Local computation of β-reduction
A concrete presentation of Game Semantics

William Blum and C.H. Luke Ong

Oxford University Computing Laboratory

Abstract

We show that ...

Key words: Lambda calculus, beta-reduction traversal theory

Contents

Preprint submitted to Elsevier *October 8, 2008*

Todo list

Analyzing the effect that a syntactic restriction (such as safety) has on the game-semantic model is a difficult task since the main feature of game semantics is precisely to be syntax-independent. The aim of this chapter is to establish an explicit correspondence between the game denotation of a term and its syntax. This will be used in the next chapter to give a characterization of the game semantics of the safe lambda calculus.

Our approach follows ideas recently introduced by Ong [1], namely the notion of *computation tree* of a simply-typed lambda-term and *traversals* over the computation tree. A computation tree is just an abstract syntax tree (AST) representation of the η-long normal form of a term. Traversals are justified sequences of nodes of the computation tree respecting some formation rules. They are meant to describe the computation of the term, but at the same time they carry information about the syntax of the term in the following sense: the *P-view* of a traversal (computed in the same way as P-view of plays in game semantics) is a path in the computation tree. Traversals provide a way to perform *local computation* of β-reductions as opposed to a global approach where β-redexes are contracted using substitution.

The culmination of this chapter is the *Correspondence Theorem* (Theorem 2.2). It states that traversals over the computation tree are just representations of the uncovering of plays in the strategy-denotation of the term. Hence there is an isomorphism between the strategy denotation of a term and its revealed game denotation. In a nutshell, the revealed denotation is computed similarly to the standard strategy denotation except that internal moves are not hidden after composition. In order to make a connection with the standard game denotation, we define an operation that extracts the *core* of a traversal by eliminating occurrences of "internal nodes". These node occurrences are the counterparts of internal moves that are hidden when performing strategy composition in game semantics. This leads to a correspondence between the standard game denotation of a term and the set traversal cores over its computation tree.

Using this correspondence, it possible to analyze the effect that a syntactic restriction has on the strategy denotation of a term. This is illustrated in the next chapter where we rely on the Correspondence Theorem to analyze the game semantics of the safety restriction.

Related works: The useful transference technique between plays and traversals was originally introduced by Ong for studying the decidability of monadic second-order theories of infinite structures generated by higher-order grammars [1]. In this setting, the Σ-constants or terminal symbols are at most order 1, and are *uninterpreted*. Here we present an extension of this framework to the general case of the simply-typed lambda calculus with free variables of any order. Further the term considered is not required to be of ground type contrary to higher-order grammars. This requires us to add new traversal rules to handle variables whose value is undetermined (*i.e.*, those that cannot be resolved through redex-contraction). We also extend computation trees with additional nodes accounting for answer moves of game semantics. This enables our framework to be extended to languages with interpreted constants such as PCF and Idealized Algol.

A notion of local computation of β-reduction has also been investigated through the use of special graphs called "virtual nets" that embed the lambda calculus [2].

Asperti et al. introduced [3] a syntactic representation of lambda-terms based on Lamping's graphs [4]. They unified various notions of paths (regular, legal, consistent and persistent paths) that have appeared in the literature as ways to implement graph-based reduction of lambda-expressions. We can regard a traversal as an alternative notion of path adapted to the graph representation of lambda-expressions given by computation trees.

1. Computation tree

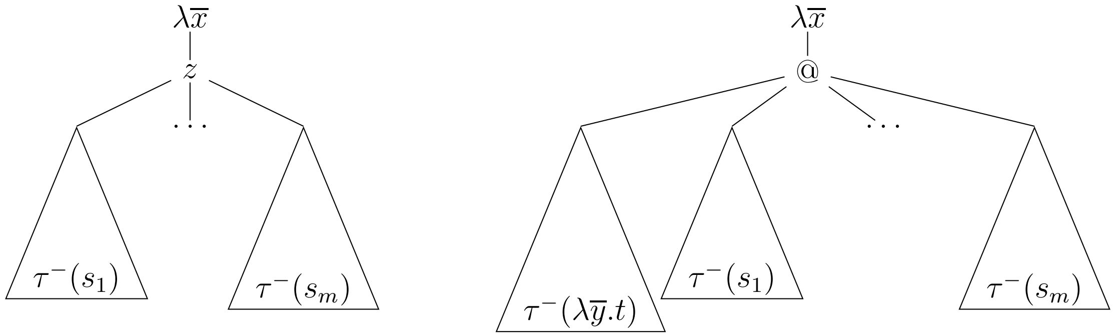
We work in the general setting of the simply-typed lambda calculus extended with a fixed set Σ of higher-order uninterpreted constants.[1] We fix a simply-typed term-in-context $\Gamma \vdash M : T$ for the rest of the section.

1.1. Definition

We define the *computation tree* of a simply-typed lambda-term as an abstract syntax tree representation of its *η-long normal form* (Def. **??**). Our definition generalizes the notion of computation tree for higher-order recursion schemes [1].

We recall that a term M in η-long normal form is of the form $\lambda \overline{x}.s_0 s_1 \ldots s_m$ where $\overline{x} = x_1 \ldots x_n$ for $n \geq 0$ and $s_0 s_1 \ldots s_m$ is of ground type, each s_j for $j \in 1..m$ is in η-long nf, and either s_0 is a variable or a constant and $m \geq 0$; or s_0 is an abstraction $\lambda \overline{y}.s$ and

[1] A constant $c \in \Sigma$ is *uninterpreted* if the small-step semantics of the language does not contain any rule of the form $c\ M_1 \ldots M_k \cdots \rightarrow f_c(M_1, \ldots, M_k)$ for some function f_c over closed normal terms $M_1, \ldots, M_k$. Think of such constant as a data constructor.



$\tau^-(\lambda\overline{x}.zs_1\ldots s_m : o)$
$m \geq 0$ and $z \in \mathcal{V} \cup \Sigma$

$\tau^-(\lambda\overline{x}.(\lambda\overline{y}.t)s_1\ldots s_m : o)$
$m \geq 1$

Table 1: The tree $\tau^-(M)$.

$m \geq 1$ where s is in η-long nf. If M is of ground type then its η-long nf is of the form $\lambda.N$; although the symbol 'λ' does not correspond to a real lambda-abstraction—we call it 'dummy lambda'—it will still be convenient to keep it in expressions representing eta-long normal forms.

Definition 1.1. Let $\Gamma \vdash_{\mathsf{st}} M : T$ be a simply-typed term with variable names from $\mathcal{V}$ and constants from Σ. The *pre-computation* tree $\tau^-(M)$ with labels taken from $\{@\} \cup \Sigma \cup \mathcal{V} \cup \{\lambda x_1 \ldots x_n \mid x_1, \ldots, x_n \in \mathcal{V}, n \in \mathbb{N}\}$, is defined inductively on its η-long normal form as follows.

$$\begin{aligned}
\text{For } m \geq 0,\ z \in \mathcal{V} \cup \Sigma\text{: } \tau^-(\lambda\overline{x}.zs_1\ldots s_m : o) &= \lambda\overline{x}\langle z\langle \tau^-(s_1), \ldots, \tau^-(s_m)\rangle\rangle \\
\text{for } m \geq 1\text{: } \tau^-(\lambda\overline{x}.(\lambda y.t)s_1\ldots s_m : o) &= \lambda\overline{x}\langle @\langle \tau^-(\lambda y.t), \tau^-(s_1), \ldots, \tau^-(s_m)\rangle\rangle\ ,
\end{aligned}$$

where we write $l\langle t_1, \ldots, t_n\rangle$ for $n \geq 0$ to denote the *ordered tree* whose root is labelled l and has n child-subtrees t_1, ..., t_n. The trees from the equations above are illustrated in Table 1.

By convention the first level of a tree (where the root lies) is numbered 0. In the tree $\tau^-(M)$, nodes at odd-levels are variable, constant or application nodes; and at even-levels lies the λ-nodes. A single λ-node can represent several consecutive abstractions or it can just be a *dummy lambda* (if the corresponding subterm is of ground type).

Definition 1.2. Let M be a simply-typed term not necessarily in η-long normal form. Let $\mathcal{D}$ denote the set of values of base type o. The ***computation tree*** of M, written $\tau(M)$ is the tree obtained from $\tau^-(\lceil M \rceil)$ by attaching leaves to each node as follows: for every node $n \in \tau^-(M)$, the corresponding node in $\tau(\lceil M \rceil)$ has a child leaf labelled v_n, called ***value-leaf***, for every possible value $v \in \mathcal{D}$.

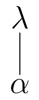
Inner nodes of the tree are thus of three kinds:

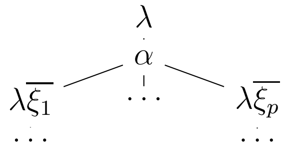
- λ-nodes labelled $\lambda\overline{x}$ for some list of variables $\overline{x}$ (Note that a λ-node represents several consecutive variable abstractions),

- application nodes labelled @,
- variable or constant nodes with labels in $\Sigma \cup \mathcal{V}$.

A node is said to be ***prime*** if it is the $0^{th}l$ child of an @-node. An inner node whose parent is a @-node or a Σ-node is called a ***spawn*** node.

Example 1.1.

- The computation tree of a ground type variable or constant α is 

;
- The computation tree of a higher-order variable or constant $\alpha : (A_1, \ldots, A_p, o)$ has the following form: 

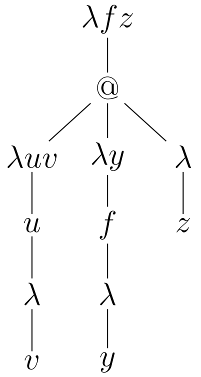
;

Example 1.2. Take $\vdash_{\mathsf{st}} \lambda f^{o \to o}.(\lambda u^{o \to o}.u) f : (o \to o) \to o \to o$.

Its η-long normal form is:

$$\begin{aligned}&\vdash_{\mathsf{st}} \lambda f^{o\to o} z^o.\\ &\qquad (\lambda u^{o\to o} v^o.u(\lambda.v))\\ &\qquad (\lambda y^o.f y)\\ &\qquad (\lambda.z)\\ &: (o \to o) \to o \to o\end{aligned}$$

Its computation tree is:

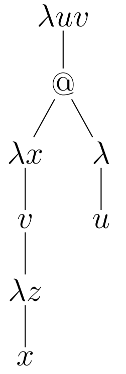


Example 1.3. Take $\vdash_{\mathsf{st}} \lambda u^o v^{((o \to o) \to o)}.(\lambda x^o.v(\lambda z^o.x)) u : o \to ((o \to o) \to o) \to o$.

Its η-long normal form is:

$$\begin{aligned}&\vdash_{\mathsf{st}} \lambda u^o v^{((o\to o)\to o)}.\\ &\qquad (\lambda x^o.v(\lambda z^o.x)) u\\ &: o \to ((o \to o) \to o) \to o\end{aligned}$$

Its computation tree is:



Notations 1.1 We write $\circledast$ to denote the root of $\tau(M)$. We write E to denote the parent-child relation of the tree, V for the set of vertices (*i.e.*, leaves and inner nodes) of the tree, N for the set of inner nodes and L for the set of value-leaves. Thus $V = N \cup L$.

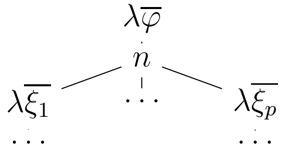
We write N_Σ for the set of Σ-labelled nodes, $N_@$ for the set of @-labelled nodes, N_{var} for the set of variable nodes, N_{fv} for the subset of N_{var} consisting of free-variable nodes, N_{prime} for the set of prime nodes and N_{spawn} for the set of spawn nodes ($= N \cap E(\!|N_@ \cup N_\Sigma|\!)$).

For \$ ranging over $\{@, \lambda, \mathsf{var}, \mathsf{fv}\}$, we write $L_\$$ to denote the set of value-leaves which are children of nodes from $N_\$$; formally $L_\$ = \{v_n \mid n \in N_\$, v \in \mathcal{D}\}$. We write $V_\$$ for $N_\$ \cup L_\$$.

For every lambda node n in N_λ we write $M^{(n)}$ for the subterm rooted at n and $V^{(n)}$ for the set of vertices of the sub-computation tree $\tau(M^{(n)})$; formally $V^{(n)} = E^*(\{n\})$ where E^* denotes the transitive, reflexive closure of the parent-child relation E.

Each subtree of the computation tree $\tau(M)$ represents a subterm of $\lceil M \rceil$. We define the function $\kappa : N \to \Lambda^{\mathrm{Ch}}_{\to}$ (where $\Lambda^{\mathrm{Ch}}_{\to}$ denotes the set of Church typed lambda-terms) that maps a node $n \in N$ to the subterm of $\lceil M \rceil$ corresponding to the subtree of $\tau(M)$ rooted at n. In particular $\kappa(\circledast) = \lceil M \rceil$.

Remark 1.1 Since the computation tree is computed from the eta-long normal form, for every subtree of $\tau(M)$ of the form


$\lambda\overline{\varphi}$
n
$\lambda\overline{\xi_1}$
$\lambda\overline{\xi_p}$

, we have $\operatorname{ord}\kappa(n) = 0$.

Definition 1.3 (Type and order of a node)**.** Suppose $\Gamma \vdash M : T$. The ***type*** of an inner-node $n \in N$ of $\tau(M)$ written $\mathsf{type}(n)$ is defined as follows:

$$\begin{array}{rrcl} & \mathsf{type}(\circledast) & = & \Gamma \to T, \\ \text{for } n \in (N_\lambda \cup N_@) \setminus \{\circledast\}\text{:} & \mathsf{type}(n) & = & \text{type of the term } \kappa(n), \\ \text{for } n \in N_{\mathsf{var}} \cup N_\Sigma\text{:} & \mathsf{type}(n) & = & \text{type of the variable labelling } n. \end{array}$$

where the notation $\Gamma \to T$ is an abbreviation for $(A_1, \ldots, A_p, T)$ and $A_1, \ldots, A_p$ are the types of the variables in the context Γ.

The ***order*** of a node n, written $\operatorname{ord} n$, is defined as follows: a value-leaf $v \in L$ has order 0 and the order of an inner node $n \in N$ is defined as the order of its type. In particular, the type of a lambda node different from the root is the type of the term represented by the sub-tree rooted at that node, and the type of a variable-node is the type of the variable labelling it.

Since the computation tree is calculated from the η-long normal form, all the @-nodes have order 0 ($\operatorname{ord} @ = 0$); for every lambda node $\lambda\overline{\xi} \neq \circledast$ we have $\operatorname{ord} \lambda\overline{\xi} = 1 + \max_{z \in \overline{\xi}} \operatorname{ord} z$; and if the root $\circledast$ is labelled $\lambda\overline{\xi}$ then $\operatorname{ord} \circledast = 1 + \max_{z \in \overline{\xi} \cup \Gamma} \operatorname{ord} z$ with the convention $\max \emptyset = -1$.

Definition 1.4 (Binder)**.** We say that a variable node n labelled x is ***bound*** by a node m, and m is called the ***binder*** of n, if m is the closest node in the path from n to the root such that m is labelled $\lambda\overline{\xi}$ with $x \in \overline{\xi}$.

1.2. *Pointers and justified sequence of nodes*

1.2.1. *Definitions*

Definition 1.5 (Enabling)**.** The ***enabling relation*** $\vdash$ is defined on the set of nodes of the computation tree as follows. We write $m \vdash n$ and we say that m enables n if and only if $m \in L \cup N_\lambda \cup N_{\mathsf{var}}$ and one of the following conditions holds:

- $n \in N_{\mathsf{fv}}$ and m is the root $\circledast$;
- $n \in N_{\mathsf{var}} \setminus N_{\mathsf{fv}}$ and m is n's binder, in which case we write $m \vdash_i n$ to precise that n is the i^{th} variable bound by m;
- $n \in N_\lambda$ and m is n's parent;
- $n \in L$ and m is n's parent ($i.e.$, $n = v_m$ for some $v \in \mathcal{D}$).

Formally:

$$\begin{array}{rcl} \vdash & = & \{(\circledast, n) \mid n \in N_{\mathsf{fv}}\} \\ & & \cup\ \{(\lambda\overline{x}, x) \mid x \in N_{\mathsf{var}} \setminus N_{\mathsf{fv}} \wedge \lambda\overline{x} \text{ is } x\text{'s binder}\} \\ & & \cup\ \{(m, \lambda\overline{\eta}) \mid m \text{ is } \lambda\overline{\eta}\text{'s parent and } \lambda\overline{\eta} \in N_\lambda\} \\ & & \cup\ \{(m, v_m) \mid v \in \mathcal{D}, m \in N\} \end{array}$$

Note that in particular, free variable nodes are enabled by the root. Table 2 recapitulates the possible node types for the enabler node depending on the type of n.

If $n \in \_$	then $m \in \_$
N_λ	$N_{\mathsf{var}} \cup N_\Sigma \cup N_@$
L_{var}	N_{var}
$L_@$	$N_@$
L_Σ	N_Σ
N_{var}	N_λ
N_Σ	n.a.
$N_@$	n.a.
L_λ	N_λ

Table 2: Type of the enabler node in "$m \vdash n$".

We say that a node n_0 of the computation tree is ***hereditarily enabled*** by $n_p \in N$ if there are nodes $n_1, \ldots, n_{p-1} \in N$ such that n_{i+1} enables n_i for all $i \in 0..p-1$.

For every sets of nodes $S, H \subseteq N$ we write $S^{H\vdash}$ to denote the subset $S \cap \vdash^* (H)$ of S consisting of nodes hereditarily enabled by some node in H. Formally:

$$S^{H\vdash} = \{n \in S \mid \exists n_0 \in H \text{ s.t. } n_0 \vdash^* n\} \ .$$

If H is a singleton $\{n_0\}$ then we abbreviate $S^{\{n_0\}\vdash}$ into $S^{n_0\vdash}$.

We have $V_{\mathsf{var}}^{\circledast\vdash} = V \setminus (V_{\mathsf{var}}^{N_{@}\vdash} \cup V_{\mathsf{var}}^{N_{\Sigma}\vdash})$. The elements of $N_{\mathsf{var}}^{\circledast\vdash}$ (*i.e.*, variable nodes that are hereditarily enabled by the root of $\tau(M)$) are called ***input-variables nodes***.

We use the following numbering conventions: The first child of a @-node—a prime node—is numbered 0; the first child of a variable or constant node is numbered 1; and variables in $\overline{\xi}$ are numbered from 1 onward ($\overline{\xi} = \xi_1 \ldots \xi_n$). We write $n.i$ to denote the i^{th} child of node n.

Definition 1.6 (Justified sequence of nodes)**.** A ***justified sequence of nodes*** is a sequence of nodes s of the computation tree $\tau(M)$ with pointers. Each occurrence in s of a node n in $L \cup N_{\lambda} \cup N_{\mathsf{var}}$ has a link pointing to some preceding occurrence of a node m satisfying $m \vdash n$; and occurrences of nodes in $N_{@} \cup N_{\Sigma}$ do not have pointer.

If an occurrence n points to an occurrence m in s then we say that m ***justifies*** n. If n is an inner node then we represent this pointer in the sequence as $\overset{\overset{i}{\curvearrowleft}}{m \ldots n}$ where the label indicates that either n is labelled with the i^{th} variable abstracted by the λ-node m or that n is the i^{th} child of m. The pointer associated to a leaf v_m, for some value $v \in \mathcal{D}$ and internal node $m \in N$, is represented as $\overset{\overset{v}{\curvearrowleft}}{m \cdot \ldots \cdot v_m}$.

To sum-up, a pointer in a justified sequence of nodes has one of the following forms:

$$\begin{array}{lll}
 & \overset{\curvearrowleft}{r \cdot \ldots \cdot z} & \text{for some occurrences } r \text{ of } \tau(M)\text{'s root and } z \in N_{\mathsf{fv}} \ ; \\
\text{or} & \overset{\overset{i}{\curvearrowleft}}{\lambda\overline{\xi} \cdot \ldots \cdot \xi_i} & \text{for some variable } \xi_i \text{ bound by } \lambda\overline{\xi},\ i \in 1..|\overline{\xi}| \ ; \\
\text{or} & \overset{\overset{j}{\curvearrowleft}}{@ \cdot \ldots \cdot \lambda\overline{\eta}} & j \in \{1..(arity(@) - 1)\} \ ; \\
\text{or} & \overset{\overset{k}{\curvearrowleft}}{\alpha \cdot \ldots \cdot \lambda\overline{\eta}}, & \text{for } \alpha \in N_{\Sigma} \cup N_{\mathsf{var}},\ k \in \{1..arity(\alpha)\} \ ; \\
\text{or} & \overset{\overset{v}{\curvearrowleft}}{m \cdot \ldots \cdot v_m} & \text{for some value } v \in \mathcal{D} \text{ and internal node } m \in N \ .
\end{array}$$

We say that an inner node n in of a justified sequence of nodes is ***answered***[2] by the value-leaf v_n if there is an occurrence of v_n for some value v in the sequence that points to n, otherwise we say that n is ***unanswered***. The last unanswered node is called the ***pending node***. A justified sequence of nodes is ***well-bracketed*** if each value-leaf occurring in it is justified by the pending node at that point.

For every justified sequence of nodes t we write $?(t)$ to denote the subsequence of t consisting only of unanswered nodes. Formally:

$$\begin{aligned}
?(u_1 \cdot \overset{\curvearrowleft}{n \cdot u_2 \cdot v_n}) &= ?(u_1 \cdot n \cdot u_2) \setminus \{n\} && \text{for some value } v \in \mathcal{D} \ , \\
?(u \cdot n) &= ?(u) \cdot n && \text{for } n \notin L \ ,
\end{aligned}$$

where $u \setminus \{n\}$ denotes the subsequence of u obtained by removing the occurrence n.

[2] This terminology is deliberately suggestive of the correspondence with game-semantics.

If u is a well-bracketed sequences then $?(u)$ can be defined as follows:

$$\begin{aligned} ?(u \cdot \overparen{n \ldots v_n}) &= ?(u) && \text{for some value } v \in \mathcal{D} \ , \\ ?(u \cdot n) &= ?(u) \cdot n && \text{where } n \notin L \ . \end{aligned}$$

Notations 1.2 We write $s = t$ to denote that the justified sequences s and t have same nodes *and* pointers. Justified sequence of nodes can be ordered using the prefix ordering: $t \leqslant t'$ if and only if $t = t'$ or the sequence of nodes t is a finite prefix of t' (and the pointers of t are the same as the pointers of the corresponding prefix of t'). Note that with this definition, infinite justified sequences can also be compared. This ordering gives rise to a complete partial order. We say that a node n_0 of a justified sequence is ***hereditarily justified*** by n_p if there are nodes $n_1, n_2, \ldots n_{p-1}$ in the sequence such that n_i points to n_{i+1} for all $i \in \{0..p-1\}$. We write t^ω to denote the last element of the sequence t.

1.2.2. Projection

We define two different projection operations on justified sequences of nodes.

Definition 1.7 (Projection on a set of nodes)**.** Let A be a subset of V, the set of vertices of $\tau(M)$, and t be a justified sequence of nodes then we write $t \upharpoonright A$ for the subsequence of t consisting of nodes in A. This operation can cause a node n to lose its pointer. In that case we reassign the target of the pointer to the last node in $t_{\leqslant n} \upharpoonright A$ that hereditarily justifies n (This node can be found by following the pointers from n until reaching a node appearing in A); if there is no such node then n just loses its pointer.

Definition 1.8 (Hereditary projection)**.** Let t be a justified sequence of nodes of $\mathcal{T}rav(M)$ and n be some occurrence in t. We define the justified sequence $t \upharpoonright n$ as the subsequence of t consisting of nodes hereditarily justified by n in t.

Lemma 1.1. *The projection function $\_ \upharpoonright n$ defined on the cpo of justified sequences ordered by the prefix ordering is continuous.*

Proof. Clearly $\_ \upharpoonright n$ is monotonous. Suppose that $(t_i)_{i \in \omega}$ is a chain of justified sequences. Let u be a finite prefix of $(\bigvee t_i) \upharpoonright n$. Then $u = s \upharpoonright n$ for some finite prefix s of $\bigvee t_i$. Since s is finite we must have $s \leqslant t_j$ for some $j \in \omega$. Therefore $u \leqslant t_j \upharpoonright n \leqslant \bigvee (t_j \upharpoonright n)$. This is valid for every finite prefix u of $(\bigvee t_i) \upharpoonright n$ thus $(\bigvee t_i) \upharpoonright n \leqslant \bigvee (t_j \upharpoonright n)$. □

The nodes occurrences that do not have pointers in a justified sequence are called ***initial occurrences***. An initial occurrence is either the root of the computation tree, an @-node or a Σ-node. Let n be occurrence in a justified sequence of nodes t. The subsequence of t consisting of occurrences that are hereditarily justified by the same *initial occurrence* as n is called ***thread*** of n. Thus each thread in a traversal contains a single initial occurrence. The thread of n is given by $n \upharpoonright i$ where i is the first node in t hereditarily justifying n; i is called the ***initial occurrence of the thread of*** n.

1.2.3. Views

The notion of ***P-view*** $\ulcorner t \urcorner$ of a justified sequence of nodes t is defined the same way as the P-view of a justified sequences of moves in Game Semantics:

Definition 1.9 (P-view of justified sequence of nodes)**.** The P-view of a justified sequence of nodes t of $\tau(M)$, written $\ulcorner t \urcorner$, is defined as follows:

$$\begin{array}{rcll} \ulcorner \epsilon \urcorner & = & \epsilon & \\ \ulcorner s \cdot n \urcorner & = & \ulcorner s \urcorner \cdot n & \text{for } n \in N_{\mathsf{var}} \cup N_{\Sigma} \cup N_{@} \cup L_{\lambda} \ ; \\ \ulcorner s \cdot \overparen{m \cdot \ldots \cdot n} \urcorner & = & \ulcorner s \urcorner \cdot \overparen{m \cdot n} & \text{for } n \in L_{\mathsf{var}} \cup L_{\Sigma} \cup L_{@} \cup N_{\lambda} \ ; \\ \ulcorner s \cdot r \urcorner & = & r & \text{if } r \text{ is an occurrence of } \circledast \ (\tau(M)\text{'s root}) \ . \end{array}$$

The equalities in the definition determine pointers implicitly. For instance in the second clause, if in the left-hand side, n points to some node in s that is also present in $\ulcorner s \urcorner$ then in the right-hand side, n points to that occurrence of the node in $\ulcorner s \urcorner$.

The O-view of s, written $\llcorner s \lrcorner$, is defined dually.

Definition 1.10 (O-view of justified sequence of nodes)**.** The O-view of a justified sequence of nodes t of $\tau(M)$, written $\llcorner t \lrcorner$, is defined as follows:

$$\begin{array}{rcll} \llcorner \epsilon \lrcorner & = & \epsilon & \\ \llcorner s \cdot n \lrcorner & = & \llcorner s \lrcorner \cdot n & \text{for } n \in L_{\mathsf{var}} \cup L_{\Sigma} \cup L_{@} \cup N_{\lambda} \ ; \\ \llcorner s \cdot \overparen{m \cdot \ldots \cdot n} \lrcorner & = & \llcorner s \lrcorner \cdot \overparen{m \cdot n} & \text{for } n \in N_{\mathsf{var}} \cup L_{\lambda} \ ; \\ \llcorner s \cdot n \lrcorner & = & n & \text{for } n \in N_{@} \cup N_{\Sigma} \ . \end{array}$$

We borrow some terminology from game semantics:

Definition 1.11. A justified sequence of nodes s satisfies:

- ***Alternation*** if for every two consecutive nodes in s, one is in V_{λ} and not the other one;

- ***P-visibility*** if for every occurrence in s of a node in $N_{\mathsf{var}} \cup L_{\lambda}$, its justifier occur in the P-view a that point;

- ***O-visibility*** if the justifier of each lambda node in s occurs in the O-view a that point.

We then have the same basic property as in game semantics: The P-view (resp. O-view) of a justified sequence satisfying P-visibility (resp. O-visibility) is a well-formed justified sequence satisfying P-visibility (resp. P-visibility). (This property follows by an easy induction.)

1.3. Traversal of the computation tree

We now define the notion of *traversal* over the computation tree $\tau(M)$. We first consider the simply-typed lambda calculus without interpreted constants; everything remains valid in the presence of *uninterpreted* constants as we can just consider them as free variables. In the second section, we extend the notion of traversal to a more general setting with interpreted constants.

1.3.1. Traversals for simply-typed λ-terms

Informally, a traversal is a justified sequence of nodes of the computation tree where each node indicates a step that is taken during the evaluation of the term.

Definition 1.12 (Traversals for simply-typed lambda-terms)**.** The set $\mathcal{T}rav(M)$ of ***traversals*** over $\tau(M)$ is defined by induction over the rules of Table 3. A traversal that cannot be extended by any rule is said to be *maximal*.

Initialization rules

(Empty) $\epsilon \in \mathcal{T}rav(M)$.

