lambda \quad x \quad \lambda y \quad y \quad \lambda\!\!\lambda^1 \quad \theta^1 \quad \lambda\!\!\lambda^1 \quad \theta^2 \quad \lambda\!\!\lambda^2 \quad \theta^2 \quad \lambda\!\!\lambda^2 \quad \theta^1$$

3.2. Imaginary traversals subsume STLC traversals

Traversals for simply-typed languages were previously studied in [8]. The traversal rules defined in Table 1 closely match those of the simply-typed lambda calculus and PCF from [8] with some important differences:

No interpreted constants Unlike PCF, there are no interpreted constants in the present setting therefore the rule (Value) and (InputValue) from the original presentation are not needed.

No η-long expansion In the original STLC traversals, the term is eta-long expanded prior to calculating the set of traversals. This guarantees that the operand of an application always exists in the tree. Imaginary traversals, on the other hand, are defined on the unmodified tree representation of the term.

On-the-fly η-expansion In the untyped setting, eta-long expansion is an infinite process, so instead of eta-long expanding the term prior to traversing it, imaginary traversals perform eta-expansion 'on the fly'. Eta-expansion occurs in rule (Var) when the arity of a variable in operand position is too low to statically determine the operand of an application (case $i > |n|$). When the variable arity is high enough ($i \geq |n|$), the definition of the rule coincides with STLC and the static tree representation of the term dictates the next node to visit.

Free variables Eta-expansion can also occur on free variables, so like for rule (Var), the input-variable rule allows for infinitely many eta-expansions ($k > 0$).

Traversal core In the typed setting, the core is obtained by just filtering nodes with respect to the tree root. In the untyped settings, lambda nodes also get relabelled after filtering.

Traversing ghost nodes There is an additional rule ($\mathsf{Lam}^{\lambda\!\!\lambda}$) for the case where a traversal ends with a ghost lambda node. In the concrete sub-case, rule (Lam) visits the unique child node of the last lambda node in the traversal. In the eta-expanded sub-case where the last node is a ghost lambda node, there is no such child node, so we visit an imaginary one: the variable node that would be created if we were to eta-expand the sub-term under the lambda. The ghost placeholder θ thus represents an occurrence of the jth variable that would be bound by lambda node α if the sub-term at node α were eta-expanded $i - |n|$ times: hence $j = |\alpha| + i - |n|$. (Observe that in this case we necessarily have $i > |n|$ since the ith child of n is a ghost variable node.)

Let's fix a simply-typed term-in-context $\Gamma \vdash M : T$, with typed-context Γ (a set of typed variables), and simple type T. Its **_eta-long form_** is defined inductively on the type T and is obtained by recursively eta-expanding every subterm as many times as possible with respect to the type of the subterm [12, 8]. By abuse of language we say that an untyped term M is in eta-long form if it inhabits a simple type T such that it equates its own eta-long expansion with respect to T, modulo variable renaming.

STLC traversals from [8] are defined for eta-long form only. The rules consist of the same rules as the imaginary traversal from Table 1 with the exclusion of all sub-cases required to implement 'on-the-fly eta-expansion' or to traverse ghost nodes:

Definition 3.4 (STLC traversal [8])**.** Let $\Gamma \vdash M : T$ be a simply-typed term in context. The set of traversals $\mathcal{T}rav_{\mathsf{STLC}}(\Gamma \vdash M : T)$ from [8] is the set of traversals of the *eta-long form* of M generated from the rules of Table 1 with the exclusion of ($\mathsf{Lam}^{\lambda\!\!\lambda}$) and the 'eta-expanded' sub-cases of (Var) and (IVar).

ULC and STLC traversals coincide on eta-long forms in the following sense:

Proposition 3.5. *Let M be an untyped term inhabiting a simple type T for some context Γ ($\Gamma \vdash M : T$) and $\eta_{\mathsf{lf}}(M)$ denote its eta-long normal form with respect to T. Then we have*

$$\mathcal{T}rav_{\mathsf{STLC}}(\Gamma \vdash M : T) = \mathcal{T}rav'(\eta_{\mathsf{lf}}(M))$$

where $\mathcal{T}rav'$ denote the set of traversals obtained with rules of Table 1 with the exclusion of the eta-expanded case in (IVar)*.*

Proof. Because the term is eta-long, the condition $i > |m|$ in rules (Var) never holds while traversing the term. This is shown by induction on the rules, observing that the arity of an @ node is necessarily equal to the arity of its 0th child lambda node, and the arity of a variable node with binding index i is necessarily equal to the arity of the ith child of its binder's parent. And since the eta-expanded rule (IVar) is excluded, traversals in $\mathcal{T}rav'(\eta_{\mathsf{lf}}(M))$ do not contain any ghost node. Other rules coincide with STLC, therefore the equality holds. $\square$

3.3. Property of ghost nodes

It is helpful to think of ghost nodes as the counterpart of complex numbers which are sometimes used in mathematics to prove trigonometry identities: they are introduced intermediately to perform some calculation (e.g. using De Moivre's Theorem) but do not appear in the final result. Just like the imaginary number i is created out of the impossibility of calculating the square root of -1, ghost nodes are defined from the impossibility of "traversing" a beta-redex of a lambda term due to an insufficient number of operands.

Ghost nodes appear in a traversal when the arity of a node is too low to be able to structurally traverse it:

Property 3.6. *If $\overset{\overset{i}{\frown}}{x \cdots y} \in \mathcal{T}rav$ then $y \in \theta \cup \lambda\!\!\lambda \iff i > |x|$. In particular for all $n \in \mathcal{N}_{\mathsf{var}}$ and $\alpha \in \mathcal{N}_\lambda$: (i) $\overset{\overset{i}{\frown}}{n \cdots \lambda\!\!\lambda} \in \mathcal{T}rav \implies i > |n|$ (ii) $\overset{\overset{i}{\frown}}{\alpha \cdots \theta} \in \mathcal{T}rav \implies i > |n|$.*

Definition 3.5. We call **_ghost materialization_** any application of a traversal rule where the last occurrence in the traversal prior to applying the rule is a ghost node (θ or $\lambda\!\!\lambda$), and the node traversed after applying the rule is a structural node of the tree (in $\mathcal{N}$).

Remark 3.1 (Materialization rule). Observe that among all the rules defined in Table 1, the rule (Var) is the only rule that can materialize a structural node in $\mathcal{N}$ from a traversal ending with a ghost node. This means that after traversing ghost nodes, the only way to 'return' to structural nodes is to visit a ghost variable node θ with an application of rule (Var) of the following form:

$$\text{(Var)} \quad t \cdot \beta \cdot \overset{\overset{\overset{i}{\frown}}{\overset{i}{\frown}}}{y \cdot \alpha \ldots \theta} \cdot \lambda\overline{x} \in \mathcal{T}rav$$

where $y \in \mathcal{N}_{\mathsf{var}}$, $\lambda\overline{x} \in \mathcal{N}_\lambda$ is the ith child lambda node of $y \in \mathcal{N}_{\mathsf{var}}$, α is either a structural lambda node in $\mathcal{N}_\lambda$ or a ghost lambda node in $\lambda\!\!\lambda$, $0 \leq \alpha < i \leq |y|$.

3.4. Correspondence with Game semantics

In [8] we formalized the correspondence between the theory of traversals and Game Semantics in the setting of simply-typed languages: there is a bijection between the set of traversals of a term and its revealed interaction game denotation. Furthermore, the core projection yields a bijection with the standard innocent game denotation. In other words, the traversal cores are precisely plays from the game denotation of the term. Formally:

Theorem 3.7 (Traversal-Play Correspondence for STLC (Theorem 4.96 in [8])). *The following two bijections hold for every simply-typed term $\Gamma \vdash M : T$ with eta-long normal form $\eta_{\mathsf{lf}}(M)$:*

$$\begin{aligned}
\mathcal{T}rav_{\mathsf{STLC}}(M) &\cong \langle\!\langle \Gamma \vdash M : T \rangle\!\rangle \\
\mathcal{T}rav_{\mathsf{STLC}}{}^\pi(\eta_{\mathsf{lf}}(M)) &\cong [\![\Gamma \vdash M : T]\!] \ .
\end{aligned}$$

where $\langle\!\langle \Gamma \vdash M : T \rangle\!\rangle$ and $[\![\Gamma \vdash M : T]\!]$ denote respectively the revealed game denotation *and* innocent game denotation *of $\Gamma \vdash M : T$; and $travstlc^\pi$ denotes the image of $travstlc$ by π (i.e., set of justified sequences that are* core *of some STLC-traversal).*

Game semantics Correspondence for ULC. What would be the equivalent of Theorem 3.7 in the untyped case? In his thesis, Andrew Ker studied Game models for the untyped lambda calculus [11]. We conjecture that the traversal-game semantics correspondence for STLC also holds for ULC Table 1 between the traversals from Table 1 and the game model of ULC from Ker's thesis, *i.e.*there is an isomorphism between the set of imaginary traversals and the revealed game denotation of the ULC term, and further, an isomorphism between the standard game semantics and the set of traversal cores:

Conjecture 3.1 (Traversal-Play Correspondence for ULC)**.** For every untyped lambda-term M there are two bijections $\mathcal{T}rav(M) \cong \langle\!\langle M \rangle\!\rangle$ and $\mathcal{T}rav^\pi(M) \cong [\![M]\!]$ where $[\![M]\!]$ denotes the innocent *effectively almost everywhere copycat(EAC)* game denotation of M defined in [11], $\langle\!\langle M \rangle\!\rangle$ denotes the corresponding interaction game denotation where internal moves are not hidden during strategy composition, and $\mathcal{T}rav^\pi(M)$ denotes the set of traversal cores of M.

A possible proof of this conjecture might consist in a similar argument to the STLC case [8] but based on Ker's game model of the untyped lambda calculus instead of the innocent game model of STLC. There seems, for instance, to exist a possible connection between the "on-the-fly" eta-expansion of imaginary traversals and the morphism $Fun : U \to (U \Rightarrow U)$ of Ker's game category where U denotes the maximal game arena.

4. Normalizing Traversals

In this section we show how traversals can help evaluate lambda terms and find their normal form when it exists. The crux of the algorithm lies in the *Path Characterization* (Proposition 4.5 and Theorem 7.11) which states that the set of traversals *core P-views* captures what is strictly required from traversals to reconstruct the normal form of a term. For STLC, the characterization result was shown using the Game Semantics correspondence; we prove its untyped counterpart using instead a term-rewriting argument.

Path Characterization suggests a normalization method based on enumeration of all maximal traversals. Unfortunately, due to the generous non-determinism of the free variable rule, traversals can be arbitrarily long and therefore the set of traversals may be infinite, even for terms with a beta-normal from! Not all traversals need to be enumerated, fortunately. It is sufficient to enumerate a subset of traversals that covers all the possible core P-views. Some traversals are redundant in the sense that there exist shorter traversals with identical core P-view. We capture this notion by considering equivalence classes on traversals: two traversals are in the same class just if they have the same core P-view. It then suffices to exhibit a traversal subset that is *complete* for the equivalence classes, in the sense that it contains at least one traversal for each equivalence class. If such subset is effectively computable and finite then it yields a normalization procedure. We show how this can be done for both STLC and ULC.

For the simply-typed lambda calculus, we consider the subset of *branching traversals* which verifies the desired completeness property. For simply typed terms in eta-long form this subset is finite. This yields a normalization procedure for STLC. Soundness follows from the characterization theorem which is an immediate consequence of the Game Semantics correspondence of [8].

For the untyped lambda calculus, *branching traversals* are not adequate as they can still be infinite even for terms with a normal form. We introduce another kind of traversals, called *normalizing traversals* to limit the non-determinism of the traversals and derive from it a procedure to calculate β-normal forms of untyped lambda terms when they exist.

4.1. Quotienting

We introduce a quotient relation that identifies traversals with identical core P-views:

Definition 4.1 (Quotienting)**.** The core P-view function $\ulcorner\pi(\_)\urcorner\colon \mathcal{T}rav \longrightarrow \mathcal{J}$ is the composition of $\ulcorner\_\urcorner\colon \mathcal{J} \longrightarrow \mathcal{J}$ with the core projection $\pi\colon \mathcal{T}rav \longrightarrow \mathcal{J}$ that is $t \longmapsto \ulcorner\pi(t)\urcorner$. We define $\sim$ as the equivalence relation over $\mathcal{T}rav$ induced by $\ulcorner\pi(\_)\urcorner$ up to relabelling (same structure but not necessarily the same labels). Formally:

$$t \sim u \quad \text{iff} \quad \ulcorner\pi(t)\urcorner \equiv \ulcorner\pi(u)\urcorner.$$

We write $\mathcal{T}rav/\sim$ for the set of equivalence classes of $\mathcal{T}rav$. We identify a $\sim$-equivalence class with the structure of the P-view core of the traversals it contains. A subset $T \subseteq \mathcal{T}rav$ is $\sim$-**complete** if it contains at least one element for each $\sim$-equivalence class; that is if $T/\sim = \mathcal{T}rav/\sim$.

In the next sub-section, we explore *branching traversals*, a $\sim$-complete traversal subset that can be effectively computed for lambda terms inhabiting some simple type.

4.2. Branching traversals

A normalization procedure based on enumerating all traversals is not practical because the set of traversals can be infinite. This is expected for non-normalizing term such as $\Omega = (\lambda x.xx)(\lambda y.yy)$ which has infinitely long traversals of the form $\lambda x \cdot x \cdot \lambda y \cdot y \cdot x \cdot \lambda y \cdot y \cdot \ldots$. But this is also the case for terms having a normal form. Take for example $M = \lambda f.f(\lambda x.x)$, then for all $k \geq 0$ the justified sequence $t_k = \lambda f \cdot f \cdot (\lambda x \cdot x)^k$ (with appropriate pointers) is a traversal. Remember that rule (IVar) offers two non-deterministic choices when traversing a free variable:

(J) The variable to pick in the O-view (the justifier),

(L) The child lambda node to pick amongst the children of variable picked in (J).

The choice (J) alone gives rise to the pattern displayed in t_k where the node λx from the O-view is repeatedly picked *ad infinitum* within a single traversal.

Remark 4.1 (Game-semantic intuition)*.* The Game Semantics correspondence explains why traversals can be infinite: In the game denotation of a lambda term M, at every point in a play where it is Opponent's turn to play, all possible Opponent moves must be accounted for. More generally, the game denotation must allow Opponent's moves modeling the behaviour of *all* contexts that can possibly interact with M. The t_ks traversals thus accounts for all possible denotations of the function parameter f of M: for each k, there exists a term that applies its argument k times: $F_k = \lambda g.g(g(\ldots(gz)))$ with k applications of g, and there got to be plays in the game denotation of the term M to accounts for those possible values of argument f. Fortunately, traversals accounting for all those contexts are redundant for normalization: due to the absence of side-effects, calling the same argument multiple times always involves the same underlying computation in M. We formalize this intuition with the notion of *branching traversals* (Definition 4.2) which prevent such repetitive behaviour while still covering all $\sim$-equivalence classes (Proposition 4.2).

One may view traversals as a mechanism to explore the beta-normal form of the term in a dept-first search manner. Under such a view, one can interpret the non-determinism in (IVar) as a 'branching point' in the exploration. In the absence of side-effect, it is sufficient to explore each possible branch only once. *Branching traversals* implement this idea by restricting the rule (IVar) so as to traverse only nodes leading to paths in the computation tree that are yet unexplored: when choosing the next lambda node to visit, it forces choice '(J)' to be the *latest* variable node in the traversal, and restricts choice '(L)' to be some child lambda node of that variable node.

Definition 4.2. The set of **branching traversals** $\mathcal{T}rav^{\mathsf{branch}}$ is the subset of $\mathcal{T}rav$ defined by induction with the rules of Table 1 where the justifier node in the input-variable rule (IVar) is restricted to be necessarily the last node in the traversal ($m = n$). We will use the subscript '$_{\mathsf{branch}}$' to refer to this system of rules. The input variable rule can thus be stated by the following derivation rule:

$$\frac{t \cdot n \in \mathcal{T}rav^{\mathsf{branch}} \qquad n \in \mathcal{N}^{\mathsf{ext}} \cap \mathcal{N}_{\mathsf{var}} \qquad n \vdash_i \alpha \qquad i \geq 1}{t \cdot \overset{i}{\overset{\frown}{n \cdot \alpha}} \in \mathcal{T}rav^{\mathsf{branch}}} \qquad (\mathsf{IVar}_{\mathsf{branch}})$$

Remark 4.2 (Game semantic intuition). Lambda nodes correspond to opponent moves in game semantics. Restriction on the opponent moves limit the set of contexts in which a term can appear. By limiting the possible lambda nodes to traverse after a free variable, the branching restriction essentially eliminates all contexts containing a sub-expression where the same argument is referred more than once, or two arguments of the same higher-order function are called in the same function body. This for instance exclude the Kierstead contexts (in which the argument f is called twice) but also contexts like $\lambda f \lambda g. f(\lambda f(\lambda x. g(\lambda y. x)))$ where the consecutive lambdas abstractions $\lambda f \lambda g$ have in their body an occurrence of both f and g.

Property 4.1. *Let $t \in \mathcal{T}rav$ be a traversal which does not contain any ghost occurrence, and m be an occurrence in t of an external λ-node (i.e., $m \in \mathcal{N}_\lambda \cap \mathcal{N}^{\mathsf{ext}}$). Then $\pi(t_{<m}) = \pi(t)_{<m}$.*

Proof. By an easy induction on t using the fact that when recursively calculating $\pi(t)$, external lambda nodes reset the stack of pending lambdas. $\square$

Proposition 4.2 (Branching traversals are $\sim$-complete). *$\mathcal{T}rav^{\mathsf{branch}}$ is $\sim$-complete.*

Property 4.3 (Infinite branching traversals). *Let $t \in \mathcal{T}rav^{\mathsf{branch}}$ be a branching traversal. If t is infinite then it necessarily contains a ghost node occurrence.*

Proof. Suppose t does not contain any ghost node then t is obtained without using ($\mathsf{Lam}^{\lambda\lambda}_{\mathsf{branch}}$) and without the 'eta-expansion' subcases of the variable rules. The remaining rules correspond precisely to the traversal rules of [12] (where (IVar) is renamed (Sig)). We can therefore appeal to the Spinal Decomposition Lemma [12, Lemma 14] which shows that if t is infinite then there is an infinite sequence of prefixes of t whose P-views (the *spine* of t) is strictly increasing. By the Path correspondence this means that there is an infinite path in the computation tree of M which gives a contradiction. Hence t is either finite or contains a ghost node. $\square$

Property 4.4. *Let M be an untyped term in eta-long form (with respect to some simple type it inhabits). Then (i) All traversals in $\mathcal{T}rav^{\mathsf{branch}}(M)$ are finite; (ii) The set $\mathcal{T}rav^{\mathsf{branch}}(M)$ is finite.*

Proof. By Proposition 3.5, traversals in $\mathcal{T}rav^{\mathsf{branch}}(M)$ do not contain any ghost node therefore Prop. 4.3 implies (i). (ii) Traversal rules that are not involving ghost nodes all have bounded non-determinism therefore a traversal only has a finite number of immediate extensions. Since traversals are finite by (i), this implies that the set of traversals is itself finite. $\square$

4.3. Normalization procedure for eta-long forms (STLC)

 The normalization procedure for the simply-typed lambda calculus is given by Algorithm B.

Algorithm B Eta-long normalization by traversals for STLC

Input: A term M admitting an eta-long form *wrt* some simple-type T that it inhabits.
Output: The tree of the eta-long beta-normal form of M.
 1. Calculate the eta-long form $\eta_{lf}(M)$ of M *wrt* T;
 2. Enumerate maximal *branching* traversals of $\eta_{lf}(M)$ from Def. 4.2:
    - For each traversal t, get the *traversal core* $\pi(t)$ by removing all internal nodes;
    - Calculate the P-view of the traversal core $\ulcorner \pi(t) \urcorner$;
    - Interpret $\ulcorner \pi(t) \urcorner$ as a path in the tree representation of the eta-long β-nf of M;
 3. Reconstruct from the paths thus obtained the tree of the eta-long β-nf of M.