(Root) The sequence consisting of a single occurrence of $\tau(M)$'s root is a traversal.

Structural rules

(Lam) If $t \cdot \lambda\overline{\xi}$ is a traversal then so is $t \cdot \lambda\overline{\xi} \cdot n$ where n denotes $\lambda\overline{\xi}$'s child and:

- If $n \in N_{@} \cup N_{\Sigma}$ then it has no justifier;
- if $n \in N_{\mathsf{var}} \setminus N_{\mathsf{fv}}$ then it points to the only occurrence[a] of its binder in $\ulcorner t \cdot \lambda\overline{\xi} \urcorner$;
- if $n \in N_{\mathsf{fv}}$ then it points to the only occurrence of the root $\circledast$ in $\ulcorner t \cdot \lambda\overline{\xi} \urcorner$.

(App) If $t \cdot @$ is a traversal then so is $t \cdot @ \cdot \overset{\curvearrowleft 0}{n}$.

Input-variable rules

(InputVar) If t is a traversal where $t^{\omega} \in N_{\mathsf{var}}^{\circledast\vdash} \cup L_{\lambda}^{\circledast\vdash}$ and x is an occurrence of a variable node in $\llcorner t \lrcorner$ then so is $t \cdot n$ for every child λ-node n of x, n pointing to x.

(InputValue) If $t_1 \cdot x \cdot t_2$ is a traversal with pending node $x \in N_{\mathsf{var}}^{\circledast\vdash}$ then so is $t_1 \cdot x \cdot t_2 \cdot \overset{\curvearrowleft v}{v_x}$ for all $v \in \mathcal{D}$.

Copy-cat rules

(Var) If $t \cdot n \cdot \lambda\overline{x} \ldots \overset{\curvearrowleft i}{x_i}$ is a traversal where $x_i \in N_{\mathsf{var}}^{@\vdash}$ then so is $t \cdot n \cdot \lambda\overline{x} \ldots \overset{\curvearrowleft i}{x_i} \cdot \overset{\curvearrowleft i}{\lambda\overline{\eta_i}}$.

(Value) If $t \cdot m \cdot n \ldots \overset{\curvearrowleft v}{v_n}$ is a traversal where $n \in N$ then so is $t \cdot m \cdot n \ldots \overset{\curvearrowleft v}{v_n} \cdot \overset{\curvearrowleft v}{v_m}$.

Table 3: Traversal rules for the simply-typed lambda calculus.

[a] Prop. 1.1 will show that P-views are paths in the tree thus n's enabler occurs exactly once in the P-view.

Example 1.4. The following justified sequence is a traversal of the computation tree from Example 1.2:

$$t = \lambda fz \cdot @ \cdot \lambda uv \cdot u \cdot \lambda y \cdot f \cdot \lambda \cdot y \cdot \lambda \cdot v \cdot \lambda \cdot z \ .$$

Remark 1.2

1. The rule (Value) from Table 3 can be equivalently reformulated into four distinct rules ($\mathsf{Value}^{\lambda\mapsto @}$), ($\mathsf{Value}^{@\mapsto\lambda}$), ($\mathsf{Value}^{\lambda\mapsto\mathsf{var}}$) and ($\mathsf{Value}^{\mathsf{var}\mapsto\lambda}$), each one dealing with a different possible category for the nodes n and m:

   ($\mathsf{Value}^{\lambda\mapsto @}$) If $t \cdot @ \cdot \lambda\overline{z} \ldots v_{\lambda\overline{z}}$ is a traversal then so is $t \cdot @ \cdot \lambda\overline{z} \ldots v_{\lambda\overline{z}} \cdot v_{@}$.

   ($\mathsf{Value}^{@\mapsto\lambda}$) If $t \cdot \lambda\overline{\xi} \cdot @ \ldots v_{@}$ is a traversal then so is $t \cdot \lambda\overline{\xi} \cdot @ \ldots v_{@} \cdot v_{\lambda\overline{\xi}}$.

   ($\mathsf{Value}^{\lambda\mapsto\mathsf{var}}$) If $t \cdot y \cdot \lambda\overline{\xi} \ldots v_{\lambda\overline{\xi}}$ is a traversal with $y \in N^{@\vdash}_{\mathsf{var}}$ then so is $t \cdot y \cdot \lambda\overline{\xi} \ldots v_{\lambda\overline{\xi}} \cdot v_y$.

   ($\mathsf{Value}^{\mathsf{var}\mapsto\lambda}$) If $t \cdot \lambda\overline{\xi} \cdot x \ldots v_x$ is a traversal where $x \in N_{\mathsf{var}}$ then so is $t \cdot \lambda\overline{\xi} \cdot x \ldots v_x \cdot v_{\lambda\overline{\xi}}$.

   In the rest of this chapter we will prove various resulting by induction on the structure of a traversal and by case analysis on the last rule used to form it. Some of these proofs will rely on the above-defined reformulation of (Value) instead of its original definition.

2. In the rule (InputValue), the last node in the traversal $t_1 \cdot x \cdot t_2$ necessarily belongs to $N_{\mathsf{var}} \cup L_\lambda$. Indeed, since the pending node x is a variable node, the traversal is of the form

   $$\ldots \cdot x \cdot \lambda\overline{\eta}_1 \ldots v^1_{\lambda\overline{\eta}_1} \lambda\overline{\eta}_2 \ldots v^2_{\lambda\overline{\eta}_2} \ldots \lambda\overline{\eta}_k \ldots v^k_{\lambda\overline{\eta}_k}$$

   for some nodes $\lambda\overline{\eta}_k$, values $v^k \in \mathcal{D}$ and $k \geq 0$; thus the last occurrence belongs to N_{var} if $k = 0$ and to L_λ if $k \geq 1$.

   Furthermore, the pending node appears necessarily in the O-view.

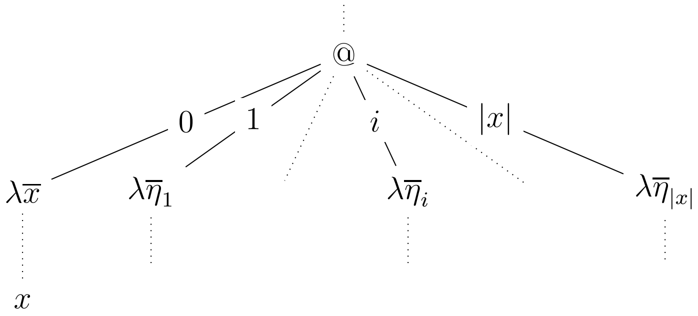
   These two observations show that the rule (InputValue) is essentially a specialization of (InputVar) to value-leaves. The only difference is that (InputVar) allows the visited node to be justified by *any* variable node occurring in the O-view whereas (InputValue) constrains the node to be justified by the pending node (which necessarily occurs in the O-view). This restriction is here to ensure that traversals are well-bracketed.

3. In the rule (Value), it is possible to replace the condition "$n \in N$" by the stronger "$n \in N \setminus N^{\circledast\vdash}_{\lambda}$". Indeed a later result (Lemma 1.6) will show that if n belongs to $N^{\circledast\vdash}_{\lambda}$ then the preceding occurrence m is necessarily an input-variable. Furthermore, another result (Prop. 1.1) shows that traversals are well-bracketed, therefore m is necessarily the pending node. Hence the rule (InputValue) can be use in place of (Value) to visit v_m.

   The advantage of this alternative formulation is that the traversal rules have disjoint domains of definition.

A traversal always starts with the root node and mainly follows the structure of the tree. The exception is the (Var) rule which permits the traversal to jump across the computation tree. The idea is that after visiting a non-input variable node x, a jump can be made to the node corresponding to the subterm that would be substituted for x if all the β-redexes occurring in the term were to be reduced. Let $\lambda\overline{x}$ be x's binder and suppose x is the i^{th} variable in $\overline{x}$. The binding node necessarily occurs previously in the traversal (This will be proved in Prop. 1.1). Since x is not hereditarily justified by the root, $\lambda\overline{x}$ is not the root of the tree and therefore it is not the first node of the traversal. We do a case analysis on the node preceding $\lambda\overline{x}$:

- If it is an @-node then $\lambda\overline{x}$ is necessarily the first child node of that node and it has exactly $|\overline{x}|$ siblings:

  
  @
0
1
i
$|x|$
$\lambda\overline{x}$
$\lambda\overline{\eta}_1$
$\lambda\overline{\eta}_i$
$\lambda\overline{\eta}_{|x|}$
x

  In that case, the next step of the traversal is a jump to $\lambda\overline{\eta_i}$—the i^{th} child of @—which corresponds to the subterm that would be substituted for x if the β-reduction was performed:

  $$t' \cdot @ \cdot \lambda\overline{x} \cdot \ldots \cdot \overset{i}{x} \cdot \overset{i}{\lambda\overline{\eta_i}} \cdot \ldots \in \mathcal{T}rav(M) \enspace .$$

- If it is a variable node y, then the node $\lambda\overline{x}$ was necessarily added to the traversal $t_{\leq y}$ using the (Var) rule. (Indeed, if it was visited using (InputVar) then $\lambda\overline{x}$ would be hereditarily justified by the root, but this is not possible since x_i, bound by $\lambda\overline{x}$, is not an input-variable.) Therefore y is substituted by the term $\kappa(\lambda\overline{x})$ during the evaluation of the term.

  Consequently, during reduction, the variable x will be substituted by the subterm represented by the i^{th} child node of y. Hence the following justified sequence is also a traversal:

  $$t' \cdot y \cdot \lambda\overline{x} \cdot \ldots \cdot \overset{i}{x} \cdot \overset{i}{\lambda\overline{\eta_i}} \cdot \ldots$$

Remark 1.3 Our notions of computation tree and traversal differ slightly from the original definitions by Ong [1]. In his setting:

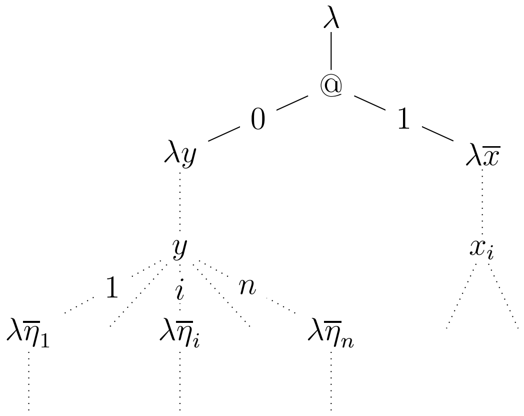
- computation trees contain (uninterpreted first-order) constants. Here we have not accounted for constants but as previously observed, uninterpreted constants can just be regarded as free variables, thus we do not lose any expressivity here.

- constants are restricted to order one at most. (Terms are used as generators of trees where first-order constants act as tree-node constructors). Here we do not need this restriction: as long as constants are uninterpreted we can regard them as free variables, even at higher-orders.

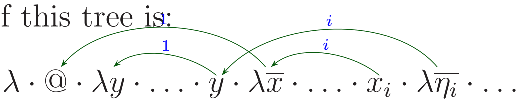
- one rule ((Sig)) suffices to model the first-order constants. In contrast our setting accounts for higher-order variables, thus the more complicated rules (InputValue) and (InputVar) are required.

- computation trees do not have value-leaves. These are not necessary to model the pure simply-typed lambda calculus. There will be necessary, however, when it comes to model interpreted constants such as those of PCF or IA.

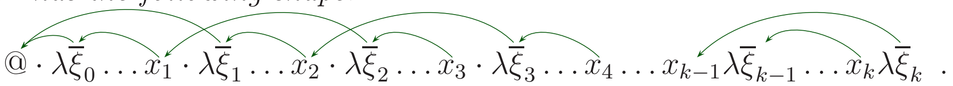
Example 1.5. Consider the following computation tree:


λ
@
0
1
λy
$\lambda \overline{x}$
y
x_i
1
i
n
$\lambda \overline{\eta}_1$
$\lambda \overline{\eta}_i$
$\lambda \overline{\eta}_n$

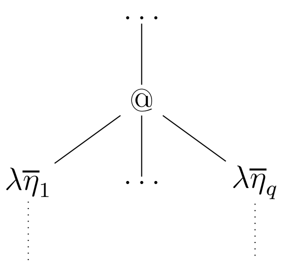
An example of traversal of this tree is:


1
i
1
i
$\lambda \cdot @ \cdot \lambda y \cdot \ldots \cdot y \cdot \lambda \overline{x} \cdot \ldots \cdot x_i \cdot \lambda \overline{\eta_i} \cdot \ldots$

Lemma 1.2. *Take a traversal t ending with an inner node hereditarily justified by an application node @. Then if we represent only the nodes appearing in the O-view, the thread of t^ω has the following shape:*


$@ \cdot \lambda \overline{\xi}_0 \ldots x_1 \cdot \lambda \overline{\xi}_1 \ldots x_2 \cdot \lambda \overline{\xi}_2 \ldots x_3 \cdot \lambda \overline{\xi}_3 \ldots x_4 \ldots x_{k-1} \lambda \overline{\xi}_{k-1} \ldots x_k \lambda \overline{\xi}_k$.

Suppose that the initial node @ occurs in the computation as follows:



Let τ_i denote the sub-tree rooted at $\lambda\overline{\eta}_i$ for $i \in \{1..q\}$. Then for every $j \in \{1..k\}$, x_j and $\lambda\overline{\xi}_j$ must belong to two different subtrees τ_i and $\tau_{i'}$. Furthermore, x_j is hereditarily justified by some occurrence of $\lambda\overline{\eta}_i$ in t and $\lambda\overline{\xi}_j$ is hereditarily justified by some occurrence of $\lambda\overline{\eta}_{i'}$ in t (and therefore $\lambda\overline{\xi}_j \in V^{\lambda\overline{\eta}_i\vdash}$ and $x_j \in V^{\lambda\overline{\eta}_{i'}\vdash}$).

Proof. The proof is by an easy induction. □

1.3.2. Traversal rules for interpreted constants

The framework that we have established up to now aims at providing a computation model of simply-typed lambda-terms. It is possible to extend it to other extensions of the simply-typed lambda calculus. This is done by completing the traversal rules from Table 3 with new rules describing the behaviour of the interpreted constants of the language considered. For instance in the case of PCF, we need to define rules for the interpreted constant `cond` that replicate the behaviour of the conditional operation. (In a forthcoming section of this chapter we will give a complete definition of the constant traversal rules for PCF and IA.)

We mentioned before that uninterpreted constants can be regarded as free variables. In the same way, we can consider interpreted constants as a *generalization* of free variables: for both of them, the "code" describing their computational behaviour is not defined within the scope of the term, it is instead assumed that the environment knows how to interpret them. Free variables, however, are more restricted than interpreted constants: When evaluating an applicative term with a free variable in head position, the evaluation of the head variable does not depend on the result of the evaluation of its parameters; whereas for applicative term with an interpreted constant in head position, the outcome of the evaluation may depend on the result of the evaluation of its parameters (*e.g.*, the PCF constant `cond` branches between two control points depending on the result of the evaluation of its first parameter).

We can thus derive a prototype for constant traversal rules by generalizing the input-variable rules ($\mathsf{InputValue}$) and ($\mathsf{InputVar}$):

Definition 1.13 (Constant traversal rule)**.** A ***constant traversal*** has one of the following two forms:

$$(\Sigma\text{-Value})\ \frac{t = t_1 \cdot \alpha \cdot t_2 \in \mathcal{T}rav(M) \quad \alpha \in N_\Sigma \cup N_{\mathsf{var}}^{N_\Sigma\vdash} \quad ?(t)^\omega = \alpha \quad P(t)}{t' = t_1 \cdot \overset{\curvearrowleft}{\alpha \cdot t_2 \cdot v(t)} \in \mathcal{T}rav(M)}$$

or

$$(\Sigma)/(\Sigma\text{-Var})\ \frac{t \in \mathcal{T}rav(M) \quad t^\omega \in N_\Sigma \cup N^{N_\Sigma\vdash} \cup L_\lambda \quad P(t)}{t \cdot n(t) \in \mathcal{T}rav(M)}$$

where:

- $P(t)$ is a predicate expressing some condition on t;
- $v(t)$ is a value-leaf of the node α that is determined by the traversal t;
- $n(t)$ is a lambda-node determined by t, and its link—also determined by t—points to some occurrence of its parent node in $\llcorner t \lrcorner$.

Clearly, such rules preserve well-bracketing, alternation and visibility.

Remark 1.4 The extra power of the constant rules over the input-variable rules (InputValue) and (InputVar) comes from their ability to base their choice of next visited node on the shape of the traversal t.

From now on, to make our argument as general as possible, we consider a simply-typed lambda calculus language extended with higher-order interpreted constants for which some constant traversal rules have been defined (in the sense of Def. 1.13). Furthermore, we complete the set of rules with the following additional copy-cat rule:

$$(\mathsf{Value}^{\Sigma\mapsto\lambda})\ t \cdot \lambda\overline{\xi} \cdot \overset{v}{\overparen{c \ldots v_c}} \in \mathcal{T}rav(M) \wedge\ c \in \Sigma \quad\Longrightarrow\quad t \cdot \overset{v}{\overparen{\lambda\overline{\xi} \cdot \overset{v}{\overparen{c \ldots v_c}} \cdot v_{\lambda\overline{\xi}}}} \in \mathcal{T}rav(M)\ .$$

Definition 1.14. A constant traversal rules is ***well-behaved*** if for every traversal $t \cdot \overparen{\alpha \cdot u \cdot n}$ formed with the rule we have $?(u) = \epsilon$.

An example is the rule (Σ-Value) which is well-behaved due to the fact that traversals are well-bracketed. The rule (Σ)/(Σ-Var), however, is not well-behaved since the node $n(t)$ does not necessarily points to the pending node in t.

Lemma 1.3. *If Σ-constants have order 1 at most, then constant rules are necessarily all well-behaved.*

Proof. In the computation tree, an order-1 constant hereditarily enables only its immediate children (which are all dummy lambda nodes λ). Hence a traversal formed with the rule (Σ)/(Σ-Var) is of the form:

$$t = \ldots \cdot \overparen{\alpha \cdot u \cdot \lambda}$$

where α appears in $\llcorner t \lrcorner$.

If $u = \epsilon$ then the result trivially holds. Otherwise, u's first node has necessarily been visited with the rule (Σ)/(Σ-Var) thus u's first node is a dummy lambda node λ' pointing to α. Since α occurs in $\llcorner t \lrcorner$ and since the node λ' enables only its value-leaf in the computation tree, t must be of the following shape:

$$t = \ldots \cdot \overparen{\alpha \cdot \underbrace{\overparen{\lambda' \ldots v_{\lambda'}} \ldots}_{u} \lambda}$$

for some value leaf $v_{\lambda'}$ of λ'.

Again, the node following $v_{\lambda'}$ must be a dummy lambda node pointing to α. By iterating the same argument we obtain that the segment u is a repetition of segments of the form $\overparen{\lambda' \cdot \ldots v_{\lambda'}}$. Hence $?(u) = \epsilon$. □

1.3.3. Property of traversals

Proposition 1.1. *Let t be a traversal. Then:*

(i) t is a well-defined justified sequence satisfying alternation, well-bracketing, P-visibility and O-visibility;

(ii) If the last element of t is not a value-leaf whose parent-node is a lambda node (i.e., $t^\omega \not\in L_\lambda$) then $\ulcorner t \urcorner$ is the path in the computation tree going from the root to the node t^ω.

Proof. This is the counterpart of another result proved by Ong in the paper where he introduces the theory of traversals [5, proposition 6]. The original proof—an induction on the traversal rules—can be adapted to take into account the constant rules and the presence of value-leaves in the traversal. We detail the case (Lam) only. We need to show that n's binder occurs only once in the P-view at that point. By the induction hypothesis (ii) we have that $\ulcorner t \cdot \lambda\overline{\xi} \urcorner$ is a path in the computation tree from the root to $\lambda\overline{\xi}$. But n's binder occurs only once in this path, therefore the traversal $t \cdot \lambda\overline{\xi} \cdot n$ is well-defined and satisfies P-visibility. Thus (i) is satisfied. Furthermore n is a child of $\lambda\overline{\xi}$ therefore (ii) also holds. □

Lemma 1.4. *If $t \cdot n$ is a traversal with $n \in N_{\mathsf{var}} \cup N_\Sigma \cup N_@$ then $t \neq \epsilon$ and t^ω is n's parent in $\tau(M)$ (and is thus a lambda node).*

Proof. By inspecting the traversal rules, we observe that (Lam) is the only rule which can visit a node in $N_{\mathsf{var}} \cup N_\Sigma \cup N_@$. Hence t is not empty and t^ω is n's parent in $\tau(M)$. □

Lemma 1.5. *Suppose that M is β-normal. Let t be a traversal of $\tau(M)$ and n be a node occurring in t. Then the root $\circledast$ does not hereditarily enable n if and only if n is hereditarily enabled by some node in N_Σ. Formally:*

$$n \not\in N^{\circledast\vdash} \quad \iff \quad n \in N^{N_\Sigma\vdash} \ .$$

Proof. In a computation tree, the only nodes that do not have justification pointer are: the root $\circledast$, @-nodes and Σ-constant nodes. But since M is in β-normal form, there is no @-node in the computation tree. Hence nodes are either hereditarily enabled by $\circledast$ or hereditarily enabled by some node in N_Σ. Moreover $\circledast$ is not in N_Σ therefore the "or" is exclusive: a node cannot be both hereditarily enabled by $\circledast$ and by some node in N_Σ. □

Lemma 1.6 (The O-view is contained in a single thread)**.** *Let $t \in \mathcal{T}rav(M)$.*

(a) If $t = \ldots \cdot m \cdot n$ where $m \in N_{\mathsf{var}} \cup N_\Sigma \cup N_@ \cup L_\lambda$ and $n \in N_\lambda \cup L_{\mathsf{var}} \cup L_\Sigma \cup L_@$ then m and n are in the same thread in t: they are hereditarily justified by the same initial occurrence (which is either $\tau(M)$'s root, a Σ-constant or an @-node);

(b) All the nodes in $\llcorner t \lrcorner$ belong to the same thread.

Proof. Clearly (b) follows immediately from (a) due to the way the O-view is computed. We show (a) by induction on the last traversal rule used to form t. The results trivially hold for the base cases (Empty) and (Root). Step case: Take $t = t' \cdot n$. If $n \in N_\lambda \cup L_{\mathsf{var}} \cup L_\Sigma \cup L_@$ then we do not need to show (a). Otherwise $n \in N_\lambda \cup L_{\mathsf{var}} \cup L_\Sigma \cup L_@$. By O-visibility, n points in $\llcorner t' \lrcorner$, thus by the I.H., it must belong to the same thread as all the nodes in $\llcorner t' \lrcorner$ and in particular to the thread of t'^ω. Therefore both (i) and (ii) hold. □

1.3.4. Traversal core

Occurrences of input-variable nodes correspond to point of the computation at which the term interacts with its context. At these points, a traversal can be extended in a non-deterministic way. In contrast, after a node that is hereditarily enabled by an @-node or by a constant node, the next visited node is uniquely determined. We can therefore think of such nodes as being "internal" to the computation: their semantics is predefined and cannot be altered by the context in which the term appears. If we want to extract the essence of the computation from a traversal, a natural way to proceed thus consists in keeping only occurrences of nodes that are hereditarily enabled by the root:

Definition 1.15. The ***core of a traversal*** t, written $t \upharpoonright \circledast$, is defined as $t \upharpoonright V^{\circledast\vdash}$ (*i.e.*, the subsequence of t consisting of the occurrences of nodes that are hereditarily enabled by the root $\circledast$ of the computation tree). The set of traversal cores of M is denoted by $\mathcal{T}rav(M)^{\upharpoonright\circledast}$:

$$\mathcal{T}rav(M)^{\upharpoonright\circledast} \overset{\mathsf{def}}{=} \{t \upharpoonright \circledast \ : \ t \in \mathcal{T}rav(M)\} \ .$$

Example 1.6. The core of the traversal given in example 1.4 is:

$$\begin{tikzcd}[column sep=tiny] t \upharpoonright \lambda f z = \lambda f z & \cdot & f \arrow[ll, bend right] & \cdot & \lambda \arrow[ll, bend right] & \cdot & z \ . \arrow[llllll, bend right] \end{tikzcd}$$

Remark 1.5

- The root occurs at most once in a traversal, therefore if t is a non-empty traversal then its core is given by $t \upharpoonright r$ where r denotes the only occurrence of $\circledast$ in t. Thus we have:

$$\mathcal{T}rav(M)^{\upharpoonright\circledast} = \{t \upharpoonright r \ : \ t \in \mathcal{T}rav(M) \text{ and } r \text{ is the only occurrence of } \circledast \text{ in } t\} \ .$$

- Since @-nodes and Σ-constants do not have pointers, the traversal cores contains only nodes in $V_\lambda \cup V_{\mathsf{var}}$.

1.3.5. Removing @-nodes and Σ-nodes from traversals

Application nodes are essential in the definition of computation trees: they are necessary to connect together the operator and operands of an application. They also have another advantage: they ensure that the lambda-nodes are all at even level in the computation tree, which subsequently guarantees that traversals respect a certain form of alternation between lambda nodes and non-lambda nodes. Application nodes are however redundant in the sense that they do not play any role in the computation of the term. In fact it will be necessary to filter them out in order to establish the correspondence with interaction game semantics.

Definition 1.16 (@-free traversal)**.** Let t be a traversal of $\tau(M)$. We write $t - @$ for the sequence of nodes-with-pointers obtained by

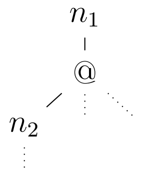
- removing from t all occurrences of @-nodes and their children value-leaves;
- replacing any link pointing to an @-node by a link pointing to the immediate predecessor of @ in t.

Suppose $u = t - @$ is a sequence of nodes obtained by applying the previously defined transformation on the traversal t, then t can be partially recovered from u by reinserting the @-nodes as follows. For each @-node in the computation tree with parent node denoted by p, we perform the following operations:

1. replace every occurrence of the pattern $p \cdot n$ for some λ-node n, by $p \cdot @ \cdot n$;
2. replace any link in u starting from a λ-node and pointing to p by a link pointing to the inserted @-node;
3. for each occurrence in u of a value-leaf v_p pointing to p, insert the value-leaf $v_@$ immediately before v_p and make it point to the immediate successor of p (which is precisely the @-node inserted in step 1.).

We write $u + @$ for this second transformation.

These transformations are well-defined because in a traversal, an @-node is always immediately preceded by its parent node n_1, and immediately followed by its first child n_2:



Example 1.7. Let f be a Σ-constant and $t = \lambda\overline{\xi} \cdot @ \cdot \lambda x \cdot f \cdot \lambda \cdot x$. Then

$$t - @ = \lambda\overline{\xi} \cdot \lambda x \cdot f \cdot \lambda \cdot x \ .$$

Example 1.8. Let t be the traversal given in example 1.4, we have:

$$t - @ = \lambda f z \cdot \lambda u v \cdot u \cdot \lambda y \ f \cdot \lambda \cdot y \cdot \lambda \cdot v \cdot \lambda \cdot z \ .$$

We also want to remove Σ-nodes form the traversals. To that end we define the operation $-\Sigma$ and $+\Sigma$ in the exact same way as $-@$ and $+@$. Again these transformations are well-defined since in a traversal, a Σ-node f is always immediately preceded by its parent node p, and a value-node v_p is always immediately preceded by a value-node v_f.

Note that the operations $-@$ and $-\Sigma$ are commutative: $(t - @) - \Sigma = (t - \Sigma) - @$.

Lemma 1.7. *For every non-empty traversal $t = t' \cdot t^\omega$ in $\mathcal{T}rav(M)$:*

$$(t - @) + @ = \begin{cases} t, & \text{if } t^\omega \notin V_@ \ ; \\ t', & \text{if } t^\omega \in V_@ \ ; \end{cases}$$

$$(t - \Sigma) + \Sigma = \begin{cases} t, & \text{if } t^\omega \notin V_\Sigma \ ; \\ t', & \text{if } t^\omega \in V_\Sigma \ . \end{cases}$$

Proof. The result follows immediately from the definition of the operation -@ and +@ (resp. $-\Sigma$ and $+\Sigma$). □

Remark 1.6 Sequences of the form $t - @$ (resp. $t - \Sigma$) are not, strictly speaking, proper justified sequences of nodes since after removing @-nodes, all the prime λ-nodes become justified by their parent's parent which are also λ-nodes! Moreover, these sequences do not respect alternation since two λ-nodes may become adjacent after removing a @-node.