Correctness of STLC normalization

We now state the Paths Characterization of simply-typed terms as an immediate consequence of the Game-Semantic Correspondence Theorem for STLC [8]:

Proposition 4.5 (Normalized Paths Characterization for STLC). *For every term M inhabiting some simple type T, let $\eta\beta_{lnf}(M)$ denote the eta-long beta-normal form of M, then we have:*

$$\mathcal{T}rav_{\mathsf{STLC}}(M)/\sim \;= \mathcal{P}aths(\eta\beta_{lnf}(M))$$

Proof. By soundness of Game Semantics, beta-eta equivalent terms have the same game denotation, therefore by Theorem 3.7, the set of traversals of the eta-long beta-normal form of M is also given by the set of cores of traversals of $\eta_{lf}(M)$. The Path-View correspondence (Proposition 3.2) and Proposition 3.5 then show that the P-views of traversal cores of $\eta_{lf}(M)$ give the tree paths of the eta-long beta normal form. $\square$

Theorem 4.6 (Correctness of STLC normalization). *Algorithm B terminates and returns the eta-long beta-normal form of the input term.*

Proof. Soundness A beta-normal term is uniquely determined by the set of maximal paths in its tree representation (Property 2.1). By Proposition 4.5, this set corresponds precisely to the set of core P-views, and by Proposition 4.2 its is also given by the core P-views of *branching* traversals. *Termination* By Property 4.4 for terms in eta-long form the set of branching traversals is finite and each traversal is itself finite, therefore the enumeration in Algorithm B terminates. $\square$

Implementation. The normalization procedure from algorithm B was first implemented in the HOG program[5, 8, 6]. The program takes as input any simply-typed lambda term and lets the user interactively generate all the traversals by "playing the traversal game" over the tree representation of the term. It also offers an environment to calculate and perform operations over the traversals, including filtering, views and core projection.

4.4. Arity threshold and normalizing traversals

We showed in the previous section that for eta-long forms, there is a finite number of branching traversals, which in turn implies termination of the normalization procedure of Algorithm B. For untyped terms, however, branching traversals can be infinite because of *on-the-fly eta-expansion* in rule (IVar): the justification label $i > 0$ in the traversal rule is unbounded. In this section, we show that, for the purpose of term evaluation, i can be bounded by a computable quantity called the **arity threshold** of a traversal.

When traversing a lambda term, ghost nodes are introduced on-demand each time an eta-expansion is deemed necessary to lookup the argument of an application. We should not need to eta-expand ad-infinitum as ghost nodes are only useful if they eventually lead to traversing some structural node of the tree. The intuition behind the **arity threshold** is that after a sufficiently large number of eta-expansions, we are guaranteed to keep on traversing ghost nodes that never materialize back to structural nodes.

This section formalizes this intuition by defining the *arity threshold* as the maximum number of times necessary to eta-expand a given subterm (using rule (IVar)) in order to discover all set of paths of the beta-normal form of the term. In the next section we will show that such limit is sufficient to characterize the normal form of a lambda term when it exists.

Definition 4.3 (Strand). A **strand** of a traversal t, is any sub-sequence of consecutive nodes from t, with even length, starting with an external lambda node and ending with an external variable node, and such that all the occurrences in-between are internal.

From the parity property of traversals, a strand consists of alternations of lambda nodes and variable/application nodes. It is convenient to highlight strands within a traversal by underlining the first and last occurrence of the strand. We will often write strands as follows:

$$ t = \cdots \underline{\alpha_k} \ n_k \ \alpha_{k-1} \ n_{k-1} \ \cdots \ \alpha_2 \ n_2 \ \alpha_1 \ \underline{n_1} \cdots $$

for some $k \geq 1$ where for j ranging from k down to 1, α_j is a lambda node in $\mathcal{N}_\lambda^{\lambda}$, n_j is a variable/application node in $\mathcal{N}_{\text{var}}^\theta$, and the $2k - 2$ nodes occurring between α_k and n_1 are the internal nodes of the strand.

For any occurrence n in t of an external node, we call **strand ending at** n the sequence of nodes ending at n that constitutes a strand. It can be obtained by taking the longest subsequence of nodes preceding n consisting only of internal nodes.

Definition 4.4 (Traversal arity threshold). Let t be traversal ending with an external variable node. Let $\underline{\alpha_k} \ n_k \ \alpha_{k-1} \ n_{k-1} \ \cdots \ \alpha_2 \ n_2 \ \alpha_1 \ \underline{n_1}$ be the strand of t ending at t^ω, for some $k > 0$ where $\alpha_j \in \mathcal{N}_\lambda^{\lambda}$, and $n_j \in \mathcal{N}_{\text{var}}^\theta$ for all $1 \leq j \leq k$. We define the **arity threshold** of t as:

$$ \text{arth}(t) = \max_{q=1..k-1} \left(|n_q| + \sum_{j=1..q-1} (|n_j| - |\alpha_j|) \right) . $$

Or equivalently, using the abbreviation $\delta_j = |n_j| - |\alpha_j|$:

$$ \text{arth}(t) = \max_{q=1..k-1} \left(|n_q| + \sum_{j=1..q-1} \delta_j \right) = \max_{q=1..k-1} \left(|\alpha_q| + \sum_{j=1..q} \delta_j \right) $$

Remark 4.3 (Calculating the arity threshold). The arity threshold can be rewritten as $\text{arth}(t) = \max_{q=1..k-1} b_q$ where $b_q = |\alpha_q| + \sum_{j=1..q} \delta_j = |n_1| - |\alpha_1| + |n_2| - |\alpha_2| + \cdots + |n_{q-1}| - |\alpha_{q-1}| + |n_q|$, for $1 \leq q \leq k - 1$ or equivalently defined by the induction $b_1 = |n_1|$, $b_{q+1} = b_q + |n_{q+1}| - |\alpha_q|$, for $1 \leq q \leq k - 2$. In other words, calculating the arity threshold boils down to adding and subtracting the arity of the nodes starting from the last variable occurrence in the traversal strand and reading backwards until reaching the previous external variable. The maximal value calculated in that summation gives the arity threshold.

Observe that traversal rule (IVar) leaves infinitely many choices for the link label: any value greater than 1. The following property shows that if the link label exceeds the arity threshold then the traversal ends up traversing only ghost nodes that never materialize back to a structural node.

Proposition 4.7 (Weaving). *Let $t \in \mathcal{T}rav^{\mathsf{branch}}$ be a branching traversal ending with an external variable node, and suppose its last strand is $\underline{\alpha_k}\ n_k\ \alpha_{k-1}\ n_{k-1}\ \cdots\ \alpha_2\ n_2\ \alpha_1\ \underline{n_1}$ of length $2k$, for some $k \geq 1$. Let $t_{\max} \in \mathcal{T}rav^{\mathsf{branch}}$ be a maximal traversal extension of t. By rule (IVar$_{\mathsf{branch}}$), the last occurrence in t^ω is necessarily immediately followed in $t_{\max}$ by an external lambda node justified by t^ω with label $i \geq 1$.*

(i) *If $i > \mathsf{arth}(t)$ then t^ω is followed in $t_{\max}$ by a strand of length $2k$ consisting only of ghost nodes with labels defined as follows:*

$$t_{\max}\ =\ \cdots\ \underline{\alpha_k}\ n_k\ \alpha_{k-1}\ \cdots\ \alpha_2\ n_2\ \alpha_1\ \underline{t^\omega}\ \lambdabar\!\!\lambdabar\ \theta\ \lambdabar\!\!\lambdabar\ \theta\ \cdots\ \theta\ \lambdabar\!\!\lambdabar\ \underline{\theta}\ \cdots$$

$$i_{q+1}\ =\ i + \sum_{j=1..q}(|\alpha_j| - |n_j|) = i - \sum_{j=1..q}\delta_j,\quad \text{for } 0 \leq q \leq k$$

where underlined occurrences indicate external nodes.

(ii) *If $i > \mathsf{arth}(t)$ then all the nodes following t^ω in $t_{\max}$ are ghost occurrences.*

(iii) *If $i \leq \mathsf{arth}(t)$ then for some $0 \leq r < k$ and link labels i_q, $0 \leq q \leq r$ defined as in (i), we have:*

$$t_{\max}\ =\ \underline{\alpha_k}\ n_k\ \cdots\ \alpha_r\ n_r\ \alpha_{r-1}\ \cdots\ \alpha_2\ n_2\ \alpha_1\ \underline{t^\omega}\ \lambdabar\!\!\lambdabar\ \theta\ \lambdabar\!\!\lambdabar\ \theta\ \cdots\ \theta\ \lambda\overline{x}\ \cdots$$

where $\lambda\overline{x}$ is a structural internal lambda node, and consequently $i_r \leq |n_r|$.

This weaving property suggests that instantiations of the external variable rule that exceeds the arity threshold are not relevant for normalization since they essentially expand external ghost variable nodes into deeper external ghost variable nodes. This observation gives rise to a stricter version of traversals:

Definition 4.5 (Normalizing traversals). The set $\mathcal{T}rav^{\mathsf{norm}}$ of **normalizing traversals**, is the subset of branching traversals defined by the rules of Def. 4.2 where the index i in rule (IVar) is bounded by the arity threshold of the traversal. Table 2 recapitulates the rules thus obtained using judgement derivations, where the judgement notations '$\models t$' means '$t \in \mathcal{T}rav^{\mathsf{norm}}$'.

Remark 4.4. Observe that $\mathcal{T}rav^{\mathsf{norm}}$ is *not* $\sim$-complete. Indeed, take $M = \lambda x.x$ and the traversal

$t = \lambda x\ \cdot\ x\ \cdot\ \lambdabar\!\!\lambdabar\ \cdot\ \theta$ in $\mathcal{T}rav^{\mathsf{branch}}$. We have $\ulcorner\pi(t)\urcorner = t$ which is not equivalent to $\ulcorner\pi(u)\urcorner$ for any u in $\mathcal{T}rav^{\mathsf{norm}} = \{\epsilon, \lambda x, \lambda x \cdot x\}$.

<div style="border: 2px solid blue;">

Normalizing imaginary traversals of λ-calculus

PROGRAM - Structural rules

$$\frac{r \text{ is the root of } CT(M)}{\models r} \qquad (\mathsf{Root_{norm}})$$

$$\frac{\models t \cdot @ \qquad @ \vdash_0 \alpha}{\models t \cdot \overset{\overset{0}{\frown}}{@ \cdot \alpha} \qquad (\alpha \in \mathcal{N}_\lambda)} \qquad (\mathsf{App_{norm}})$$

$$\frac{\models t \cdot \alpha \qquad \alpha \in \mathcal{N}_\lambda \qquad \mathsf{Ch}(\alpha) \in \mathcal{N}_@}{\models t \cdot \alpha \cdot \mathsf{Ch}(\alpha) \qquad \mathsf{Ch}(\alpha) \text{ has no pointer}} \qquad (\mathsf{Lam^@_{norm}})$$

$$\frac{\models u \cdot \beta \cdot v \cdot \alpha \quad \alpha \in \mathcal{N}_\lambda \quad \mathsf{Ch}(\alpha) \in \mathcal{N}_{\mathsf{var}} \quad \beta \vdash_i \mathsf{Ch}(\alpha), i \geq 1 \quad \beta \text{ visible at } \alpha}{\models u \cdot \overset{\overset{i}{\frown}}{\beta} \cdot v \cdot \alpha \cdot \mathsf{Ch}(\alpha)} \qquad (\mathsf{Lam^{var}_{norm}})$$

$$\frac{\models t \cdot \alpha \cdot \overset{\overset{i}{\frown}}{m} \ldots \lambda\!\lambda}{\models t \cdot \overset{\overset{|\alpha| + i - |m|}{\frown}}{\alpha} \cdot \overset{\overset{i}{\frown}}{m} \ldots \lambda\!\lambda \cdot \theta \qquad (\alpha \in \mathcal{N}^{\lambda\!\lambda}_\lambda, m \in \mathcal{N}^\theta_{\mathsf{var}})} \qquad (\mathsf{Lam^{\lambda\!\lambda}_{norm}})$$

PROGRAM - Copy-cat rules

$$\frac{\models t \cdot m \cdot \overset{\overset{i}{\frown}}{\alpha} \ldots n \qquad n \in \mathcal{N}^\theta_{\mathsf{var}} \setminus \mathcal{N}^{\mathsf{ext}} \qquad m \vdash_i \beta \qquad i > 0}{\models t \cdot \overset{\overset{i}{\frown}}{m} \cdot \overset{\overset{i}{\frown}}{\alpha} \ldots n \cdot \beta \qquad (m \in \mathcal{N}^\theta_{\mathsf{var}} \cup \mathcal{N}_@, \alpha, \beta \in \mathcal{N}^{\lambda\!\lambda}_\lambda)} \qquad (\mathsf{Var_{norm}})$$

DATA - Input-variable rules

$$\frac{\models t \cdot n \qquad n \in \mathcal{N}^\theta_{\mathsf{var}} \cap \mathcal{N}^{\mathsf{ext}} \qquad n \vdash_i \alpha \qquad 1 \leq i \leq \mathsf{arth}(t)}{\models t \cdot \overset{\overset{i}{\frown}}{n \cdot \alpha} \qquad (\alpha \in \mathcal{N}_\lambda)} \qquad (\mathsf{IVar_{norm}})$$

where t, u, v range over (sub-sequences of) justified sequences of nodes.

Table 2: Normalizing traversals $\mathcal{T}rav^{\mathsf{norm}}$ of the untyped lambda calculus. The judgement rules define $\mathcal{T}rav^{\mathsf{norm}}$ as a strict subset of imaginary traversals $\mathcal{T}rav$ from Table 1 where only rules $\mathsf{IVar_{norm}}$ and IVar differ.

</div>

Algorithm C describes the normalization procedure to compute the β-normal of an untyped lambda term if it exists. It generalizes Algorithm B for STLC based on *branching* traversals to ULC using *normalizing* traversals. We will prove correctness of the procedure in Section 7.

Algorithm C Normalization by traversals for the Untyped Lambda Calculus

---

Input: Untyped term M having a beta-normal form
Output: Computation tree of the beta-normal form of M modulo α-conversion
    1. Calculate $\mathcal{P} \leftarrow \{\ulcorner\pi(t)\urcorner \mid t \in \mathcal{T}rav^{\mathsf{norm}}\}$ as follows:
      • Enumerate maximal *normalizing* traversals $\mathcal{T}rav^{\mathsf{norm}}(M)$ using the rules of Table 2;
      • For each traversal, apply transformation π to get the traversal core;
      • Take the P-view of the traversal core.
    2. Apply Algorithm A to reconstruct the tree of the β-nf of M from $\mathcal{P} \equiv \mathcal{P}aths(\beta_{\mathsf{nf}}(M))$.

---

5. Examples of Normalization by Traversals

This section walks you through the ULC normalization algorithm C and traversal enumeration of Table 2 on several examples of lambda terms.

For each example we will enumerate the normalizing traversal of Table 2. We write t_ϵ for the initial traversal up to the first non-deterministic choice. For every $s \in \mathbb{N}^*$, we write $t_{s \cdot k}$ for the maximal traversal obtained after extending t_s using a single application of the eta-expanded subcase of rule (IVar) with link-label $k \in \mathbb{N}$. In other words, the subscript s in t_s records the list of non-deterministic choices made so far by rule (IVar).

Example 5.1 (Baby example)**.** Take $M \equiv (\lambda x.xx)(\lambda y.y)$. Repeatedly applying the structural traversal rules until reaching a non-deterministic choice yields traversal

$$t_\epsilon = \lambda \quad @ \quad \lambda x \quad x \quad \lambda y \quad y \quad \lambda \quad x \quad \lambda y \quad y \quad \lambda\!\lambda^1 \quad \theta^1 \quad \lambda\!\lambda^1 \quad \theta^2 \quad \lambda\!\lambda^2 \quad \theta^2 \quad \lambda\!\lambda^2 \quad \theta^1.$$

Since the last node is an external variable (because it is justified by the root) rule (IVar) could extend t_ϵ if considered as a pure *imaginary* traversals. Since, the traversal's arity threshold is $\mathsf{arth}(t) = 0$, however, there is no more *normalizing* traversal to explore: t_ϵ is maximal. The P-view core is $\ulcorner\pi(t_\epsilon)\urcorner = \lambda \quad \theta^1$ thus the beta-normal form of M is, up to α-conversion $\lambda y.y$.

Example 5.2 (Church increment)**.** Consider Church numerals $k \equiv \lambda sz.s^k z$ for $k \geq 0$.

Consider a term M implementing the increment function so that Mk reduces to $k+1$ for all k. Such term can be defined as $M \equiv$ add 1 where add $\equiv \lambda xysz.x\, s(y\, s\, z)$. The right figure represents the computation tree of M.

$$
\begin{array}{c}
\lambda \\
| \\
@ \\
\diagup\ \diagdown \\
\lambda xysz \quad \lambda sz \\
| \qquad\quad | \\
x \qquad\quad s \\
\diagup\ \diagdown \qquad | \\
\lambda \quad \lambda\ \ \lambda \\
| \quad\ \ |\ \ | \\
s \quad\ \ y\ \ z \\
\diagup\ \diagdown \\
\lambda \quad \lambda \\
| \quad\ | \\
s \quad\ z
\end{array}
$$
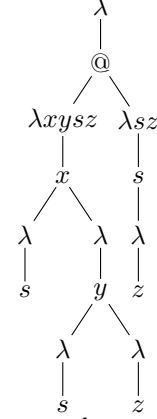
- The maximal normalizing traversal obtained without making any non-deterministic choice is:

$t_\epsilon = \lambda \quad @ \quad \lambda xysz \quad x \quad \lambda sz \quad s \quad \lambda \quad s \quad \lambda\!\!\lambda^3 \quad \theta^2$. The core of t_ϵ is $\pi(t_\epsilon) = \lambda \cdot \theta^2$ and therefore $\ulcorner \pi(t_\epsilon) \urcorner = \lambda \cdot \theta^2$ is a path in $\beta_{\mathsf{nf}}(M)$. This means that $\beta_{\mathsf{nf}}(M)$ must be of the form $\lambda ys \dots \cdot yN_1 \dots \dots N_q$ for some fresh variable y and s and $q \geq 0$.
- To determine each argument N_k from the normal form, we enumerate every possible non-deterministic choice corresponding to eta-expansion of each possible argument for $k \geq 1$ using rule (IVar). We then maximally extend the traversal until the next non-deterministic choice.
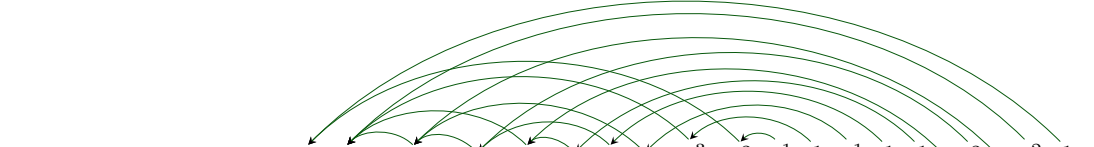
For $k=1$ this gives $t_1 = \lambda \quad @ \quad \lambda xysz \quad x \quad \lambda sz \quad s \quad \lambda \quad s \quad \lambda\!\!\lambda^3 \quad \theta^2\, \lambda\!\!\lambda^1\, \theta^1\, \lambda\!\!\lambda^1\, \theta^1\, \lambda^1\, z\, \lambda^2\, y\, \lambda\!\!\lambda^2\, \theta^1$. Its core P-view is $\ulcorner \pi(t_1) \urcorner = \lambda \cdot \theta^2 \cdot \lambda\!\!\lambda^1 \cdot \theta^1$ therefore the normal form is of the shape $\lambda ys \dots \cdot s(yR_1 \dots R_{q_2})N_2 \dots N_q$ for some terms $R_1, \dots R_{q_2}$, and $q, q_2 \geq 0$.
- How many more k do we need to look at? The answer: we need to keep iterating until the point where applying the traversal rules is guaranteed to only produce ghost variables and ghost lambda-nodes. Because there is a finite number of nodes in the computation tree, the variable node arities are bounded. Therefore, for high enough index k, the eta-expansion case from rule (Var) will never be met: after applying rule (IVar) on t_ϵ, all subsequent extensions of the traversal will be constructed using repeated applications of rule (Lam^θ) or the eta-expanded case of rule (Var).

The upper-bound q for k is precisely given by the *arity threshold* of the traversal t_ϵ as defined in 4.4:

$$
\mathsf{arth}(t_\epsilon) = \max\{|s^2|, |s^1| + (|s^2| - |\lambda|), |x| + (|s^1| - |\lambda sz|) + (|s^2| - |\lambda|)\}
$$
$$
= \max\{0, 1 + (0-0), 2 + (1-2) + (0-0)\} = 1
$$

Thus, we do not have to consider higher value of k at t_ϵ. Hence we have $\beta_{\mathsf{nf}}(M) = \lambda ys \dots \cdot s(yR_1 \dots R_{q_2})$.

26

- The arity threshold of t_1 is $\mathsf{arth}(t_1) = |z| + |y| - |\lambda^2| = 0 + 2 - 0 = 2$ hence $\beta_{\mathsf{nf}}(M)$ is of the form $\lambda ys \ldots \cdot s(yR_1R_2)$.
- Let's eta-expand using rule (IVar) for child index k_2 ranging from 1 to $q_2 = 2$. For the case $k_2 = 1$, we obtain the traversal:

$$t_{11} = \lambda \quad @ \quad \lambda xysz \quad x \quad \lambda sz \quad s \quad \lambda \quad s \quad \lambda\!\!\lambda^3 \quad \theta^2 \; \lambda\!\!\lambda^1 \; \theta^1 \; \lambda\!\!\lambda^1 \; \theta^1 \; \lambda^1 \; z \; \lambda^2 \; y \; \lambda\!\!\lambda^2 \; \theta^1 \; \lambda\!\!\lambda^1 \; \theta^1 \; \lambda \; s \; \lambda\!\!\lambda^3 \; \theta^2$$
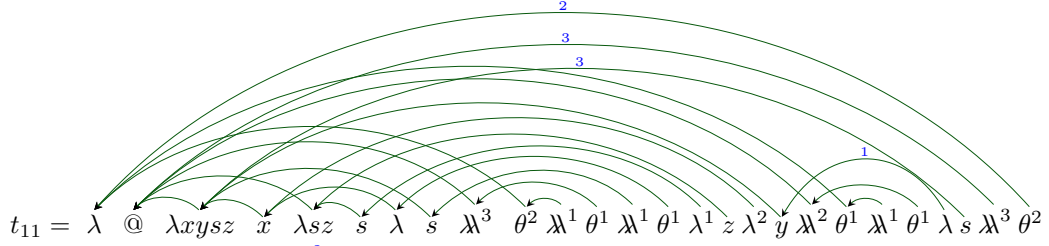
Thus $\ulcorner \pi(t_{11}) \urcorner = \lambda \quad \theta^2 \; \lambda\!\!\lambda^1 \; \theta^1 \; \lambda\!\!\lambda^1 \; \theta^2$. Hence $\beta_{\mathsf{nf}}(M)$ is of the form $\lambda ys \ldots \cdot s \ (y \ s \ R_2)$.

- Extending t_1 with (IVar^λ) for $k_2 = 2$ gives:

$$t_{12} = \lambda \quad @ \quad \lambda xysz \quad x \quad \lambda sz \quad s \quad \lambda \quad s \quad \lambda\!\!\lambda^3 \quad \theta^2 \; \lambda\!\!\lambda^1 \; \theta^1 \; \lambda\!\!\lambda^1 \; \theta^1 \; \lambda^1 \; z \; \lambda^2 \; y \; \lambda\!\!\lambda^2 \; \theta^1 \; \lambda\!\!\lambda^2 \; \theta^2 \; \lambda \; z \; \lambda\!\!\lambda^4 \; \theta^3$$
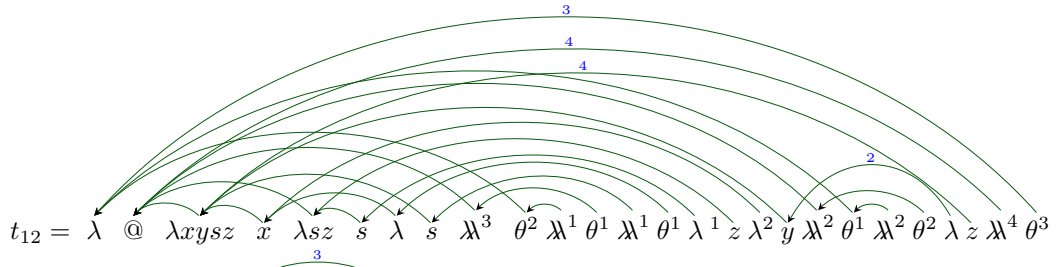
Thus $\ulcorner \pi(t_{12}) \urcorner = \lambda \quad \theta^2 \; \lambda\!\!\lambda^1 \; \theta^1 \; \lambda\!\!\lambda^2 \; \theta^3$.

Hence $\beta_{\mathsf{nf}}(M) = \lambda ysz \cdot s \ (y \ s \ z)$.

Example 5.3 ("Missing operand" example by Neil Jones). This small example illustrates how on-the-fly eta-expansion helps resolve the "missing argument" problem faced when using the traversal rules of STLC. Take $M = (\lambda u.u \ (y_1 \ u))(\lambda v.v \ y_2)$. The computation tree and its only two maximal normalizing traversals are shown below.

$$\begin{array}{c}\lambda \\ | \\ @ \\ \lambda u \quad \lambda v \\ | \qquad | \\ u \quad\quad v \\ | \qquad | \\ \lambda \quad\quad \lambda \\ | \qquad | \\ y_1 \quad\quad y_2 \\ | \\ \lambda \\ | \\ u \end{array}$$
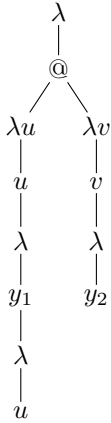
With STLC traversals, one can traverse t_1 all the way to variable v at which point one gets stuck due to the lack of operand applied to the last occurrence of u (the occurrence at the bottom of the left branch in the tree).

$$t_1 = \lambda \ @ \ \lambda u \ u \ \lambda v \ v \ \lambda \ y_1 \ \lambda \ u \ \lambda v \ v \ \lambda\!\!\lambda^1 \ \theta^1 \ \lambda\!\!\lambda^1 \ \theta^1 \ \lambda \ y_2$$
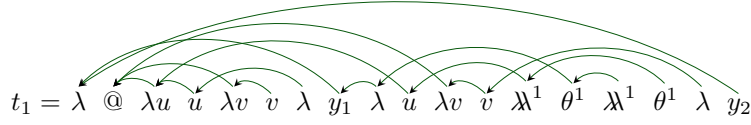
Using imaginary traversals, that operands gets created on-the-fly through eta-expansion and is represented by ghost lambda node $\lambda\!\!\lambda^1$. The other maximal traversal is

$$t_2 = \lambda \ @ \ \lambda u \ u \ \lambda v \ v \ \lambda \ y_1 \ \lambda\!\!\lambda^2 \ \theta^1 \ \lambda \ y_2$$
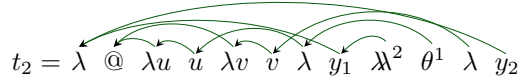
The two core P-views give the two maximal paths in the beta-normal form: $\ulcorner\pi(t_1)\urcorner = \lambda \ y_1 \ \lambda \ \theta^1 \ \lambda\!\!\lambda^1 \ y2$ and $\ulcorner\pi(t_2)\urcorner = \lambda \ y_1 \ \lambda\!\!\lambda^2 \ y_2$. The beta-normal form of M is thus $y_1(\lambda z.z \ y_2)y_2$.

Example 5.4. Take the fixed-point combinator $\Omega = (\lambda x.xx)(\lambda y.yy)$.

$$\begin{array}{c}\lambda \\ | \\ @ \\ \lambda x \quad \lambda y \\ | \qquad | \\ x_1 \quad\quad y_1 \\ | \qquad | \\ \lambda \quad\quad \lambda \\ | \qquad | \\ x_2 \quad\quad y_2 \end{array}$$
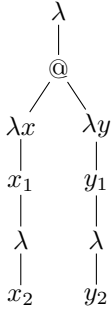
It's a well-known fact that Ω does not have a normal form. The term has a unique (infinite) traversal represented below, which starts with

$$\lambda \ @ \ \lambda x \ x_1 \ \lambda y \ y_1 \ \lambda \ x_2 \ \lambda y \ y_1 \ \dots$$ and then follows a repeated pattern with an increasing number of ghost nodes between occurrences of y_1 and y_2:

$$\lambda\ @\ \lambda x\ x\ \lambda y\ y\ \lambda\ x\ \lambda y\ y\ \lambda\!\!\lambda^1\ \theta^1\ \lambda\ y\ \lambda\ x\ \lambda y\ y\ \lambda\!\!\lambda^1\ \theta^1\ \lambda\!\!\lambda^1\ \theta^1\ \lambda\ y\ \lambda\!\!\lambda^1\ \dots$$
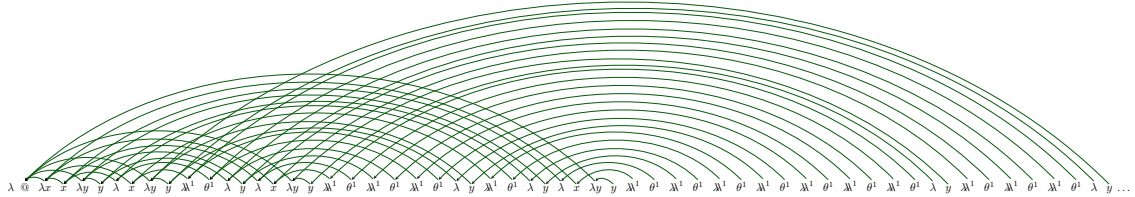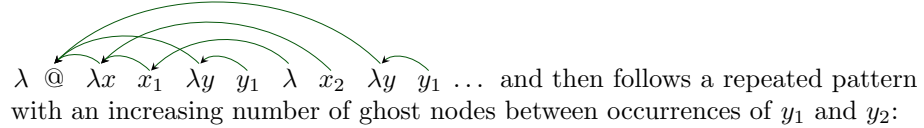
Consider each section of the traversal starting with y_1 and ending with y_2 with no other variable occurrence in-between, and define its length as the number of ghost nodes in-between. Then the lengths of successive such traversal sections have the following pattern: $2, 6, 14, 30, 62 \dots$ One can show that the length of the ith section is precisely given by $2 \times (2^i - 1)$, for $i \geq 1$. So as the term is being evaluated, it takes exponentially longer to determine that y_2 is the argument of y_1. The

28

infinite traversal t can be expressed as follows (omitting justification pointers):

$$t_\Omega = \lambda \cdot @ \cdot \lambda x \cdot x_1 \cdot \lambda y \cdot y_1 \cdot \sum_{i \geq 1} \left[\lambda \cdot x_2 \cdot \lambda y \cdot y_1 \cdot (\lambda\!\!\lambda \ \theta)^{2^i - 1} \cdot \lambda \cdot y_2 \sum_{i \leq j < i} \left((\lambda\!\!\lambda \ \theta)^{2^j - 1} \cdot \lambda \cdot y_2 \right) \right]$$

where $\sum$ represents sequence concatenation and u^i represents the concatenation of i copies of u for $i \geq 0$. Represented more succinctly by keeping only variable nodes and omitting redundant lambda and @-nodes: $t_\Omega \upharpoonright \mathcal{N}_{\mathsf{var}} = x_1 \cdot y_1 \cdot \sum_{i \geq 1} \left[x_2 \cdot y_1 \cdot \theta^{2^i - 1} \cdot y_2 \sum_{1 \leq j < i} (\theta^{2^j - 1} \cdot y_2) \right]$. Without ghost nodes, Ω's traversal semantics can thus be summarized as $t_\Omega \upharpoonright (\mathcal{N}_{\mathsf{var}} \cap \mathcal{N}^{\mathsf{ext}}) = x_1 \cdot y_1 \cdot \sum_{i \geq 1} x_2 \cdot y_1 \cdot y_2^i$.

Example 5.5 (Neil Jones' varity example). Take *varity two* where $two \equiv \lambda s_2 z_2 . s_2 (s_2 \ z_2)$ and $varity \equiv \lambda t . t (\lambda nax . n(\lambda sz . as(xsz)))(\lambda a.a)(\lambda z_0 . z_0)$. The initial traversal t_ϵ up to the first non-deterministic choice is:
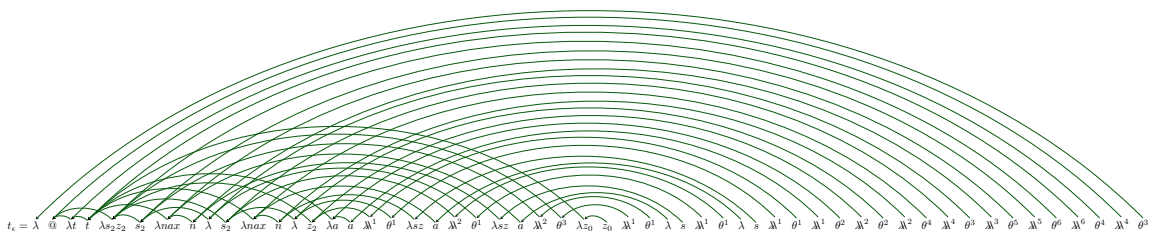
Enumerating normalizing traversals yields the set $\{t_\epsilon, t_{11}, t_{12}, t_{121}, t_{122}\}$. Keeping only maximal traversals gives $\mathcal{T}rav_{\max}^{\mathsf{norm}}(M) = \{t_{11}, t_{121}, t_{122}\}$ from Table 3 in appendix. The set of maximal paths in the beta-normal form is thus given by the core P-views $\ulcorner \pi(t_{11}) \urcorner = \lambda \ \theta^3 \ \lambda\!\!\lambda^1 \ \theta^1 \ \lambda\!\!\lambda^1 \ \theta^3$, $\ulcorner \pi(t_{121}) \urcorner = \lambda \ \theta^3 \ \lambda\!\!\lambda^1 \ \theta^1 \ \lambda\!\!\lambda^2 \ \theta^2 \ \lambda\!\!\lambda^1 \ \theta^3$ and $\ulcorner \pi(t_{122}) \urcorner = \lambda \ \theta^3 \ \lambda\!\!\lambda^1 \ \theta^1 \ \lambda\!\!\lambda^2 \ \theta^2 \ \lambda\!\!\lambda^2 \ \theta^4$ Therefore the beta-normal from of *varity* 2 is $\lambda x_1 x_2 x_3 x_4 . x_3 (x_1 x_3 (x_2 x_3 x_4)) \equiv \lambda xysz . s(xs(ysz))$.

6. Leftmost linear reduction

6.1. Lambda calculus background and notations

We recall standard results of the lambda calculus. A **redex** is a sub-term of the form $(\lambda x.M)N$. Reducing redex $(\lambda x.M)N$, or also *firing* the redex, means substituting all free occurrences of x in M by the term N using capture avoiding substitution (the bound variable is renamed afresh when recursively substituting under a lambda). A term is said to be in **normal form** if it does not contain any redex. A term is in **head normal form** if it can be written $\lambda x_1 \ldots x_n . y A_1 \ldots A_m$ for $n, m \geq 0$. If a term is not in head normal form then its **head beta-redex** is the left-most redex, otherwise the term does not have any head beta-redex. For any reduction relation $\rightarrow$ between terms we will write $\rightarrow^*$ to denote its reflexive transitive closure.

The **head reduction**, denoted $\rightarrow_h$, fires the head redex of a term if it exists. It can be shown that $\rightarrow_h^*$ yields the head-normal form. The **normal-order reduction strategy**, also called leftmost-outermost reduction strategy, performs head reduction until reaching the head-normal form and then recursively applies head reduction on each operand of the head variable. A standard result is that this reduction strategy yields the normal form if it exists.

6.2. Head-linear reduction

In the lambda calculus, a redex is necessarily formed by the outermost lambda in operator position of an application: if the operator consists of consecutive lambda abstractions (*e.g.*, as in $(\lambda x \lambda y.M)A_1 A_2$) then the outermost lambda (*e.g.*, λx) is the one that will form the redex. The notion of redex can be generalized to allow evaluation of arguments in any order. In particular, one can allow any of the consecutive λ-abstractions to form a redex (*e.g.*, the abstraction λy and corresponding argument A_2 would be a valid redex). This is formalized by the notion of *generalized redex*–a generalization of the notion of *prime redex* from [10]:

Definition 6.1 (Generalized redex)**.** The set of generalized redexes of a term M, written $gr(M)$, is a set of pairs $(\lambda x, A)$ where λx is some abstraction in M and A is a subterm called the argument of λx. The head λ list of M, written $\lambda_l(M)$ is a list of lambda abstractions of M. They are defined by induction:

$$
\begin{aligned}
gr(v) &= \emptyset & \lambda_l(v) &= \epsilon \\
gr(\lambda x.U) &= gr(U) & \lambda_l(\lambda x.U) &= \lambda x \cdot \lambda_l(U) \\
gr(UV) &= \{(\lambda x, V)\} \cup gr(U) \cup gr(V) & \lambda_l(UV) &= l & \text{if } \lambda_l(U) = \lambda x \cdot l \\
gr(UV) &= gr(U) \cup gr(V) & \lambda_l(UV) &= \epsilon & \text{if } \lambda_l(U) = \epsilon.
\end{aligned}
$$

where v ranges over variable occurrences, x ranges over variable names, U, V range over subterms of M, and ϵ denotes the empty list.

Example 6.1. For any term M, N, A_1, A_2 we have $gr((\lambda x \lambda y.M)A_1 A_2) = \{(\lambda x, A_1), (\lambda y, A_2)\}$ and $gr((\lambda z.(\lambda x \lambda y.M)N)A_1 A_2) = \{(\lambda z, A_1), (\lambda x, N), (\lambda y, A_2)\}$.

To define head-linear reduction, one needs to consider specific *occurrences* of variables and subterms in a given term. In particular, let's emphasize that a generalized redex $(\lambda x, A)$ refers to specific *occurrence* of λx and subterm A.

We say that a variable occurrence is ***involved*** in the generalized redex $(\lambda x, A)$ if the variable occurrence is bound by λx. A variable occurrence can therefore be involved in at most one generalized redex. We define the ***linear substitution*** of x for A as the term obtained by performing capture-avoiding substitution of that single occurrence of x by A. (Compare this to the standard substitution that applies to *every* occurrence of x in M.) When performing such substitution we say that we ***linearly fire*** the generalized redex $(\lambda x, A)$ for that occurrence of x.

The ***head variable occurrence*** (abbreviated *hoc*) of a term is the left-most variable occurring in the term (*i.e.*, the first variable found by depth-first traversal of the term tree.) If the head variable occurrence of a term is involved in a generalized redex then we call that redex the ***head-linear redex***. A term that does not have a head-linear redex is said to be in ***quasi head-normal form***. The ***head-linear reduction*** $\rightarrow_{hl}$ is defined as the reduction that linearly fires the head linear redex, if it exists. It can be shown that the reflexive transitive closure $\rightarrow_{hl}^*$ yields the quasi head-normal form [10, 9].