We write $t^\star$ to denote the sequence obtained from t by removing all the @-nodes as well as the constant nodes together with their associated value-leaves:

$$t^\star \overset{\mathsf{def}}{=} t - @ - \Sigma \ .$$

Example 1.9. Let f be a Σ-constant. We have

$$\left(\lambda\overline{\xi} \cdot @ \cdot \lambda x \cdot f \cdot \lambda \cdot x\right)^\star = \lambda\overline{\xi} \cdot \lambda x \cdot \lambda \cdot x \ .$$

We introduce the set

$$\mathcal{T}rav(M)^\star = \{t^\star \mid t \in \mathcal{T}rav(M)\} \ .$$

Remark 1.7 If M is a β-normal term and if it contains no Σ-constant (as for pure simply-typed terms) then $\tau(M)$ does not contain any @-node or Σ-node, thus all nodes are hereditarily enabled by $\circledast$ and we have $\mathcal{T}rav(M) = \mathcal{T}rav(M)^{\upharpoonright \circledast} = \mathcal{T}rav(M)^\star$.

Lemma 1.8. *For every traversal t we have $t^\star \upharpoonright V^{\circledast\vdash} = t \upharpoonright \circledast$.*

Proof. This is because nodes removed by the operation $\_^\star$ are not hereditarily enabled by the root of the tree. □

The notion of P-view extends naturally to sequences of the form $t^\star$: it is defined by the same induction as for P-views of traversals. It is then easy to check that if t^ω is not in $L_@ \cup L_\Sigma$ then the P-view of $t^\star$ is obtained from $\ulcorner t \urcorner$ by keeping only the non @/Σ-nodes:

$$\ulcorner t^\star \urcorner = \ulcorner t \urcorner \setminus (V_@ \cup V_\Sigma) \ . \tag{1}$$

We define a projection operation for sequences of the form $t^\star$ as follows:

Definition 1.17. Let t be a traversal such that $t^\omega \not\in L_@ \cup L_\Sigma$ and r_0 be an occurrence of some lambda-node n. Then the projection $t^\star \upharpoonright V^{(n)}$ is defined as the subsequence of $t^\star$ consisting of nodes of $V^{(n)}$ only. If a variable node loses its pointer in $t^\star \upharpoonright V^{(n)}$ then its justifier is reassigned to the only occurrence of n in $\ulcorner t^\star \urcorner$.

Note that this operation is well-defined. Indeed if a variable x loses its pointer in $t^\star \upharpoonright V^{(n)}$ then it means that x is free in $M^{(n)}$. But then n must occur in the path to the root $\circledast$ which is precisely $\ulcorner t_{\leqslant x} \urcorner$. Thus by (1), n must occur in $\ulcorner {t_{\leqslant x}}^\star \urcorner$.

1.3.6. Subterm projection (with respect to a node occurrence)

Let n_0 be a node-occurrence in a traversal t. The ***subterm projection*** $t \upharpoonright\!\upharpoonright n_0$ is defined as the subsequence of t consisting of the occurrences whose P-view at that point contain the node n_0. Formally:

Definition 1.18. Let $t \in \mathcal{T}rav(M)$ and n_0 be an occurrence in t. The subsequence $t \upharpoonright\!\upharpoonright n_0$ of t is defined inductively on t as follows:

- $(t \cdot n_0) \upharpoonright\!\upharpoonright n_0 = n_0$;
- If $n \in N_\lambda \cup L_{\mathsf{var}} \cup L_\Sigma \cup L_@$ and $n \neq n_0$ then

$$(t \cdot n) \upharpoonright\!\upharpoonright n_0 = \begin{cases} (t \upharpoonright\!\upharpoonright n_0) \cdot n, & \text{if } n\text{'s justifier appears in } t \upharpoonright\!\upharpoonright n_0 \ ; \\ t \upharpoonright\!\upharpoonright n_0, & \text{otherwise ;} \end{cases}$$

- If $n \in N_{\mathsf{var}} \cup N_\Sigma \cup N_@ \cup L_\lambda$ and $n \neq n_0$ then

$$(t \cdot n) \upharpoonright\!\upharpoonright n_0 = \begin{cases} (t \upharpoonright\!\upharpoonright n_0) \cdot n, & \text{if } t^\omega\text{'s appears in } t \upharpoonright\!\upharpoonright n_0 \ ; \\ t \upharpoonright\!\upharpoonright n_0, & \text{otherwise ;} \end{cases}$$

  where in the first subcase, if n loses its justifier in $t \upharpoonright\!\upharpoonright r_0$ then it is reassigned to r_0.

We call this transformation the *subterm projection with respect to a node occurrence* because it keeps only nodes that appear in the sub-tree rooted at some reference node. If n_0 is an occurrence of a lambda node $n \in N_\lambda$ then we say that $t \upharpoonright\!\upharpoonright n_0$ a ***sub-traversal of the computation tree*** $\tau(M)$. This name is suggestive of the forthcoming Proposition 1.5 stating that $t \upharpoonright\!\upharpoonright n_0$ is a traversal of the sub-computation tree of $\tau(M)$ rooted at n.

Remark 1.8 There is an alternative way to define $t \upharpoonright\!\upharpoonright r_0$: For every traversal t we write t^+ to denote the sequence-with-pointers obtained from t by adding pointers as follows: For every occurrence of a @ or Σ-node m in t we add a pointer going from m to its predecessor in t (which is necessarily an occurrence of its parent node). Further, for every variable node x we add auxiliary pointers going to each lambda node occurring in the P-view at that point after x's binder. Conversely, for every sequence-with-pointers u we define u^- as the sequence obtained from u by removing the links associated to @ and Σ-nodes and where for each occurrence of a variable node, only the "longest" link is preserved. (The length of a link being defined as the distance between the source and the target occurrence.) Clearly the operation $\_^-$ is the inverse of $\_^+$: For every traversal t we have $t = (t^+)^-$. Then it can be easily shown that the sequence $t \upharpoonright\!\upharpoonright n$ is precisely the subsequence of t consisting of nodes hereditarily justified by n *with respect to the justification pointers of* t^+:

$$t \upharpoonright\!\upharpoonright n = (t^+ \upharpoonright n)^- \ .$$

(Note that since the operation $\_^+$ changes the justification pointers, the hereditary justification relation in a traversal t is different from the hereditary justification relation in t^+ and therefore we have $(t \upharpoonright n)^+ \sqsubseteq t^+ \upharpoonright n$ but $(t \upharpoonright n)^+ \neq t^+ \upharpoonright n$.) End of remark.

The following lemmas follow directly from the definition of $t \upharpoonright\!\upharpoonright r_0$:

Lemma 1.9. *Let t be a traversal and r_0 be an occurrence of a lambda node r' in t.*

(a) Suppose that $t = \ldots \overset{\curvearrowleft}{m \ldots n}$ with $n \in N_\lambda \cup L_@ \cup L_\Sigma \cup L_{\mathsf{var}}$ and $n \neq r_0$. Then n appears in $t \upharpoonright\!\upharpoonright r_0$ if and only if m appears in $t \upharpoonright\!\upharpoonright r_0$.

(b) Suppose that $t = \ldots \cdot n$ where $n \in N_{\mathsf{var}} \cup N_@ \cup N_\Sigma \cup L_\lambda$. Then n appears in $t \upharpoonright\!\upharpoonright r_0$ if and only if the last lambda node in $\ulcorner t \urcorner$ does.

(c) Suppose that $t = \ldots \overset{\curvearrowleft}{m \ldots v_m}$ with $v_m \in L = L_\lambda \cup L_@ \cup L_\Sigma \cup L_{\mathsf{var}}$. Then v_m appears in $t \upharpoonright\!\upharpoonright r_0$ if and only if m does.

Proof. (a) holds by definition of $t \upharpoonright\!\upharpoonright r_0$. (b) is proved by induction on t: It follows easily from the fact that in the definition of $t \upharpoonright\!\upharpoonright r_0$, the inductive cases follow those from the definition of traversal P-views. (c) If $v_m \in L_@ \cup L_\Sigma \cup L_{\mathsf{var}}$ then it falls back to (a). Otherwise $v_m \in L_\lambda$ and by (b), v_m appears in $t \upharpoonright\!\upharpoonright r_0$ if and only if the last lambda node in $\ulcorner t \urcorner$ does. But the last node in $\ulcorner t \urcorner$ is necessarily m (since v_m is necessarily visited with a copy-cat rule). □

Lemma 1.10. *Let $t \in \mathcal{T}rav(M)$ and r_0 be the occurrence in t of a λ-node. We have:*

$$?(t \upharpoonright\!\upharpoonright r_0) = ?(t) \upharpoonright\!\upharpoonright r_0 \ .$$

Proof. Take a prefix u of t ending with a value-leaf v_n of an occurrence n. By Lemma 1.9(c), the operation $\_ \upharpoonright\!\upharpoonright r_0$ removes v_n from t if and only if it also removes n. □

1.3.7. O-view and P-view of the subterm projection

P-view projection.

Lemma 1.11 (P-view Projection for traversals)**.** *Let t be a traversal and r_0 be an occurrence in t of a lambda node $r' \in N_\lambda$. Then:*

(i) If t^ω appears in $t \upharpoonright\!\upharpoonright r_0$ then:

a. r_0 appears in $\ulcorner t \urcorner$, all the nodes occurring after r_0 in $\ulcorner t \urcorner$ appear in $t \upharpoonright\!\upharpoonright r_0$ and all the nodes occurring before r_0 in $\ulcorner t \urcorner$ do not appear in $t \upharpoonright\!\upharpoonright r_0$;

b. $\ulcorner t \upharpoonright\!\upharpoonright r_0 \urcorner^{M^{(r')}} = \ulcorner t \urcorner^{M}_{\geqslant r_0} = r_0 \cdot \ldots$;

c. if t^ω also appears in $t \upharpoonright\!\upharpoonright r_1$ for some occurrence r_1 r' then $r_0 = r_1$;

d. if $t = \ldots \overset{\curvearrowleft}{m \ldots n}$ and m does not appear in $t \upharpoonright\!\upharpoonright r_0$ then r_0 occurs after m in t and m is a free variable node in the sub-computation tree $\tau(M^{(r')})$.

(ii) Suppose $t = \ldots r_0 \ldots \overset{\curvearrowleft}{m \ldots n}$. Then the node n appears in $t \upharpoonright\!\upharpoonright r_0$ if and only if m does.

Proof. (i) A trivial induction shows both a. and b.(The inductive steps in the definition of the projection operation $\_ \upharpoonright\!\upharpoonright r_0$ correspond precisely to those from the definition of P-views.)

c. By a., both r_0 and r_1 appears in the P-view. But the P-view is the path from t^ω to the root, hence it cannot contain two different occurrences of the same node r'.

d. Since t^ω appears in $t \upharpoonright\!\upharpoonright r_0$ and its justifier m is not in $t \upharpoonright\!\upharpoonright r_0$, by a., the justifier m necessarily precedes r_0 in t, and by Lemma 1.9, n is necessarily a variable node. Thus m occurs before r_0 in the P-view $\ulcorner t \urcorner$. In other words, r_0 lies in the path from n to its binder m. Consequently, n is a free variable node in $\tau(M^{(r')})$.

(ii) The case $n \notin N_{\mathsf{var}}$ is handled by Lemma 1.9(a) and (c).

Suppose that $n \in N_{\mathsf{var}}$. If n appears in $t \upharpoonright\!\upharpoonright r_0$ then by (i) all the nodes occurring in $\ulcorner t \urcorner$ up to r_0 appear in $t \upharpoonright\!\upharpoonright r_0$. By P-visibility, m appears in $\ulcorner t \urcorner$ and since r_0 precedes it by assumption, m also appears in $t \upharpoonright\!\upharpoonright r_0$. If m appears in $t \upharpoonright\!\upharpoonright r_0$ then since m appears in the P-view at x, by definition of $t \upharpoonright\!\upharpoonright r_0$, x must also appear in $t \upharpoonright\!\upharpoonright r_0$. □

Lemma 1.12. *Let $t \in \mathcal{T}rav(M)$ such that $t^\omega \notin L_\lambda$. Let r' be some lambda node in N_λ.*

The node t^ω belongs to the subtree of $\tau(M)$ rooted at r' (i.e., $t^\omega \in V^{(r')}$) if and only if t^ω appears in $t \upharpoonright\!\upharpoonright r_0$ for some occurrence r_0 of r' in t.

Proof. Only if part: Since t's last move in not a lambda leaf, by Proposition 1.1, the P-view $\ulcorner t \urcorner$ is the path to the root $\circledast$. Hence since t^ω belongs to the subtree of $\tau(M)$ rooted at r', $\ulcorner t \urcorner$ must contain (exactly) one occurrence r_0 of r'. But then by definition of $t \upharpoonright\!\upharpoonright r_0$, all the nodes following r_0 occurring in the P-view must also belong to $t \upharpoonright\!\upharpoonright r_0$, so in particular, t^ω does.

If part: By Lemma 1.11(i), r_0 must occur in $\ulcorner t \urcorner$ and therefore r_0 lies in the path from t^ω to the root $\circledast$ of the computation tree $\tau(M)$. Consequently, t^ω necessarily belongs to the subtree of $\tau(M)$ rooted at r'. □

Lemma 1.13. *Let t be a traversal and r_0 be an occurrence in t of some lambda node r'. Then an occurrence $n \notin V_@ \cup V_\Sigma$ of t is hereditarily justified by n_0 in $t^\star \upharpoonright V^{(r')}$ if and only if n appears in $t \upharpoonright\!\upharpoonright r_0$.*

Proof. We proceed by induction on $t_{\leqslant n}$. If $n = r_0$ or if r_0 does not occur in $t_{\leqslant n}$ then the result holds trivially. Suppose that r_0 occurs in $t_{<n}$. Let m be n's justifier in t. We do a case analysis on n. The case $n \in L_@ \cup L_\Sigma \cup N_@ \cup N_\Sigma$ is excluded by assumption.

Suppose $n \in L_\lambda \cup L_{\mathsf{var}} \cup N_\lambda$ then

$$\begin{aligned}
n \text{ appears in } t \upharpoonright\!\upharpoonright r_0 &\iff m \text{ appears in } t \upharpoonright\!\upharpoonright r_0 && \text{by Lemma 1.9(a)}\\
&\iff m \text{ her. just. by } n_0 \text{ in } t^\star \upharpoonright V^{(r')} && \text{by I.H. on } t_{\leqslant m}\\
&\iff n \text{ her. just. by } n_0 \text{ in } t^\star \upharpoonright V^{(r')} && \text{since } m \text{ is } n\text{'s parent in } \tau(M^{(r')}).
\end{aligned}$$

Suppose that $n \in N_{\mathsf{var}}$ then

$$\begin{aligned}
n \text{ appears in } t \upharpoonright\!\upharpoonright r_0 &\iff r_0 \text{ appears in } \ulcorner t \urcorner && \text{by Lemma 1.12 and 1.11(i)}\\
&\iff \left\{\begin{array}{l} r_0 \text{ precedes } m \text{ in } \ulcorner t \urcorner, \text{ and thus } n \text{ is a bound variable in } M^{(r')}\\ \text{or } r_0 \text{ appears strictly after } m \text{ in } \ulcorner t \urcorner \text{ and } n \text{ is free in } M^{(r')} \end{array}\right.\\
&\iff \left\{\begin{array}{ll} m \text{ appears in } t \upharpoonright\!\upharpoonright r_0 & \text{by Lemma 1.11(i)}\\ \text{or } n \text{ points to } r_0 \text{ in } t^\star \upharpoonright V^{(r')} & \text{by def. of } \_ \upharpoonright V^{(r')} \end{array}\right.\\
&\iff \left\{\begin{array}{ll} m \text{ her. just. by } n_0 \text{ in } t^\star \upharpoonright V^{(r')} & \text{by I.H. on } t_{\leqslant m}\\ \text{or } n \text{ points to } r_0 \text{ in } t^\star \upharpoonright V^{(r')} & \end{array}\right.\\
&\iff \left\{\begin{array}{ll} n \text{ her. just. by } n_0 \text{ in } t^\star \upharpoonright V^{(r')} & n \text{ is in } V^{(r')} \text{ iff its binder } m \text{ is}\\ \text{or } n \text{ points to } r_0 \text{ in } t^\star \upharpoonright V^{(r')} & \end{array}\right.\\
&\iff n \text{ is her. just. by } n_0 \text{ in } t^\star \upharpoonright V^{(r')} \quad .
\end{aligned}$$

□

Lemma 1.14. *Take a traversal t. Let r' be a node in N_λ and r_0 an occurrence of r' in t. Suppose that t^ω appears in $t \upharpoonright\!\upharpoonright r_0$ and that the thread of t^ω is initiated by $\alpha \in N_@ \cup N_\Sigma$.*

(i) If r_0 precedes α in t then all the nodes occurring in the thread appear in $t \upharpoonright\!\upharpoonright r_0$.

(ii) If α precedes r_0 in t then t^ω is hereditarily enabled by r' in $\tau(M^{(r')})$.

Proof. (i) By definition of a thread, the nodes occurring in the thread are all hereditarily justified by α. Since r_0 precedes α and t^ω appears in $t \upharpoonright\!\upharpoonright r_0$, by Lemma 1.11(ii) all the nodes in the thread must also appear in $t \upharpoonright\!\upharpoonright r_0$.

(ii) Let q be the first node in t that hereditarily justifies t^ω in t and that appears in $t \upharpoonright\!\upharpoonright r_0$.

If $q \in N_\lambda$ then necessarily $q = r_0$. Otherwise by definition of $\_ \upharpoonright\!\upharpoonright r_0$, q's justifier also appears in $t \upharpoonright\!\upharpoonright r_0$ which contradicts the definition of q. Hence the result holds trivially.

If $q \in N_@ \cup N_\Sigma$ then necessarily $q = \alpha$, since links always point inside the current thread and since a thread contains by definition only one node in $N_@ \cup N_\Sigma$. But α precedes r_0 therefore α cannot be hereditarily justified by r_0 hence this case is not possible.

If $q \in N_{\mathsf{var}}$ then by Lemma 1.11(i.d), q is an free variable in $\tau(M^{(r')})$ and therefore it is enabled by r' in $\tau(M^{(r')})$. Hence since t^ω is hereditarily justified by r_0, it must be hereditarily enabled by r' in $\tau(M^{(r')})$. □

O-view projection. In this paragraph we will spend some time proving the following Proposition:

Proposition 1.2 (O-view projection for traversals)**.** *Let t be a traversal of $\mathcal{T}rav(M)$ such that its last node appears in $t \upharpoonleft\upharpoonright r_0$ for some occurrence r_0 in t of a lambda node r' in N_λ. Then $\llcorner t \lrcorner_M \upharpoonleft\upharpoonright r_0 \sqsubseteq \llcorner t \upharpoonleft\upharpoonright r_0 \lrcorner_{M^{(r')}}$.*

One may recognize that this result bears resemblance with another non trivial result of game semantics from the seminal paper by Hyland and Ong on full abstraction of PCF [6]:

Proposition 1.3 (P-view projection in game semantics)**.** *[6, Prop.4.3] Let s be a legal position of a game $A \rightarrow B$. If s^ω is in B then $\ulcorner s \urcorner^{A \rightarrow B} \upharpoonright B \sqsubseteq \ulcorner s \upharpoonright B \urcorner^B$.*

Since such result is relatively hard to prove, it would be nice if we could just reuse the above proposition to show our result. Unfortunately, the two settings are not exactly analogues of each other so we cannot immediately deduce one proposition from the other. Indeed, the proof of the previous proposition relies on several properties of a legal position s [6]:

- (w1) Initial question to start: The first move played in s is an initial move and there is no other occurrence of initial moves in the rest of s;
- (w2) Alternation: P-moves and O-moves alternate in s;
- (w3) Explicit justification: *every* move, except the first one, has a pointer to a preceding move,
- (w4) Well-bracketing: The pending question is answered first;
- (w5) Visibility: s satisfies P-visibility and O-visibility.

Also, further assumptions are made on the legal positions of the game $A \rightarrow B$:

- (w6) For every occurrence n in the position, $n \in A \iff n \notin B$;
- (w7) Switching condition: The Proponent is the only player who can switch from game A to B or from B to A.
- (w8) Justification in $A \rightarrow B$: Suppose m justifies n in s. Then
  - $n \in B$ implies $m \in B$;
  - if n is a non-initial move in A the $n \in A$;
  - if n is an initial move in A the $n \in B$.

Most of these requirements coincide with properties that we have already shown for traversals. However traversals do not strictly satisfy explicit justification since there are some nodes—the @-nodes and Σ-nodes—that do not have justification pointers. The solution to this problem is simple: we just add justification pointers to @-nodes and Σ-nodes!

Take a justified sequence of nodes t. We define $\mathsf{ext}(t)$, the ***extension of*** t, to be the sequence of nodes-with-pointers obtained from $\diamond \cdot t$ (where $\diamond$ is a dummy node) by adding justification pointers going from occurrences of the root $\circledast$, @-nodes and Σ-nodes to their immediate predecessor in t.

Example 1.10. Let $f \in \Sigma$. We have

$$\begin{tikzcd}[column sep=tiny] \mathsf{ext}(\lambda\overline{\xi} \cdot & @ \cdot & \lambda x \cdot \arrow[l, bend right] & f \cdot & \lambda \cdot \arrow[l, bend right] & x) = \arrow[lll, bend right] & \diamond \cdot & \lambda\overline{\xi} \cdot \arrow[l, bend right] & @ \cdot \arrow[l, bend right] & \lambda x \cdot \arrow[l, bend right] & f \cdot \arrow[l, bend right] & \lambda \cdot \arrow[l, bend right] & x. \arrow[lll, bend right] \end{tikzcd}$$

It is an immediate fact that for every two justified sequences t_1 and t_2 we have:

$$\mathsf{ext}(t_1) \sqsubseteq \mathsf{ext}(t_2) \qquad \Longleftrightarrow \qquad t_1 \sqsubseteq t_2 \tag{2}$$

and for every justified sequence t:

$$\mathsf{ext}(t) \upharpoonright\!\upharpoonright r_0 = \mathsf{ext}(t \upharpoonright\!\upharpoonright r_0) \ . \tag{3}$$

Since a traversal extension $\mathsf{ext}(t)$ may contain @/Σ-nodes with pointers, it is not a proper justified sequence of nodes as defined in Def. 1.6. Nevertheless, the basic transformations that we have defined for justified sequences—such as hereditary projection, P-view and O-view—apply naturally to traversal extensions (without any modification in their definition). The views of a traversal extension can be expressed in term of the traversal's views as follows:

$$\llcorner \mathsf{ext}(t) \lrcorner = \llcorner t \lrcorner \tag{4}$$

$$\ulcorner \mathsf{ext}(t) \urcorner = \begin{cases} \epsilon, & \text{if } t = \epsilon \ ; \\ \diamond \cdot \mathsf{ext}(\ulcorner t \urcorner), & \text{otherwise.} \end{cases} \tag{5}$$

The transformations $\ulcorner\_\urcorner$ and $\llcorner\_\lrcorner$, however, do not convey the appropriate notion of view for extended traversals. We define an alternative notion of view more appropriate to traversal extensions, called O-e-view and P-e-view, as follows:

Definition 1.19. The O-e-view of a traversal extension $\mathsf{ext}(t)$, written, $\llcorner \mathsf{ext}(t) \lrcorner_{\mathsf{e}}$ is defined as

$$\llcorner \mathsf{ext}(t) \lrcorner_{\mathsf{e}} \stackrel{\mathsf{def}}{=} \ulcorner \mathsf{ext}(t) \urcorner \ .$$

The P-e-view of $\mathsf{ext}(t)$, written, $\llcorner \mathsf{ext}(t) \lrcorner_{\mathsf{e}}$ is defined by induction:

$$\begin{aligned} \ulcorner \epsilon \urcorner^{\mathsf{e}} &= \epsilon \\ \ulcorner u \cdot n \urcorner^{\mathsf{e}} &= \ulcorner u \urcorner^{\mathsf{e}} \cdot n \qquad \text{for } n \in L_{\mathsf{var}} \cup L_{\Sigma} \cup L_{@} \cup N_{\lambda} \ ; \end{aligned}$$

$$\begin{tikzcd}[column sep=tiny] \ulcorner u \cdot & m \cdot \ldots \cdot & n \urcorner^{\mathsf{e}} \arrow[l, bend right] & = & \ulcorner u \urcorner^{\mathsf{e}} \cdot & m \cdot & n \arrow[l, bend right] & \text{for } n \in N_{\mathsf{var}} \cup L_{\lambda} \cup N_{@} \cup N_{\Sigma} \ . \end{tikzcd}$$

Inserting a dummy node $\diamond$ at the beginning of the traversal changes the parity of the alternation between nodes in $N_{\mathsf{var}} \cup L_\lambda \cup N_@ \cup N_\Sigma$ and $N_\lambda \cup L_{\mathsf{var}} \cup L_\Sigma \cup L_@$. Thus the role of O and P is interchanged for traversal extensions. This explains why the O-e-view is calculated from the P-view.

For the P-e-view, the definition is almost the same as the traversal O-view $\llcorner\_\lrcorner$ except that the computation does not stop when reaching a node in $N_@ \cup N_\Sigma$—this is sometimes referred as the *long O-view* [7]. (The O-view contains only one thread whereas the long-O-view may contain several; the O-view is a suffix of the long O-view.) This is possible because occurrences of nodes from $N_@ \cup N_\Sigma$ in a traversal extension all have a justification pointer. The O-view of t is a suffix of its P-e-view:

$$\ulcorner t \urcorner^{\mathsf{e}} = w \cdot \llcorner t \lrcorner \quad \text{for some sequence } w. \tag{6}$$

We are now fully equipped to establish an analogy between the traversal extension setting and the game-semantic setting. The reason why we make this analogy is purely to reuse the proof of Proposition 1.3 [6, Prop. 4.3]. The reader must not confuse it with another correspondence that we will establish in a forthcoming section, between plays of game semantics and traversals of the computation tree. (In particular the colouring of nodes used here in term of P-move/O-move is the opposite of the one used in the Correspondence Theorem.) The following analogy is made:

Traversal setting	**Game-semantic setting**
Extended traversal $\mathsf{ext}(t)$	Play s
Nodes in $n \in N_{\mathsf{var}} \cup L_\lambda \cup N_@ \cup N_\Sigma \cup \{\diamond\}$	O-moves $\bullet$
Nodes in $n \in N_\lambda \cup L_{\mathsf{var}} \cup L_\Sigma \cup L_@$	P-moves $\circ$
P-view $\ulcorner \mathsf{ext}(t) \urcorner^{\mathsf{e}}$	P-view $\ulcorner s \urcorner$
O-view $\llcorner \mathsf{ext}(t) \lrcorner_{\mathsf{e}}$	O-view $\llcorner s \lrcorner$
Occurrence n appearing in $t \upharpoonleft\!\upharpoonright r_0$	Occurrence $n \in B$
Occurrence n not appearing in $t \upharpoonleft\!\upharpoonright r_0$	Occurrence $n \in A$
No notion of initiality (All nodes are considered to be non-initial).	Distinction between initial and non-initial move.

Clearly sequences of the form $\mathsf{ext}(t)$ satisfy the requirements (w1) to (w5): For (w1), the initial node becomes $\diamond$. Explicit justification (w4) holds since we have added pointers to $@/\Sigma$-nodes. Finally, alternation (w3), well-bracketing (w4) and visibility (w5) of the traversal t (Prop. 1.1) are preserved by the extension operation (where visibility is defined with respect to the appropriate notion of P-view and O-view).

The property (w6) trivially holds: $n \in t \upharpoonleft\!\upharpoonright r_0$ iff $\neg(n \not\in t \upharpoonleft\!\upharpoonright r_0)$. So does the switching condition (w7): if $t = \ldots \cdot m \cdot n$ where $n \in N_{\mathsf{var}} \cup L_\lambda \cup N_@ \cup N_\Sigma$ and $m \in N_\lambda \cup L_{\mathsf{var}} \cup L_\Sigma \cup L_@$ then, by definition of $t \upharpoonleft\!\upharpoonright r_0$, m appears in $t \upharpoonleft\!\upharpoonright r_0$ if and only if n does. For (w8): Using the analogy of the preceding table and since all nodes are considered "non-initial" in $\mathsf{ext}(t)$, this condition can be stated as:

(w8) Suppose m justifies n in $\mathsf{ext}(t)$. Then $n \in t \upharpoonleft\!\upharpoonright r_0$ if and only if $m \in t \upharpoonleft\!\upharpoonright r_0$.

Unfortunately, as we have seen previously, the direct implication does not hold in general! (Indeed, a variable node can very well appear in $t \upharpoonleft\!\upharpoonright r_0$ even though its justifier does not.) Consequently, the proof of Proposition 1.3 cannot be directly reused in our setting. A weaker version of condition (w8) holds however: if r_0 occurs before n's justifier then, by Lemma 1.11(i), n appears in $t \upharpoonleft\!\upharpoonright r_0$ if and only if its justifier does; this condition turns out to be sufficient to reuse most of the proof of Proposition 1.3 [6].

We reproduce here some definition used in this proof. Let s be a position of the game $A \to B$. A bounded segment is a segment θ of s of the form $\overset{\curvearrowleft}{\overset{x}{\circ} \ldots \overset{y}{\bullet}}$. If x is in A, and hence so does y, then θ is an A-bounded segment. Respectively if x and y are in B then it is a B-bounded segment. By an abuse of notation we define $\ulcorner \theta \upharpoonright B \urcorner$ to be the subsequence of $\ulcorner s_{\leqslant y} \upharpoonright B \urcorner$ consisting only of moves in θ appearing after (and not including) x.