Soundness. The set of ***spine subterms*** is defined by induction: a term is a spine subterm of itself; the spine subterms of UV and $\lambda x.U$ are those of U. A ***prime redex*** (ala Danos-Regnier) is a generalized redex $(\lambda x, A)$ such that the operator of the redex ($\lambda x.U$ for some U) is a spine subterm. Danos-Regnier showed the following result (Theorem 2 in [10]):

Theorem 6.1 (Soundness and completeness of head-linear reduction [10])**.**

- *If $T \to_{hl}^* T$ then T and T' are β-equivalent.*
- *If T is in quasi-head normal form and has n prime redexes then the head reduction of T leads to a head normal form in exactly n steps.*
- *If T is any term, the head linear reduction of T terminates iff the head reduction of T terminates.*

6.3. Leftmost linear reduction

As the name indicates, the head-linear reduction is a linear version of the *head* reduction. It thus only yields the quasi *head*-normal form, not the normal form, and therefore is not complete for normalization.

In the standard lambda calculus, normal-order reduction strategy is obtained by repeatedly applying head reduction to get to the head-normal form, and then continuously applying the head-reduction on each argument of the head variable. This ultimately yields the normal form of the term if it exists. We now define the linear counterpart of the normal-order reduction: Informally, the **leftmost linear reduction strategy** is the strategy that performs head-linear reduction if possible, and otherwise, if the term is in quasi-head normal form, continuously (and recursively) applies the head-linear reduction on each argument (from left to right) of the head variable occurrence. Formally:

Definition 6.2 (Leftmost linear reduction)**.** Given a term M, we write $\mathcal{O}cc(M)$ for the set of variable occurrences in M and $\mathcal{O}cc^\perp(M)$ for $\{\perp\} + \mathcal{O}cc(M)$ where the bottom element $\perp$ represents the 'undefined' occurrence. We introduce the partial order $\sqsubseteq$ on $\mathcal{O}cc^\perp(M)$ defined by: for all $x, y \in \mathcal{O}cc^\perp(M)$, $x \sqsubseteq y$ if and only if $x = y$ or $x = \perp$. We define the partial function $lloc_M$ from subterms of M to variable occurrences by induction on the subterms of M:

$$lloc_M(v) = \begin{cases} v & \text{if } v \text{ is involved in a generalized redex in } M, \\ \perp & \text{otherwise.} \end{cases}$$

$$lloc_M(\lambda x.U) = lloc_M(U)$$

$$lloc_M(UV) = \begin{cases} lloc_M(U) & \text{if } lloc_M(U) \neq \perp, \\ lloc_M(V) & \text{if } lloc_M(U) = \perp \text{ and } \lambda_l(U) = \epsilon, \\ \perp & \text{if } lloc_M(U) = \perp \text{ and } \lambda_l(U) \neq \epsilon. \end{cases}$$

where v ranges over variable occurrences in M, and for any subterm N, $lloc_M(N) = \perp$ denotes that $lloc$ is undefined at N.

The **leftmost linear variable occurrence** of M is defined as $lloc_M(M)$ if it exists and the generalized redex it is involved in is called the **lloc redex**, otherwise M is said to be in **quasi normal form**, or *qnf* for short. The **leftmost linear reduction** strategy, written $\to_{ll}$, is defined as the strategy that linearly fires the generalized redex involving the leftmost linear variable occurrence.

Leftmost linear reduction proceeds by first locating the left-most variable occurrence and, if it is involved in a redex, linearly fires it. In comparison, the standard left-most outermost reduction strategy first locates the leftmost redex and then fires it by substituting all the variables involved in it.

Since the *lloc* is uniquely defined, a given term has a unique left-most linear reduction sequence $M \to_{ll} M_1 \to_{ll} M_2 \to_{ll} \ldots$ (also abbreviated as **linear reduction sequence**).

Example 6.2. Take $M = (\lambda x.xxN)z$. We have $M \to_{hl} (\lambda x.zxN)z$ which is in quasi head normal form because the head variable z is not involved in any generalized redex. But since the left-most occurrence x is involved in the redex $(\lambda x, z)$ the leftmost linear reduction gives $(\lambda x.zxN)z \to_{ll} (\lambda x.zzN)z$.

The *lloc* generalizes the notion of *hoc* to arguments of the head variable. Observe, however, that contrary to the *hoc*, the definition requires the *lloc* to be involved in a generalized redex.

Example 6.3. In the term $M = (\lambda xy.zyx)(\lambda u.u)(\lambda v.v)$, the *hoc* variable z is not involved in any generalized redex, whereas the *lloc* variable y is involved in generalized redex $(\lambda y, (\lambda v.v))$.

On the other hand, if the *hoc* is involved in a generalized redex then by definition it must coincide with the *lloc*.

Property 6.2. *If* $\lambda_l(M) = \langle \lambda x_1 \ldots x_n \rangle$ *and* M *has a* β-*normal form then its beta-normal form starts with at least* n *lambda abstractions, i.e., up to variable renaming it is of the shape* $\lambda x_1 \ldots x_n.M'$ *for some beta-normal term* M'.

Proof. Trivial induction. □

Property 6.3. *Let* M *be an untyped term and* $yA_1 \ldots A_n$ *be a subterm of* M *for* $n \geq 1$ *where* y *is a variable not involved in any generalized redex then:*

$$lloc_M(yA_1 \ldots A_n) = \begin{cases} lloc_M(A_j) & \text{where } j = \min\{1 \leq i \leq n \mid lloc_M(A_i) \neq \bot\} \\ \bot & \text{if } lloc_M(A_i) = \bot \text{ for all } 1 \leq i \leq n. \end{cases}$$

Proof. Immediate from the definition of *lloc* since $\lambda_l(y) = \epsilon$. □

We say that a standard redex $(\lambda x.N)A$ is **trivial** if x does not occur freely in N in which case the redex can be *trivially reduced* to just N.

Property 6.4. *Some immediate properties:*

(i) *If* N *is a subterm of* M *then* $lloc_N(N) \sqsubseteq lloc_M(N)$;
(ii) *If* M *is beta-normal then* $lloc_M(M) = \bot$;
(iii) *If* UV *is in quasi normal form then so is* U;
(iv) *If* $\lambda x.U$ *is in quasi normal form then so is* U;
(v) *If* M *has a* lloc *redex then* M *has a redex*;
(vi) *If all the redexes in* M *are trivial then* M *is in* qnf.

Proof. (i) Immediate from the fact that generalized redexes of N are also generalized redexes of M. (ii) If $lloc_M(M) \neq \bot$ then M must contain a generalized redex and therefore must also contain a standard redex hence M is not beta-normal. (iii) If UV is in quasi normal form then $lloc_{UV}(UV) = \bot$ so by definition of *lloc* we must have $lloc_{UV}(U) = \bot$. By (i) since U is a subterm of UV this implies $lloc_{UV}(U) = \bot$. (iv) We have $\bot = lloc_{\lambda x.U}(\lambda x.U) = lloc_{\lambda x.U}(U) \sqsupseteq lloc_U(U)$. (v) and (vi) are immediate from the definition of *lloc* redex. □

The following example shows that the converse of (vi) does not hold:

Example 6.4. Take $M = (\lambda z.u)((\lambda x.x)y)$. Then $lloc_M(M) = \bot$ since $\lambda_l(\lambda z.y) \neq \epsilon$ and $lloc_M(\lambda z.u) = \bot$. So M is in quasi-normal form, even though it has a non-trivial redex in the argument $(\lambda x.x)y$.

Definition 6.3 (Arguments of the hoc). Let z be the *hoc* of M. The arguments of the *hoc* occurrence, written $\mathsf{Args}(M)$, is a list of subterms of M defined by induction:

$$\mathsf{Args}(z) = \epsilon$$
$$\mathsf{Args}(\lambda x.U) = \mathsf{Args}(U)$$
$$\mathsf{Args}(UV) = \begin{cases} \mathsf{Args}(U) \cdot V & \text{if } \lambda_l(U) = \epsilon \\ \mathsf{Args}(U) & \text{if } \lambda_l(U) \neq \epsilon \end{cases}$$

Note that the variable case where $x \neq z$ is not needed since the induction proceeds by spinal descent to the head occurrence.

Property 6.5. *If M is in quasi-head-normal form, then the head variable of its head-normal form has precisely $|\mathsf{Args}(M)|$ arguments.*

Proof. By Theorem 6.1, the *hoc* of a quasi-head normal form becomes the head variable of the head-normal form after reducing the prime redexes, and it is easy to show by structural induction that standard beta-reduction preserves the number of pending hoc arguments. □

We now introduce a β-reduction strategy (in the standard sense). We define the **leftmost spinal-innermost** standard β-redex of M as the β-redex $(\lambda x.N)A$ such that there is no other redex in M occurring left of it and such that N contains no other redex; the argument A on the other-hand, may contain other redexes. The idea consists in reducing M's redexes in 'inside-out left-to-right' order. This definition should be distinguished from the *leftmost innermost* reduction strategy from the lambda calculus literature. In particular $(\lambda x.y)\Omega$ reduces to y with the leftmost spinal-innermost strategy, while the leftmost innermost strategy does not terminate.

Proposition 6.6 (Trivial reduction of quasi normal forms). *Let M be a term in* qnf.
  (i) *If M is not in beta-normal form then its leftmost spinal-innermost standard β-redex $(\lambda x.N)A$ is trivial. (i.e., x does not occur freely in N).*
  (ii) *M has a beta-normal form and the leftmost spinal-innermost strategy normalizes M.*

Observe that if M is in *qnf*, its redexes are not all necessarily trivial, but the non-trivial redexes get eliminated by the trivial reductions:

Example 6.5. The term $M = (\lambda x.u)((\lambda z.z)y)$ is in *qnf* and trivially reduces to u but contains the non-trivial redex $(\lambda z.z)y$.

Example 6.6. Take $M = (\lambda x.z((\lambda wy.y)x))U$ for any term U. Then M is in quasi-head normal form with one prime spine redex (x, U). Performing one head reduction gives $M \rightarrow_h z((\lambda wy.y)U)$ which brings the *hoc* in head position. M is also in quasi normal form. Indeed $lloc_M(M) = lloc_M(\lambda wy.y) = \bot$. Reducing the term with the leftmost spinal-innermost reduction strategy gives $M \rightarrow (\lambda x.z(\lambda y.y))U \rightarrow z(\lambda y.y)$ which yields a beta-normal form.

Theorem 6.7 (Soundness of leftmost linear reduction). *If $M \rightarrow_l N$ then M and N are beta-equivalent.*

Proof. The proof is like that of Theorem 6.1. We adapt the notion of *consecutive redex* from [10] to leftmost-linear reduction: Let $r = (\lambda x, V)$ and $s = (\lambda y, W)$ be two generalized redexes. We say that r contains s if the node λy lies in the scope of λx. We say that r and s are *consecutive* if r

contains s; no other generalized redex other than r and s is contained in r and contains s; and s is the leftmost generalized redex contained in r (*i.e.*, any other redex t contained in r resides on the left of node λy in the usual tree representation of the term). Let $r_0, \ldots, r_p$ denote the consecutive generalized redexes for $p \geq 0$ where r_0 is not contained in any other generalized redexes and r_p is the *lloc redex*. (This sequence necessarily exists by definition of *lloc*.) An induction on p shows that M and N reduce to the same term by exactly $p-1$ standard reduction steps (*i.e.*, the strategy reducing the leftmost outermost redex), observing that at every step, no redex can be introduced above or on the left of the *lloc*. $\qquad\square$

Property 6.8. *Let M_1, M_2 be two lambda terms. (i) If $M_1 \to_\beta M_2$ then $|\lambda_l(M_2)| \geq |\lambda_l(M_1)|$; (ii) If $M_1 \to_{ll} M_2$ then $|\lambda_l(M_2)| \geq |\lambda_l(M_1)|$; (iii) If M_1 trivially reduces to M_2 then $\lambda_l(M_2) = \lambda_l(M_1)$.*

Proof. (i) and (ii) Performing substitution in an operand of an application does not affect its lambda-list; while substituting a variable in operator position in the application, effectively appends the lambda-list of the substituted term to the lambda-list of the application. (iii) follows trivially by definition of λ_l. $\qquad\square$

Example 6.7. Take $M_1 = (\lambda x.x)(\lambda yz.y)$. We have $\lambda_l(M_1) = \epsilon$ but $M \to_\beta \lambda yz.y = M_2$, $M_1 \to_{ll} (\lambda x.(\lambda yz.y))(\lambda yz.y) = M_3$ and and $\lambda_l(M_2) = \lambda_l(M_3) = (\lambda y, \lambda z)$. Take $N = (\lambda x.U)V$ where x does not occur in U then N trivially reduces to U and $\lambda_l(N) = \lambda_l(U)$.

Property 6.9. *(i) If M is in beta-nf and $\lambda_l(M) = \epsilon$ then M is not an abstraction.*
(ii) If M is in qnf and M has a beta-nf then its beta-normal form is not an abstraction.

Proof. (i) If M were an abstraction $\lambda x.N$ we would have $\lambda_l(M) = \lambda x$ which contradicts the assumption. (ii) By Proposition 6.6(ii), we have that M trivially reduces to its beta-nf so by Property 6.8, M and its normal form must have the same lambda list. We can then conclude using (i). $\qquad\square$

Theorem 6.10 (Completeness of leftmost linear reduction)**.** *If M has a beta-normal form then its left-most linear reduction terminates (and thus yields a quasi normal form).*

The technical proof is in appendix.

7. Correctness of ULC normalization

Correctness of the evaluation procedure relies, like in the STLC case, on a Path Characterization Theorem. Unlike STLC, however, we show this characterization without appealing to the Game Semantics correspondence, since its untyped counterpart remains a conjecture (Conjecture 3.1). The proof relies instead on a soundness result showing that $\sim$-equivalence classes are preserved by *leftmost-linear reduction* from Section 6. It is based on a bisimulation argument requiring careful analysis of the traversals structure.

Definition 7.1. A subsequence of a traversal is a (i) ***spinal descent*** if it is a path in the computation tree consisting solely of lambda and @-nodes, and where each lambda node (except the first occurrence if it is a lambda node) is the 0th child of the preceding @-node. (ii) ***pending argument lookup*** if it consists of an alternation of ghost lambda nodes and ghost variables nodes, starting with an external ghost lambda node and terminated by a structural internal lambda node in $\mathcal{N}_\lambda$. (iii) ***branching descent*** if it consists of an alternation of external structural lambda nodes and external structural variable nodes.

Definition 7.2 (lloc of a node)**.** For any tree node n we write $lloc_M(n)$ to denote $lloc_M(N)$ where N is the subterm of M rooted at n and $lloc$ denotes the leftmost linear occurrence from Def. 6.2.

Property 7.1. *Let t be a traversal of M and u be a subsequence of t of the form $u = \underline{\lambda\overline{x}} \cdot v \cdot \underline{x}$ for some subsequence v of t, external structural variable node x and external structural lambda node $\lambda\overline{x}$. If v is empty then the node following u in t, if it exists, is necessarily a structural child lambda node of x (i.e., it cannot be a ghost node).*

Proof. If v is empty then the strand ending at x is just $\underline{\lambda\overline{x}} \cdot \underline{x}$ therefore t's arity threshold is precisely $|x|$, in which case rule (IVar) can only visit a structural child of x. $\qquad\square$

Definition 7.3 (Strand types)**.** We distinguish two types of strands:
- Structural argument strands (S) of the form $\underline{\lambda\overline{\eta}} \cdot u \cdot \underline{x}$ where
    - $\lambda\overline{\eta}$ is an external structural lambda node verifying $lloc_M(\lambda\overline{\eta}) = \perp$,
    - u is a (possibly empty) *spinal descents* of internal nodes,
    - x is an external structural variable node;
- Ghost argument strands (G) of the form $\underline{\lambda\!\!\lambda} \cdot v \cdot \lambda\overline{y} \cdot u \cdot \underline{x}$ where
    - $\lambda\!\!\lambda$ is an external ghost lambda node,
    - v is a *pending parameter lookup* (ghost nodes) of length shorter than u,
    - $\lambda\overline{y}$ is an internal lambda node verifying $lloc_M(\lambda\overline{y}) = \perp$,
    - x is an external structural variable node.

Proposition 7.2 (Quasi-normal forms strand decomposition)**.** *Any maximal traversal $t_{\max}$ of a qnf M consists of a succession of S and G strands determined by the following transition system with states $\{S, G\}$:*

State (strand type)	Transition	Current strand	Next strand
Initial state	$\to S$	ϵ	$\underline{\lambda\overline{\eta_1}} \cdot u_1 \cdot \underline{x_1}$
S	$S \to G$	$\underline{\lambda\overline{\eta_1}} \cdot u_1 \cdot \underline{x_1}$	$\underline{\lambda\overline{\eta_2}} \cdot u_2 \cdot \underline{x_2}$
	$S \to S$		$\underline{\lambda\!\!\lambda} \cdot v \cdot \lambda\overline{y_2} \cdot u_2 \cdot \underline{x_2}$
G	$G \to S$	$\underline{\lambda\!\!\lambda} \cdot v \cdot \lambda\overline{y_1} \cdot u_1 \cdot \underline{x_1}$	$\underline{\lambda\overline{\eta_2}} \cdot u_2 \cdot \underline{x_2}$
	$G \to G$		$\underline{\lambda\!\!\lambda} \cdot v \cdot \lambda\overline{y_2} \cdot u_2 \cdot \underline{x_2}$

where $\lambda\overline{y_2}$ is justified by a @-node in u_1.

Figure 1 represents the more fine-grained state machine underpinning the stand decomposition result. It has three states representing each possible type of external node: lambda, ghost lambda or input variable. The bottom two states correspond to the start of a new strand in the traversal. The solid-arrows represent consecutive applications of deterministic traversal rules while dashed-arrows represent the two possible non-deterministic choices in rule (IVar): one for structural child lambda nodes (left) and one for ghost children (right).

Proposition 7.3. *Let $t \in \mathcal{T}rav^{\mathsf{norm}}(M)$ for some quasi normal term M. We have:*
(a) $lloc_M(\alpha) = \perp$ for all external lambda node $\alpha \in \mathcal{N}_\lambda$ occurring in t.
(b) t does not contain any internal variable node (i.e., all the internal nodes are @ nodes);
(c) t is finite.

Proof. (a) and (b) are direct consequences of Proposition 7.2. (c) Let's consider the cases of the strand decomposition of Proposition 7.2 observe that for the two transitions $\to S$ and $S \to S$,
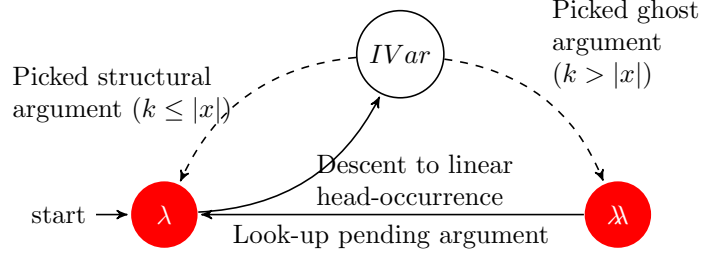
Figure 1: State machine underpinning the *qnf* strand decomposition.

because u_1 and u_2 are spinal descents, the last node in the next strand (x_1 or x_2) must be a descendant (in the computation tree of M) of the first node ($\lambda\overline{\eta_1}$) from the previous strand. For transition $S \to G$, since both u_1 and u_2 are spinal descents, there are also paths in the tree, therefore since $\lambda\overline{y_2}$ points to u_1, x_2 must also be a descendant of $\lambda\overline{\eta_1}$. For transitions $G \to S$ and $G \to G$, since u_1 and u_2 are spinal descents, x_2 is a descendant of $\lambda\overline{y_2}$, itself a descendant of $\lambda\overline{y_1}$.