We then have:

Lemma 1.15. *[6, Lemma A.3] Let θ be an A-bounded segment in s with end-moves x and y.*

(i) $\ulcorner \theta \upharpoonright B \urcorner = \overset{\curvearrowleft}{\overset{p_r}{\circ} \cdot \overset{q_r}{\bullet}} \ldots \overset{\curvearrowleft}{\overset{p_1}{\circ} \cdot \overset{q_1}{\bullet}}$ for some $r \geq 0$. Note that each segment $p_i \ldots q_i$ is B-bounded in s, for $1 \leq i \leq r$.

(ii) For every P-move m in θ which appears in $\llcorner s_{<y} \lrcorner$, m does not belong to any of the B-bounded segments $p_i \ldots q_i$ for $1 \leq i \leq r$.

This lemma assumes that the segment θ satisfies the assumptions (w1) to (w8). As we have seen, (w8) does not always hold for extended traversals. But using our analogy with extended traversals, a segment θ is "A-bounded" if θ is bounded by two nodes appearing in $t \upharpoonleft\!\upharpoonright r_0$. This can only happen if r_0 occurs before θ in t or if θ's left bound is r_0. Thus the condition (w8) holds at least for the nodes of the segment θ. The previous lemma thus translates into:

Lemma 1.16. *Let t be a traversal and θ be a segment of $\mathsf{ext}(t)$ bounded by nodes x and y appearing in $t \upharpoonleft\!\upharpoonright r_0$.*

(i) $\ulcorner \theta \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} = \overset{\curvearrowleft}{p_r \cdot q_r} \ldots \overset{\curvearrowleft}{p_1 \cdot q_1}$ for some $r \geq 0$ where $p_i \in N_\lambda \cup L_{\mathsf{var}} \cup L_\Sigma \cup L_@$ and $q_i \in N_{\mathsf{var}} \cup L_\lambda \cup N_@ \cup N_\Sigma$, for $1 \leq i \leq r$.

(ii) For every node m in $N_\lambda \cup L_{\mathsf{var}} \cup L_\Sigma \cup L_@$ occurring in θ and appearing in $\llcorner \mathsf{ext}(t)_{<y} \lrcorner_{\mathsf{e}}$, m does not belong to any of the segments $p_i \ldots q_i$ for $1 \leq i \leq r$.

We now show the analogue of Proposition 1.3 in the context of extended traversals:

Proposition 1.4. *Let t be a traversal and r_0 be an occurrence of some lambda node r'. If $\mathsf{ext}(t)$'s last node appears in $t \upharpoonleft\!\upharpoonright r_0$ then $\ulcorner \mathsf{ext}(t) \urcorner^{\mathsf{e}} \upharpoonleft\!\upharpoonright r_0 \sqsubseteq \ulcorner \mathsf{ext}(t \upharpoonleft\!\upharpoonright r_0) \urcorner^{\mathsf{e}}$.*

Proof. By (3) we can equivalently show that: $\ulcorner \mathsf{ext}(t) \urcorner^{\mathsf{e}} \upharpoonleft\!\upharpoonright r_0 \sqsubseteq \ulcorner \mathsf{ext}(t) \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}}$. By induction on the length of t. The base case is immediate. For the inductive case, we do a case analysis:

- $t = t' \cdot r_0$. We have $\mathsf{ext}(t) \upharpoonleft\!\upharpoonright r_0 = r_0$ and $\ulcorner \mathsf{ext}(t) \urcorner^{\mathsf{e}} \upharpoonleft\!\upharpoonright r_0 = r_0 = \ulcorner \mathsf{ext}(t) \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}}$.
- $t = t' \cdot n$ with $n \in N_\lambda \cup L_{\mathsf{var}} \cup L_\Sigma \cup L_@$ where n is not the occurrence r_0.

  There are two cases.

  - Suppose that the last node in t' appears in $t \upharpoonleft\!\upharpoonright r_0$. Then by the I.H. we have $\ulcorner \mathsf{ext}(t') \urcorner^{\mathsf{e}} \upharpoonleft\!\upharpoonright r_0 \sqsubseteq \ulcorner \mathsf{ext}(t') \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}}$ thus

$$\begin{aligned}
\ulcorner \mathsf{ext}(t) \urcorner^{\mathsf{e}} \upharpoonleft\!\upharpoonright r_0 &= \ulcorner \mathsf{ext}(t') \urcorner^{\mathsf{e}} \upharpoonleft\!\upharpoonright r_0 \cdot n && \text{(P-view for extended justified sequences of nodes of } M\text{)} \\
&\sqsubseteq \ulcorner \mathsf{ext}(t') \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} \cdot n && \text{(induction hypothesis)} \\
&= \ulcorner \mathsf{ext}(t') \upharpoonleft\!\upharpoonright r_0 \cdot n \urcorner^{\mathsf{e}} && \text{(P-view for extended justified sequences of nodes of } M^{(r')}\text{, } n \text{ belongs to } V^{(r')} \text{ by Lemma 1.12)} \\
&= \ulcorner \mathsf{ext}(t' \cdot n) \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} && (n \text{ occurs in } t \upharpoonleft\!\upharpoonright r_0) \\
&= \ulcorner \mathsf{ext}(t) \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} && \text{(definition of } t\text{).}
\end{aligned}$$

  - Suppose that the last node y_1 in t' does not appear in $t \upharpoonleft\!\upharpoonright r_0$. Let $\underline{m}$ be the last node preceding m in $\ulcorner \mathsf{ext}(t) \urcorner^{\mathsf{e}}$ that appears in $t \upharpoonleft\!\upharpoonright r_0$. Then for some $q \geq 0$ we have

$$\ulcorner \mathsf{ext}(t) \urcorner^{\mathsf{e}} = \ulcorner \mathsf{ext}(t)_{\leqslant \underline{m}} \urcorner^{\mathsf{e}} \cdot \underbrace{\overset{\curvearrowleft}{x_q \cdot y_q} \ \ldots\ \overset{\curvearrowleft}{x_1 \cdot y_1}}_{\text{all appear in } t \upharpoonleft\!\upharpoonright r_0 \cdot m}$$

  where the x_is are in $N_\lambda \cup L_{\mathsf{var}} \cup L_\Sigma \cup L_@$ and the y_is are in $N_{\mathsf{var}} \cup N_\Sigma \cup N_@ \cup L_\lambda$. Therefore the sequence $\mathsf{ext}(t)$ must be of the following form:

$$\mathsf{ext}(t)_{\leqslant \underline{m}} \cdot \underbrace{x_q \ldots y_q}_{\theta_q} \ \cdots\ \underbrace{x_1 \cdots y_1}_{\theta_1} \cdot m$$

  where each segment θ_i is bounded by nodes appearing in $t \upharpoonleft\!\upharpoonright r_0$. By Lemma 1.16, when computing the P-view of $\mathsf{ext}(t)$, pointers going from a segment θ to a node outside the segment are never followed! In other words:

$$\ulcorner \mathsf{ext}(t) \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} = \ulcorner \mathsf{ext}(t)_{\leqslant \underline{m}} \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} \cdot \ulcorner \theta_q \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} \cdot \ \cdots\ \cdot \ulcorner \theta_1 \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} \cdot m \ .$$

  Hence:

$$\begin{aligned}
\ulcorner \mathsf{ext}(t) \urcorner^{\mathsf{e}} \upharpoonleft\!\upharpoonright r_0 &= \ulcorner \mathsf{ext}(t)_{\leqslant \underline{m}} \urcorner^{\mathsf{e}} \upharpoonleft\!\upharpoonright r_0 \cdot n \\
&\sqsubseteq \ulcorner \mathsf{ext}(t)_{\leqslant \underline{m}} \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} \cdot n && \text{(I.H.)} \\
&\sqsubseteq \ulcorner \mathsf{ext}(t)_{\leqslant \underline{m}} \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} \cdot \ulcorner \theta_q \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} \cdot \ldots \cdot \ulcorner \theta_1 \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} \cdot n \\
&= \ulcorner \mathsf{ext}(t) \upharpoonleft\!\upharpoonright r_0 \urcorner^{\mathsf{e}} && \text{(by the previous equation).}
\end{aligned}$$

- $t = t' \cdot \overparen{m \cdot u \cdot n}$ where $n \in N_{\mathsf{var}} \cup N_{\Sigma} \cup N_{@} \cup L_{\lambda}$. We have $m \in N_{\lambda} \cup L_{\mathsf{var}} \cup L_{\Sigma} \cup L_{@}$. Suppose that r_0 appears in $t' \cdot m$, then since n appears in $t \upharpoonright\!\upharpoonright r_0$, by Lemma 1.11(i) so does m. Thus we can apply the I.H. on $t' \cdot m$:

$$\begin{aligned}
\ulcorner \mathsf{ext}(t) \urcorner^{\mathsf{e}} \upharpoonright\!\upharpoonright r_0 &= \ulcorner \mathsf{ext}(t') \cdot \overparen{m \cdot \overline{u} \cdot n} \urcorner^{\mathsf{e}}_{M} \upharpoonright\!\upharpoonright r_0 && \text{(definition of } t\text{)}\\
&= (\ulcorner \mathsf{ext}(t') \cdot \overparen{m \urcorner^{\mathsf{e}} \cdot n}) \upharpoonright\!\upharpoonright r_0 && \text{(P-eview computation in } M \text{)}\\
&= \ulcorner \mathsf{ext}(t' \cdot \overparen{m) \urcorner^{\mathsf{e}} \upharpoonright\!\upharpoonright r_0 \cdot n} && (n \text{ appears in } t \upharpoonright\!\upharpoonright r_0)\\
&\sqsubseteq \ulcorner (\mathsf{ext}(t' \cdot \overparen{m)) \upharpoonright\!\upharpoonright r_0 \urcorner^{\mathsf{e}} \cdot n} && \text{(induction hypothesis on } t' \cdot m\text{)}\\
&= \ulcorner \mathsf{ext}(t') \upharpoonright\!\upharpoonright r_0 \cdot \overparen{m \urcorner^{\mathsf{e}} \cdot n} && (m \text{ appears in } t \upharpoonright\!\upharpoonright r_0)\\
&= \ulcorner \mathsf{ext}(t') \upharpoonright\!\upharpoonright r_0 \cdot \overparen{m \cdot (\mathsf{ext}(u) \upharpoonright\!\upharpoonright r_0) \cdot n} \urcorner^{\mathsf{e}} && \text{(P-eview in } M^{(r')}\text{, nodes in } m \cdot (\mathsf{ext}(u) \upharpoonright\!\upharpoonright r_0) \cdot n \text{ are all in } V^{(r')}\text{)}\\
&= \ulcorner (\mathsf{ext}(t') \cdot \overparen{m \cdot \mathsf{ext}(u) \cdot n}) \upharpoonright\!\upharpoonright r_0 \urcorner^{\mathsf{e}} && (m \text{ and } n \text{ both appear in } t \upharpoonright\!\upharpoonright r_0)\\
&= \ulcorner \mathsf{ext}(t) \upharpoonright\!\upharpoonright r_0 \urcorner^{\mathsf{e}} && \text{(definition of } t\text{).}
\end{aligned}$$

Suppose that r_0 appears in u then:

$$\begin{aligned}
\ulcorner \mathsf{ext}(t) \urcorner^{\mathsf{e}} \upharpoonright\!\upharpoonright r_0 &= \ulcorner \mathsf{ext}(t' \cdot \overparen{m) \urcorner^{\mathsf{e}} \upharpoonright\!\upharpoonright r_0 \cdot n} \\
&= n && (r_0 \text{ occurs after } m)\\
&\sqsubseteq \ulcorner (\mathsf{ext}(t' \cdot \overparen{m)) \upharpoonright\!\upharpoonright r_0 \urcorner^{\mathsf{e}} \cdot n} \\
&= \ulcorner \mathsf{ext}(t) \upharpoonright\!\upharpoonright r_0 \urcorner^{\mathsf{e}} \ . && \square
\end{aligned}$$

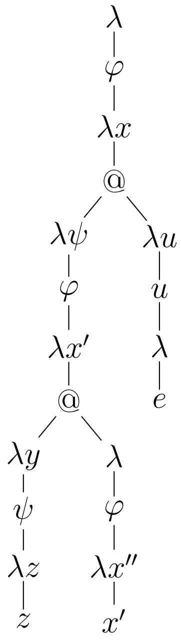
We can now prove Proposition 1.2:

Proof of Proposition 1.2. We have:

$$\begin{aligned}
\llcorner t \lrcorner \upharpoonright\!\upharpoonright r_0 &= \llcorner \mathsf{ext}(t) \lrcorner \upharpoonright\!\upharpoonright r_0 && \text{by (4)}\\
&\sqsubseteq \ulcorner \mathsf{ext}(t) \urcorner^{\mathsf{e}} \upharpoonright\!\upharpoonright r_0 && \text{by (6)}\\
&\sqsubseteq \ulcorner \mathsf{ext}(t \upharpoonright\!\upharpoonright r_0) \urcorner^{\mathsf{e}} && \text{by Proposition 1.4}\\
&= w \cdot \llcorner \mathsf{ext}(t \upharpoonright\!\upharpoonright r_0) \lrcorner && \text{for some } w\text{, by (6)}\\
&= w \cdot \llcorner t \upharpoonright\!\upharpoonright r_0 \lrcorner && \text{by (4).}
\end{aligned}$$

Thus $\llcorner t \lrcorner \upharpoonright\!\upharpoonright r_0 \sqsubseteq w \cdot \llcorner t \upharpoonright\!\upharpoonright r_0 \lrcorner$. But by definition of the operator $\_ \upharpoonright\!\upharpoonright$, both $\llcorner t \lrcorner \upharpoonright\!\upharpoonright r_0$ and $\llcorner t \upharpoonright\!\upharpoonright r_0 \lrcorner$ start with the occurrence r_0, we thus have $\llcorner t \lrcorner \upharpoonright\!\upharpoonright r_0 \sqsubseteq \llcorner t \upharpoonright\!\upharpoonright r_0 \lrcorner$. □

Example 1.11. Take $\varphi : 2, e : o \vdash \varphi(\lambda x.(\lambda \psi.\varphi(\lambda x'.(\lambda y.\psi(\lambda z.z))(\varphi(\lambda x''.x'))))(\lambda u.ue))$. The computation tree is represented below together with an example of traversal t:



$$t = \lambda\ \varphi\ \lambda x\ @\ \lambda\psi\ \varphi\ \lambda x'\ @\ \lambda y\ \psi\ \lambda u\ u\ \lambda z\ z\ \lambda$$

$$\llcorner t \lrcorner = @\ \lambda\psi\ \psi\ \lambda u\ u\ \lambda z\ z\ \lambda$$

$$\llcorner t \lrcorner \upharpoonright\!\upharpoonright r_0 = \lambda\psi\ \psi\ \lambda z\ z$$

$$t \upharpoonright\!\upharpoonright r_0 = \lambda\psi\ \varphi\ \lambda x'\ @\ \lambda y\ \psi\ \lambda z\ z$$

$$\llcorner t \upharpoonright\!\upharpoonright r_0 \lrcorner = \lambda\psi\ \psi\ \lambda z\ z\ .$$

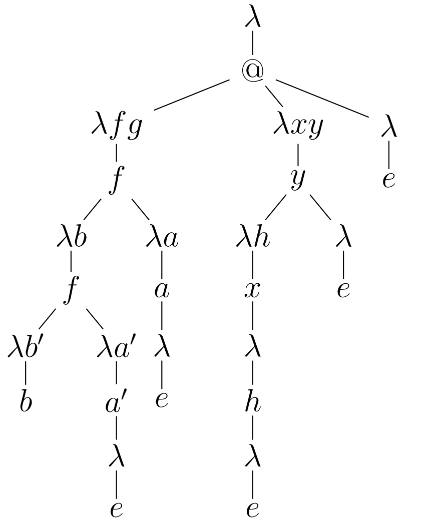
Example 1.12. Take the term-in-context:

$$e : o \vdash (\lambda f g . f(\lambda b . f(\lambda b' . b)(\lambda a' . a' e))(\lambda a . a e))(\lambda x y . y(\lambda h . x(h e))e)e\ .$$

Take the traversal:

$$t = \lambda\ @\ \lambda f g\ f\ \lambda x y\ y\ \lambda a\ a\ \lambda h\ x\ \lambda b\ f\ \lambda x y\ y\ \lambda a'\ a'\ \lambda h\ x\ \lambda b'\ b\ \lambda\ h$$

then we have the following relations:



$$\llcorner t \lrcorner = @\ \lambda f g\ f\ \lambda x y\ y\ \lambda a\ a\ \lambda h\ h$$

$$\llcorner t \lrcorner \upharpoonright\!\upharpoonright r_0 = \lambda f g\ f\ \lambda a\ a$$

$$t \upharpoonright\!\upharpoonright r_0 = \lambda f g\ f\ \lambda a\ a\ \lambda b\ f\ \lambda a'\ a'\ \lambda b'\ b$$

$$\llcorner t \upharpoonright\!\upharpoonright r_0 \lrcorner = \lambda f g\ f\ \lambda a\ a\ \lambda b\ b\ .$$

1.3.8. Subterm projections are sub-traversals

We now show an important result that relies on all the lemmas and propositions from the previous two sections:

Proposition 1.5 (Subterm projections are sub-traversals)**.** *Let $t \in \mathcal{T}rav(M)$. For every occurrence r_0 in t of some lambda node $r' \in N_\lambda$ we have $t \upharpoonright\!\upharpoonright r_0 \in \mathcal{T}rav(M^{(r')})$.*

Proof. We proceed by induction on the traversal rules. The base cases (Empty) and (Root) are trivial. *Step case:* Take a traversal $t \in \mathcal{T}rav(M)$ and suppose that the result holds for every traversal shorter than t.

Suppose that t^ω does not appear in $t \upharpoonright\!\upharpoonright r_0$ then the result follows by applying the induction hypothesis on the immediate prefix of t. Suppose that t^ω appears in $t \upharpoonright\!\upharpoonright r_0$ then we do a case analysis on the last traversal rule used to form t:

- (Lam) We have $t = t' \cdot n$ with $t' = \ldots \cdot \lambda\overline{\xi}$. By the induction hypothesis, $t' \upharpoonright\!\upharpoonright r_0 \in \mathcal{T}rav(M^{(r')})$.

Since n is a variable node appearing in $t \upharpoonright\!\upharpoonright r_0$, by definition of $t \upharpoonright\!\upharpoonright r_0$ its immediate predecessor $\lambda\overline{\xi}$ must occur in $t \upharpoonright\!\upharpoonright r_0$ and therefore must be the last occurrence in $t' \upharpoonright\!\upharpoonright r_0$. Thus we can use the rule (Lam) in $\tau(M^{(r')})$ to produce the traversal $u = (t' \upharpoonright\!\upharpoonright r_0) \cdot n$ of $M^{(r')}$.

We have $t \upharpoonright\!\upharpoonright r_0 = (t' \upharpoonright\!\upharpoonright r_0) \cdot n$, but in order to state that $u = t \upharpoonright\!\upharpoonright r_0$ it remains to prove that n has the same link in $t \upharpoonright\!\upharpoonright r_0$ and in u.

Suppose $n \in N_@ \cup N_\Sigma$ then n has no justifier in both u and $t \upharpoonright\!\upharpoonright r_0$. Otherwise $n \in N_{\mathsf{var}}$. Let m_u denote the occurrence in t of n's justifier in u, m_t for the occurrence in t of n's justifier in t, and m for the occurrence in t of n's justifier in $t \upharpoonright\!\upharpoonright r_0$. We want to show that $m_u = m$. By the rule (Var), m_u is defined as the only occurrence of n's enabler in $\ulcorner t' \upharpoonright\!\upharpoonright r_0 \urcorner$ and m_t is the only occurrence of n's enabler in $\ulcorner t' \urcorner$.

If r_0 occurs before m_t then by Lemma 1.11(ii), m_t appears in $t \upharpoonright\!\upharpoonright r_0$ thus by definition of $\_ \upharpoonright\!\upharpoonright$ we have $m = m_t$. Moreover, since m_t appears in $t \upharpoonright\!\upharpoonright r_0$, it must appear after r_0 by Lemma 1.11(i.a), thus since it is in the P-view at t', it must be in $\ulcorner t \urcorner_{\geqslant r_0}$ which is equal to $\ulcorner t' \upharpoonright\!\upharpoonright r_0 \urcorner$ by Lemma 1.11(i.b). Hence we necessarily have $m_u = m_t$ (since r' occurs only once in the P-view $\ulcorner t' \upharpoonright\!\upharpoonright r_0 \urcorner$).

If r_0 occurs after m_t then m_t does not appear in $t \upharpoonright\!\upharpoonright r_0$ thus $m = r_0$ by definition of $\_ \upharpoonright\!\upharpoonright$. Moreover by Lemma 1.11(i), n's binder occurs in the path from r' to the root $\circledast$. Thus n is a free variable in $\tau(M^{(r')})$ and consequently the only enabler of n occurring in $\ulcorner t' \upharpoonright\!\upharpoonright r_0 \urcorner$ is necessarily r_0: $m_u = r_0$.

This proves the equality $t \upharpoonright\!\upharpoonright r_0 = u$ and thus $t \upharpoonright\!\upharpoonright r_0$ is a valid traversal of $M^{(r')}$.

- (App) $t = \ldots \cdot \lambda\overline{\xi} \cdot @ \cdot n$. Since n appears in $t \upharpoonright\!\upharpoonright r_0$, so does $@$ (by definition of $t \upharpoonright\!\upharpoonright r_0$). Hence $@$ is the last occurrence in $t' \upharpoonright\!\upharpoonright r_0$. By the induction hypothesis, $t' \upharpoonright\!\upharpoonright r_0$ is a traversal of $\tau(M^{(r')})$ therefore we can use the rule (App) in $\tau(M^{(r')})$ to produce the traversal $(t' \upharpoonright\!\upharpoonright r_0) \cdot n = t \upharpoonright\!\upharpoonright r_0$ of $M^{(r')}$.

- $(\mathsf{Value}^{@\mapsto\lambda})$ Take $t = t' \cdot \lambda\overline{\xi} \cdot @ \ldots v_@ \cdot v_{\lambda\overline{\xi}}$.

The occurrence $v_{\lambda\overline{\xi}}$ appears $t \upharpoonright\!\upharpoonright r_0$ therefore since r_0 is not a lambda node, its justifier $\lambda\overline{\xi}$ also appears in $t \upharpoonright\!\upharpoonright r_0$. Moreover since $@$ and $v_@$ are hereditarily justified by $\lambda\overline{\xi}$, they must also appear in $t \upharpoonright\!\upharpoonright r_0$.

By the induction hypothesis $t' \upharpoonright\!\upharpoonright r_0$ is a traversal of $\tau(M^{(r')})$ therefore since the occurrence $\lambda\overline{\xi}$, $@$, $v_@$, $v_{\lambda\overline{\xi}}$ all appear in $t \upharpoonright\!\upharpoonright r_0$ we can use the rule $(\mathsf{Value}^{@\mapsto\lambda})$ in $M^{(r')}$ to form the traversal $(t' \upharpoonright\!\upharpoonright r_0) \cdot n = t \upharpoonright\!\upharpoonright r_0$ of $M^{(r')}$.

- ($\mathsf{Value}^{\lambda\mapsto @}$) Take $t = t' \cdot @ \cdot \lambda\overline{z} \ldots v_{\lambda\overline{z}} \cdot v_{@}$. Again, since $v_{@}$ appears in $t \upharpoonright\!\upharpoonright r_0$, necessarily the occurrences $@$, $\lambda\overline{z}$, $v_{\lambda\overline{z}}$ and $v_{@}$ must all appear in $t \upharpoonright\!\upharpoonright r_0$. Hence using the induction hypothesis and the rule ($\mathsf{Value}^{\lambda\mapsto @}$) in $M^{(r')}$ we obtain that $t \upharpoonright\!\upharpoonright r_0$ is a traversal of $M^{(r')}$.
- ($\mathsf{Value}^{\mathsf{var}\mapsto\lambda}$) Take $t = t' \cdot \lambda\overline{\xi} \cdot x \ldots v_x \cdot v_{\lambda\overline{\xi}}$. Since $v_{\lambda\overline{\xi}}$ is in $t \upharpoonright\!\upharpoonright r_0$, so must be x, v_x and $\lambda\overline{\xi}$, by definition of $t \upharpoonright\!\upharpoonright r_0$. Hence we can use the I.H. to form the traversal $t \upharpoonright\!\upharpoonright r_0$ of $M^{(r')}$.
- ($\mathsf{InputValue}$) Take $t = t_1 \cdot x \cdot t_2 \cdot v_x$ for some $v \in \mathcal{D}$ where x is the pending node in $t_1 \cdot x \cdot t_2$ and $x \in N^{\circledast\vdash}_{\mathsf{var}}$. Since v_x appears in $t \upharpoonright\!\upharpoonright r_0$, so does x hence by Lemma 1.10, x is also the pending node in $(t_1 \cdot x \cdot t_2) \upharpoonright\!\upharpoonright r_0$. Furthermore since $M^{(r')}$ is a subterm of M, x is necessarily an input-variable node in $\tau(M^{(r')})$. Hence we can conclude using the I.H. and the rule ($\mathsf{InputValue}$).
- ($\mathsf{InputVar}$) Take $t = t' \cdot n$ where $n \in N_\lambda$ points to an occurrence of its parent node $y \in N^{\circledast\vdash}_{\mathsf{var}}$ in $\llcorner t \lrcorner$. By Lemma 1.9(a), y must also appear in $t \upharpoonright\!\upharpoonright r_0$, therefore y also occurs in $\llcorner t \upharpoonright\!\upharpoonright r_0 \lrcorner \sqsubseteq \llcorner t \lrcorner \upharpoonright\!\upharpoonright r_0$. Hence we can conclude using the rule ($\mathsf{InputVar}$) in $M^{(r')}$.
- (Var) Take $t = t' \cdot p \cdot \lambda\overline{x} \ldots x_i \cdot \lambda\overline{\eta_i}$ for some variable x_i in $N^{@\vdash}_{\mathsf{var}}$. If $\lambda\overline{\eta_i}$ is the occurrence r_0 then the traversal $t \upharpoonright\!\upharpoonright r_0 = r_0$ can be formed using the rule (Root).
Suppose that $\lambda\overline{\eta_i}$ is not the occurrence r_0. Then both $\lambda\overline{\eta_i}$ and its justifier p must appear in $t \upharpoonright\!\upharpoonright r_0$. The nodes $\lambda\overline{x}$ and x_i, however, do not necessarily appear in $t \upharpoonright\!\upharpoonright r_0$.
Consider the node $@$ that initiates the thread of $\lambda\overline{\eta_i}$.
  - Suppose that r_0 precedes $@$ in t then by Lemma 1.14(i), the nodes $\lambda\overline{\eta_i}$, p, $\lambda\overline{x}$ and x_i as well as $@$ all appear in $t \upharpoonright\!\upharpoonright r_0$. Moreover since $@$ appear in $t \upharpoonright\!\upharpoonright r_0$, it must be an occurrence of an application node that appear in the subtree rooted at r' thus $@ \in N^{r'\vdash}_{\mathsf{var}}$. Hence we can use the use the rule (Var) in $M^{(r')}$ to form the traversal $t \upharpoonright\!\upharpoonright r_0$ of $M^{(r')}$.
  - Suppose that $@$ precedes r_0 in t then by Lemma 1.14(ii), p is necessarily an input variable node in $\tau(M^{(r')})$. We have $p \in \llcorner t \lrcorner \upharpoonright\!\upharpoonright r_0 \sqsubseteq \llcorner t \upharpoonright\!\upharpoonright r_0 \lrcorner$ by Proposition 1.2. Furthermore we can easily check (by alternation and using the fact that if an occurrence in $N_\lambda \cup L_{\mathsf{var}} \cup L_{@} \cup L_\Sigma \cup N_{@} \cup N_\Sigma$ appears in $t \upharpoonright\!\upharpoonright r_0$ then so does its immediate successor) that the penultimate node in $t \upharpoonright\!\upharpoonright r_0$ is necessarily in $N_{\mathsf{var}} \cup L_\lambda$. Hence we can make use of the rule ($\mathsf{InputVar}$) in $M^{(r')}$ (in its alternative form) to produce the traversal $t \upharpoonright\!\upharpoonright r_0$ of $M^{(r')}$.
- ($\mathsf{Value}^{\lambda\mapsto\mathsf{var}}$) Take $t = t' \cdot y \cdot \lambda\overline{\xi} \ldots v_{\lambda\overline{\xi}} \cdot v_y$ for some variable y in $N^{@\vdash}_{\mathsf{var}}$. The proof is similar to the previous case using the rule ($\mathsf{InputValue}$) instead of ($\mathsf{InputVar}$) in the second subcase.
- (Σ)/(Σ-var) The proof is similar to the case (App) and (Var).
- (Σ-Value) The proof is similar to the case ($\mathsf{Value}^{\lambda\mapsto\mathsf{var}}$). □

The following Lemma will be useful to prove the Correspondence Theorem:

Lemma 1.17. *Let t be a traversal and r_0 be an occurrence of a lambda node r'. We have*

$$(t \upharpoonright\!\upharpoonright r_0)^\star = t^\star \upharpoonright V^{(r')} \upharpoonright r_0 \ .$$

Proof. By the previous Lemma, $t \upharpoonright\!\upharpoonright r_0$ is indeed a traversal (of $\tau(M^{(r')})$) thus the expression "$(t \upharpoonright\!\upharpoonright r_0)^\star$" is well-defined. We show the result by induction on t: It is true for the empty traversal. Take $t = t' \cdot n$.