Suppose t is infinite then from the above strand decomposition we can construct an infinite path in the tree of M which contradicts the fact that M is a finitary term. $\qquad\square$

Lemma 7.4 (Generalized redex argument lookup). *Let x be a variable occurrence involved in a generalized redex in $gr(M)$.*

(i) *The path from the parent node of the redex argument down to x's binder is of the form: $@_r \cdot \lambda\overline{x_r} \cdots @_2 \cdot \lambda\overline{x_2} \cdot @_1 \cdot \lambda\overline{x}$ for $r \geq 0$, where $@_r$ denotes the parent node of the redex argument, and $\lambda\overline{x}$ denotes x's binder, such that λx is the ith lambda in the bulk lambdas $\lambda\overline{x}$ for $i \geq 0$.*

(ii) *The argument of x is precisely given by the i_kth child of node $@_k$ where k is the smallest index verifying $i_k < |@_k|$ where: $i_1 = i$ and $i_{j+1} = i_j - |@_j| + |\lambda\overline{x_{j+1}}|$ for $j \geq 0$.*

Traversal bisimulation. We define $\mathcal{T}rav^\dagger(M)$ as the subset of traversals of $\mathcal{T}rav^{\mathsf{norm}}(M)$ ending with either a structural node or an external ghost variable (*i.e.*, not ending with an internal ghost variable nor a ghost lambda). Observe that every traversal in $\mathcal{T}rav^{\mathsf{norm}}$ has a traversal extension in $\mathcal{T}rav^\dagger$. Indeed, if $t \in \mathcal{T}rav^{\mathsf{norm}}$ ends with an internal ghost variable or a ghost lambda then repeatedly applying rules (Var) and (Lmd) yields a traversal ending with an external node.

Definition 7.4 (Traversal bisimulation). Let M and N be two terms with variables in $\mathcal{V}$. Let $\phi \colon \mathcal{J}(N) \to \mathcal{J}(M)$ be a function from the justified sequences of N to justified sequences of M. We define the state transition system $(X, R_\phi, \to_X)$ as:

- *States*: X is the disjoint union $\mathcal{T}rav^\dagger(M) + \mathcal{T}rav^\dagger(N)$.
- *Binary relation $R_\phi \subseteq X \times X$* defined for any traversals $t, u \in X$ as: $t \, R_\phi \, u$ just if $t \in \mathcal{T}rav^\dagger(M)$, $u \in \mathcal{T}rav^\dagger(N)$ and $\phi(u) = t$.
- *Transitions* For all $t, t' \in \mathcal{T}rav^\dagger(M)$ we write $t \to_M t'$ just if t' is a minimal extension of t that remains in $\mathcal{T}rav^\dagger(M)$ (*i.e.*, t' extends t using rules of Table 2 and any occurrence strictly between the last occurrence in t and t' is an either internal ghost variable or a ghost lambda node.) We define $\to_N$ identically and write $t \to_X t'$ if either $t \to_M t'$ or $t \to_N t'$ holds for $t, t' \in X$.

M and N are ϕ-**bisimilar** if R_ϕ defines a bisimulation over X with respect to $\to_X$, that is:

36

- (B1) for all $t_1, t_2 \in \mathcal{T}rav^\dagger(M)$, $u_1 \in \mathcal{T}rav^\dagger(N)$ with $t_1\ R_\phi\ u_1$ and $t_1 \to_M t_2$ there is $u_2 \in \mathcal{T}rav^\dagger(N)$ such that $u_1 \to_N u_2$ and $t_2\ R_\phi\ u_2$.
- (B2) for all $t_1 \in \mathcal{T}rav^\dagger(M)$, $u_1, u_2 \in \mathcal{T}rav^\dagger(N)$ with $t_1\ R_\phi\ u_1$ and $u_1 \to_N u_2$ there is $t_2 \in \mathcal{T}rav^\dagger(M)$ such that $t_1 \to_M t_2$ and $t_2\ R_\phi\ u_2$.

Lemma 7.5. *If M and N are ϕ-bisimilar; the restriction of ϕ to $\pi(\mathcal{T}rav^\dagger(N))$ is injective; and there is a mapping $\rho : \mathcal{V}^*(N) \to \mathcal{V}^*(M)$ such that for any variable names $\alpha \in \mathcal{V}^*(N)$, ϕ and π_α commute modulo ρ: (i.e., for all u, $\pi_{\rho(\alpha)}(\phi(u)) = \phi(\pi_\alpha(u))$), then $\pi(\mathcal{T}rav(N)) \cong \pi(\mathcal{T}rav(M))$.*

Proof. Consider ϕ as a function from $\pi(\mathcal{T}rav^\dagger(N))$ to $\pi(\mathcal{T}rav^\dagger(M))$. It is *well-defined*: $\mathcal{T}rav^\dagger(N)$ is by definition the transitive closure of the empty traversal by $\to_N$, so for any traversal u of $\mathcal{T}rav^\dagger(N)$, repeatedly applying (B2) for each step of its $\to_N$ derivation shows that $t = \phi(u) \in \mathcal{T}rav^\dagger(M)$. By commutativity of π and ϕ this yields $\pi(t) = \phi(\pi(u)) \in \pi(\mathcal{T}rav^\dagger(M))$. Similarly, it is *surjective* by (B1) because $\mathcal{T}rav^\dagger(M)$ is the transitive closure of the empty traversal by $\to_M$. Finally it is *injective* by assumption. Hence we have $\pi(\mathcal{T}rav^\dagger(N)) \cong \pi(\mathcal{T}rav^\dagger(M))$, and because π eliminates internal nodes from traversals, we also have $\pi(\mathcal{T}rav(N)) \cong \pi(\mathcal{T}rav(M))$. $\square$

The sets of normalizing traversals before and after linear reduction are identical, up to some mapping between nodes of the term and its reduct.

Proposition 7.6 (Traversals are sound for linear reduction). *Let M be an untyped term. If $M \to_{ll} N$ then $\pi(\mathcal{T}rav^{\mathsf{norm}}(M)) \cong \pi(\mathcal{T}rav^{\mathsf{norm}}(N))$.*

Proposition 7.7 (Traversals are sound for trivial reduction). *Let M be an untyped term containing only trivial redexes, and N the reduct obtained by firing the* leftmost spinal-innermost *trivial redex, then $\pi(\mathcal{T}rav^{\mathsf{norm}}(M)) \cong \pi(\mathcal{T}rav^{\mathsf{norm}}(N))$.*

Example 7.1. The term $M = \lambda x.(\lambda yz.z)u$ from Example 2.1 trivially reduces to $N = \lambda xz.z$. Take the traversal t from Example 2.1, then we have $\pi(t) = \overset{2}{\overset{\frown}{\lambda x z}} \cdot \theta$ which is equivalent to $\overset{2}{\overset{\frown}{\lambda x^{[z]}}} \cdot \theta$, itself equivalent to traversal $\overset{2}{\overset{\frown}{\lambda x^{[z]}}} \cdot z$ in N.

Corollary 7.8. *If M is in* qnf *and has beta-normal form N then $\pi(\mathcal{T}rav^{\mathsf{norm}}(M)) \cong \pi(\mathcal{T}rav^{\mathsf{norm}}(N))$.*

Proof. By Proposition 6.6(ii) *qnf* trivially reduce to their beta-nf: there is a reduction sequence $M = M_1 \to_\beta M_2 \ldots \to_\beta M_q = N$, $q \geq 1$ where in each reduction step, M_{i+1} is obtained by reducing the leftmost spinal-innermost standard redex of M_i (of the form $(\lambda x.U)T$ such that U does not contain any redex). By Proposition 6.6(i), each such redex is trivial. By Proposition 7.7 the set $\pi(\mathcal{T}rav^{\mathsf{norm}})$ is an invariant (up to isomorphism) throughout this trivial reduction sequence, which yields the desired result. $\square$

Proposition 7.9. *Let M be an untyped term with a beta-normal form. Then (i) All traversals in $\mathcal{T}rav^{\mathsf{norm}}(M)$ are finite; (ii) The set $\mathcal{T}rav^{\mathsf{norm}}(M)$ is finite.*

Proof. (i) If M has a normal form then by Theorem 6.10 its linear reduction sequence yields a quasi-normal form Q and by Theorem 6.7 Q and M are beta-equivalent. Thus by Prop. 7.6 the sets of traversals of M and Q are isomorphic. So if M has an infinite traversal then so does Q, which contradicts Proposition 7.3. (ii) Traversals rules defining $\mathcal{T}rav^{\mathsf{norm}}$ all have bounded non-determinism therefore by (i) $\mathcal{T}rav^{\mathsf{norm}}$ is finite. $\square$

Proposition 7.10. *Algorithm C terminates.*

Proof. By Proposition 7.9 there is a finite number of traversals in $\mathcal{T}rav^{\mathsf{norm}}$ and each of them can be obtained with finitely many applications of traversal rules. □

Theorem 7.11 (ULC Paths Characterization). *Let M be an untyped term with normal form T.*

$$\mathcal{T}rav^{\mathsf{norm}}(M)/\sim \; \cong \; \mathcal{P}aths(T) \; .$$

Proof. Let's first assume that M is in beta-normal form. Then its computation tree cannot contain any @-node and all nodes must be external. Hence the arity threshold of a traversal ending with external variable x is precisely $|x|$, the arity of x. A trivial induction on the traversal rules shows, that since the (Var) is never used, we necessarily have $\mathcal{T}rav^{\mathsf{norm}}(M) = \mathcal{P}aths(M)$. Further, since all nodes are externals, the projection π and $\ulcorner \_ \urcorner$ functions both coincides with the identity function thus $\ulcorner \pi(\mathcal{T}rav^{\mathsf{norm}}(M)) \urcorner = \mathcal{T}rav^{\mathsf{norm}}(M) = \mathcal{P}aths(M)$.

Otherwise, since M has a beta-nf, by Theorem 6.10 its *leftmost linear reduction sequence* terminates and yields a *qnf*. By Prop. 7.6 the set of traversals is preserved by *linear reduction* (up to an isomorphism) thus the set of traversals of the *qnf* is isomorphic to the set of traversals of M. Corollary 7.8 permits us to conclude. □

Soundness. of the normalization procedure follows from Theorem 7.11 and the fact that a term is uniquely characterized by its set of justified paths $\mathcal{P}aths$ (Property 2.1).

8. Conclusion and further directions

We presented a novel method to evaluate untyped lambda terms by generalizing the theory of traversals, originally introduced in a typed setting, to the untyped lambda calculus. We introduced the leftmost linear reduction, a generalization of the non-standard *head-linear reduction* strategy[10], and showed soundness of the evaluation procedure by showing that traversals implement leftmost linear reduction.

Connection with Berezun-Jones traversals of ULC

Daniil Berezun and Neil Jones were first to introduce a notion of traversals for the untyped lambda calculus [2]. Their presentation has notable differences with ours. Firstly, they require two types of pointers: binding pointers and control pointers, instead of a single justification pointer for imaginary traversals. The control pointers would be what Danos-Regnier call the "price to pay for having no stacks or environments" in their definition of the argument lookup transition of the Pointer Abstract Machine[10].

In imaginary traversals, however, node occurrence have at most one justification pointer. Instead, we pay for the absence of stacks and environment by introducing ghost occurrences. In particular, we performs the necessary "gymnastics to find the argument of a subterm"[10] by traversing the 'pending argument lookup' (Def. 7.1) consisting of ghost occurrences, until a structural node gets materialized[2]

---

[2]The ghost materialization loop implemented by rule ($\mathsf{Lam}^{\lambda\!\lambda}$) and (Var) essentially calculates the quantity "$|\alpha| + i - |m|$" at each iteration, which is the exact same quantity $(r - a + l)$ used in step 2. (b) of the PAM transition from [10].

Another difference is that Berezun-Jones define a single traversal for a given term, whereas our definition involves one traversal for every path in the resulting normal form. The 'read-out' algorithm to reconstruct the normal form from the traversals (to be defined in a forthcoming paper by the same authors) involves recursively applying rewriting rules to the traversal. In contrast, our read-out algorithm reconstructs the tree representation of the normal from from tree-path extracted from each maximal traversal.

The connection between the two presentations lies in their respective correspondence with head-linear reduction: we have shown that imaginary traversals essentially implement the recursive application of the head-linear reduction. Berezun and Jones established a similar correspondence in their forthcoming paper.

Game semantics connection

In the typed setting, Danos *et al.* showed the connection between game models of typed lambda and *head-linear reduction* and proposed *Pointer Abstract Machine (PAM)* as an abstract machine implementation of this reduction [9]. More recently, the theory of traversals [13] was shown to correspond to Game Semantics in a very concrete way for typed languages like the simply-typed lambda calculus, PCF and Idealized Algol [8]: the set of traversals is in one to one correspondence with the game denotation of a term where all internal moves are revealed. We conjecture that such Game Semantic correspondence can be generalized to untyped terms using the game model of the untyped lambda calculus introduced in Andrew Ker's thesis [11] which denotes lambda-terms by *effectively almost everywhere copycat* strategies (Conjecture 3.1).

Acknowledgement

[1] S. Abramsky. Algorithmic game semantics: a tutorial introduction. In *Proceedings of 2001 Marktoberdorf International Summer School*, 2001.

[2] D. Berezun and N. Jones. Compiling, untyped lambda calculus to lower-level code by game semantics and partial evaluation. *Galop*, 2017.

[3] D. Berezun and N. D. Jones. Partial evaluation and normalization by traversals. pages 452–461, January 2016. Galop 2016.

[4] W. Blum. A tool for constructing structures generated by higher-order recursion schemes and collapsible pushdown automata. *Technical report*, 2007.

[5] W. Blum. A concrete presentation of game semantics. *Galop*, 2008.

[6] W. Blum. Game semantic traversal evaluator (HOG). `https://github.com/blumu/dphil.tools`, 2008. Accessed: 2018-02-26.

[7] W. Blum. Local computation of beta-reduction - a concrete presentation of game semantics. *Technical Report - Oxford University Computing Laboratory*, 2008.

[8] W. Blum. *The Safe Lambda Calculus.* PhD thesis, University of Oxford, 2009. https://ora.ox.ac.uk/objects/uuid:537d45e0-01ac-4645-8aba-ce284ca02673.

[9] V. Danos, H. Herbelin, and L. Regnier. Game semantics and abstract machines. In *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*, pages 394–405, Jul 1996.

[10] V. Danos and L. Regnier. Head linear reduction. submitted for publication, 2004.

[11] A. D. Ker. Innocent game models of untyped lambda-calculus. *DPhil Thesis, Oxford University*, 2000.

[12] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *Logic in Computer Science, 2006 21st Annual IEEE Symposium on*, pages 81–90, 12-15 Aug. 2006.

[13] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *Proceedings of IEEE Symposium on Logic in Computer Science.*, pages 81–90. Computer Society Press, 2006. Extended abstract.

[14] C. L. Ong. Normalisation by traversals. *CoRR*, abs/1511.02629, 2015.

A. Tables
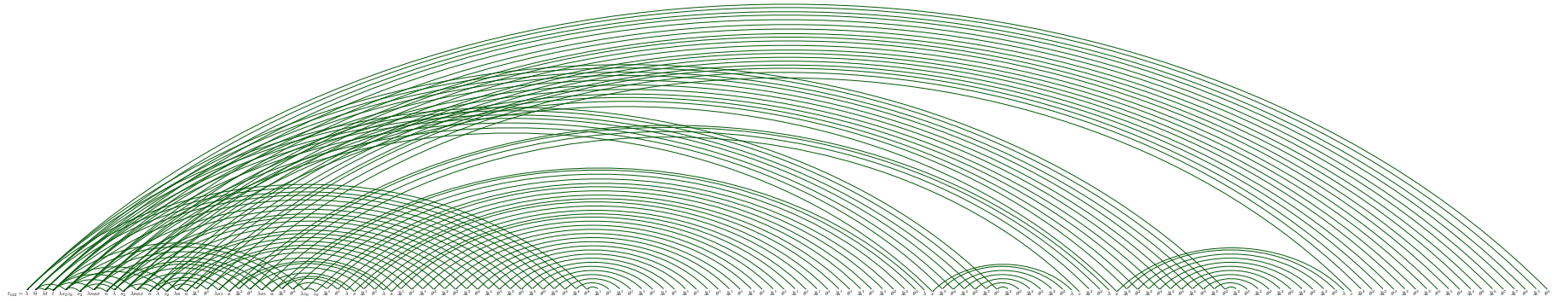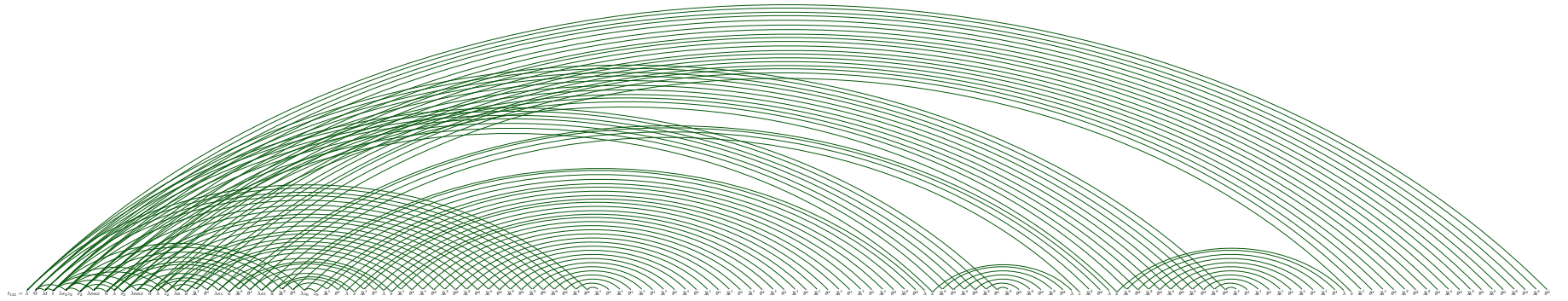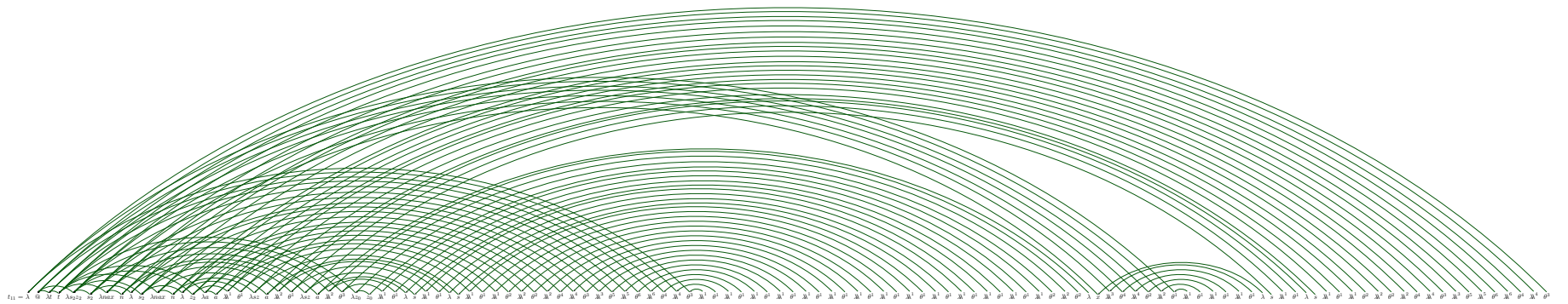
Table 3: Maximal traversals of *varity* 2 from Example 5.5

B. Proofs

Proof of Proposition 3.4. The tree $CT(M)$ being by definition a subtree of $CT(M^t)$, induces a one-to-one mapping from $\mathcal{N}(M)$ to $\mathcal{N}(M^t)$. We extend it to ghost nodes by mapping ghost occurrences in t to the corresponding node resulting from the eta-expansions of subterms of M in M^t, and all other ghost nodes not occurring in t to the corresponding ghost nodes in $CT(M^t)$. Formally, we define η_t by induction on the traversal t, observing that in rules (Var) and (IVar), the on-the-fly eta-expansion of Definition 3.2 increments the arity of the justifier node m, which guarantees that the corresponding structural node exists in $CT(M^t)$.

(i) By induction on u. Reusing the rule used to traverse u to traverse $\eta_t(u)$. For the case (Var) and (IVar), when the *eta-expanded* subcase of the rule is used to traverse a ghost variable that also appeared in t, we use instead the *concrete* sub-case used to traverse $\eta_t(u)$.

(ii) By induction on v, applying to u each rule that is applied to v. For the case (Var) and (IVar), if the index i is greater than the arity of the node m in M then the lambda node from v gets replaced by a ghost lambda node in u.

(iii) By (i) the function is well defined, it is injective by definition, and by (ii) it is surjective. $\square$

Proof of Proposition 4.2. Let $t \in \mathcal{T}rav$ and let's first assume that t does not contain any ghost occurrence. We show by strong induction on t that there is a subsequence $u \in \mathcal{T}rav^{\mathsf{branch}}$ of t such that $\ulcorner \pi(t) \urcorner = \ulcorner \pi(u) \urcorner$, and so in particular $\ulcorner \pi(t) \urcorner \equiv \ulcorner \pi(u) \urcorner$. We do a case analysis on the last node t^ω of t.

- $t^\omega \in \mathcal{N}^{\mathsf{ext}}$: Suppose t^ω is a variable or an @ node then we can conclude immediately from the induction hypothesis on $t_{<t^\omega}$ and using the rules (Lmd) of $\mathcal{T}rav^{\mathsf{norm}}$.

  Suppose t^ω is a lambda node. If it has no justifier then it is the root in which case we conclude by taking $u = t = t^\omega$. Otherwise let m denote t^ω's justifier in t. By the I.H. on $t_{\leq m}$ there is $u' \in \mathcal{T}rav^{\mathsf{norm}}$ such that $\pi(u') = \pi(t_{\leq m})$. Take $u = u' \cdot t^\omega$ where t^ω points to its immediate predecessor m. Then u is clearly a $\mathcal{T}rav^{\mathsf{norm}}$-traversal by rule (IVar) and:

$$
\begin{aligned}
\ulcorner \pi(u) \urcorner &= \ulcorner \pi(u') \urcorner \cdot t^\omega && \text{Def. of } \pi \\
&= \ulcorner \pi(t_{\leq m}) \cdot t^\omega \urcorner && \text{By I.H. on } t_{\leq m} \\
&= \ulcorner \pi(t)_{\leq m} \cdot t^\omega \urcorner && \text{By Prop. 4.1 since } m \text{ is necessarily followed in } t \text{ by} \\
& && \text{a } \lambda\text{-node in } \mathcal{N}^{\mathsf{ext}}.
\end{aligned}
$$

  Now because m justifies t^ω, by Def. of P-view, *up to renaming of lambda variables* the sequences $\ulcorner \pi(t)_{\leq m} \cdot t^\omega \urcorner$ and $\ulcorner \pi(t) \urcorner$ are equal: $\ulcorner \pi(t)_{\leq m} \cdot t^\omega \urcorner \equiv \ulcorner \pi(t) \urcorner$. But because m is a variable node, its label is kept untouched by the transformation π and therefore the equality holds.

- $t^\omega \notin \mathcal{N}^{\mathsf{ext}}$: Let n be the last occurrence in t that is in $\mathcal{N}^{\mathsf{ext}}$, and let $m_1 \ldots m_q$, $q > 0$ be the occurrences of internal nodes following n in t (so that $m_q = t^\omega$). By definition of the traversal rules, a variable in $\mathcal{N}^{\mathsf{ext}}$ is necessarily followed by a lambda node in $\mathcal{N}^{\mathsf{ext}}$, therefore n is necessarily a lambda node. By definition of π, n is therefore also the last occurrence in $\pi(t)$ therefore $\pi(t) = \pi(t)_{\leq n}$. But *up to relabelling*, $\pi(t)_{\leq n} = \pi(t_{\leq n})$; more precisely, $\pi(t)_{\leq n}$ is obtained from $\pi(t_{\leq n})$ by pre-pending to n's label the stack of pending lambdas of $t_{>m}$ (the internal nodes following n in t). Applying the induction hypothesis on $t_{\leq n}$ gives $\pi(t_{\leq n}) = \pi(u')$ for some $u' \in \mathcal{T}rav^{\mathsf{norm}}$. To conclude it therefore suffices that to show that $u = u' \cdot m_1 \ldots m_q$ is also a traversal of $\mathcal{T}rav^{\mathsf{norm}}$.

42

We prove by finite induction on q that $u_{\leq m_q} \in \mathcal{T}rav^{\mathsf{norm}}$ and m_q's justifying node and label are same in u and t. For $q = 0$, we just apply rule (Lam) on u'. For $q > 0$: by case analysis on the rule used to visit m_q in t. For structural rules (Root), (App) and (Lam) it follows immediately by induction. Rule (Var): If $m_q \in \mathcal{N}_{\mathsf{var}}$ then by the Path-View correspondence, $\ulcorner u_{<m_q} \urcorner$ is a path in the tree from the root to m_q therefore m_q's binder necessarily occur in u, we can therefore conclude using the I.H. on $u_{<m_q}$ and applying (Var) on $u_{<m_q}$ to get $u_{\leq m_q}$.

Suppose that t contains ghost occurrences then we consider the term M^t. By Prop. 3.4(i), $\eta_t(t)$ is a traversal of M^t. Hence by the above, there exists $u' \in \mathcal{T}rav^{\mathsf{norm}}(M^t)$ such that $\ulcorner \pi(u') \urcorner = \ulcorner \pi(\eta_t(t)) \urcorner$. By Prop. 3.4(ii), there exists $u \in \mathcal{T}rav(M)$ such that $u' = \eta_t(u)$. Recall that the normalizing traversals is defined as the subset of traversals where external lambda nodes always point to their immediate predecessor. Therefore since η_t is pointer-preserving u must necessarily belong to $\mathcal{T}rav^{\mathsf{norm}}(M)$.

By definition of π we have $\pi(t) = \pi(\eta_t(t))$, and since u is a subsequence of t we also have $\pi(u) = \pi(\eta_t(u))$. Hence $\ulcorner \pi(t) \urcorner = \ulcorner \pi(\eta_t(t)) \urcorner = \ulcorner \pi(u') \urcorner = \ulcorner \pi(\eta_t(u)) \urcorner = \ulcorner \pi(u) \urcorner$. $\qquad\square$

Proof of Proposition 4.7. (i) By the alternation property of traversals, the last strand of t consists of a succession of lambda nodes α_q and variable or @-nodes n_q, with indices going from k down to 1.

We show by finite induction on $1 \leq q \leq k$ that the first $2k$ nodes after t^ω are successive pairs of ghost lambda and ghost variable nodes justified in order by $n_1, \alpha_1, n_2, \alpha_2, n_3, \ldots, n_k, \alpha_k$ with respective labels $i_1, i_2, i_2, i_3, i_3, \ldots, i_k, i_k, i_{k+1}$ defined by:

$$\begin{cases} i_1 = i \\ i_{q+1} = i_q + |\alpha_q| - |n_q|, \; 1 \leq q \leq k. \end{cases}$$

- Base case $q = 1$: Because t^ω is a ghost variable node hereditarily justified by the root, the only next rule that can be applied is ($\mathsf{IVar_{branch}}$), and by assumption the following node is a ghost lambda node justified by t^ω with label $i_1 = i$. Then by rule ($\mathsf{Lam^{\lambda\!\lambda}_{branch}}$) the next node is a ghost variable justified by α with label $i_2 = i + |\alpha_1| - |t^\omega|$.

$$t_{\max} = \; \cdots \; \alpha_1 \quad t^{\omega} \quad \lambda\!\lambda \quad \theta \; \cdots$$

- Let $1 < q < k$, applying the induction hypothesis on q gives:

$$t_{\max} = \; \cdots \; \alpha_q \quad n_q \quad \alpha_{q-1} \quad n_{q-1} \cdots t^{\omega} \; \cdots \; \lambda\!\lambda \quad \theta \; \cdots$$

We then have $i_q > |n_q|$ since:

$$i_q = i - \sum_{j=1..q-1} \delta_j \qquad \text{(by induction hypothesis)}$$

$$> \mathsf{arth}(t) - \sum_{j=1..q-1} \delta_j \qquad \text{(assumption } i > \mathsf{arth}(t))$$

$$= \max_{r=1..k-1} \left(|n_r| + \sum_{j=1..r-1} \delta_j \right) - \sum_{j=1..q-1} \delta_j \qquad \text{(definition of } \mathsf{arth})$$

$$\geq |n_q| + \sum_{j=1..q-1} \delta_j - \sum_{j=1..q-1} \delta_j \qquad \text{(take } r = q)$$

$$= |n_q|$$

By definition of the strand, α_{q-1} is hereditarily justified by an @ node. Since $i_q > |n_q|$, by rule ($\mathsf{Var_{branch}}$) the next node is necessarily a ghost lambda node justified by n_q with label i_q. With rule ($\mathsf{Lam}^{\lambda}_{branch}$) the following node is a ghost variable node θ justified by α_q and labelled by $i_{q+1} = i_q + |\alpha_q| - |n_q|$.

(ii) By (i) the next strand following t consists solely of ghost nodes and ends with a ghost variable θ hereditarily justified by the root. Since ghost nodes have arity 0, the arity threshold at that point is 0. Hence, any label value j chosen to extend the traversal at that point will be strictly greater than the arity threshold: thus by (i) the next strand consists solely of ghost nodes. Applying the argument repeatedly shows that all the occurrences following t are necessarily ghost nodes.

(iii) Take r to be the smallest $q \geq 1$ such that $i_q \leq |n_q|$, it exists because:

$$i \leq \mathsf{arth}(t) \iff i \leq \max_{q=1..k-1} \left(|n_q| + \sum_{j=1..q-1} \delta_j \right) \quad \text{(Definition of } \mathsf{arth}(t))$$

$$\implies i \leq |n_r| + \sum_{j=1..r-1} \delta_j \quad \text{for some } 1 \leq r \leq k-1$$

$$\iff i - \sum_{j=1..r-1} \delta_j \leq |n_r| \quad \text{for some } 1 \leq r \leq k-1$$

$$\iff i_r \leq |n_r| \quad \text{for some } 1 \leq r \leq k-1 \quad \text{(Definition of } i_r).$$

We then conclude by applying the same argument as (i) for all $q < r$. $\qquad\qquad\square$

Proof of Proposition 6.6. (i) By induction on M. Suppose $M = (\lambda x.N)A$. By assumption N is in beta-normal form therefore if x occurs freely in N it is involved in the generalized redex $(\lambda x, A)$ of M and by bottom-up evaluation of *lloc* this gives $lloc_M(M) = x$ which contradict the assumption that M is in *qnf*.

Suppose $M = \lambda z.U$ then U is in *qnf* by Property 6.4(iv) and the result follows by the induction hypothesis since the redexes of M are those of U and the free variables of M are a subset of those of U.

Suppose $M = UV$. We consider the three sub-cases depending on where the redex is located within UV:

(1) The redex $(\lambda x.N)A$ is in U. Since M is in *qnf*, by Property 6.4(iii) so is U, we can thus conclude by the induction hypothesis.

(2) The redex $(\lambda x.N)A$ is in V. If $\lambda_l(U) = \epsilon$ then by definition of *lloc* we have $lloc_M(UV) = lloc_M(V) = \bot$ so we conclude by the induction hypothesis. If $\lambda_l(U) \neq \epsilon$, then either U contains a redex or U is an abstraction. But U cannot be a redex since by assumption N does not contain any redex, and U being abstraction would make $M = UV$ itself a redex which would contradict the fact that redex $(\lambda x.N)A$ in V is the leftmost redex in M.

(3) $U = \lambda x.N$ and $V = A$. This is the base case of the induction already treated above.

(ii) By (i) each reduction of the leftmost spinal-innermost reduction strategy is trivial and it is easy to see that *lloc* is preserved by trivial reduction, therefore each term in the reduction sequence is in *qnf*. Since a trivial reduction cannot possibly create new β-redexes, the reduction sequence necessarily terminates and yields the beta-normal form.

$\square$

Proof of Theorem 6.10. Suppose M has a beta-normal form then by a standard result of lambda calculus its head-reduction terminates, therefore by Theorem 6.1 the $\rightarrow_{hl}$-reduction of M terminates and yields its quasi-*head*-normal form. Since $\rightarrow_{hl}$ coincides with $\rightarrow_{ll}$, for terms not in *qhnf*, we can without loss of generality assume that M is in *qhnf*. We prove by induction on the *size* of M that

(H1) If M has a beta-nf then its linear reduction sequence terminates.

For the variable case, $\rightarrow_{ll}$ trivially terminates. If $M = \lambda x.U$ then U must also admit a beta-nf so we can conclude by induction. Suppose $M = UV$. If M is in *qnf* then we conclude immediately. Otherwise, M must have a *lloc* occurrence that is distinct from the *hoc* occurrence.

(i) Suppose that the *lloc* of M is in U ($lloc_M(U) \neq \bot$). We show by finite *structural* induction on subterms T of U that:

(H2) If $lloc_M(T) \neq \bot$ and $hoc(M)$ is in T then there are at most a finite number of steps in the linear reduction sequence of M that involve a linear substitutions in (reducts of) T.

- T is a variable. Since M's *hoc* is in T, it must be T itself. Because M is in quasi-head-normal form, the *hoc* is not involved in any generalized redex and therefore T is not involved in any substitution of the linear reduction sequence of M.
- $T = \lambda x.T'$ then the *hoc* and *lloc* must be in T' and we can conclude by the induction hypothesis (H2).
- $T = T_1 T_2$. By definition, the *hoc* is necessarily in T_1.

(a) Suppose that *lloc* is in T_2. Then we have $lloc_M(T_1) = \bot$ therefore none of the steps in the linear reduction sequence of M will involve substitutions in T_1 (there is no variable in T_1 involved in a general redex therefore subsequent linear reduction steps can only introduce general redexes on the right of T_1).

- Suppose T_1 is a variable x then it must be the *hoc* and we have $T = x\, T_2$.
  We show the result by induction (H3) on the number of pending arguments of the *hoc*.
  - Base: z has a single argument which must be T_2.
    By Theorem 6.1, reducing all the prime redexes in M yields its head-normal form with x as the head variable. Let γ denotes the sequence of substitutions performed on M to reduce the prime redexes, so that $\gamma(M)$ is the head normal form of M. Then $\gamma(T_2)$

45

will becomes the (unique) argument of x in the head normal form. And because M has a beta-nf, the term $\gamma(T_2)$ must also have a beta-nf.

Because z has a single hoc argument, all the subsequent linear substitutions in M must necessarily occur under T_2 and its reducts: the *lloc* will never get out of T_2. (Indeed, otherwise by the inductive definition of *lloc* there would be a subterm $U'V'$ of M where U' contains zT_2 and $\lambda_l(U;) \neq \epsilon$, which by definition of Args would imply that V' is another hoc argument, contradicting the assumption that the hoc has a single argument.)

Therefore if we consider the term M' obtained by replacing the subterm T by just T_2, the $\rightarrow_{ll}$-reduction sequence of M coincides with that of M'.

Now M' must necessarily have a beta-normal form. Otherwise it's standard beta-reduction sequence would not terminate, which would imply that $\gamma(T_2)$ does not terminate, which contradicts the fact that $\gamma(T_2)$ is the argument of the head variable in the head normal form!

Thus, since M' is by definition smaller than M,by the induction hypothesis H1, its linear-reduction sequence terminates. We can then conclude since we have shown that the linear reduction sequences of M and M' coincide.

  – Inductive case: suppose the result holds for terms with $n \geq 1$ *hoc* arguments, and suppose $\mathsf{Args}(M) = n + 1$.

   Let $U_{n+1}V_{n+1}$ denote the subterm of M such that V_{n+1} is the last *hoc* argument of M. We consider the term M' obtained from M by removing the last *hoc* argument of M (*i.e.* replacing $U_{n+1}V_{n+1}$ by U_{n+1}). Observe that the last argument of the hoc cannot be possibly be argument of any generalized redex, therefore the linear reduction of M must coincide with that of M' up until the *hoc* of the reduct of M lies under the last *hoc* argument U_{n+1}.

   By induction hypothesis (H3), the $\rightarrow_{ll}$-reductions sequences of M' involves a finite number of substitutions under T_2. Consequently, the same holds in M.

 • Suppose T_1 is not a variable. Let z denote a fresh variable in M, and $M' = M[z/T_1]$ be the term obtained by replacing the subterm T_1 in M by the fresh variable z.

  Since T_1 is not a variable, M' must be strictly smaller than M. And since M has a beta-nf, so must M', thus by the induction hypothesis (H1), the $\rightarrow_{ll}$ reduction sequence of M' terminates. Consequently, there is only a finite number of steps in the linear reduction of M' that involve substitutions in reducts of $z\,T_2$.

  Clearly, the linear reduction steps under T_2 in M' coincide with those of T_2 in M. Hence, we can conclude that at most a finite number of substitutions take place under $T = T_1 T_2$ in the linear reduction sequence of M.

(b) Suppose that *lloc* is in T_1. Then by the induction hypothesis (H2) there is a finite number of steps in the linear reduction sequence of M that involve (reducts of) T_1. Let us consider the first reduct R of M in the $\rightarrow_{ll}$ reduction sequence such that the *lloc* is not under a reduct of T_1. Observe that in the linear reduction sequence of T, the subterm T_2 remains unmodified as long as the *lloc* remains in (the reduct) of T_1.

At R, if the *lloc* is not under T_2 then none of the further steps in the linear reduction sequence of R will involve T_2 (*i.e.*, subsequent substitutions will occur outside of T – on its right, more precisely). We can therefore conclude.

If instead at R the *lloc* lies under the subterm T_2: $lloc_R(T2) \neq \bot$, then T_1 cannot be a variable, otherwise it would be both the *hoc* and the *lloc* of M which contradict the

assumption. We can therefore conclude by applying the same reasoning as case (a) above to the reduct R.

(ii) Suppose that the lloc is in V. Then by definition of the *lloc* for the application case, we must have $lloc_M(U) = \bot$ and $\lambda_l(U) = \epsilon$. Hence U is a term in *qnf* and by Proposition 6.6 it must have a beta-normal form. By Property 6.9 U's normal form is not an abstraction. Hence since M has a normal form so must V and we have $\beta_{\mathsf{nf}}(M) = \beta_{\mathsf{nf}}(U)\beta_{\mathsf{nf}}(V)$. By the induction hypothesis (H1) the $\rightarrow_{ll}$-reduction of V must terminate. We can then conclude by observing that the linear-substitutions performed in the linear-reduction sequence of M map one-to-one to those in the linear-reduction sequence of V (*i.e.*, if $V \rightarrow_{ll} V_2 \rightarrow_{ll} \ldots V_i$ then $M = UV \rightarrow_{ll} UV_2 \rightarrow_{ll} \ldots UV_i$ for some $i \geq 0$)

$\square$

Proof of Proposition 7.2. By case analysis on the current state:

(Init) The first strand is obtained by applying the rule (Root) followed by repeated applications of (App) and (Lam), which yields u, a path from the root $\lambda\overline{\eta}$ to some variable node x in the tree of M, so that u is a spinal descent consisting only of lambda nodes and application nodes. Suppose that x is an internal variable then by definition it is involved in some generalized redex and therefore we have $lloc_M(\lambda\eta) = x$, but since M is in *qnf* we have $lloc_M(\lambda\overline{\eta}) = lloc_M(M) = \bot$ which gives a contradiction. Hence x is an external variable which ends the first strand of type S.