If n belongs to $V_@ \cup V_\Sigma$ then

$$((t' \cdot n) \upharpoonright\!\upharpoonright n_0)^\star = (t' \upharpoonright\!\upharpoonright n_0)^\star \cdot \begin{cases} n, & \text{if } n \text{ appears in } t \upharpoonright\!\upharpoonright n_0;\\ \epsilon, & \text{otherwise.} \end{cases}$$

$$\text{and } ((t' \cdot n)^\star \upharpoonright V^{(r')}) \upharpoonright n_0 = (t'^\star \upharpoonright V^{(r')}) \upharpoonright n_0 \cdot \begin{cases} n, & \text{if } n \text{ is her. just. by } n_0 \text{ in } t^\star \upharpoonright V^{(r')};\\ \epsilon, & \text{otherwise.} \end{cases}$$

Since $t^\omega \not\in V_@ \cup V_\Sigma$, by Lemma 1.13 we have that n is hereditarily justified by n_0 in $t^\star \upharpoonright V^{(r')}$ if and only if n appears in $t \upharpoonright\!\upharpoonright n_0$. Hence we can conclude using the I.H. on t'.

If n does not belong to $V_@ \cup V_\Sigma$ then

$$\begin{aligned} ((t' \cdot n) \upharpoonright\!\upharpoonright n_0)^\star &= (t' \upharpoonright\!\upharpoonright n_0)^\star \\ &= (t'^\star \upharpoonright V^{(r')}) \upharpoonright n_0 && \text{by the I.H. on } t' \\ &= ((t' \cdot n)^\star \upharpoonright V^{(r')}) \upharpoonright n_0 && \square \end{aligned}$$

Consequently, by Lemma 1.7, if $t^\omega \not\in V_@ \cup V_\Sigma$ then $t \upharpoonright\!\upharpoonright r_0 = (t^\star \upharpoonright r_0) + \Sigma + @$.

1.3.9. O-view and P-view projection with respect to root

Lemma 1.18 (O-view projection with respect to the root)**.** *Let t be a non-empty traversal of M and r denote the only occurrence of $\tau(M)$'s root in t. If t^ω appears in $t \upharpoonright r$ then:*

$$\llcorner t \upharpoonright r \lrcorner = \llcorner t \lrcorner \upharpoonright r = \llcorner t \lrcorner \ .$$

Proof. It follows immediately from the fact that, by Lemma 1.6, all the occurrences in $\llcorner t \lrcorner$ belong to the same thread and therefore are all hereditarily justified by r. □

Lemma 1.19 (P-view projection with respect to the root)**.** *Let t be a non-empty traversal of M and r denote the only occurrence of $\tau(M)$'s root in t. If t^ω appears in $t \upharpoonright r$ then:*

$$\ulcorner t \urcorner \upharpoonright r \sqsubseteq \ulcorner t \upharpoonright r \urcorner \ .$$

Proof. We just sketch the proof. We proceed exactly in the same way as for the proof of Proposition 1.2. Again we establish an analogy between traversals and plays of game semantics:

Traversal setting	**Game-semantic setting**
Traversal t	Play s
Nodes in $n \in N_\lambda \cup L_{\mathsf{var}} \cup L_\Sigma \cup L_@$	O-moves $\bullet$
Nodes in $n \in N_{\mathsf{var}} \cup L_\lambda \cup N_@ \cup N_\Sigma \cup \{\diamond\}$	P-moves $\circ$
P-view $\ulcorner t \urcorner$	P-view $\ulcorner s \urcorner$
O-view $\llcorner t \lrcorner$	O-view $\llcorner s \lrcorner$
Occurrence n her. just. by r in t	Occurrence $n \in B$
Occurrence n not her. just. by r in t	Occurrence $n \in A$
No notion of initiality (all nodes are considered to be non-initial).	Distinction between initial and non-initial move.

Clearly the conditions (w1) to (w8) hold. Hence we can reuse Proposition 4.3 form [6] which gives the desired result. □

The previous result gives us only an inequality. In the particular case where interpreted constants are well-behaved, however, and if we consider the subsequence of a traversal consisting of unanswered nodes only, then we obtain an equality:

Lemma 1.20. *Suppose that M is in β-normal form and all the Σ-constants are well-behaved. Let t be a non-empty traversal of M and r denote the only occurrence in t of $\tau(M)$'s root.*

(a) If t's last occurrence is not a leaf then $\ulcorner t \urcorner \upharpoonright r = \ulcorner ?(t) \upharpoonright r \urcorner = \ulcorner ?(t \upharpoonright r) \urcorner = ?(\ulcorner t \upharpoonright r \urcorner)$;

(b) If t's last occurrence is not a leaf and is hereditarily justified by r then $\ulcorner t \urcorner \upharpoonright r = \ulcorner t \upharpoonright r \urcorner$.

Proof. (a) It is easy to show that $?(t) \upharpoonright r = ?(t \upharpoonright r)$. This implies the second equality. The third equality can be shown by an easy induction and by observing that in a traversal core, variable occurrences are always immediately preceded by a lambda node (and not by a leaf). We show the first equality by induction. The base case $t = \epsilon$ is trivial. Consider a traversal t and suppose that the property is satisfied for all traversals shorter than t. Observe that since t contains at most a single occurrence r of the root $\circledast$, an occurrence n in t is hereditarily justified by r if and only if the corresponding node in $\tau(M)$ is hereditarily enabled by $\circledast$. Thus $t \upharpoonright r = t \upharpoonright N^{\circledast\vdash}$. We do a case analysis on t's last node:

- $t^\omega \in N_@$. This case does not happen since M is β-normal.
- $t = t' \cdot n$ with $n \in N_{\mathsf{var}} \cup N_\Sigma$ then t'^ω is not a leaf (otherwise n would also be a leaf by rule (Value)) thus we can use the I.H. on t' which, by an easy calculation, gives the desired equality.

Suppose that t^ω is a lambda node. There are three subcases:

- $t^\omega \in N_\lambda^{@\vdash}$. Since the term is in β-normal form, there is no @-node in $\tau(M)$ so the rules (App) and (Var) are unused, hence this case does not happen.
- $t^\omega \in N_\lambda^{N_\Sigma\vdash}$. We have $t = t' \cdot \overset{\curvearrowleft}{m \cdot u \cdot n}$ with $n \in N_\lambda^{N_\Sigma\vdash}$ and $m \in N_{\mathsf{var}} \cup N_\Sigma$. The occurrence n is necessarily visited with a (Σ)-rule. Since, by assumption, these rules are well-behaved we have $?(u) = \epsilon$. Hence:

$$\begin{aligned}
\ulcorner t \urcorner \upharpoonright r &= \ulcorner t' \cdot \overset{\curvearrowleft}{m \cdot u \cdot n} \urcorner \upharpoonright r && \text{(def. of } t\text{)}\\
&= (\ulcorner t' \urcorner \cdot \overset{\curvearrowleft}{m \cdot n}) \upharpoonright r && \text{(P-view computation)}\\
&= \ulcorner t' \urcorner \upharpoonright r && (m, n \not\in N^{\circledast\vdash})\\
&= \ulcorner ?(t') \upharpoonright r \urcorner && \text{(induction hypothesis)}\\
&= \ulcorner ?(t' \cdot \overset{\curvearrowleft}{m \cdot n}) \upharpoonright r \urcorner && (m, n \not\in N^{\circledast\vdash})\\
&= \ulcorner ?(t' \cdot \overset{\curvearrowleft}{m \cdot u \cdot n}) \upharpoonright r \urcorner && (?(u) = \epsilon)\\
&= \ulcorner ?(t) \upharpoonright r \urcorner && \text{(since } u = \epsilon\text{).}
\end{aligned}$$

- $t^\omega \in N_\lambda^{\circledast\vdash}$. If $t = r$ then the result holds trivially. Otherwise $t = t' \cdot m \cdot u \cdot n$ for some $n \in N_\lambda^{\circledast\vdash}$. An easy calculation using the induction hypothesis on $t' \cdot m$ shows the desired equality.

(b) If t's last occurrence is hereditarily justified by r then the last occurrence of $t \upharpoonright r$ is precisely the last occurrence of t and is therefore not a leaf. In a traversal core, variable nodes are immediately preceded by lambda nodes thus since the last node in $t \upharpoonright r$ is not a leaf, an easy induction shows that all the nodes in $\ulcorner t \upharpoonright r \urcorner$ are not leaves. Consequently $?(\ulcorner t \upharpoonright r \urcorner) = \ulcorner t \upharpoonright r \urcorner$. □

The hypothesis that the term is beta-normal is crucial in this Lemma. Take for instance the term $\lambda x^o\, f^{(o,o)}.(\lambda y^o.f\ y)x$. A possible traversal is

$$t = \lambda x f \cdot @ \cdot \lambda y \cdot f \cdot \lambda \cdot y \cdot \lambda \cdot x\ .$$

But $\ulcorner t \urcorner \upharpoonright r = \lambda x f \cdot x$ is only a strict subsequence of $\ulcorner t \upharpoonright r \urcorner = \lambda x f \cdot f \cdot \lambda \cdot x$.

2. Game semantics correspondence

We work in the general setting of an applied simply-typed lambda calculus with a given set of higher-order constants Σ. The operational semantics of these constants is given by certain reduction rules. We assume that a fully abstract model of the calculus is provided by means of a category of well-bracketed games. For instance, if Σ consists of the PCF constants then we work in the category of games and innocent well-bracketed strategies [6, 8]. A strategy is commonly defined in the literature as a set of plays closed by even-length prefixing. For our purpose, however, it is more convenient to represent strategies using *prefix-closed* set of plays. This will spare us some considerations on the parity of traversal length when showing the correspondence between traversals and game semantics. For the rest of the section we fix a simply-typed term $\Gamma \vdash M : T$. We write $[\![\Gamma \vdash M : T]\!]$ for its strategy denotation (in the standard cartesian closed category of games and innocent strategies [8, 6]). We use the notation $\mathsf{Pref}(S)$ to denote the prefix-closure of the set S.

2.1. Revealed game semantics

In standard game semantics, terms are denoted by strategies that are computed inductively on the structure of the term: calculating the denotation of a term boils down to performing the composition of strategies denoting some of its subterms. Strategy composition is the CSP-like "composition + hiding" operation where all the internal moves are hidden.

It is possible to use an alternative notion of composition where the internal moves are not hidden. Game model based on such notion of composition have appeared in the literature under the name *revealed semantics* [9] and *interaction semantics* [10]. In such game models, the denotation is computed inductively on the syntax of the term as in the standard game semantics, but certain internal moves may be uncovered after composition.

There is not just one revealed semantics as one may desire to hide/uncover different internal moves. Such semantics will help to establish a correspondence between the game semantics of a term and the traversals of its computation tree.

This section presents a general setting in which revealed semantics can be defined. At the end of the section we will provide an example of such an revealed semantics that is calculated inductively on the syntax of the η-long normal form of the term.

2.1.1. Revealed strategies

Definition 2.1. We consider ordered trees whose leaves are labelled with PCF simple types and inner nodes are labelled with symbols in $\{;, \langle \_ , \_ \rangle, \Lambda\}$ where '$;$' and '$\langle \_ , \_ \rangle$' are of arity 2 and 'Λ' is of arity one. We write $\langle T_1, T_2\rangle$ for the tree obtained by attaching T_1 and T_2 to a $\langle \_ , \_ \rangle$-node, and similarly we use the notations $T_1; T_2$ and $\Lambda(T_1)$.

The set of ***interaction type trees***, or just ***interaction types***, is defined inductively as follows:

- *Leaf*: If T is a leaf annotated by a type A then T is an interaction type, and we define $\mathit{type}(T)$ to be A;
- *Currying*: If T is an interaction type with $\mathit{type}(T) = A \times B \rightarrow C$ then $\Lambda(T)$ is also an interaction type and $\mathit{type}(\Lambda(T)) = A \rightarrow (B \rightarrow C)$;
- *Pairing*: If T_1 and T_2 are interaction types with $\mathit{type}(T_1) = C \rightarrow A$ and $\mathit{type}(T_2) = C \rightarrow B$ then $\langle T_1, T_2\rangle$ is also an interaction type and $\mathit{type}(\langle T_1, T_2\rangle) = C \rightarrow A \times B$ (Pairing generalizes straightforwardly to a p-tuple operator $\langle \Sigma_1, \ldots, \Sigma_p\rangle$ for $p \geq 2$, in which case the tree has p child subtrees.);
- *Composition*: If T_1 and T_2 are interaction types with $\mathit{type}(T_1) = A \rightarrow B$ and $\mathit{type}(T_2) = B \rightarrow C$ then $T_1; T_2$ is also an interaction type and $\mathit{type}(T_1; T_2) = A \rightarrow C$.

We call $\mathit{type}(T)$ the ***underlying type*** (or just type) of the interaction type T. We sometimes write T^A to indicate that $\mathit{type}(T) = A$.

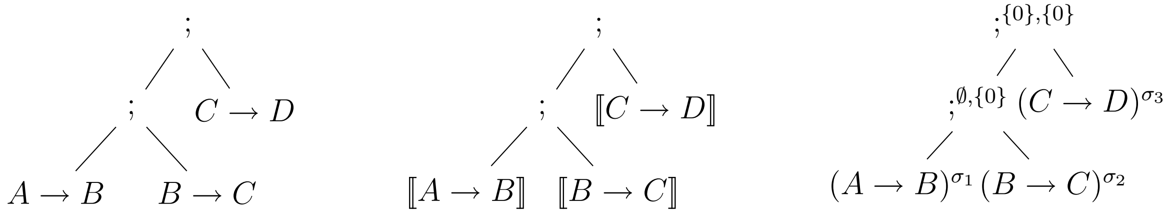
Let T be an interaction type tree. Each node of type A in T can be mapped to the (standard) game $[\![A]\!]$. By taking the image of T across this mapping we obtain a tree whose leaves and nodes are labelled by games. This tree, written $\langle\!\langle T\rangle\!\rangle$, is called an ***interaction game***. A ***revealed strategy*** Σ on the interaction game $\langle\!\langle T\rangle\!\rangle$ is a compositions of several standard strategies in which certain internal moves are not hidden. Formally:

Definition 2.2. A ***revealed strategy*** Σ on an interaction game $\langle\!\langle T\rangle\!\rangle$, written $\Sigma : \langle\!\langle T\rangle\!\rangle$, is an annotated interaction type tree T where

- each leaf $[\![A]\!]$ of T is annotated with a (standard) strategy σ on the game $[\![A]\!]$;
- each ;-node is annotated with two sets of indices $S, P \subseteq \mathbb{N}$ called respectively the *superficial* and *profound* uncovering indices.

The intuition behind this definition is that if a ;-node has children $\Sigma_1 : \langle\!\langle A \to B \rangle\!\rangle$ and $\Sigma_2 : \langle\!\langle B \to C \rangle\!\rangle$ then the two sets of indices S, P indicate which components of B should be uncovered when performing composition. The set S indicates which ***superficial*** internal moves (*i.e.*, those that are created by the top-level composition between Σ_1 and Σ_2) to uncover; whereas the set P indicates the ***profound*** internal moves (*i.e.*, those that are already present in the revealed strategies Σ_1 and Σ_2) to uncover. This notion of uncovering is made concrete in the next paragraph where we define *revealed strategies* by means of *uncovered positions.*

Example 2.1. The diagrams below represent an interaction type tree T (left), the corresponding interaction game $\langle\!\langle T \rangle\!\rangle$ (middle) and a revealed strategy Σ (right):



For convenience, a revealed strategy will be written as an expression in infix form: for instance the strategy of the example above is written $\Sigma = (\sigma_1 ;^{\emptyset,\{0\}} \sigma_2) ;^{\{0\},\{0\}} \sigma_3$.

A revealed strategy induces a strategy in the usual sense: the standard strategy $\sigma : A$ ***induced*** by a reveled strategy $\Sigma : T^A$ is obtained by replacing each occurrence of the operator '$;^{S,P}$' for some S, P by '$;^{\emptyset,\emptyset}$' (also abbreviated '$;$') in the expression of Σ. For instance the strategy Σ from the example above induces the strategy $(\sigma_1 ; \sigma_2) ; \sigma_3 : A \to D$.

2.1.2. Uncovered play

The analogue of a play in the revealed semantics is called an *uncovered play* or *uncovered position*; it is a play whose moves are interleaved with internal moves. Each move in such a play may belong to multiple games from different nodes of the interaction game; they are thus implicitly tagged so that one can retrieve the components of the node-games to which the move belongs.

Definition 2.3. The ***set of possible moves*** M_T of an interaction game $\langle\!\langle T \rangle\!\rangle$ is defined as $\mathcal{M}_T/\!\sim_T$, the quotient of the set $\mathcal{M}_T$ by the equivalence relation $\sim_T \subseteq \mathcal{M}_T \times \mathcal{M}_T$ defined as follows: For a single leaf tree T labelled by a type A we define $\mathcal{M}_T = M_A$ and $\sim_T = id_{M_A}$;

for other cases:

$$\begin{aligned}\mathcal{M}_{\Lambda(T^{A\times B\to C})} &= \mathcal{M}_T + M_{A\to B\to C}\\ \sim_{\Lambda(T^{A\times B\to C})} &= (\sim_T \cup ((A\times B\to C)\leftrightarrow(A\to(B\to C))))^{=}\end{aligned}$$

$$\begin{aligned}\mathcal{M}_{\langle T_1^{C^1\to A^1}, T_2^{C^2\to B^2}\rangle} &= \mathcal{M}_{T_1} + \mathcal{M}_{T_2} + M_{C\to(A\times B)}\\ \sim_{\langle T_1^{C^1\to A^1}, T_2^{C^2\to B^2}\rangle} &= \big(\sim_{T_1} \cup \sim_{T_2} \cup(C^1\leftrightarrow C)\cup(C^2\leftrightarrow C)\cup(A^1\leftrightarrow A)\cup(B^2\leftrightarrow B)\big)^{=}\end{aligned}$$

$$\begin{aligned}\mathcal{M}_{T_1^{A\to B};T_2^{B\to C}} &= \mathcal{M}_{T_1} + \mathcal{M}_{T_2} + M_{A\to C}\\ \sim_{T_1^{A^1\to B^1};T_2^{B^2\to C^2}} &= \big(\sim_{T_1} \cup \sim_{T_2} \cup(A^1\leftrightarrow A)\cup(B^1\leftrightarrow B^2)\cup(C\leftrightarrow C^2)\big)^{=}\end{aligned}$$

where $A \leftrightarrow B$ denotes the canonical bijection between M_A and M_B for two isomorphic games A and B; and $R^{=}$ denotes the smallest equivalence relation containing R.

It is easy to check that for every sub-type tree T' of T, the equivalence classes of $M_{T'}$ are subsets of equivalence classes of M_T. Thus $M_{T'}$ can be viewed as a subset of M_T.

We call ***internal move*** of the game $\langle\langle T\rangle\rangle$, any $\sim$-class from M_T that does not contain any move from $M_{type(T)}$. We denote the set of all internal moves by M_T^{int}. The complement of M_T^{int} in M_T, called the set of ***external moves***, is denoted by M_T^{ext}. For every subgame A occurring in some node of the interaction game T, we write $M_{T,A}^{\mathsf{int}}$ (resp. $M_{T,A}^{\mathsf{ext}}$) for the subset of moves of M_T^{int} (resp. M_T^{ext}) consisting of $\sim$-classes containing some move in M_A.

A ***justified interaction sequence*** of moves on the interaction game $\langle\langle T\rangle\rangle$ is a sequence of moves from M_T together with pointers where each move in the sequence except the first one has a link attached to it pointing to some preceding move in the sequence. We write J_T to denote the set of justified interaction sequences over $\langle\langle T\rangle\rangle$.

Definition 2.4 (Projection)**.** Let $s \in J_T$ for some interaction game T. We define the following projection operations:

(a) Let M' be a subset of M_T. The projection $s \upharpoonright M'$ is defined as the subsequence of s consisting of $\sim$-equivalence classes from M';

(b) Let A be a sub-game of $[\![type(T)]\!]$. We define the projection operator $s \upharpoonright A$ to be the subsequence of s consisting of the $\sim$-classes that contain some move in M_A. Formally $s \upharpoonright A \overset{\mathsf{def}}{=} s \upharpoonright \{[m] \mid m \in M_A\}$ where $[m]$ denotes the $\sim$-equivalence class of m.

(c) Let m be a $[\![type(T)]\!]$-initial move occurring in s. We define $s \upharpoonright m$ as the subsequence of s consisting of moves that are *hereditarily justified* by that occurrence of m in $s \upharpoonright [\![type(T)]\!]$.

(d) Let T' be an immediate subtree of T. The projection $s \upharpoonright T'$ is defined as follows:

(i) the sequence $s \upharpoonright T'$ viewed as a sequence of moves without pointers is defined as $s \upharpoonright M_{T'}$ (*i.e.*, the subsequence of s consisting of the $\sim$-equivalence classes that contain some equivalence class of $M_{T'}$; see (a));

(ii) the justification pointers of $s \upharpoonright T'$ are those of s except that if an element m loses its pointer (*i.e.*, if its justifier does not appear in $s \upharpoonright T'$) then its justifier is redefined as the only occurrence of an initial $[\![type(T')]\!]$-move in $\ulcorner s \upharpoonright M_{T'} \upharpoonright [\![type(T')]\!] \urcorner$ (*cf.* (a) and (b)).

(e) Let T' be a non-immediate subtree of T. We define the projection $s \upharpoonright T'$ as $(\ldots(s \upharpoonright T^0) \upharpoonright \ldots \upharpoonright T^{k-1}) \upharpoonright T^k$ where $T^0, \ldots, T^k$ is the uniquely defined sequence of subtrees of T satisfying $T = T^0$, $T' = T^k$ and such that for every $1 \leq l \leq k$, T^l is an immediate subtree of T^{l-1}.

(f) Let T' be some subtree of T and A be a sub-game of $[\![type(T')]\!]$. Then we write $s \upharpoonright A$ for $s \upharpoonright T' \upharpoonright A$.

By extension, we also define these operations on *sets* of justified interaction sequences.

We now characterize revealed strategies by means of sets of justified sequences of moves called *uncovered positions* or *uncovered plays*. This set is calculated by a bottom-up computation on the strategy tree. At each ;-node, we apply the composition operation of game semantics. In accordance with standard game semantics, justification pointers are adjusted when composing two interaction strategies $\Sigma_l : T_l^{A \to B}$ and $\Sigma_r : T_r^{B \to C}$: if an initial A-move a is justified by an initial B-move itself justified by an initial C-move c then a's justifier is set to c (see definition of the projection $\_ \upharpoonright A, C$ [11]). This guarantees that for every interaction position u of $\Sigma_l ; \Sigma_r$, the subsequence consisting of moves in A and C only—filtering out B-moves as well as the internal moves coming from compositions taking place at deeper level in the revealed semantics—is a valid position of the *standard* strategy underlying $\Sigma_l ; \Sigma_r$. In contrast with the standard game semantics, however, not all internal moves are hidden during composition.

Definition 2.5. A revealed strategy Σ (defined by means of an annotated type tree) is characterized by its set of ***uncovered positions*** defined inductively as follows:

- *Leaf* labelled with type A and annotated by the strategy σ: The set of positions of the revealed strategy is precisely the set of positions of the standard strategy σ.
- *Currying*: Let $\Sigma : \langle\!\langle T \rangle\!\rangle$.

$$\Lambda(\Sigma) = \{u \in J_{\Lambda(T)} \mid \rho(u) \in \Sigma\} \ ,$$

where ρ denotes the canonical bijection from $M_{\Lambda(T)}$ to M_T.

- *Pairing*: Let $\Sigma_1 : \langle\!\langle T_1 \rangle\!\rangle$ and $\Sigma_2 : \langle\!\langle T_2 \rangle\!\rangle$.

$$\begin{aligned} \langle \Sigma_1, \Sigma_2 \rangle \ = \ \{u \in J_{\langle T_1, T_2 \rangle} \mid \ & (u \upharpoonright T_1 \in \Sigma_1 \wedge \ u \upharpoonright T_2 = \epsilon) \\ & \vee \ (u \upharpoonright T_1 = \epsilon \wedge u \upharpoonright T_2 \in \Sigma_2)\} \ . \end{aligned}$$

- *Uncovered composition*: Let $\Sigma_1 : \langle\!\langle T_1 \rangle\!\rangle$ and $\Sigma_2 : \langle\!\langle T_2 \rangle\!\rangle$ where $\mathit{type}(T_1) = A \to B_0 \times \ldots \times B_l$ and $\mathit{type}(T_2) = B_0 \times \ldots \times B_l \to C$.

$$\begin{array}{rl} \Sigma_1 \| \Sigma_2 = \{u \in J_{T_1;T_2} \mid & u \upharpoonright T_2 \in \Sigma_2 \\ & \wedge \text{ for all occurrence } b \text{ in } u \text{ of an initial } [\![\mathit{type}(T_1)]\!]\text{-} \\ & \quad \text{move, } u \upharpoonright T_1 \upharpoonright b \in \Sigma_1 \\ & \wedge \text{ for every initial } A\text{-move } a \text{ justified in } u \upharpoonright T_1 \text{ by} \\ & \quad b \in B_j, \text{ itself justified by } c \in C \text{ in } u \upharpoonright T_2, \text{ we} \\ & \quad \text{have that } m \text{ is justified by } c \text{ in } u.\ \} \quad . \end{array}$$

- *Partially covered composition*: Let $\Sigma_1 : \langle\!\langle T_1 \rangle\!\rangle$ and $\Sigma_2 : \langle\!\langle T_2 \rangle\!\rangle$ where $\mathit{type}(T_1) = A \to B_0 \times \ldots \times B_l$ and $\mathit{type}(T_2) = B_0 \times \ldots \times B_l \to C$.

$$\Sigma_1 \;;^{S,P}\; \Sigma_2 = \{\mathsf{hide}(u, \{0..l\} \setminus S, \{0..l\} \setminus P) \mid u \in \Sigma_1 \| \Sigma_2\}$$
$$\text{where } \mathsf{hide}(u, S, P) = u \upharpoonright (M_T \setminus H(S,P))$$
$$H(S,P) = \bigcup_{j \in S} \underbrace{M^{\mathsf{ext}}_{T_1,B_j} \cup M^{\mathsf{ext}}_{T_2,B_j}}_{\text{superficial } B_j\text{-moves}} \cup \bigcup_{j \in P} \underbrace{M^{\mathsf{int}}_{T_1,B_j} \cup M^{\mathsf{int}}_{T_2,B_j}}_{\text{profound } B_j\text{-moves}} \quad .$$

Observe that in particular $\Sigma_1 \| \Sigma_2 = \Sigma_1 ;^{\{0..l\},\{0..l\}} \Sigma_2$.

In words, the *uncovered composition* of $\Sigma_1 \|\ \Sigma_2$ is the set of uncovered plays obtained by performing the usual composition of the standard strategies underlying Σ_1 and Σ_2 while preserving the internal moves already in Σ_1 and Σ_2 as well as the internal movea produced by the composition.

On the other hand, given a product game $B = B_0 \times \ldots \times B_l$, the *partially covered composition* $\Sigma_1 ;^{S,P} \Sigma_2$ keeps only the superficial internal moves from the component B_k for $k \in S$ as well as the profound internal moves from the component B_k for $k \in P$.

As expected, this notion of set of uncovered positions is coherent with the usual notion of positions of a standard strategy:

Lemma 2.1. *Let $\Sigma : T$ be a revealed strategy* inducing *the standard strategy $\sigma : [\![\mathit{type}(T)]\!]$. Then for all $u \in \Sigma$, $u \upharpoonright [\![\mathit{type}(T)]\!] \in \sigma$.*

Proof. The proof is by induction on the structure of Σ. It follows from the fact that the operations on revealed strategies from Def. 2.5 are defined identically to their counterparts in the standard game semantics. □

2.1.3. Fully-revealed and syntactically-revealed semantics

We call *revealed semantics* any game model of a language in which a term is denoted by some revealed strategy as defined in the previous section. As we have already observed, depending on the internal moves that we wish to hide, we obtain different possible revealed strategies for a given term. Thus there is not a unique way to define a revealed semantics. In this section we give two examples of such semantics.