(Structural) The previous strand is of form $\lambda\overline{\eta_1} \cdot u_1 \cdot \underline{x_1}$.

If the node following x_1 is a structural node $\lambda\overline{\eta_2}$ then by rule (IVar), it's a child lambda node of x_1. Because u_1 is a spinal descent, the subterm rooted at $\lambda\overline{\eta}_1$ is of the form $\lambda\overline{\eta}_1.x_1 A_1 \ldots A_q$ for some $q \geq 1$, therefore we have $lloc_M(\lambda\overline{\eta}_1.x_1 A_1 \ldots A_q) = \bot$. Since x_1 is external, by Property 6.3 we have $lloc_M(A_j) = \bot$ for all $1 \leq j \leq q$, so in particular $lloc_M(\lambda\overline{\eta_2}) = \bot$. The same logic used in the *Initialization* case lets us conclude that u_2 is a spinal descent to external node x_2. We have shown that the next strand is of type S.

Now suppose instead that the node following x_1 is a ghost node then by Property 7.1, u_1 is necessarily non-empty: we have $u_1 = @_1 \cdot \lambda\overline{\xi_1} \cdots @_q \cdot \lambda\overline{\xi_q}$ for some $q \geq 1$. The subterm at $@_r$, represented on Figure 2, is of the form

$$(\lambda\overline{\xi_r}.A_0)A_1 \ldots A_{k_r-1} \, (\lambda\overline{y_2}.\ldots)A_{k_r+1} \, \ldots A_{|@_r|}.$$

By Proposition 4.7(iii), x_1 is followed by a *pending parameter lookup* v necessarily shorter than u_1 and terminated by a structural lambda node $\lambda\overline{y_2}$ justified by some application node $@_r$ in u_1 for some $1 \leq r \leq q$:

$$t = \cdots \underline{\lambda\overline{\eta}} \cdot @_1 \cdot \lambda\overline{\xi_1} \cdots \cdot @_r \, \lambda\overline{\xi_r} \cdots \cdot @_q \cdot \lambda\overline{\xi_q} \cdot \underline{x} \cdot \lambdabar \cdot v \cdot \lambda\overline{y_2}$$
$$v = \theta_{k_q} \cdot \lambdabar_{k_q} \cdot \theta_{k_{q-1}} \cdot \lambdabar_{k_{q-1}} \cdots \cdot \theta_{k_{r-1}} \cdot \lambdabar_{k_{r-1}} \cdot \theta_{k_r}$$

47

where, for all i ranging from r to q, ghost variable θ_{k_i} and ghost lambda $\lambda\!\lambda_{k_i}$ point respectively to $\lambda\overline{\xi_i}$ and $@_i$ with label $k_i \geq 1$ defined by:

$$k_i = k - |x| + |\lambda\overline{\xi}_q| + \sum_{i \leq j < q} (|\lambda\overline{\xi}_j| - |@_{j+1}|)$$

$$= k - |x| + |\lambda\overline{\xi}_i| + \sum_{i < j \leq q} (|\lambda\overline{\xi}_j| - |@_j|)$$

Since all the nodes in v are ghost nodes, by Property 3.6 their justification label is greater than the justifier's arity; the opposite holds for $\lambda\overline{y_2}$ which is a structural node. Therefore:

$$k > |x| \tag{1}$$

$$k_i > |@_i| \quad \text{for all } r+1 \leq i \leq q \tag{2}$$

$$k_r \leq |@_r| \tag{3}$$

To prove that $lloc(\lambda\overline{y_2}) = \bot$ we first show that for all i ranging from $r+1$ to q we have $k_i > |\lambda_l(\lambda\overline{\xi_i})|$. We prove the result by a finite induction proceeding bottom-up from the lower tree node $\lambda\overline{\xi_q}$ up to $\lambda\overline{\xi_r}$.

Base case: We have $i = q$. The subterm rooted at $\lambda\overline{\xi_q}$ is in head normal form $\lambda\overline{\xi_q}.x \cdots$ therefore its head lambda-list consists precisely of the lambda abstractions $\lambda\overline{\xi_q}$, and thus $|\lambda_s(\lambda\overline{\xi_q})| = |\lambda\overline{\xi_q}|$. We have $k_q = k - |x| + |\lambda\overline{\xi_q}|$ and by Equation 1 we have $k > |x|$, hence $k_q > |\lambda\overline{\xi_q}|$.

Induction case: Let $r+1 \leq i \leq q$. We suppose that the result holds for i we show that it must hold for $i-1$. By definition, the head lambda-list at $\lambda\overline{\xi_i}$ is precisely given by the variables in the abstraction concatenated with the head lambda-list at $@_{i+1}$, thus $|\lambda_l(\lambda\overline{\xi_i})| = |\lambda\overline{\xi_i}| + |\lambda_l(@_{i+1})|$. The lambda-list at $@_i$, is by definition, obtained by popping $|@_i|$ lambdas from the lambda-list at $\lambda\overline{\xi_i}$; or is the empty list if the arity is greater than the number of pending lambdas. In terms of lengths this means:

$$|\lambda_l(@_i)| = \max(0, |\lambda_l(\lambda\overline{\xi_i})| - |@_i|)$$

$$= \max\left(0, |\lambda_l(@_{i+1})| + |\lambda\overline{\xi_i}| - |@_i|\right) \qquad \text{for } r \leq i \leq q-1.$$

Furthermore by definition of k_{i-1} we have:

$$k_{i-1} = k_i + |\lambda\overline{\xi_{i-1}}| - |@_i| \tag{4}$$

Hence,

$$
\begin{aligned}
|\lambda_l(\lambda\overline{\xi_{i-1}})| - k_{i-1} &= |\lambda\overline{\xi_{i-1}}| + |\lambda_l(@_i)| - k_{i-1} && (\text{Def of } \lambda_l)\\
&= |\lambda\overline{\xi_{i-1}}| + \max\left(0, |\lambda_l(\lambda\overline{\xi_i})| - |@_i|\right) - k_{i-1} && (\text{Def of } \lambda_l)\\
&= \max\left(|\lambda\overline{\xi_{i-1}}| - k_{i-1}, |\lambda\overline{\xi_{i-1}}| + |\lambda_l(\lambda\overline{\xi_i})| - |@_i| - k_{i-1}\right)\\
&= \max\left(|@_i| - k_i, |\lambda_l(\lambda\overline{\xi_i})| - k_i\right) && (\text{Eqn 4})
\end{aligned}
$$

By the induction hypothesis we have $|\lambda_l(\lambda\overline{\xi_i})| < k_i$, and by Equation 2 we have $|@_i| < k_i$, therefore the maximum of the two quantities in the last equation is negative which shows the desired result $|\lambda_l(\lambda\overline{\xi_{i-1}})| < k_{i-1}$.

Figure 2: Relevant sub-tree for a traversal strand of type G (assuming $q > r + 1$).

We now show that $lloc_M(\lambda\overline{y_2}) = \bot$. By assumption we have that $lloc_M(\lambda\overline{\eta}) = \bot$ so since $@_r$ occurs in the spine of M, by definition of $lloc_M$ we must have $lloc_M(@_r) = \bot$, and thus also $lloc_M(\lambda\overline{\xi_r}) = \bot$. We have just shown that $|\lambda_l(\lambda\overline{\xi_r}| < k_r$, consequently, the sub-term $(\lambda\overline{\xi_r}.A_0)A_1\ldots A_{k_r-1}$ has more operands than pending lambdas in the operator's lambda list. By definition of lambda list this implies that $\lambda_l((\lambda\overline{\xi_r}.A_0)A_1\ldots A_{k_r-1})$ is empty. Hence, we necessarily have $lloc(\lambda\overline{y_2}) = \bot$, otherwise by definition of $lloc$ we would have $lloc((\lambda\overline{\xi_r}.A_0)A_1\ldots A_{k_r-1}(\lambda\overline{y_2}\ldots)) = lloc(\lambda\overline{y_2}) \neq \bot$, which subsequently implies $lloc(\lambda\overline{\eta_1}) \neq \bot$, contradicting the assumption. The sequence u_2 is shown to be a spinal descent by the same argument used in the *structural argument* case above, using the fact that $lloc_M(\lambda\overline{y_2}) = \bot$. We have thus shown that the next strand is of type G.

(Ghost) If the node following x_1 is a structural node then, like we have shown in the previous case, the next strand is necessarily a spinal descent of type S.

If the node following x_1 is a ghost. Once again, the weaving property shows that the ghost node after x_1 is necessarily followed by a *pending parameter lookup* terminated by structural lambda node $\lambda\overline{y_2}$ pointing to some node in $v \cdot \lambda\overline{y_1} \cdot u_1$. But since v consists solely of ghost node, the justifier is necessarily in u_1. We can thus conclude with the same argument as in the previous case that the next strand is necessarily of type G.

$\square$

Proof of Lemma 7.4. (i) By definition of generalized redexes and lambda lists (Def. 6.1), $@_r$ necessarily occurs in the path from $\lambda\overline{x}$ to the root. Further the path from $@_r$ to $\lambda\overline{x}$ is necessarily a spinal descent (*i.e.*, it contains only lambda and application nodes, and no variable node). Indeed, if it contained a variable node z then the lambda-list at the subterm rooted at z would be empty, in which case no argument could possibly form a generalized redex with λx.

(ii) We show by induction on k that x is the variable abstracted by the i_kth lambda in $\lambda_l(\lambda\overline{x_k})$, which implies the result since by definition λx forms a generalized redex with the i_kth argument of $@_k$. Base case $k = 1$. By assumption, λx is the ith lambda in the bulk lambda $\lambda\overline{x}$. Let $k \geq 1$. By the induction hypothesis λx is the i_{k-1} lambda in $\lambda_l(\lambda\overline{x_{k-1}})$. substitution involved iBy definition of k we have $i_{k-1} \geq |@_{k-1}|$, therefore by definition of λ_l, the long-application $@_{k-1}$ has the effect of popping exactly $|@_{k-1}|$ arguments from the lambda-list, and the following abstraction $\lambda\overline{x_k}$ then adds $|\lambda\overline{x_k}|$ more lambdas in front of the lambda list. Hence λx gets moved to position $i_{k-1} - |@_{k-1}| + |\lambda\overline{x_k}| = i_k$ in the lambda list at $\lambda\overline{x_k}$. $\square$

Proof of Proposition 7.6. Suppose that $M \to_{ll} N$. Let x denote the leftmost linear variable occurrence in M and let $(\lambda x, A)$ be the generalized redex involving it, so that N is the term obtained after firing the generalized redex $(\lambda x, A)$. A property of linear head reduction is that the reduct contains the original term itself, modulo alpha conversion and relabelling of the *lloc* node. This induces an implicit map Φ from nodes of the reduct N to those of M:

$$\Phi : \tilde{\mathcal{N}}(N) \to \tilde{\mathcal{N}}(M)$$

which maps nodes from the substituted subterm in N to the corresponding nodes under argument A in M; and every node in N that is not involved in the substitution, to its counterpart in M.

The substitution involved in the leftmost linear reduction affects the computation tree in three possible ways depending on the structure and number of x's operands:

Case 1 x has at least one operand ($|x| > 0$) and A is an abstraction;

Case 2 x has at least one operand ($|x| > 0$) and A is not an abstraction;

Case 3 x is unapplied ($|x| = 0$).

We proceed by showing that for each of the three cases, there is an injection ϕ such that M and N are ϕ-bisimilar.

Case 1 The reduction is represented on Figure 3 where x has some operand ($|x| > 0$), and A is an abstraction $\lambda \overline{y}.R$.

Let $\lambda \overline{x}$ denote x's binding lambda node in M, i denote x's binding index, and $@_a$ denote the parent node of the subterm A.

After reduction: (i) Node x gets replaced by an application node $@_x$ and its existing children are preserved. (ii) The subterm $\lambda \overline{y}'.R'$, obtained from R by renaming variable afresh to avoid variable capture, becomes the operator child of $@_x$. (Note that we write $\lambda \overline{y}'$ to distinguish the root node of R' from the root $\lambda \overline{y}$ of R in N, even though the bound variable names are $\overline{y}$ for both nodes.)

We thus have $\Phi(@_x) = x$, $\Phi(\lambda \overline{y}') = \lambda \overline{y}$, and Φ maps each node from subterm $\lambda \overline{y}'.R'$ to the corresponding node under $\lambda \overline{y}.R$.
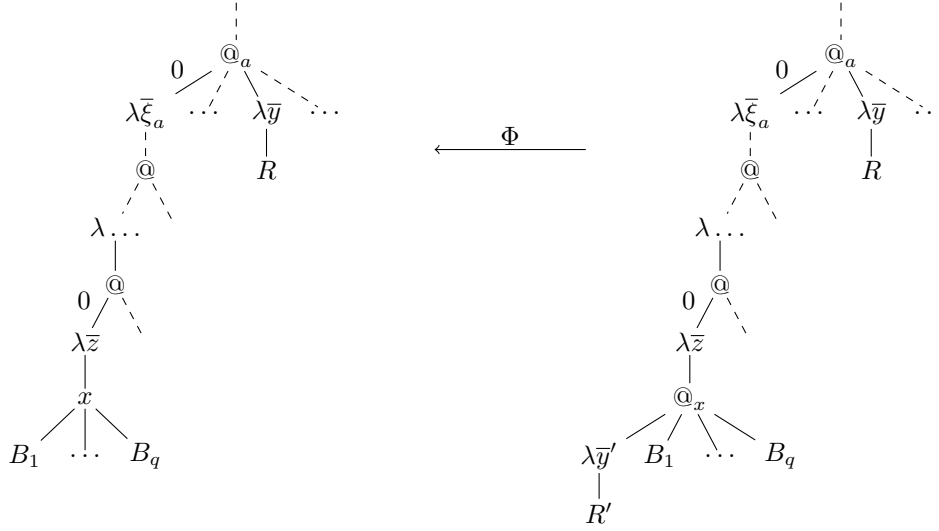

Figure 3: (Case 1) Reduction $M \to_{ll} N$ where node x has at least one operand, and A is a non-dummy abstraction $\lambda \overline{y}.R$. Linearly firing generalized redex $(\lambda x, A)$ for x in M (left) yields N (right) where R' is a copy of R with fresh names to avoid variable capture. Function Φ maps nodes from the right tree to the left tree.

Observe that Φ has the following properties:

(i) Φ preserves the parent-child relation for lambda and variable nodes: for any lambda or variable node n, the child nodes of n's image by Φ are the image of the n's children. (This is not true for application node $@_x$, however);

(ii) Φ is naming-consistent: any two variable nodes sharing the same name in N map to variable nodes with same name in M;

(iii) Φ preserves variable binding: it maps the binder lambda node of a variable node z in N to the binder in M of $\Phi(z)$;

(iv) hence Φ preserves the enabling relation: if $n_1 \vdash_i n_2$ in N for $i \geq 0$, $n_1, n_2 \in \mathcal{N}(N)$ then $\Phi(n_1) \vdash_i \Phi(n_1)$ as long as $(n_1, n_2) \neq (@_x, \lambda \overline{y}')$.

(v) Φ preserves node types: images of external and internal nodes are respectively external and internal nodes; images of variable/@-nodes are variable/@-nodes; images of lambda nodes are lambda nodes; images of ghost and structural nodes are respectively ghost and structural;

(vi) Φ preserves node arity. In particular $|x| = |@_x|$ since $@_x$'s operator does not contribute to its arity;

(vii) Φ is not injective: since for any node in R' there is a corresponding node in R that is mapped to the same node in M.

Let $\mathcal{J}^P$ denote the subset $\mathcal{J}$ consisting of justified sequences verifying the P-view-path correspondence: the P-view at every occurrence of an internal node is precisely the path from the root to that node in the computation tree. By Property 2.1 we have $\mathcal{T}rav \subseteq \mathcal{J}^P$.

We define ϕ as an extension of Φ to justified sequences of $\mathcal{J}^P$ of N as follows: (i) maps all other occurrences to their image by Φ; (ii) it preserves all justification pointers except for $@_x$ and $\lambda \overline{y}'$; (iii) inserts in-between each occurrence of x and its immediately following lambda-node, the corresponding *argument lookup* of x. Formally $\phi = \phi_1 \circ \phi_2$ where

$$\phi_1 \colon \mathcal{J}^P(N) \to \mathcal{J}(M)$$

$$\epsilon \mapsto \epsilon$$

$$t \cdot n \mapsto \phi(t) \cdot \Phi(n) \qquad \Phi(n) \text{ has same link label and length as } n, \text{ for } n \notin \{@_x, \lambda \overline{y}\}$$

$$t \cdot @_x \mapsto \phi(t) \cdot x \qquad\qquad x \text{ points to the first occurrence of } x\text{'s binder in } \ulcorner t \urcorner$$

$$t \cdot \lambda \overline{y}' \to \phi(t) \cdot \lambda \overline{y} \qquad\qquad \lambda \overline{y} \text{ points to the last occurrence of } @_a \text{ in } \ulcorner t \urcorner.$$

and ϕ_2 is the function that inserts the argument lookup sequence of ghost nodes between each occurrence of x and $\lambda \overline{y}$ in a justified sequence. Observe that the argument lookup of x is entirely statically determined and thus can be reconstructed from any (visible) traversal $t_{\leq x}$ of M by repeatedly instantiating rules (Lmd) and (Var) starting from $t_{\leq x}$ until reaching node $\tilde{\lambda}\overline{y}$.

The following example illustrates how ϕ is calculated on a generic traversal u of N involving the lloc variable x:

$$u \quad = \quad \ldots \cdot \lambda \overline{\xi_a} \cdot \ldots @_1 \cdot \lambda \overline{\xi_1} \cdot @ \ldots \lambda \overline{z} \cdot \overset{0}{@_x} \cdot \lambda \overline{y}'$$

$$\phi_1(u) \quad = \quad \ldots \cdot \lambda \overline{\xi_a} \cdot \ldots @_1 \cdot \lambda \overline{\xi_1} \cdot @ \ldots \lambda \overline{z} \cdot \overset{i}{x} \cdot \lambda \overline{y}$$

$$\phi(u) = \phi_2(\phi_1(u)) \quad = \quad \ldots \cdot \lambda \overline{\xi_a} \cdot \ldots @_1 \cdot \lambda \overline{\xi_1} \cdot @ \ldots \lambda \overline{z} \cdot \overset{i}{x} \cdot \lambda\!\!\lambda \cdot \theta \ldots \lambda\!\!\lambda \cdot \theta \cdot \lambda \overline{y}$$

Some immediate properties:

- ϕ_1, and ϕ are well defined since by the P-view-path property, x's binder and (resp. $@_a$) must necessarily occurs in the P-view at x (resp. $\lambda\overline{y}'$).

- The restriction of ϕ to $\pi(\mathcal{T}rav^\dagger(N))$ coincides precisely with the element-wise structure-preserving[3] extension of Φ over justified sequences. This is because ϕ_1 only alters justification pointers of internal nodes ($@_x$ and $\lambda\overline{y}$) and ϕ_2 only inserts internal nodes, while justified sequences in $\pi(\mathcal{T}rav^\dagger(N))$ do not contain any external node.

- The restriction of ϕ to $\pi(\mathcal{T}rav^\dagger(N))$, is injective. Suppose that $\pi(u) \neq \pi(v)$ for $u, v \in \mathcal{T}rav^\dagger(N)$ then we must have $u \neq v$. Let s denote the longest common prefix of u and v; and n_1, n_2 the nodes immediately following s in u and v respectively. We have $s \to_N s \cdot n_1$ and $s \to_N s \cdot n_2$ with $n_1 \neq n_2$ therefore n_1, n_2 are necessarily traversed by the non-deterministic rule (IVar). Hence they must both be external nodes and therefore they are not eliminated after applying π: the last occurrence of $\pi(s \cdot n_1)$ and $\pi(s \cdot n_2)$ are different. Since Φ maps distinct external sibling nodes in N to distinct sibling nodes in M, we necessarily have $\phi(\pi(s \cdot n_1)) \neq \phi(\pi(s \cdot n_2))$.

- π and ϕ commute modulo the implicit renaming function $\rho : \mathcal{V}^*(N) \to \mathcal{V}^*(M)$ underlying Φ. Formally, for any variable names $\alpha \in \mathcal{V}^*$ occurring in N we have:

$$\phi(\pi_\alpha(u)) = \pi_{\rho(\alpha)}(\phi(u)) \ .$$

  This is shown by an easy induction and by assuming, without loss of generality, that variable names are not reused in either M or N: each variable name is introduced by at most one lambda node in each term. (This condition can always be met via alpha-conversion.)

We now show that M and N are in ϕ-bisimulation by induction on the traversals rules.

Base case: $(\mathsf{Root})_{\mathsf{norm}}$ The empty traversal in M is trivially the image by ϕ of the empty traversal in N. Same for the two singleton traversals consisting of the root nodes of M and N.

Induction: Let $u \in \mathcal{T}rav^\dagger(N)$, $t \in \mathcal{T}rav^\dagger(M)$ with $t = \phi(u)$ and m, n denote respectively the last node in t and u. We prove both directions by case analysis on the *first* traversal rule used to extend t (resp. u).

Observe that the result holds trivially up until the traversal in M reaches occurrence x since one can use the exact same rules used to traverse M in order to traverse the corresponding node in N, and reciprocally for traversals of N until reaching node $@_x$. The crucial part of the induction takes place when traversing the *lloc* variable x (or $@_x$ in N).

- $(\mathsf{App})_{\mathsf{norm}}$ We have $m = \Phi(n)$.
(B1) Suppose t extends to t' with the application rule then $m, n \in \mathcal{N}_@$. Then the last occurrence m' in t' is the unique node verifying $m \vdash_0 m'$. Since Φ does not map $@_x$ to an application node we necessarily have $n \neq @_x$. Thus ϕ preserves the enabling relation at n, and we can use the application rule in N to extend u into $u' = u \cdot n'$ where $n \vdash_0 n'$ and $\phi(n') = m'$.
(B2) Suppose u' extends traversal u with the application rule then the last occurrence n in u is an application node. If $n \neq @_x$ then ϕ preserves the enabling relation at n and therefore the application rule can be used in M to simulate u'.

---

[3] Recall that two justified sequences over two distinct terms share the same structure just if their constituting nodes are of the same kind (variable/@ nodes or lambda node), have the same link labels, and same pointer lengths.

Otherwise, $n = @_x$ and since $t = \phi(u)$ we must have $m = \Phi(@_x) = x$. and $u' = u \cdot \lambda \overline{y}'$. Let $@_j$ and $\lambda \overline{\xi_j}$ for $1 \leq j \leq a$, $a \geq 1$ denote respectively the application and lambda nodes in the spinal descent from $@_a$ to x's binder. So that $\lambda \overline{\xi_1}$ is x's binder, $@_1$ its parent, and $@_a$ is $\lambda \overline{y}$'s parent. By Lemma 7.4, $@_a$ is precisely the lowest application node in the spinal descent from $@_a$ to $\lambda \overline{\xi_1}$ verifying $|@_a| > i_a$, where $i_a = i + \sum_{1 \leq j < a}(|\lambda \overline{\xi_j}| - |@_j|)$. This formula precisely captures the "pending argument lookup" algorithm implicitly implemented by the rules (Lmd) and (Var): with exactly i_a instantiations of those two rules, traversal t gets extended by $2(i_a - 1)$ ghost nodes followed by the root of A:

$$t \rightarrow_M t \cdot v \cdot \lambda \overline{y}$$

where v is a *pending argument lookup*. Therefore by definition of ϕ if v denote the argument lookup of x we have: $\phi(u') = \phi(u \cdot \lambda \overline{y}') = \phi(u) \cdot \phi(\lambda \overline{y}') = t \cdot v \cdot \lambda \overline{y} \in \mathcal{T}rav^\dagger(M)$.

- $(\mathsf{Lam})_{\mathsf{norm}}^@$ (B1) follows from the fact that parent-child relation is preserved by Φ (B2) Suppose $n \neq \lambda \overline{z}$ then it follows again from the fact that parent-child relation is preserved by Φ. Otherwise, we have $n = \lambda \overline{z}$ and $u \rightarrow_N u \cdot \lambda \overline{y}'$ can be simulated in M using rule $(\mathsf{Lam})^{\mathsf{var}}$: $t \rightarrow_M t \cdot x$.

- $(\mathsf{Lam})_{\mathsf{norm}}^{\mathsf{var}}$ We have already shown that ϕ preserves the local child-parent relationship for lambda nodes; node types, and arity of lambda nodes. It remains to show that any $\vdash$-enabler of m that is visible at u is mapped to an enabler of n that is visible in $\phi(u)$.

  Since the last node of t and u are structural lambda nodes, their P-view do not contain any ghost node nodes, and in particular the argument lookups inserted by ϕ_2 are not involved in the P-view calculation. Hence $\ulcorner\phi(u)\urcorner = \ulcorner\phi_1(u)\urcorner$. But by the P-view path characterization of traversals $\ulcorner\phi_1(u)\urcorner$ is precisely the path in M from $\Phi(n)$ to the root of M, and $\ulcorner u \urcorner$ is the path in N from n to the root of N.

  If n is not under $\lambda \overline{y}'.R'$ then the path to the root is not impacted by the reduction and therefore the path in M and N are identical modulo ϕ: we thus have $\ulcorner\phi_1(u)\urcorner = \phi_1(\ulcorner u \urcorner)$. Otherwise if n is under $\lambda \overline{y}'.R'$ then the path from m to the root in M is precisely the element-wise image by Φ of the path from n to the root in N with the exclusion of the spinal descent from $@_a$ to $@_x$. But by construction of N, none of the variable in R' are bound by the lambda nodes in the spinal descent. Hence, the rule $(\mathsf{Lam})_{\mathsf{norm}}^{\mathsf{var}}$ used to extend u can also be used to extend t, and reciprocally.

- $(\mathsf{Lam}_{\mathsf{norm}}^{\lambda\!\!\lambda})$ This case is excluded since t and u belong to $\mathcal{T}rav^\dagger(M)$ and $\mathcal{T}rav^\dagger(N)$ respectively and therefore their last occurrence cannot be a ghost lambda.

- $(\mathsf{Var}_{\mathsf{norm}})$ (B2) Since Φ maps external variable nodes to external variable nodes,preserves the parent-child relation for variable nodes and ϕ preserves the justification pointer of external variable nodes, if this rule can extend traversal u with node n' then the same rule can be used to extend t with $\phi(n')$.

  (B1) Suppose that t ends with *lloc* variable x in which case $t' = t \cdot \lambda \overline{y}$, then we can just take $u = u' \cdot @_x$ and conclude exactly like we did for case $(\mathsf{App})_{\mathsf{norm}}$ (B2). Otherwise, if $m \neq x$, we conclude like in case (B1).

- $(\mathsf{IVar}_{\mathsf{norm}})$ Φ preserves node types, arity of variable nodes. It remains to show that it preserves *arity threshold* of traversals: In an argument lookup, all occurrences are ghost with arity 0 therefore they do not affect the arity threshold calculation. Hence $\mathsf{arth}(\phi(u)) = \mathsf{arth}(\phi_1(u))$. Further Φ preserves arity and maps external and internal nodes to external and internal nodes respectively, thus $\mathsf{arth}(\phi_1(u)) = \phi_1(\mathsf{arth}(u))$. Hence the rule $(\mathsf{Var}_{\mathsf{norm}})$ used to extend u can be used to extend t, and reciprocally.

Case 2 where x has at least one operand ($|x| > 0$) and A is an application ($|\overline{y}| = 0$). The root lambda node of A is therefore a dummy lambda: $A = \lambda.R$. Let r denote the root node of R, which is either a variable if R is in head-normal form, or an @-node otherwise. The tree of the reduct N is obtained from $CT(M)$ by substituting x's label with that of r (modulo variable renaming to avoid capture if r is a variable node) and prepending r's children to existing x's children. Other nodes, including x's parent lambda node, remain untouched. Let r' denote this updated node in the reduct N. This transformation yields an implicit mapping function $\Phi : \tilde{\mathcal{N}}(N) \to \tilde{\mathcal{N}}(N)$ such that $\Phi(r') = x$. Although Φ does not preserve the arity of x, it does not affect the arity threshold of traversals since $|r'| = |x| + |r|$. We can then extend Φ to a function ϕ on justified sequence and show by induction that it yields a bisimulation similarly to the previous case.

Case 3 where x is unapplied ($|x| = 0$). After reduction, node x gets replaced by subterm A and the label of x's parent lambda node gets concatenated with the label of A's root node: the parent lambda node $\lambda\overline{z}$ of x in M becomes $\lambda\overline{z}\overline{y}'$ in the reduct N. Like in the first case, we can define a function ϕ such that traversals of M of the form $\ldots \lambda\overline{z} \cdot x \cdot v \cdot \lambda\overline{y} \cdots$, where v is an argument lookup, are images of traversals in N of the form $\ldots \cdot \lambda\overline{z}\overline{y}' \cdots$. Once again, although the arity of x is not preserved, this does not affect the arity threshold because $|\lambda\overline{z}\overline{y}'| = |\lambda\overline{z}| + |\lambda\overline{y}|$.

Hence M and N are in ϕ-bisimulation, thus because ϕ is injective and commutes with π, by Lemma 7.5 the two sets of traversals are isomorphic. $\square$

Proof of Proposition 7.7. Let's consider the tree structure at the trivial redex on Figure 4. It is of the form $(\lambda\overline{x}.T)A_1 \ldots A_r$ for $r \geq 1$ where (x_1, A_1) forms the leftmost trivial redex. Two cases must be considered:
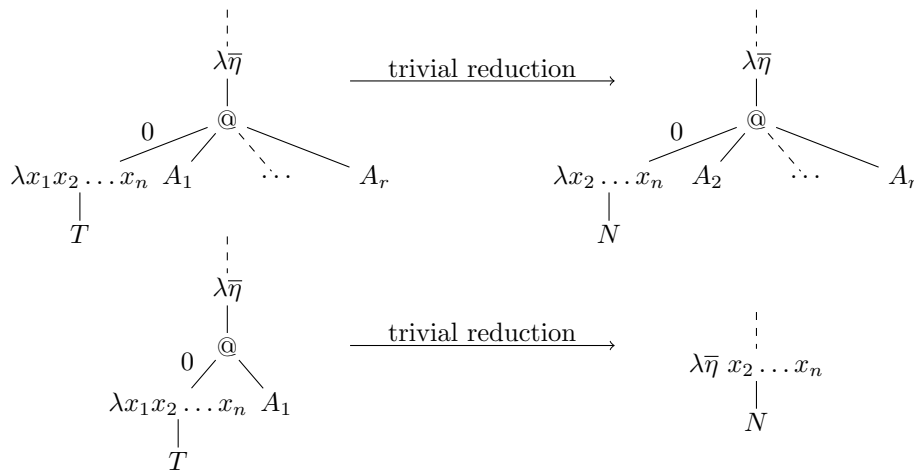

Figure 4: Reducing a (leftmost spinal-innermost) trivial redex. Case $r > 1$ (top) and $r = 1$ (bottom).

Case $r > 1$ Because the structure of the tree is preserved after reduction, there is an implicit injection Φ from nodes of the reduct N to nodes of M. Let ϕ denote its element-wise structure-reserving extension to justified sequences.

Firstly, M and N are in ϕ-bisimulation. We omit the details, the key observations are: (i) Since x_1 does not occur in T, the term A_1 will never be traversed in N. (ii) Only nodes hereditarily justified by $\lambda x_1 \ldots x_n$ see their label (or link label) changed after applying Φ. But since those are external nodes, they get filtered out by the core projection π. (iii) The arity of node $\lambda x_2 \ldots x_n$ increases by 1 after applying Φ, but it does not impact the arity threshold of traversals because the arity of the application node also increases by 1.

Secondly, since Φ is injective, so is its element-wise extension to justified sequences.

Finally, ϕ and π commute. This is shown by an easy induction, observing that nodes @ and $\lambda x_2 \ldots x_n$ necessarily occur consecutively in a traversal of N, and because $|@| = |\Phi(@)| - 1$ and $|\lambda x_2 \ldots x_n| = |\Phi(\lambda x_2 \ldots x_n)| - 1$, the core projection is preserved modulo Φ.

Hence by Lemma 7.5 the two sets of traversals are isomorphic.

Case $r = 1$ Depicted on Figure 4 (bottom). The tree structure is altered: the application node and the argument A_1 are eliminated in the reduct. There is an implicit injection Φ from nodes of N to nodes of M. We extent this function to *occurrences* of nodes as follows. Let's introduce the abbreviation $\overline{y} = \overline{\eta} x_2 \ldots x_n$.

$$\Phi \colon O_4(N) \to O_4(M)$$
$$(n, p, d, k) \longmapsto (\Phi(n), p, d, k) \qquad \qquad \text{if } n \neq \lambda\overline{y}$$
$$(\lambda\overline{y}, p, d, k) \longmapsto (\lambda\overline{\eta}, (x_2 \ldots x_n \cdot p), d, k)$$

So in particular, Φ maps occurrences of $\lambda\overline{y}$ to occurrences of $\lambda\overline{\eta}$ with binding variables $x_2 \ldots x_n$ appended to $\overline{\eta}$.

$$\Phi(\lambda\overline{y}) = \lambda\overline{\eta}^{[x_2 \ldots x_n]} \tag{5}$$

Observe also that Φ commutes with the rebinding operation: because in the case of $\lambda\overline{y}$, Φ appends variable names to the left whereas the rebinding operation appends variables names to the right.

We extend Φ to justified sequences by taking the element-wise extension ϕ with the following additional modification: any occurrence of node $\lambda\overline{y}$ in a traversal of N gets expanded by ϕ into the three occurrences $\lambda\overline{\eta} \cdot @ \cdot \lambda\overline{x}$ in M.

We have the following properties:

- ϕ is injective. It follows from the fact that Φ is itself injective.
- ϕ and π commute. This is because ϕ carries over the list of pending lambdas and the core symbol operation adequately implements merging of nodes $\lambda\overline{\eta}$ and $\lambda\overline{x}_1 \ldots x_n$ into $\lambda\overline{\eta} x_2 \ldots x_n$. Formally:
  Let u be a traversal of $\mathcal{T}rav^\dagger(N)$ ending with $\lambda\overline{y}$ that is $u = u_1 \cdot \lambda\overline{y}$ for some traversal u_1. The sequence $\phi(u)$ is of the form $\phi(u) = \phi(u_1) \cdot \lambda\overline{\eta} \cdot @ \cdot \lambda\overline{x}$. Let $\overline{\alpha}$ be the list of pending lambdas at $u_{\leq \lambda\overline{y}}$. There are two cases:

Suppose $\lambda\overline{\eta}$ is internal. We have:

$$
\begin{aligned}
\pi_{\overline{\alpha}}(\phi(u)) &= \pi_{\overline{\alpha}}(\phi(u_1) \cdot \lambda\overline{\eta} \cdot @ \cdot \lambda\overline{x}) && \text{(def. of } \phi(u)) \\
&= \pi_{\overline{x\alpha}}(\phi(u_1) \cdot \lambda\overline{\eta} \cdot @) && \text{(def. of } \pi) \\
&= \pi_{pop_1(\overline{x\alpha})}(\phi(u_1) \cdot \lambda\overline{\eta}) && \text{(def. of } \pi \text{ and } |@| = 1) \\
&= \pi_{\overline{\eta} \cdot pop_1(\overline{x\alpha})}(\phi(u_1)) && \text{(def. of } \pi) \\
&= \pi_{\overline{y\alpha}}(\phi(u_1)) && \text{(def. of } \overline{y}) \\
&= \phi(\pi_{\overline{y\alpha}}(u_1)) && \text{(by induction hypothesis)} \\
&= \phi(\pi_{\overline{\alpha}}(u_1 \cdot \lambda\overline{y})) && \text{(def of } \pi) \\
&= \phi(\pi_{\overline{\alpha}}(u)) && \text{(def. of } u).
\end{aligned}
$$

Suppose $\lambda\overline{\eta}$ is external. We have:

$$
\begin{aligned}
\pi_{\overline{\alpha}}(\phi(u)) &= \pi_{\overline{\alpha}}(\phi(u_1) \cdot \lambda\overline{\eta} \cdot @ \cdot \lambda\overline{x}) && \text{(def. of } \phi(u)) \\
&= \pi_{\overline{x\alpha}}(\phi(u_1) \cdot \lambda\overline{\eta} \cdot @) && \text{(def. of } \pi) \\
&= \pi_{pop_1(\overline{x\alpha})}(\phi(u_1) \cdot \lambda\overline{\eta}) && \text{(def. of } \pi \text{ and } |@| = 1) \\
&= \pi_{x_2...x_n\overline{\alpha}}(\phi(u_1) \cdot \lambda\overline{\eta}) && \text{(def. of } pop) \\
&= \pi_{\epsilon}(\phi(u_1)) \cdot \lambda\overline{\eta}^{[x_2...x_n\overline{\alpha}]} && \text{(def. of } \pi)
\end{aligned}
$$

And:

$$
\begin{aligned}
\phi(\pi_{\overline{\alpha}}(u)) &= \phi(\pi_{\overline{\alpha}}(u_1 \cdot \lambda\overline{y})) && \text{(def. of } u) \\
&= \phi(\pi_{\epsilon}(u_1) \cdot \lambda\overline{y}^{[\overline{\alpha}]}) && \text{(def. of } \pi) \\
&= \phi(\pi_{\epsilon}(u_1)) \cdot \phi(\lambda\overline{y}^{[\overline{\alpha}]}) && \text{(def. of } \phi) \\
&= \pi_{\epsilon}(\phi(u_1)) \cdot \phi(\lambda\overline{y}^{[\overline{\alpha}]}) && \text{(commutativity, by I.H.)} \\
&= \pi_{\epsilon}(\phi(u_1)) \cdot \phi(\lambda\overline{y})^{[\overline{\alpha}]} && (\phi \text{ commutes with rebinding)} \\
&= \pi_{\epsilon}(\phi(u_1)) \cdot (\lambda\overline{\eta}^{[x_2...x_n]})^{[\overline{\alpha}]} && \text{(by Eqn. 5)} \\
&= \pi_{\epsilon}(\phi(u_1)) \cdot \lambda\overline{\eta}^{[x_2...x_n\overline{\alpha}]} && \text{(rebinding composition)}
\end{aligned}
$$

Hence $\phi(\pi_{\overline{\alpha}}(u)) = \pi_{\overline{\alpha}}(\phi(u))$.

- $\to_M$ can simulate $\to_N$ and reciprocally. This is shown by induction on the traversal rules. One interesting case is when ghost nodes in M get materialized into structural nodes in N: Consider the traversal of M for some $i \geq 1$: $t = \lambda\overline{\eta} \cdot @ \cdot \lambda\overline{x} \cdot \ldots \cdot \overline{\lambda} \cdot \theta$. If $i > n$ then corresponding traversal in N also ends with a ghost variable: $u = \lambda\overline{\eta}x_2\ldots x_n \cdot \ldots \cdot \theta$ If $i \leq n$ then the ghost nodes θ gets materialized into variable occurrence x_i in N: $u = \lambda\overline{\eta}x_2\cdots x_n \cdot \ldots \cdot x_i$

We can then conclude by Lemma 7.5 that the two sets of traversals are isomorphic.

$\square$