Let π_i denote the i^{th} projection strategy $\pi_i : [\![X_1 \times \ldots \times X_l]\!] \to [\![X_i]\!]$.

Definition 2.6 (The fully-revealed semantics)**.** The ***fully-revealed game denotation*** of M written $\langle\!\langle \Gamma \vdash M : A \rangle\!\rangle$ is defined by structural induction on the *η-long normal form of M*:

$$\begin{aligned}
\langle\!\langle \Gamma \vdash \alpha : o \rangle\!\rangle &= \llbracket \Gamma \vdash \alpha : o \rrbracket \quad \text{where } \alpha \in \Gamma \cup \Sigma, \\
\langle\!\langle \Gamma \vdash \lambda\overline{\xi}.M : A \rangle\!\rangle &= \Lambda^{|\overline{\xi}|}(\langle\!\langle \Gamma, \overline{\xi} \vdash M : o \rangle\!\rangle) \\
\langle\!\langle \Gamma \vdash x_i N_1 \ldots N_p : o \rangle\!\rangle &= \langle \pi_i, \langle\!\langle \Gamma \vdash N_1 : A_1 \rangle\!\rangle, \ldots, \langle\!\langle \Gamma \vdash N_p : A_p \rangle\!\rangle \rangle \| ev^p, \quad X_i = A_0 \\
\langle\!\langle \Gamma \vdash f N_1 \ldots N_p : o \rangle\!\rangle &= \langle \langle\!\langle \Gamma \vdash N_1 : A_1 \rangle\!\rangle, \ldots, \langle\!\langle \Gamma \vdash N_p : A_p \rangle\!\rangle \rangle \parallel \llbracket f \rrbracket, \quad f : A_0 \in \Sigma \\
\langle\!\langle \Gamma \vdash N_0 \ldots N_p : o \rangle\!\rangle &= \langle \langle\!\langle \Gamma \vdash N_0 : A_0 \rangle\!\rangle, \ldots, \langle\!\langle \Gamma \vdash N_p : A_p \rangle\!\rangle \rangle \parallel ev^p
\end{aligned}$$

where $\Gamma = x_1 : X_1 \ldots x_l : X_l$, $A_0 = (A_1, \ldots, A_p, o)$ and ev^p denotes the evaluation strategy with p parameters where $p \geq 1$.

Fig. 1 shows tree representations of the interaction games involved in the revealed strategy $\langle\!\langle \Gamma \vdash M : A \rangle\!\rangle$ for the two application cases. These trees give us information about the constituent strategies involved in $\langle\!\langle M \rangle\!\rangle$. For instance the revealed strategy $\langle\!\langle N_0 \rangle\!\rangle$ is defined on the interaction game $\langle\!\langle T^{00} \rangle\!\rangle$ whose root game is $A \rightarrow B_0$, and the strategy ev is defined on the interaction game $\langle\!\langle T^1 \rangle\!\rangle$ whose underlying tree is constituted of a single game-node $B_0 \times \ldots \times B_p \rightarrow o$.

Example 2.2. Take the term $\lambda x.(\lambda f.fx)(\lambda y.y)$. Its fully-revealed denotation is

$$\Lambda(\langle \llbracket x : X \vdash \lambda f.fx : (o \rightarrow o) \rightarrow o \rrbracket, \llbracket x : X \vdash \lambda y.y : o \rightarrow o \rrbracket \rangle \| ev^2) \ .$$

Note that the set of fully-revealed strategies does not give rise to a category because strategy composition is not associative and there is no identity interaction strategy.

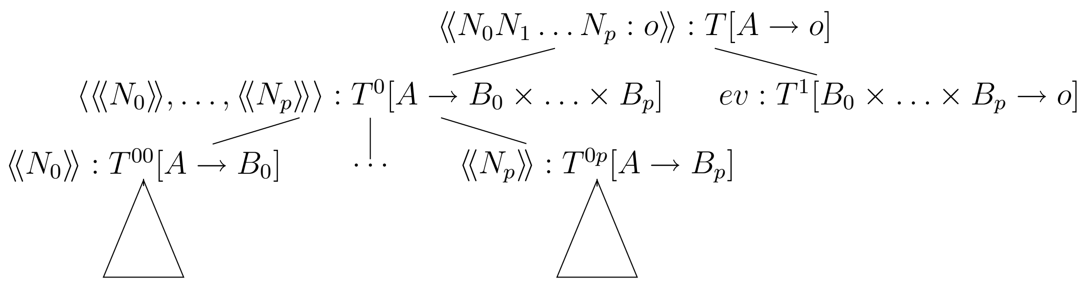
Definition 2.7 (Syntactically-revealed semantics)**.** The ***syntactically-revealed game denotation*** of M written $\langle\!\langle \Gamma \vdash M : A \rangle\!\rangle_{\mathsf{s}}$ is defined by structural induction on the *η-long normal form of M*. The equations are the same as in Def. 2.6 except for the third case:

$$\langle\!\langle \Gamma \vdash x_i N_1 \ldots N_p : o \rangle\!\rangle_{\mathsf{s}} \ = \ \langle \pi_i, \langle\!\langle \Gamma \vdash N_1 : A_1 \rangle\!\rangle_{\mathsf{s}}, \ldots, \langle\!\langle \Gamma \vdash N_p : A_p \rangle\!\rangle_{\mathsf{s}} \rangle ;^{\emptyset,\{1..p\}} ev^p, \quad X_i = A_0 \ .$$

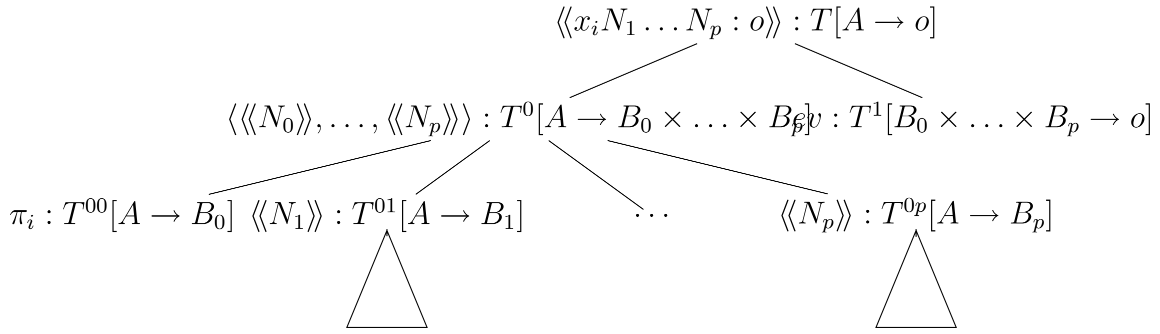
The syntactically-revealed denotation differs from the fully-revealed one in that only certain internal moves are preserved during composition: when computing the denotation of an application (joint by an @-node) in the computation tree, all the internal moves are preserved. However when computing the denotation of $\langle\!\langle y_i N_1 \ldots N_p \rangle\!\rangle_{\mathsf{s}}$ for some variable y_i, we only preserve the internal moves of N_1, ..., N_p while omitting the internal moves produced by the copy-cat projection strategy denoting y_i.

2.1.4. Relating the two revealed denotations

As one would expect, the two revealed denotations that we have just introduced are in fact equivalent. We now show how $\langle\!\langle \Gamma \vdash M : A \rangle\!\rangle$ can be obtained from $\langle\!\langle \Gamma \vdash M : A \rangle\!\rangle_{\mathsf{s}}$ and conversely.


$\langle\!\langle N_0 N_1 \ldots N_p : o \rangle\!\rangle : T[A \to o]$
$\langle \langle\!\langle N_0 \rangle\!\rangle, \ldots, \langle\!\langle N_p \rangle\!\rangle \rangle : T^0[A \to B_0 \times \ldots \times B_p]$
$ev : T^1[B_0 \times \ldots \times B_p \to o]$
$\langle\!\langle N_0 \rangle\!\rangle : T^{00}[A \to B_0]$
$\ldots$
$\langle\!\langle N_p \rangle\!\rangle : T^{0p}[A \to B_p]$

Tree-representation of the revealed strategy $\langle\!\langle \Gamma \vdash N_0 N_1 \ldots N_p : o \rangle\!\rangle$.


$\langle\!\langle x_i N_1 \ldots N_p : o \rangle\!\rangle : T[A \to o]$
$\langle \langle\!\langle N_0 \rangle\!\rangle, \ldots, \langle\!\langle N_p \rangle\!\rangle \rangle : T^0[A \to B_0 \times \ldots \times B_p]$
$ev : T^1[B_0 \times \ldots \times B_p \to o]$
$\pi_i : T^{00}[A \to B_0]$
$\langle\!\langle N_1 \rangle\!\rangle : T^{01}[A \to B_1]$
$\ldots$
$\langle\!\langle N_p \rangle\!\rangle : T^{0p}[A \to B_p]$

Tree-representation of the revealed strategy $\langle\!\langle \overline{x} : \overline{X} \vdash x_i N_1 \ldots N_p : o \rangle\!\rangle$.

A node label '$\Pi : T[G]$' indicates that Π is a revealed strategy on the interaction game T whose top-level game (at the root of the tree underlying T) is G. Each game is annotated with a string $s \in \{0..p\}^*$ in the exponent to indicate the path from the root to the corresponding node in the tree. (The digits in s tell the direction to take at each branch of the tree.)
The games A and B are given by:

$$\begin{array}{rcl} A & = & X_1 \times \ldots \times X_n \\ B & = & \underbrace{((B'_1 \times \ldots \times B'_p) \to o')}_{B_0} \times B_1 \times \ldots \times B_p \ . \end{array}$$

Figure 1: Tree-representation of the revealed strategy in the application case.

Fully-uncovered composition versus partially-uncovered composition. In this paragraph we relate the fully-uncovered composition '$\|$' with the partially-uncovered composition '$;^{\emptyset,\{1..p\}}$' used in the definition of the syntactically-revealed semantics. Take a term $M \equiv x_i N_1 \ldots N_p$. Its revealed denotation is given by $\langle\!\langle \Gamma \vdash M : o \rangle\!\rangle_{\mathsf{s}} = \Sigma_s ;^{\emptyset,\{1..p\}} ev$ where $\Sigma_s = \langle \pi_i, \langle\!\langle \Gamma \vdash N_1 : B_1 \rangle\!\rangle_{\mathsf{s}}, \ldots, \langle\!\langle \Gamma \vdash N$
We use the notations introduced in Fig. 1: the composition takes place on the game

$$X_1 \times \ldots \overbrace{((B''_1 \times \ldots \times B''_p) \to o'')}^{X_i} \ldots \times X_n \xrightarrow{\Sigma} \boxed{\overbrace{((B'_1 \times \ldots \times B'_p) \to o')}^{B_0} \times B_1 \times \ldots \times B_p} \xrightarrow{ev} o$$

where the dashed-line frame contains the internal components of the game.

In $\Sigma_s \| ev$, all the internal moves from B_k for $k \in \{0..p\}$ are preserved, whereas in $\langle\!\langle M \rangle\!\rangle_{\mathsf{s}}$, the internal B_0-moves as well as the superficial internal B_k-moves for $k \in \{1..p\}$ are hidden. By definition of the composition operator '$;^{\emptyset,\{1..p\}}$', the set $\langle\!\langle \Gamma \vdash M : o \rangle\!\rangle_{\mathsf{s}}$ is obtained from $\Sigma_s \| ev$ by eliminating the internal B-moves appropriately:

$$\langle\!\langle \Gamma \vdash M : o \rangle\!\rangle_{\mathsf{s}} = \Sigma_s ;^{\emptyset,\{1..p\}} ev = \{\mathsf{hide}(u, \emptyset, \{1..p\}) \mid u \in \Sigma_s \| ev\} \ .$$

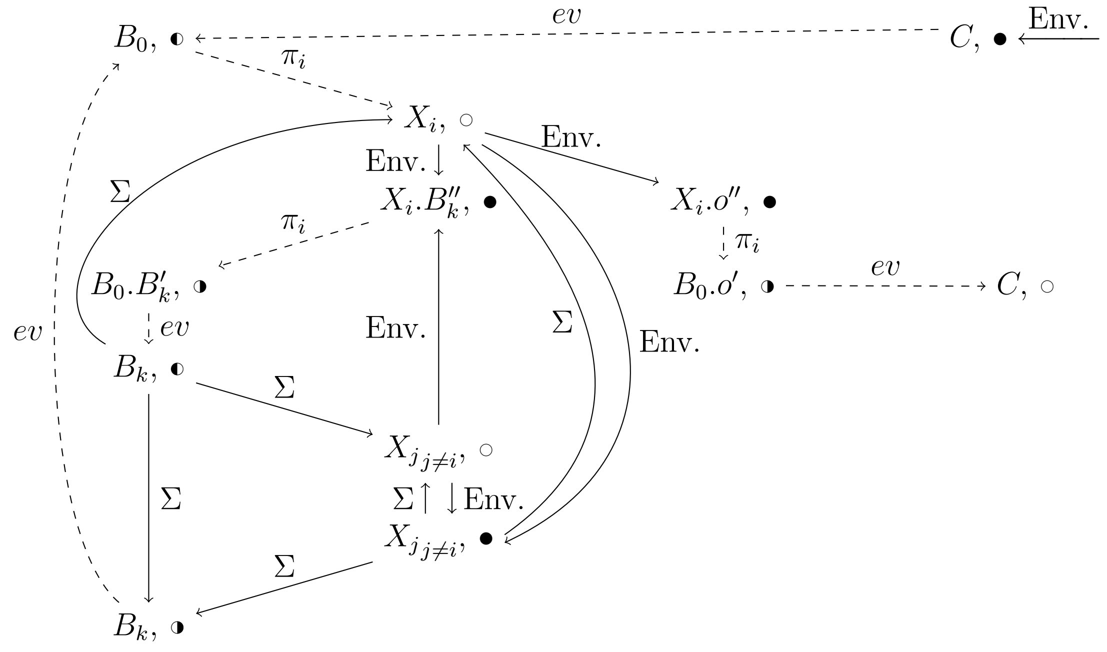
We now show that conversely, there exists a transformation mapping the set $\langle\!\langle \Gamma \vdash M : o \rangle\!\rangle_{\mathsf{s}}$ to $\Sigma_s \| ev$. More precisely we show that for every $u \in \langle\!\langle \Gamma \vdash M : o \rangle\!\rangle_{\mathsf{s}}$, there is a unique play v of $\Sigma_s \| ev$ ending with an external move such that eliminating the superficial internal moves from it gives us back u.

Let us look at the structure of an interaction play of $\Sigma \| ev$. The state-diagram in Fig. 2 describes precisely the flow of an interaction play. A node of the diagram indicates the last move that was played. Its label is of the form 'A, α' where A is the game in which the move was played, and $\alpha \in \{$ ●, ○, ◑, ◐$\}$ specifies the player that made the move. We use the symbols ◐, ◑, ●, ○ for OP-move, PO-move, O-move and P-move respectively. We use the notation '$X_i.B''_k$' to denote the sub-component B''_k of the game X_i.

An edge from node S_1 to node S_2 in the diagram indicates that the move S_2 can be played if S_1 was the last moved played. It is labelled by the name of the strategy that is responsible of making the move or by 'Env.' to denote a move played by the environment (*i.e.*, the opponent in the overall game $[\![\Gamma \to o]\!]$). For instance the edge B_k, ◑ $\xrightarrow{ev}$ B_0, ◐ tells us that if B_k, ◑ is the last move played then the evaluation strategy can respond with the move B_k, ◑. The game starts at node C, ● which corresponds to the initial move of the overall game. The dashed-edges correspond to moves played by the copy-cat strategies π_i and ev.

We observe that every (superficial) internal move played in some component B_k for $k \in \{0..p\}$ is either a copy of a previous external move, or it is subsequently copied to a external component by the copy-cat strategy ev or π_i: ◐-moves from B_0 are copies by ev of O-moves from C and ◑-moves from B_k, $k \in \{1..p\}$; ◑-moves from B_0 are copies by π_i of O-moves from X_i; ◐-moves from B_k, $k \in \{1..p\}$ are copies by ev of ◑-moves from the components B'_k of B_0; and finally ◑-moves from B_k, $k \in \{1..p\}$ are copied into B_0.

Moreover, each move on the diagram of Fig. 2 has either a single outgoing copy-cat edge—in which case the following move is uniquely determined—or it has multiple outgoing edges all labelled by Σ—in which case the strategy Σ determines which moves will



where $k \in \{1..p\}$, $i, j \in \{1..n\}$ and $p \geq 1$.

Figure 2: Flow-diagram for interaction plays of $\langle\!\langle \Gamma \vdash x_i N_1 \ldots N_p \rangle\!\rangle$.

be played next. Hence for every two consecutive moves in a play of $\langle\!\langle \Gamma \vdash M : o \rangle\!\rangle_{\mathsf{s}}$ we can uniquely recover all the internal moves occurring between the two moves in the corresponding play of $\Sigma_s \| ev$ by following the arrows of the flow diagram. This transformation is called the ***syntactical uncovering function*** with respect to Σ_s and ev and is denoted $\Upsilon_{\Sigma,ev} : \Sigma_s ;^{\emptyset,\{1..p\}} ev \rightarrow \Sigma_s \| ev$. By definition it satisfies the following property:

$$\mathsf{hide}(\Upsilon_{\Sigma,ev}(u), \emptyset, \{1..p\}) = u$$

for all $u \in \Sigma_s ;^{\emptyset,\{1..p\}} ev$ whose last occurrence is an external move (*i.e.*, in C or X_i for $i \in \{1..n\}$).

Recovering the fully-revealed semantics from the syntactically-revealed semantics. Given a term-in-context $\Gamma \vdash M : A$, its syntactically-revealed denotation $\langle\!\langle \Gamma \vdash M : A \rangle\!\rangle_{\mathsf{s}}$ can be obtained from $\langle\!\langle \Gamma \vdash M : A \rangle\!\rangle$ by recursively hiding the appropriate internal moves. Conversely, the fully-revealed denotation $\langle\!\langle \Gamma \vdash M : A \rangle\!\rangle$ can be obtained from $\langle\!\langle \Gamma \vdash M : A \rangle\!\rangle_{\mathsf{s}}$ by recursively applying the syntactical-uncovering transformation described in the previous paragraph for every subterm of the form $y_i N_1 \ldots N_p$.

2.1.5. Revealed semantics versus standard game semantics

In the standard semantics, given two strategies $\sigma : A \rightarrow B$, $\tau : B \rightarrow C$ and a sequence $s \in \sigma ; \tau$, it is possible to (uniquely) recover from the sequence s the internal moves that were hidden during composition [6, part II]. The revealed denotation of a term can be recovered from its standard game denotation by recursively uncovering the internal moves for every application occurring in the term.

Conversely, the standard denotation can be obtained from the revealed denotation by filtering out all the internal moves:

$$[\![\Gamma \vdash M : T]\!] = \langle\!\langle \Gamma \vdash M : T \rangle\!\rangle \upharpoonright [\![\Gamma \to T]\!] \ . \tag{7}$$

This equality remains valid if we replace the fully revealed denotation by the syntactically-revealed denotation.

Observe that the two sets of plays $\langle\!\langle \Gamma \vdash M : T \rangle\!\rangle$ and $[\![\Gamma \vdash M : T]\!]$ are not in bijection. Indeed, by definition the revealed denotation is prefix-closed therefore it also contains plays ending with an internal move. Thus the revealed denotation contains more plays than the standard denotation. What we can say, however, is that the set of plays $[\![\Gamma \vdash M : T]\!]$ is in bijection with the subset of $\langle\!\langle \Gamma \vdash M : T \rangle\!\rangle$ consisting of plays ending with an external move. Furthermore the set of complete plays of $[\![\Gamma \vdash M : T]\!]$ is in bijection with the set of complete interaction plays of $\langle\!\langle \Gamma \vdash M : T \rangle\!\rangle$.

2.1.6. Projection

The projection operation for justified sequences of moves of an interaction strategies (Def. 2.4) proceeds by eliminating some of the moves from the sequence. In general when projecting a sequence $s \in \Sigma$ on a subtree T', for some subtree $\Sigma' : T'$ of $\Sigma : T$, the resulting sequence is not necessarily an *interaction position* of Σ' because some internal moves may be missing from s. The following lemma shows that for strategies that are fully-revealed denotations the projection operation generates valid positions of its sub-interaction strategies.

Lemma 2.2 (Projection for fully-revealed denotations)**.** *Let $\Sigma : T$ be a fully-revealed denotation (i.e., $\Sigma = \langle\!\langle M \rangle\!\rangle$ for some term M). Then for every sub-tree $\Sigma' : T'$ of $\Sigma : T$ and $u \in \Sigma$:*

- *if T' is the first subtree of a '$;$'-node in T then for every initial $[\![type(T')]\!]$-move b occurring in u we have $u \upharpoonright T' \upharpoonright b \in \Sigma'$;*
- *otherwise (T' is the subtree of a 'Λ'-node, '$\langle \_, \_ \rangle$'-node or the l^{th} subtree of a '$;$'-node for $l > 1$) then $u \upharpoonright T' \in \Sigma'$.*

Proof. The proof is by induction on the distance between T' and T's root. The sequence $u \upharpoonright T'$ equals $u \upharpoonright T_0 \upharpoonright \ldots \upharpoonright T_k$ for some $k \geq 0$ where the T_is are the unique subtrees of T such that $T_0 = T$, $T_k = T'$, and T_i is an immediate subtree of T_{i-1} for $1 \leq i \leq k$. Let $\Sigma_i : T_i$ denote the strategy corresponding to each subtree T_i of T. We proceed by induction on $k \geq 0$. The base case is trivial. Step case: Suppose that $v = u \upharpoonright T_{k-1} \in \Sigma_{k-1}$. We do a case analysis on the type of the root node of Σ_{k-1}. The cases 'Λ' and '$\langle \_, \_ \rangle$' are trivial. The only other possible case is '$\|$' (since Σ is a fully-revealed denotation). The result then follows by definition of $\|$ with a subtlety in the case $l = 1$: we have $\Sigma_{k-1} = \Sigma' \| \Sigma_r$, $\Sigma' : T'^{A \to B}$ for some strategy $\Sigma_r : T_r^{B \to C}$. When calculating the positions of the composition $\Sigma' \| \Sigma_r$, links going from initial A-moves to initial B-moves in the positions of Σ' are changed into links pointing to initial C-moves in $\Sigma' \| \Sigma_r$. Thus in order to obtain a valid position of Σ' from v we need to recover the pointers accordingly. This is precisely what the filtering operation $\_ \upharpoonright T'$ does (see Def. 2.4): if a move in v loses its pointer in $v \upharpoonright M_{T'}$ then its justifier in $v \upharpoonright T'$ is set to the only initial move occurring in the P-view $\ulcorner v \upharpoonright M_{T'} \upharpoonright [\![type(T')]\!] \urcorner$, which is necessarily b. Hence the justification pointers are properly restored and $v \upharpoonright T' \upharpoonright b$ is indeed an uncovered position of Σ'. □

Together with Lemma 2.1 this further implies:

Lemma 2.3. *Let $\Sigma = \langle\!\langle M \rangle\!\rangle : T$. For every $u \in \Sigma$ and sub-tree $\Sigma' : T'$ of $\Sigma : T$ inducing a standard strategy $\sigma' : [\![type(T')]\!]$:*

- *if T' is the first subtree of a ';'-node in T then for every initial D-move b occurring in u we have $u \upharpoonright [\![type(T')]\!] \upharpoonright b \in \sigma'$;*
- *otherwise (T' is the subtree of a 'Λ'-node, '$\langle \_, \_ \rangle$'-node or the l^{th} subtree of a ';'-node for $l > 1$) then $u \upharpoonright [\![type(T')]\!] \in \sigma'$.*

Proof. Follows immediately from Lemma 2.2 and 2.1. □

Lemma 2.4 (Well-bracketing)**.** *Let $\Sigma : T$ be the fully-revealed denotation of some term M. Then for every sub-revealed strategies $\Sigma' : T'$ of $\Sigma : T$, the standard strategy $\sigma' : [\![type(T')]\!]$ induced by Σ' is well-bracketed.*

Proof. The leaves of a fully-revealed denotation are annotated by well-bracketed strategies therefore since well-bracketing is preserved by pairing, currying and composition, all the standard strategies induced by the sub-revealed strategies of Σ are also well-bracketed. □

Lemma 2.5 (Complete interaction play)**.** *Let $\Sigma : T$ and $\Sigma_s : T$ denote respectively the fully-revealed strategy and syntactically-revealed denotation of some term (i.e., $\Sigma = \langle\!\langle M \rangle\!\rangle$ and $\Sigma_s = \langle\!\langle M \rangle\!\rangle_{\mathsf{s}}$ for some term M). Then:*

(i) For every $u \in \Sigma$, if $u \upharpoonright [\![type(T)]\!]$ is complete (i.e., maximal and all question moves are answered) then so is u.

(ii) For every $u \in \Sigma_s$, if $u \upharpoonright [\![type(T)]\!]$ is complete then so is u.

Proof. (i) We show the contrapositive. If u is not complete then it contains an answered move b. If b is not internal then it appears in $u \upharpoonright [\![type(T)]\!]$ and therefore $u \upharpoonright [\![type(T)]\!]$ is not complete. Otherwise, let $\Sigma' : T'$ be the subtree of Σ where the internal move b is uncovered: Σ' is of the form $\Sigma_1 ;^{S,P} \Sigma_2$ for some $S, P \subseteq \mathbb{N}$ with $\Sigma_1 : \langle\!\langle T_1^{A \to B} \rangle\!\rangle$ and $\Sigma_2 : \langle\!\langle T_2^{B \to C} \rangle\!\rangle$, and b belongs to some uncovered component of B (*i.e.*, whose index is in S).

Since b is unanswered in u, it is not answered in $u \upharpoonright A, B$ and $u \upharpoonright B, C$ either; thus the sequences $u \upharpoonright A, B$ and $u \upharpoonright B, C$ are not complete. This further implies that $u \upharpoonright A, C$

is not complete (By contradiction: otherwise we would have $u \upharpoonright A \to C = \overset{\curvearrowleft}{q\,u'\,a}$ for some initial question q and answer a; but since q and a both belong to C this implies $u \upharpoonright B \to C = \overset{\curvearrowleft}{q \ldots a}$). By Lemma 2.3, $u \upharpoonright B \to C$ belongs to the standard strategy induced by Σ_2, and by Lemma 2.4 this strategy is well-bracketed, thus $u \upharpoonright B \to C$ is well-bracketed; so since its first question is answered it is necessarily complete.

We have shown that $u \upharpoonright [\![A \to C]\!] = u \upharpoonright [\![type(T')]\!]$ is not complete. We then conclude by observing that if $u \upharpoonright [\![type(T')]\!]$ is not complete for some sub-tree T' of T then $u \upharpoonright [\![type(T)]\!]$ is not complete either. This can be shown by an easy induction on the distance between the root of T' and T: The currying and pairing cases are trivial; for the composition case, the argument is similar to the one used in the previous paragraph.

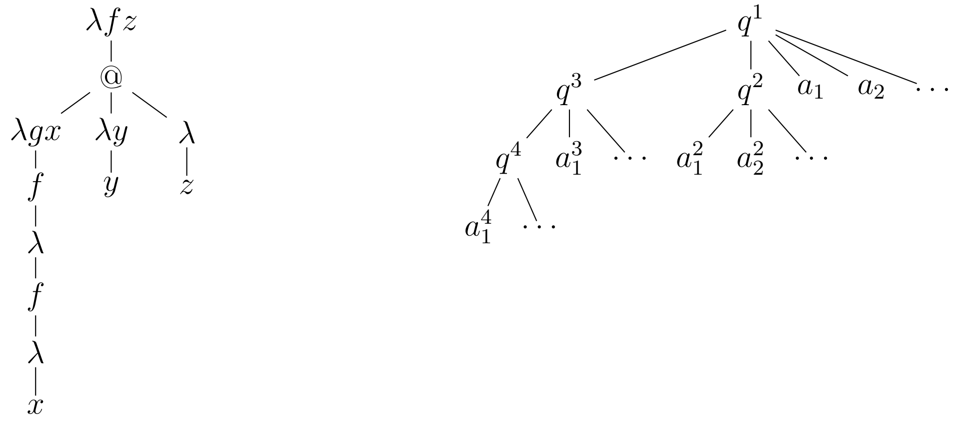
(ii) By applying the syntactical uncovering function on u we obtain a position v of Σ satisfying $u \upharpoonright [\![type(T)]\!] = v \upharpoonright [\![type(T)]\!]$. Hence by (i), v is complete, and therefore so is u (since u is the subsequence of v obtained by recursively hiding internal moves). □

2.2. Relating computation trees and games

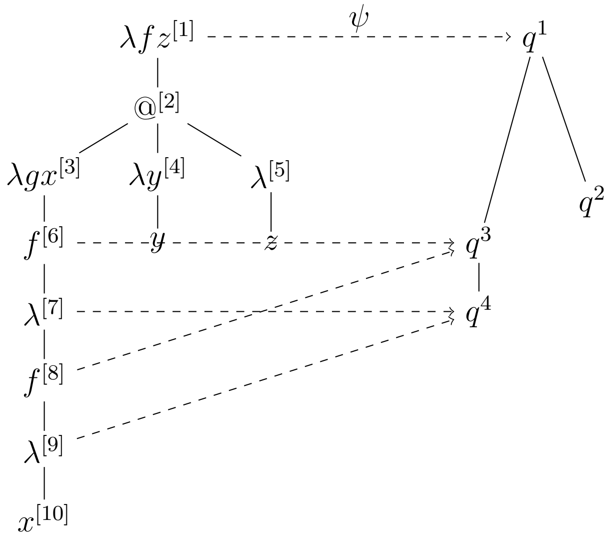
In this paragraph we relate nodes of the computation tree to moves of the game arena. First we use an example to explain the insight before giving the formal definition.

2.2.1. Example

Consider the following term $M \equiv \lambda f z.(\lambda g x.f(fx))(\lambda y.y)z$ of type $(o \to o) \to o \to o$. Its η-long normal form is $\lambda f z.(\lambda g x.f(fx))(\lambda y.y)(\lambda.z)$. The following figure represents side-by-side the computation tree of M (left) and the arena of the game $[\![(o \to o) \to o \to o]\!]$ (right):



Now consider the following partial mapping ψ (represented by a dashed line in the diagram below) from the set of nodes of the computation tree to the set of moves in the arena: (For simplicity, we now omit answer moves when representing arenas.)



Consider the justified sequence of moves:

$$s = q^1\ q^3\ q^4\ q^3\ q^4\ q^2 \in [\![M]\!] \ .$$

Its image by $\psi(r_i)$ gives a justified sequence of nodes of the computation tree:

$$r = \lambda f z \cdot f^{[6]} \cdot \lambda^{[7]} \cdot f^{[8]} \cdot \lambda^{[9]} \cdot z$$

where $s_i = \psi(r_i)$ for all $i < |s|$.

The sequence r is in fact the core of the following traversal:

$$t = \lambda f z \cdot @^{[2]} \cdot \lambda g x^{[3]} \cdot f^{[6]} \cdot \lambda^{[7]} \cdot f^{[8]} \cdot \lambda^{[9]} \cdot x^{[10]} \cdot \lambda^{[5]} \cdot z \ .$$

This example motivates the next section where we formally define the mapping ψ for any given simply-typed term.

2.2.2. Formal definition

We now establish formally the relationship between games and computation trees. We assume that a term $\Gamma \vdash M : T$ in η-long normal form is given.

NOTATIONS 2.1 We suppose that computation tree $\tau(M)$ is given by a pair (V, E) where V is the set of vertices and $E \subseteq V \times V$ is the parent-child relation. We have $V = N \cup L$ where N and L are the set of nodes and value-leaves respectively. Let $\mathcal{D}$ be the set of values of the base type o. If n is a node in N then the value-leaves attached to the node n are written v_n where v ranges in $\mathcal{D}$. Similarly, if q is a question in A then the answer moves enabled by q are written v_q where v ranges in $\mathcal{D}$.

Definition 2.8 (Mapping from nodes to moves of the standard game semantics)**.**

- Let n be a node in $N_\lambda \cup N_{\mathsf{var}}$ and q be a question move of some game A such that n and q are of type $(A_1, \ldots, A_p, o)$ for some $p \geq 0$. Let $\{q^1, \ldots, q^p\}$ (resp. $\{v_q \mid v \in \mathcal{D}\}$) be the set of question-moves (resp. answer-moves) enabled by q in A (each q^i being of type A_i).

  We define the function $\psi_A^{n,q}$ from $V^{n\vdash}$— nodes that are hereditarily enabled by n—to moves of A as:

$$\begin{array}{rcl}\psi_A^{n,q} & = & \{n \mapsto q\} \cup \{v_n \mapsto v_q \mid v \in \mathcal{D}\} \\ & & \cup \left\{ \begin{array}{ll} \bigcup_{m \in N_{\mathsf{var}} \mid n \vdash_i m} \psi_A^{m,q^i}, & \text{if } n \in N_\lambda \ ; \\ \bigcup_{i=1..p} \psi_A^{n.i,q^i}, & \text{if } n \in N_{\mathsf{var}} \ . \end{array} \right.\end{array}$$

- Suppose $\Gamma = x_1 : X_1, \ldots, x_k : X_k$. Let q_0 denote $[\![\Gamma \rightarrow T]\!]$'s initial move[3] and suppose that the set of moves enabled by q_0 in $[\![\Gamma \rightarrow T]\!]$ is $\{q_{x_1}, \ldots, q_{x_k}, q^1, \ldots, q^p\} \cup \{v_q \mid v \in \mathcal{D}\}$ where each q^i is of type A_i and q_{x_j} of type X_j.

  We define $\psi_M : V^{\circledast\vdash} \rightarrow [\![\Gamma \rightarrow T]\!]$ (or just ψ if there is no ambiguity) as:

$$\begin{array}{rcl}\psi_M = & & \{r \mapsto q_0\} \cup \{v_r \mapsto v_{q_0} \mid v \in \mathcal{D}\} \\ & \cup & \displaystyle\bigcup_{n \in N_{\mathsf{var}} \mid \circledast \vdash_i n} \psi_{[\![\Gamma \rightarrow T]\!]}^{n,q^i} \\ & \cup & \displaystyle\bigcup_{n \in N_{\mathsf{fv}} \mid n \text{ labelled } x_j, j \in \{1..k\}} \psi_{[\![\Gamma \rightarrow T]\!]}^{n,q_{x_j}} \ .\end{array}$$

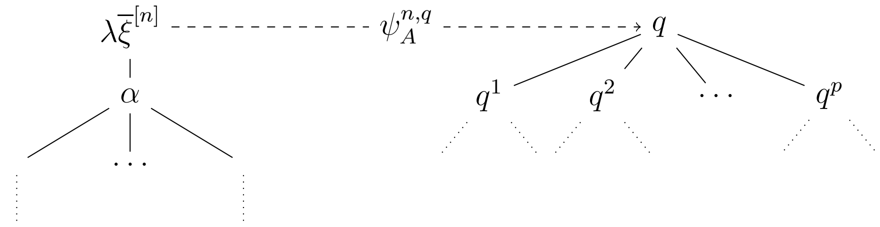
It can easily be checked that the domain of definition of $\psi_A^{n,q}$ is indeed the set of nodes that are hereditarily enabled by n and similarly, the domain of ψ_M is the set of nodes that are hereditarily enabled by the root (this includes free variable nodes and nodes that are hereditarily enabled by free variable nodes). Also, if M is closed then we have $\psi_M = \psi_{[\![\rightarrow T]\!]}^{\circledast,q_0}$.

The construction of the function $\psi_A^{n,q}$, defined above, goes as follows. Let p be the arity of the type of n and q.

- If $p = 0$ then n is a dummy λ-node or a ground type variable: $\psi_A^{n,q}$ maps n to the initial move q.

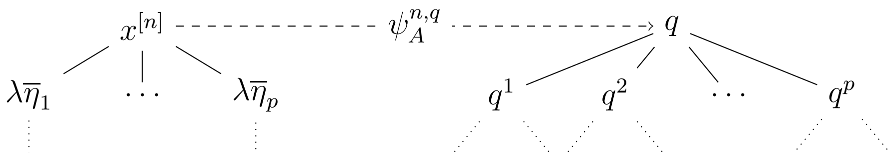
- If $p \geq 1$ and $n \in N_\lambda$ with n labelled $\lambda\overline{\xi} = \lambda\xi_1 \ldots \xi_p$ then the sub-computation tree rooted at n and the arena A have the following forms (value-leaves and answer moves are not represented for simplicity):

[3] Arenas involved in the game semantics of simply-typed lambda calculus are trees: they have a single initial move.



For each abstracted variable ξ_i there exists a corresponding question move q^i of the same order in the arena. The function $\psi_A^{n,q}$ maps each free occurrence of ξ_i in the computation tree to the move q^i.

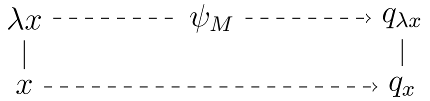
- If $p \geq 1$ and $n \in N_{\mathsf{var}}$ then n is labelled with a variable $x : (A_1, \dots, A_p, o)$ with children nodes $\lambda\overline{\eta}_1$, …, $\lambda\overline{\eta}_p$. The computation tree $\tau(M)$ rooted at n and the arena A have the following forms:



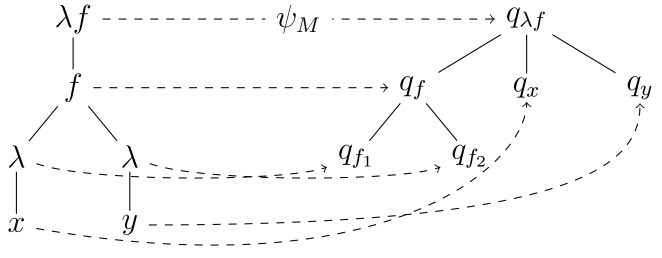
and $\psi_A^{n,q}$ maps each node $\lambda\overline{\eta}_i$ to the question move q^i.

Example 2.3. For each of the following examples of term-in-context $\Gamma \vdash M : T$, we represent the computation tree $\tau(M)$, the arena of the game $[\![\Gamma \to T]\!]$, and the function ψ_M (in dashed lines):

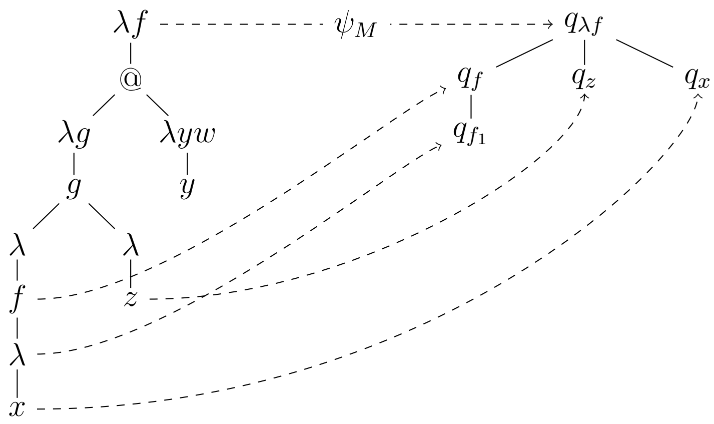
- $M = \lambda x^o.x$



- $M = \lambda f^{(o,o,o)}.fxy$



- $M = \lambda f^{(o,o)}.(\lambda g^{(o,o,o)}.g(fx)z)(\lambda y^o w^o.y)$



Lemma 2.6.

(i) ψ_M *maps* λ*-nodes to O-questions, variable nodes to P-questions, value-leaves of* λ*-nodes to P-answers and value-leaves of variable nodes to O-answers;*

(ii) ψ_M *preserves hereditary enabling: a node* $n \in V^{\circledast\vdash}$ *is hereditarily enabled by some node* $n' \in V^{\circledast\vdash}$ *in* $\tau(M)$ *if and only if the move* $\psi_M(n)$ *is hereditarily enabled by* $\psi_M(n')$ *in* $[\![\Gamma \to T]\!]$*;*

(iii) ψ_M *maps a node of a given order to a move of the same order;*

(iv) Let $s \in \mathcal{T}rav(M)^{\upharpoonright\circledast}$*. The P-view (resp. O-view) of* $\psi_M(s)$ *and* s *are computed identically (i.e., the set of positions of occurrences that need to be deleted in order to obtain the P-view (resp. O-view) is the same for both sequences).*

Proof. (i), (ii) and (iii) are direct consequences of the definition. (iv): Because of (i) and since t and $\psi_M(t)$ have the same pointers, the computations of the views of the sequence of moves and the views of the sequence of nodes follow the same steps. □

The convention chosen to define the order of the root node (see Def. 1.3) permits us to have property (iii). This explains why the order of the root node was defined differently from other lambda nodes.

By extension, we can define the function ψ_M on $\mathcal{T}rav(M)^{\upharpoonright\circledast}$, the set of traversal cores, as follows:

Definition 2.9 (Mapping traversal cores to sequences of moves)**.** The function ψ_M maps any traversal core $u = u_0 u_1 \ldots \in \mathcal{T}rav(M)^{\upharpoonright\circledast}$ to the following justified sequence of moves of the arena $[\![\Gamma \to T]\!]$: $\psi_M(u) = \psi_M(u_0)\ \psi_M(u_1)\ \psi_M(u_2) \ldots$ where $\psi_M(u)$ is equipped with u's pointers.

The pointer-free function underlying ψ_M is thus a monoid homomorphism.

2.3. Mapping traversals to interaction plays

Let I be the interaction game of the revealed strategy $\langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}}$ and M_I be the set of equivalence classes of moves from $\mathcal{M}_I$.

Let r' be a lambda node in N_{spawn} (the children nodes of @/Σ-nodes). We write $\Gamma(r') \vdash \kappa(r') : T(r')$ to denote the subterm of $\lceil M \rceil$ rooted at r' (thus $\Gamma(r') \subseteq \Gamma$). We consider the function $\psi_{\kappa(r')}$ which maps nodes of $V^{r' \vdash}$ to moves of $[\![\Gamma(r') \rightarrow T(r')]\!]$. Since $\mathcal{M}_I$ contains the moves from the standard game $[\![\Gamma(r') \rightarrow A(r')]\!]$, we can consider $\psi_{\kappa(r')}$ as a function from $V^{r' \vdash}$ to $\mathcal{M}_I$.

Every node in $n \in V \setminus (V_@ \cup V_\Sigma)$ is either hereditarily enabled by the root or by some λ-node in N_{spawn}. Therefore we can define the following relation ψ^*_M from $V \setminus (V_@ \cup V_\Sigma)$ to $\mathcal{M}_I$:

$$\psi^*_M = \psi_M \quad \cup \bigcup_{r' \in N_{\mathsf{spawn}}} \psi_{\kappa(r')} \ .$$

This relation is totally defined on $V \setminus (V_@ \cup V_\Sigma)$ since those nodes are either hereditarily justified by the root, by an @-node or by a Σ-node. Moreover it is a relation and *not* a function since for a given variable node x, for every spawn node r' occurring in the path from x to $\circledast$, x is hereditarily enabled by r' *with respect to the computation tree* $\tau(\kappa(r'))$. Thus the domains of definition of the relations $\psi_{\kappa(r')}$ for such nodes r' overlap. It can be easily check, however, that for every node $n \in V \setminus (V_@ \cup V_\Sigma)$, the moves in $\psi^*_M(n)$ are all $\sim$-equivalent, which leads us to the following definition:

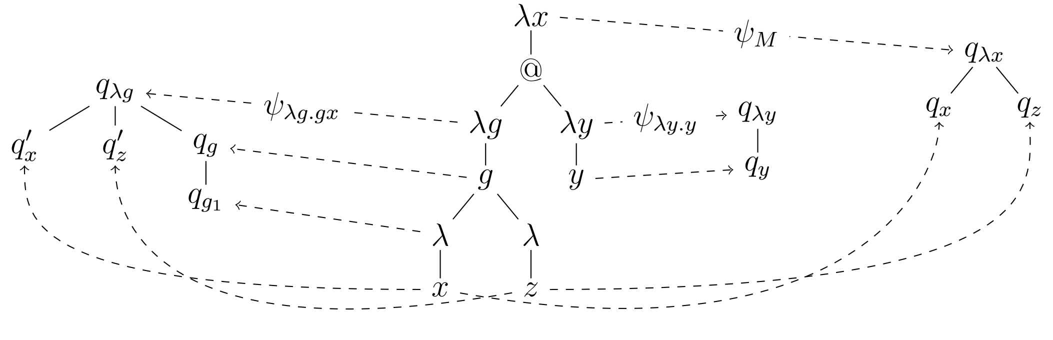
Definition 2.10 (Mapping from nodes to moves of the syntactically-revealed semantics)**.** We define the *function* $\varphi_M : V \setminus (V_@ \cup V_\Sigma) \rightarrow M_I$ as follows: For $n \in V \setminus (V_@ \cup V_\Sigma)$, $\varphi_M(n)$ is defined as the $\sim$-equivalence class containing the set $\psi^*_M(n)$. We omit the subscript in φ_M if there is no ambiguity.

Definition 2.11 (Mapping sequences of nodes to sequences of moves)**.** We define the function φ_M from $\mathcal{T}rav(M)^\star$ to justified sequence of moves in M_I as follows. If $u = u_0 u_1 \ldots \in \mathcal{T}rav(M)^\star$ then:

$$\varphi_M(s) = \varphi_M(u_0)\ \varphi_M(u_1)\ \varphi_M(u_2) \ldots$$

where $\varphi_M(u)$ is equipped with u's pointers.

Example 2.4. Take $M = \lambda x^o.(\lambda g^{(o,o)}.gxz)(\lambda y^o.y)$. The diagram below represents the computation tree (middle) and the relation $\psi^*_M = \psi_{\lambda x} \cup \psi_{\lambda g.gx} \cup \psi_{\lambda y.y}$ (dashed-lines).



where $q'_x \sim q_x$, $q'_z \sim q_z$, $q_g \sim q_{\lambda y}$, $q_{g_1} \sim q_y$ and $q_{\lambda g} \sim q_{\lambda x}$.

Lemma 2.7 (Traversal projection lemma)**.** *Let $\Delta \vdash Q : A$ be a subterm of $\lceil M \rceil$ and $\circledast_Q$ denote the root lambda node of the subtree of $\tau(M)$ corresponding to the term Q. Let $t \in \mathcal{T}rav(M)$, r_0 be an occurrence of $\circledast_Q$ in t and m_0 be the occurrence of the initial A-move $\varphi_M(r_0)$ in $\varphi_M(t^\star)$. Then:*

$$\varphi_Q(t^\star \upharpoonright V^{(\circledast_Q)} \upharpoonright r_0) = \varphi_M(t^\star) \upharpoonright \langle\!\langle \Delta \rightarrow A \rangle\!\rangle \upharpoonright m_0 \ .$$

Proof. Firstly we observe that the expression "$\varphi_Q(t^\star \upharpoonright V^{(\circledast_Q)} \upharpoonright r_0)$" is well-defined. Indeed, by Proposition 1.5 $t \upharpoonright\!\upharpoonright r_0$ is a traversal of $\mathcal{T}rav(Q)$ therefore the sequence $t^\star \upharpoonright V^{(\circledast_Q)} \upharpoonright r_0$, which is equal to $(t \upharpoonright\!\upharpoonright r_0)^\star$ by Lemma 1.17, does belong to $\mathcal{T}rav(Q)^\star$.

We now make the assumption that $\circledast_Q$ is a level-2 lambda nodes (*i.e.*, a grand-child of the root $\circledast$). The proof easily generalizes to other lambda nodes by iterating the argument at every lambda nodes occurring in the path from $\circledast_Q$ to $\circledast$.

Claim: (i) The set of occurrence positions of $t^\star$ that are removed by the operation $\_ \upharpoonright V^{(\circledast_Q)}$ is the same as the set of positions of $\varphi_M(t^\star)$ removed by the operation $\_ \upharpoonright \langle\!\langle \Delta \rightarrow A \rangle\!\rangle$. (ii) The justification pointers in the sequences of nodes $t^\star \upharpoonright V^{(\circledast_Q)}$ are the same as those of the sequence of moves $\varphi_M(t^\star) \upharpoonright \langle\!\langle \Delta \rightarrow A \rangle\!\rangle$.

Indeed: (i) follows from the fact that, by definition, the range of the function φ_M restricted to $V^{(\circledast_Q)}$ is included in $M_{\langle\!\langle \Delta \rightarrow A \rangle\!\rangle}$ (the set of moves of the interaction game of Q).

(ii) By Def. 2.11, the sequences $\varphi_M(t^\star)$ and $t^\star$ have the same justification pointers. The projections $\_ \upharpoonright V^{(\circledast_Q)}$ and $\_ \upharpoonright \langle\!\langle \Delta \rightarrow A \rangle\!\rangle$ both alter the pointers in the sequences $\varphi_M(t^\star)$ and $t^\star$, but they do so identically: the operation $\_ \upharpoonright V^{(\circledast_Q)}$ (Def. 1.17) alters pointers only for variable nodes that are free in $V^{(\circledast_Q)}$; it makes them point to the only occurrence of $\circledast_Q$ in the P-view at that point (which is also the only occurrence of a level-2 lambda node in the P-view). Similarly, the operation $\_ \upharpoonright \langle\!\langle \Delta \rightarrow A \rangle\!\rangle$ (Def. 2.4) alters pointers only for initial A-moves: it makes them point to the only occurrence of an initial B-move in the P-view at that point. Further φ_M maps free variables in $V^{(\circledast_Q)}$ to initial A-moves, and level-2 lambda nodes to initial B-moves.

Hence the claim holds which subsequently implies $\varphi_M(t^\star) \upharpoonright \langle\!\langle \Delta \rightarrow A \rangle\!\rangle = \varphi_M(t^\star \upharpoonright V^{(\circledast_Q)})$. Thus $\varphi_M(t^\star) \upharpoonright \langle\!\langle \Delta \rightarrow A \rangle\!\rangle \upharpoonright m_0 = \varphi_M(t^\star \upharpoonright V^{(\circledast_Q)}) \upharpoonright m_0 = \varphi_M(t^\star \upharpoonright V^{(\circledast_Q)} \upharpoonright r_0)$. Finally, since the function φ is defined inductively on the structure of the computation tree, the restriction of φ_M to $V^{\circledast_Q}$ coincides with φ_Q. □

The following lemma states that projecting the image of a traversal by φ gives the image of the traversal's core:

Lemma 2.8 (Core projection lemma)**.**

$$\varphi_M(\mathcal{T}rav(M)^\star) \upharpoonright \llbracket \Gamma \rightarrow T \rrbracket = \psi_M(\mathcal{T}rav(M)^{\upharpoonright \circledast}) \ .$$

Proof. Let H be the set of nodes of $\tau(M)$ which are mapped by $\psi^*(M)$ to moves that are $\sim$-equivalent to moves in $\llbracket \Gamma \rightarrow T \rrbracket$. We need to show that $H = V^{\circledast\vdash}$.

Since $\psi_M \subseteq \psi^*(M)$ and the image of $\psi(M)$ is $[\![\Gamma \rightarrow T]\!]$, H must contain the domain of $\psi(M)$ which is precisely $V^{\circledast\vdash}$. Conversely, suppose that a node $n \in V \setminus (V_@ \cup V_\Sigma)$ is mapped by $\varphi^*(M)$ to some move $m \in \mathcal{M}_I$ which is $\sim$-equivalent to some move in $[\![\Gamma \rightarrow T]\!]$. If $m = \psi_M(n)$ then $n \in V^{\circledast\vdash}$. Otherwise, $m = \psi_{\kappa(\odot)}(n)$ for some $\odot \in N_{\mathsf{spawn}}$. There may be several node $\odot$ such that n belongs to the domain of definition of $\psi_{\kappa(\odot)}$, w.l.o.g. we can take $\odot$ to be the one which is closest to the root. Let $\Gamma(\odot) \vdash \kappa(\odot) : T(\odot)$. Suppose that m is $\sim$-equivalent to a move from

- the subgame $[\![\Gamma]\!]$ of $[\![\Gamma \rightarrow T]\!]$, then this means that n is hereditarily justified by a free variable node in M and therefore $n \in V^{\circledast\vdash}$.
- the subgame $[\![T]\!]$ of $[\![\Gamma \rightarrow T]\!]$ then m must belong to the subgame $\Gamma(\odot)$ of $[\![\Gamma(\odot) \rightarrow T(\odot)]\!]$. Indeed, since $\odot$'s parent node is an application node, moves in the subgame $[\![T(\odot)]\!]$ correspond to internal moves of the application. By definition of the interaction strategy for the application case, such moves can only be $\sim$-equivalent to other internal moves and thus cannot be equivalent to a move from $[\![T]\!]$.

  Consequently, n is hereditarily justified by a free variable node z in $\kappa(\odot)$. By assumption, $\odot$ is the closest node to the root $\circledast$ (excluding $\circledast$ itself) for which n belongs to $V^{\odot\vdash}$ (the domain of definition of $\psi_{\kappa(\odot)}$). Hence z is not bound by any λ-node occurring in the path to the root. Thus $z \in V^{\circledast\vdash}$ and therefore $n \in V^{\circledast\vdash}$.

Hence $H = V^{\circledast\vdash}$. Consequently, for every traversal t we have $\varphi_M(t^\star) \upharpoonright [\![\Gamma \rightarrow T]\!] = \varphi_M(t^\star \upharpoonright V^{\circledast\vdash})$ which equals $\varphi_M(t \upharpoonright \circledast)$ by Lemma 1.8. □

2.4. The correspondence theorem for the pure simply-typed lambda calculus

In this section, we establish a connection between the revealed semantics of a simply-typed term without interpreted constants (*i.e.*, $\Sigma = \emptyset$) and the traversals of its computation tree: we show that the set $\mathcal{T}rav(M)$ of traversals of the computation tree is isomorphic to the set of uncovered plays of the strategy denotation (this is the counterpart of Ong's "Path-Traversal Correspondence" Theorem [1]), and that the set of traversal cores is isomorphic to the strategy denotation.

Preliminary lemmas

NOTATION 2.2 For every node occurrence n in a justified sequence (of nodes or of moves) u we write $\mathsf{ptrdist}_u(n)$, or just $\mathsf{ptrdist}(n)$ if there is no ambiguity, to denote the distance between n and its justifier in u if it has one, and 0 otherwise.

Lemma 2.9.

$$\left(\begin{array}{l} t \cdot n_1, t \cdot n_2 \in \mathcal{T}rav(M) \\ \wedge \ \ n_1 \neq n_2 \end{array}\right) \implies n_1, n_2 \in V^{\circledast\vdash}_\lambda \wedge (\psi(n_1) \neq \psi(n_2) \vee \mathsf{ptrdist}(n_1) \neq \mathsf{ptrdist}(n_2)) .$$

Proof. Take $t \cdot n_1, t \cdot n_2 \in \mathcal{T}rav(M)$. Suppose that n_1 and n_2 belong to two distinct categories of nodes (N_{var}, $N_@$, N_λ, N_Σ, L_{var}, $L_@$, L_λ, or L_Σ) then necessarily one must be visited with the rule ($\mathsf{InputVar}$) and the other by ($\mathsf{InputVal}$)—they are the only rules with a common domain of definition—thus one is a leaf-node and the other is an inner node which implies that $\psi(n_1) \neq \psi(n_2)$.

Otherwise n_1 and n_2 belong to the same category of nodes and we proceed by case analysis:

- If $n_1, n_2 \in N_@$ then $t \cdot n_1$ and $t \cdot n_2$ are formed using the (App) rule. Since this rule is deterministic we must have $n_1 = n_2$ which violates the second hypothesis.
- If $n_1, n_2 \in L_@$ then the traversals are formed using the deterministic rule $(\mathsf{Value}^{@\mapsto\lambda})$ which again violates the second hypothesis.
- If $n_1, n_2 \in N_\Sigma$ then they are formed using a deterministic constant rule (see Def. 1.13).
- If $n_1, n_2 \in L_\Sigma$ then they are formed using a deterministic value-constant rule.
- If $n_1, n_2 \in N_{\mathsf{var}}$ then $t \cdot n_1$ and $t \cdot n_2$ were formed using either rule (Lam) or (App). But these two rules are deterministic and their domains of definition are disjoint. Hence again the second hypothesis is violated.
- If $n_1, n_2 \in L_{\mathsf{var}}$ then either the traversals were both formed using the deterministic rule $(\mathsf{Value}^{\mathsf{var}\mapsto\lambda})$ in which case the second hypothesis is violated; or they were formed with $(\mathsf{InputValue})$ in which case n_1 and n_2 are two different value leaves belonging to $V_\lambda^{\circledast\vdash}$ and justified by the same input variable node. Thus by definition of ψ, $\psi(n_1) \neq \psi(n_2)$.
- If $n_1, n_2 \in N_\lambda$ then the traversals $t \cdot n_1$ and $t \cdot n_2$ must have been formed using either rule (Root), (App), (Var) or $(\mathsf{InputVar})$. Since all these rules have disjoint domains of definition, the same rule must have been use to form $t \cdot n_1$ and $t \cdot n_2$. But since the rules (Root), (App) and (Var) are all deterministic, the rule used is necessarily $(\mathsf{InputVar})$.

  By definition of $(\mathsf{InputVar})$, $n_1, n_2 \in N_\lambda^{\circledast\vdash}$, the parent node of n_1 and the parent node of n_2 all occur in $\llcorner t_{\leqslant x} \lrcorner$ where $x \in N_{\mathsf{var}}^{\circledast\vdash}$ denotes the pending node at t. If n_1 and n_2 have the same parent node in $\tau(M)$ then since $n_1 \neq n_2$, by definition of ψ, $\psi(n_1) \neq \psi(n_2)$. If their parent node is different, then n_1 and n_2 are necessarily justified by two different occurrences in t therefore $\mathsf{ptrdist}(n_1) \neq \mathsf{ptrdist}(n_2)$.
- If $n_1, n_2 \in L_\lambda$ then either the traversals $t \cdot n_1$ and $t \cdot n_2$ were formed using $(\mathsf{Value}^{\lambda\mapsto\mathsf{var}})$ or they were formed with $(\mathsf{Value}^{\lambda\mapsto @})$ but this is impossible since these two rules are deterministic and $n_1 \neq n_2$. □

The function φ_M regarded as a function from the set of vertices $V \setminus V_@$ of the computation tree to moves in arenas is not injective. (For instance the two occurrences of x in the computation tree of $\lambda fx.fxx$ are mapped to the same question move.) However the function φ_M defined on the set of @-free traversals is injective, and similarly the function ψ_M defined on the set of traversal cores is injective as the following lemma shows:

Lemma 2.10 (ψ_M and φ_M are injective)**.** *For every two traversals t_1 and t_2:*

(i) If $\varphi(t_1^\star) = \varphi(t_2^\star)$ then $t_1^\star = t_2^\star$;

(ii) if $\psi(t_1 \upharpoonright \circledast) = \psi(t_2 \upharpoonright \circledast)$ then $t_1 \upharpoonright \circledast = t_2 \upharpoonright \circledast$.

Proof. (i) The result is trivial if either t_1 or t_2 is empty. Otherwise, suppose that $t_1^\star \neq t_2^\star$ then necessarily $t_1 \neq t_2$. W.l.o.g. we can assume that the two traversals differ only by their last node (or last node's pointer). Thus we have $t_1 = t \cdot n_1$ and $t_2 = t \cdot n_2$ for some sequence t and some occurrences n_1, n_2 where either n_1 and n_2 are two distinct nodes in the computation tree or $\mathsf{ptrdist}(n_1) \neq \mathsf{ptrdist}(n_2)$.

If $n_1 = n_2$ and $\mathsf{ptrdist}(n_1) \neq \mathsf{ptrdist}(n_2)$ then n_1, n_2 are not @-nodes nor Σ-nodes (since for such nodes we would have $\mathsf{ptrdist}(n_1) = 0 = \mathsf{ptrdist}(n_2)$). By definition of the sequence $\varphi(t_1)$ we have $\mathsf{ptrdist}(\varphi(n_1)) = \mathsf{ptrdist}(n_1)$ and similarly $\mathsf{ptrdist}(\varphi(n_2)) = \mathsf{ptrdist}(n_2)$ thus $\varphi(t' \cdot n_1) \neq \varphi(t' \cdot n_2)$. Finally since $n_1, n_2 \notin (N_@ \cup N_\Sigma)$ we also have $\varphi((t' \cdot n_1)^\star) \neq \varphi((t' \cdot n_2)^\star)$. Hence $\varphi(t_1^\star) \neq \varphi(t_2^\star)$.

If $n_1 \neq n_2$ then by Lemma 2.9 n_1, n_2 are not @-nodes or Σ-nodes (since such nodes are not hereditarily justified by the root) and we have either $\mathsf{ptrdist}(n_1) \neq \mathsf{ptrdist}(n_2)$ or $\varphi(n_1) = \psi(n_1) \neq \psi(n_2) = \varphi(n_2)$. Hence $\varphi(t_1^\star) \neq \varphi(t_2^\star)$.

(ii) Suppose that $t_1 \upharpoonright \circledast \neq t_2 \upharpoonright \circledast$ then necessarily $t_1 \neq t_2$. W.l.o.g. we can assume that the two sequences differ only by their last occurrence. Hence we have $t_1 = t \cdot n_1$, $t_2 = t' \cdot n_2$ for some sequence t and some nodes n_1, n_2 where either $n_1 \neq n_2$ or $\mathsf{ptrdist}(n_1) \neq \mathsf{ptrdist}(n_2)$.

If $n_1 \neq n_2$ then Lemma 2.9 gives $\psi(t_1 \upharpoonright \circledast) \neq \psi(t_2 \upharpoonright \circledast)$. Otherwise $n_1 = n_2$ and $\mathsf{ptrdist}(n_1) \neq \mathsf{ptrdist}(n_2)$. The only rules that can visit the same node with two different pointers are ($\mathsf{InputVar}$) and ($\mathsf{InputValue}$), thus n_1 and n_2 must be in $V_\lambda^{\circledast\vdash}$. Hence:

$$\psi(t_i \upharpoonright \circledast) = \psi(t \upharpoonright \circledast) \cdot \psi(n_i) \text{ for } i \in \{1..2\}$$

where $\mathsf{ptrdist}_{\psi(t_i \upharpoonright r)}(\psi(n_i)) = \mathsf{ptrdist}_{t_i \upharpoonright r}(n_i)$.

Furthermore, since $\mathsf{ptrdist}(n_1) \neq \mathsf{ptrdist}(n_2)$ and $t_{1<n_1} = t_{2<n_2}$ we have $\mathsf{ptrdist}_{t_1 \upharpoonright \circledast}(n_1) \neq \mathsf{ptrdist}_{t_2 \upharpoonright \circledast}(n_2)$. Thus $\psi(t_1 \upharpoonright \circledast) \neq \psi(t_2 \upharpoonright \circledast)$. □

Corollary 2.1.

(i) φ defines a bijection from $\mathcal{T}rav(M)^\star$ to $\varphi(\mathcal{T}rav(M)^\star)$;

(ii) ψ defines a bijection from $\mathcal{T}rav(M)^{\upharpoonright \circledast}$ to $\psi(\mathcal{T}rav(M)^{\upharpoonright \circledast})$.

The following lemma says that extending a traversal locally also extends the traversal globally: the traversal t of M can be extended by extending a sub-traversal t' of some subterm of M. This is not obvious since t' is a subsequence of t which means that the nodes in t' are also present in t with the same pointers but with some other nodes interleaved in between. However these interleaved nodes are inserted in a way that allows us to apply on t the rule that was used to extend the sub-traversal t':

Lemma 2.11 (Sub-traversal progression)**.** *Let $\circledast_j$ be a lambda node in $\tau(M)$, $t = t' \cdot t^\omega$ be a justified sequence of nodes of $\tau(M)$, and r_j be an occurrence of $\circledast_j$ in t different from t^ω. If*

1. *t' is a traversal of $\tau(M)$,*

2. *t^ω appears in $t \upharpoonright\!\upharpoonright r_j$,*

3. *$t \upharpoonright\!\upharpoonright r_j$ is a traversal of $\tau(M^{(\circledast_j)})$ and its last node is visited using a rule different from* ($\mathsf{InputVar}$) *and* ($\mathsf{InputVar}^{\mathsf{val}}$),

then t is a traversal of $\tau(M)$.

Proof. Let $t_j = t \upharpoonright\!\upharpoonright r_j$. Since t' is a traversal of M, by Prop. 1.5 the sequence $t' \upharpoonright\!\upharpoonright r_j$ (which is also the immediate prefix of t_j) is a traversal of $\tau(M^{(\circledast_j)})$. We proceed by case analysis on the last rule used to produce the traversal t_j and we show that t is a traversal of M:

- (Empty), (Root). These cases do not occur since $|t_j| \geq 2$. Indeed, t_j contains at least t^ω and r_j which are two different occurrences.
- (Lam) We have $t_j = \ldots \cdot \lambda\overline{\xi} \cdot n$. Since $t_j \sqsubseteq t$, the node $\lambda\overline{\xi}$ also occurs in t. Therefore using the rule (Lam) in M we can form the traversal $t_{\leqslant \lambda\overline{\xi}} \cdot n$. But then we have $(t_{\leqslant \lambda\overline{\xi}} \cdot n) \upharpoonright \upharpoonright r_j = t_{\leqslant \lambda\overline{\xi}} \upharpoonright\!\upharpoonright r_j \cdot n = t_{j \leqslant \lambda\overline{\xi}} \cdot n = t_j = t \upharpoonright\!\upharpoonright r_j$. Thus, since t's last node and n both appear in $t \upharpoonright\!\upharpoonright r_j$, this implies that $t_{\leqslant \lambda\overline{\xi}} \cdot n = t$. Hence t is a traversal of M.
- (App) $t_j = \ldots \cdot \lambda\overline{\xi} \cdot @ \cdot n$. The same reasoning as in the previous case permits us to conclude.
- (Value$^{@\mapsto\lambda}$) $t_j = \ldots \cdot \lambda\overline{\xi} \cdot @ \ldots v_@ \cdot v_{\lambda\overline{\xi}}$. Since $t_j \sqsubseteq t$, the nodes $\lambda\overline{\xi}$, $@$, $v_@$ and $v_{\lambda\overline{\xi}}$ all appear in t. Moreover, since $\lambda\overline{\xi}$ is a lambda node appearing in $t \upharpoonright\!\upharpoonright r_j$, its immediate successor must also appear in $t \upharpoonright\!\upharpoonright r_j$. Thus the two nodes $\lambda\overline{\xi}$ and $@$ are also consecutive in t. Hence we can use the rule (Value$^{@\mapsto\lambda}$) in the computation tree $\tau(M)$ to produce the traversal $t_{\leqslant v_{\lambda\overline{\xi}}} \cdot n$ and by the same reasoning as in the previous case, we conclude that necessarily $t = t_{\leqslant v_{\lambda\overline{\xi}}} \cdot n$.
- (Value$^{\mathsf{var}\mapsto\lambda}$) $t_j = \ldots \cdot \lambda\overline{\xi} \cdot x \ldots v_x \cdot v_{\lambda\overline{\xi}}$. This case is identical to the previous case.
- (Value$^{\lambda\mapsto@}$) $t_j = \ldots \cdot @ \cdot \lambda\overline{z} \ldots v_{\lambda\overline{z}} \cdot v_@$. Same as in the previous case by observing that $@$ and $\lambda\overline{z}$ are necessarily consecutive in t.
- (InputValue) and (InputVar). By assumption these cases do not happen.
- (Var) $t_j = \ldots \cdot p \cdot \lambda\overline{x} \ldots x_i \cdot \lambda\overline{\eta_i}$ for some variable $x_i \in N^{@\vdash}_{\mathsf{var}}$.

In general, two nodes p and $\lambda\overline{x}$ appearing consecutively in t_j are not necessarily consecutive in t. For in M, t can "jump" from p to a node that do not belong to the subterm $M^{(\circledast_j)}$, and thus not appearing in $t_j = t \upharpoonright\!\upharpoonright r_j$. This situation cannot happen here, however. Indeed, suppose that $t_{\leqslant p}$ extends to $t_{\leqslant p} \cdot m$ in $\tau(M)$. All the nodes in the thread of $\lambda\overline{\eta_i}$, in t_j, are hereditarily justified by the same initial $@$-node α which necessarily occurs after r_j (the first node of t_j). Consequently p belongs to $N^{@\vdash}_{\mathsf{var}}$ and therefore the traversal $t_{\leqslant p} \cdot m$ must have been formed using the rule (Var) in $\tau(M)$. Since p appears in $t \upharpoonright\!\upharpoonright r_j$, by Lemma 1.14(i), all the nodes in the thread of p in t appear in $t \upharpoonright\!\upharpoonright r_j$. Thus m appears in $t \upharpoonright\!\upharpoonright r_j$ (since by O-visibility it points in the thread of p). Hence $(t_{\leqslant p} \cdot m) \upharpoonright\!\upharpoonright r_0 = t_{<p} \upharpoonright\!\upharpoonright r_0 \cdot p \cdot m$ which implies that m is precisely the occurrence $\lambda\overline{x}$.

Hence the nodes p, $\lambda\overline{x}$, x_i and $\lambda\overline{\eta_i}$ all appear in t with the two nodes p and $\lambda\overline{x}$ appearing consecutively. We can therefore use the rule (Var) in M to form the traversal t.

- (Value$^{\lambda\mapsto\mathsf{var}}$) Same proof as in the previous case.
- (Σ)/(Σ-var) Same as (App) and (Var).
- (Σ-Value) Same as (Value$^{\lambda\mapsto\mathsf{var}}$). □

The correspondence theorem

We now state and prove the correspondence theorem for the simply-typed lambda calculus without interpreted constants ($\Sigma = \emptyset$). This theorem establishes a correspondence between the denotation of a term in the *intentional* game model and the set of traversals of its computation tree. The result extends immediately to the simply-typed lambda calculus with *uninterpreted* constants since we can regard constants as being free variables.

Theorem 2.2 (The Correspondence Theorem)**.** *For every simply-typed term $\Gamma \vdash M : T$, φ_M defines a bijection from $\mathcal{T}rav(M)^\star$ to $\langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}}$ and ψ_M defines a bijection from $\mathcal{T}rav(M)^{\upharpoonright\circledast}$ to $\llbracket \Gamma \vdash M : T \rrbracket$:*

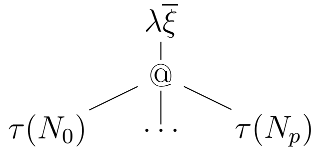
$$\begin{aligned} \varphi_M &: \mathcal{T}rav(\Gamma \vdash M : T)^\star \xrightarrow{\cong} \langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}} \\ \psi_M &: \mathcal{T}rav(\Gamma \vdash M : T)^{\upharpoonright\circledast} \xrightarrow{\cong} \llbracket \Gamma \vdash M : T \rrbracket \ . \end{aligned}$$

Remark 2.1 By Corollary 2.1, we just need to show that φ_M and ψ_M are *surjective*, that is to say: $\varphi_M(\mathcal{T}rav(M)^\star) = \langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}}$ and $\psi_M(\mathcal{T}rav(M)^{\upharpoonright\circledast}) = \llbracket \Gamma \vdash M : T \rrbracket$. Moreover the former implies the latter, indeed:

$$\begin{aligned} \llbracket \Gamma \vdash M : T \rrbracket &= \langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}} \upharpoonright \llbracket \Gamma \to T \rrbracket && \text{by (7) from Sec. 2.1.5} \\ &= \varphi_M(\mathcal{T}rav(M)^\star) \upharpoonright \llbracket \Gamma \to T \rrbracket && \text{by assumption} \\ &= \psi_M(\mathcal{T}rav(M)^{\upharpoonright\circledast}) && \text{by Lemma 2.8.} \end{aligned}$$

Therefore we just need to prove $\varphi_M(\mathcal{T}rav(M)^\star) = \langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}}$.

Since the proof is rather technical, we first give an overview of the argument: We proceed by induction on the structure of the computation tree. The only non-trivial case is the application; the computation tree $\tau(M)$ has the following form:



A traversal of $\tau(M)$ goes as follows: It starts at the root $\lambda\overline{\xi}$ of the tree $\tau(M)$ (rule (Root)), visits the node @ (rule (Lam)) and the root of $\tau(N_0)$ (rule (App)) and then proceeds by traversing the subtree $\tau(N_0)$. While doing so, some variable y_i bound by $\tau(N_0)$'s root may be reached, in which case the traversal is interrupted by a jump to $\tau(N_i)$'s root (performed with the rule (Var)) and the process goes on with $\tau(N_i)$. Again, if the traversal encounters a variable bound by $\tau(N_i)$'s root then the traversal of $\tau(N_i)$ is interrupted and the traversal of $\tau(N_0)$ resumes. This schema is repeated until the traversal of $\tau(N_0)$ is completed[4].

[4] Since we are considering simply-typed terms, the traversal does indeed terminate. However this will not be true anymore in the PCF case.

The traversal of M is therefore made of an initialization part followed by an interleaving of a traversal of N_0 and several traversals of N_i for $i = 1..p$. This schema is reminiscent of the way the evaluation copy-cat map ev works in game semantics.

The crucial idea of the proof is that every time the traversal jumps from one subterm to another, the jump is permitted by one of the "copy-cat" rules (Var), ($\mathsf{Value}^{\lambda\mapsto @}$), ($\mathsf{Value}^{\mathsf{var}\mapsto\lambda}$), ($\mathsf{Value}^{@\mapsto\lambda}$), or ($\mathsf{Value}^{\lambda\mapsto\mathsf{var}}$). We show by a second induction that these copy-cat rules implement precisely the copy-cat evaluation strategy ev.

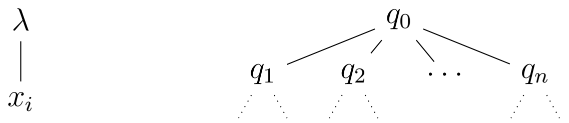
Proof. Let $\Gamma \vdash M : T$ be a simply-typed term where $\Gamma = x_1 : X_1, \ldots x_n : X_n$. We assume that M is already in η-long normal form. By remark 2.1 we just need to show that $\varphi_M(\mathcal{T}rav(M)^\star) = \langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}}$. We proceed by induction on the structure of M:

- (abstraction) $M \equiv \lambda\overline{\xi}.N : \overline{Y} \rightarrow B$ where $\overline{\xi} = \xi_1 : Y_1, \ldots \xi_n : Y_n$. On the one hand we have:

$$\begin{aligned}\langle\!\langle \Gamma \vdash \lambda\overline{\xi}.N : T \rangle\!\rangle_{\mathsf{s}} &= \Lambda^n(\langle\!\langle \overline{\xi}, \Gamma \vdash N : B \rangle\!\rangle_{\mathsf{s}}) \\ &\simeq \langle\!\langle \overline{\xi}, \Gamma \vdash N : B \rangle\!\rangle_{\mathsf{s}} \ .\end{aligned}$$

  On the other hand, the computation tree $\tau(N)$ is isomorphic to $\tau(\lambda\overline{\xi}.N)$ (up to renaming of the computation tree's root), and $\mathcal{T}rav(N)$ is isomorphic to $\mathcal{T}rav(\lambda\overline{\xi}.N)$. Hence we can conclude using the induction hypothesis.

- (variable) $M \equiv x_i$. Since M is in η-long normal form, x must be of ground type. The computation tree $\tau(M)$ and the arena $\langle\!\langle \Gamma \rightarrow o \rangle\!\rangle_{\mathsf{s}}$ are represented below (value leaves and answer moves are not represented):



  Let π_i denote the i^{th} projection of the interaction game semantics. We have:

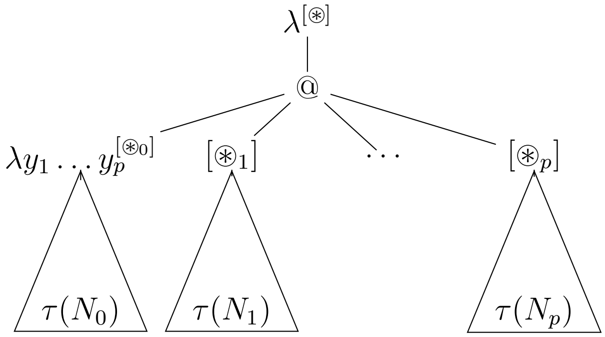
$$\langle\!\langle M \rangle\!\rangle_{\mathsf{s}} = \pi_i = \mathsf{Pref}(\{q_0 \cdot q^i \cdot v_{q^i} \cdot v_{q_0} \mid v \in \mathcal{D}\}) \ .$$

  It is easy to see that traversals of M are precisely the prefixes of $\lambda \cdot x_i \cdot v_{x_i} \cdot v_\lambda$. Since M is in β-normal we have $\mathcal{T}rav(M)^\star = \mathcal{T}rav(M)$, and since $\varphi_M(\lambda) = q_0$ and $\varphi_M(x_i) = q^i$ we have:

$$\varphi_M(\mathcal{T}rav(M)^\star) = \varphi_M(\mathcal{T}rav(M)) = \varphi_M(\mathsf{Pref}(\lambda \cdot x_i \cdot v_{x_i} \cdot v_\lambda)) = \langle\!\langle M \rangle\!\rangle_{\mathsf{s}} \ .$$

- (@-application) $M = N_0 N_1 \ldots N_p : o$ where N_0 is not a variable. We have the typing judgments $\Gamma \vdash N_0 N_1 \ldots N_p : o$ and $\Gamma \vdash N_i : B_i$ for $i \in 0..p$ where $B_0 = (B_1, \ldots, B_p, o)$ and $p \geq 1$.

  The tree $\tau(M)$ has the following form:



where $\circledast_j$ denote the root of $\tau(N_j)$ for $j \in \{0..p\}$.

We have:

$$\langle\!\langle \Gamma \vdash M : o \rangle\!\rangle_{\mathsf{s}} = \underbrace{\langle \langle\!\langle \Gamma \vdash N_0 : B_0 \rangle\!\rangle_{\mathsf{s}}, \ldots, \langle\!\langle \Gamma \vdash N_p : B_p \rangle\!\rangle_{\mathsf{s}} \rangle}_{\Sigma} \parallel ev \ .$$

In the following, we use the notations introduced in Fig. 1 from section 2.1.3 which fixes the names of the different games involved in the interaction strategy $\langle\!\langle M \rangle\!\rangle_{\mathsf{s}}$. In particular the games A, B and C are defined as:

$$\begin{array}{rcl} A & = & X_1 \times \ldots \times X_n \\ B & = & \underbrace{((B'_1 \times \ldots \times B'_p) \rightarrow o')}_{B_0} \times B_1 \times \ldots \times B_p \\ C & = & o \ . \end{array}$$

Let q_0 and q'_0 be the initial question of C and B_0 respectively.

$\subseteq$ We first prove that $\langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}} \subseteq \varphi_M(\mathcal{T}rav(M)^\star)$. Suppose $u \in \langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}}$. We give a constructive proof that there is a traversal t such that $\varphi_M(t^\star) = u$ by induction on u.

For the base case $u = \epsilon$, take t to be the empty traversal formed with (Empty). *Step case:* Suppose that $u = u' \cdot m \in \langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}}$ for some move $m \in M_T$ where $u' = \varphi_M(t'^\star)$ for some traversal t' of $\tau(M)$.

By unraveling the definition of $u \in \langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}}$ we have:

$$\left.\begin{array}{ll} (a) & u \in J_T \ ; \\ (b) & \text{For every occurrence } b \text{ in } u \text{ of an initial } B_k\text{-move, for some } k \in \{0..p\}\text{:} \\ & \left\{\begin{array}{l} u \upharpoonright T^{0k} \upharpoonright b \in \langle\!\langle N_k \rangle\!\rangle_{\mathsf{s}} \ , \\ u \upharpoonright T^{0k'} \upharpoonright b = \epsilon \quad \text{for every } k' \in \{0..p\} \setminus \{k\} \ ; \end{array}\right. \\ (c) & u \upharpoonright B_0 = u \upharpoonright B_1, \ldots, B_p, C \ . \end{array}\right\} \qquad (8)$$

We recall that each $m \in M_T$ is an equivalence class of moves from $\mathcal{M}_T$. For every game A appearing in the interaction game T we will write "$m \in A$" to mean that some element of the class m belongs to the set of moves M_A. Similarly, for every sub-interaction game T' of T, we write "$m \in T'$" to mean that some element of the class m belongs to the set of moves $\mathcal{M}_{T'}$. We proceed by case analysis on m: We either have $m \in C$ or $m \in T^0$; in the last case m is either in A, a superficial internal move in B or a profound internal move in B:

- Suppose $m \in C$. Moves in C are played by the standard strategy ev that does not contain any internal move. Hence m is either q_0 or v_{q_0} for some $v \in \mathcal{D}$.
  Suppose that $m = q_0$. Since q_0 can occur only once in u we have $u = q_0$ and the traversal $t = \lambda^{[\circledast]}$ formed with (Root) clearly satisfies $\varphi(t^\star) = u$.
  Otherwise $m = v_{q_0}$. This P-move is played by the copy-cat strategy ev therefore it is the copy of some answer $v_{q'_0}$ to the question q'_0 from the sub-game o'. The move $v_{q'_0}$ is necessarily the immediate predecessor of m in u. (Indeed the play $u_{\leqslant v_{q'_0}} \upharpoonright A, B$ is complete since its first move q'_0 is answered by $v_{q'_0}$, and therefore $u_{\leqslant v_{q'_0}} \upharpoonright T^0$ is also complete by Lemma 2.5; thus no profound internal move can be played between $v_{q'_0}$ and v_{q_0}, and therefore these two moves are consecutive.) Hence by the induction hypothesis the last move in t' is $\varphi(v_{q'_0}) = v_{\lambda y_1}$. The rules ($\mathsf{Value}^{\lambda\mapsto @}$) and ($\mathsf{Value}^{@\mapsto\lambda}$) permits us to extend the traversal t' to $t' \cdot v_{@} \cdot v_{\lambda\overline{\xi}}$ where $v_{@}$ and $v_{\lambda\overline{\xi}}$ point to the second and first node of t' respectively. Clearly we have $\varphi_M((t' \cdot v_{@} \cdot v_{\lambda\overline{\xi}})^\star) = u$.
- Suppose $m \in T^0$ and m is an initial move in B_0. Then necessarily m is $q'_0 \in [\![o']\!]$, the copy-cat move of the initial move $q_0 \in C$ of u. Hence $u = q_0 \cdot q'_0$. The rules (Root), (App) and (Lam) permit us to build the traversal $t = \lambda^{[\circledast]} \cdot @ \cdot \lambda\overline{y}^{[\circledast_0]}$ which clearly satisfies $\varphi_M(t^\star) = u$.
- Suppose $m \in T^0$ and m is an initial move in B_k for some $k \in \{1..p\}$. Then m is necessarily a copy-cat move played by the evaluation strategy, and the move m^1 immediately preceding m in u is an initial move of the component B'_k of B_0.
  Thus since $\varphi_M(t'^\omega) = m^1$, t'^ω must be an occurrence of the node y_k—the k^{th} variable bound by $\lambda\overline{y}$. We can thus form, with the rule (Var), the traversal $t = t' \cdot \circledast_k$ satisfying $\varphi_M(t^\star) = \varphi_M(t'^\star) \cdot m = u$.
- Suppose $m \in T^0$ and m is not initial in B. In $u \upharpoonright T^0$, m must be hereditarily justified by some initial move b in B_k for some $k \in \{0..p\}$. Since $u \upharpoonright T^{0k} \upharpoonright b \in \langle\!\langle N_k \rangle\!\rangle_{\mathsf{s}}$, the outermost induction hypothesis gives us:

  $$u \upharpoonright T^{0k} \upharpoonright b = \varphi_{N_k}(t_k^\star) \tag{9}$$

  for some traversal $t_k \in \mathcal{T}rav(N_k)$ where w.l.o.g. we can assume that $t_k^\omega \notin V_{@}$.

We have:

$$\begin{aligned}
\varphi_M(t_k^\omega) &= (\varphi_M(t_k^\star))^\omega && \text{since } t_k^\omega \not\in V_{@} \\
&= ((u' \cdot m) \upharpoonright T^{0k} \upharpoonright b)^\omega && \text{by (9)} \\
&= ((u' \upharpoonright T^{0k} \upharpoonright b) \cdot m))^\omega && \text{since } m \text{ is h.j. by } b \text{ and belongs to } T^{0k} \\
&= m \ .
\end{aligned}$$

Take $t = t' \cdot t_k^\omega$ where t_k^ω points in t' to the image by φ_M of the occurrence justifying m in u. Since $t_k^\omega \neq @$ we have $t^\star = t'^\star \cdot t_k^\omega$ where t_k^ω justifier in $t'^\star$ is the same as its justifier in t.
Hence we have $\varphi_M(t^\star) = \varphi_M(t'^\star) \cdot \varphi_M(t_k^\omega)$ which, by the innermost I.H. together with the previous equation, equals $u' \cdot m$ where m's justifier in u' corresponds to $\varphi_M(t_k^\omega)$'s justifier in $\varphi_M(t'^\star)$. Consequently:

$$\varphi_M(t^\star) = u \ . \tag{10}$$

We are half-done at this point, it remains to show that t is indeed a traversal of $\tau(M)$. Let r_k denote the occurrence of the root $\circledast_k$ in t that is mapped to the occurrence b in $\varphi_M(t^\star)$. We make the following claim:

$$t_k = t \upharpoonright\!\upharpoonright r_k \ . \tag{11}$$

Indeed we have:

$$\begin{aligned}
\varphi_{N_k}(t_k^\star) &= u \upharpoonright T^{0k} \upharpoonright b && \text{by (9)} \\
&= \varphi_M(t^\star) \upharpoonright T^{0k} \upharpoonright b && \text{by (10)} \\
&= \varphi_{N_k}(t^\star \upharpoonright V^{(\circledast_k)} \upharpoonright r_k) && \text{by Lemma 2.7.}
\end{aligned}$$

Since φ_{N_k} is a bijection from $\mathcal{T}rav(N_k)^\star$ to $\varphi_{N_k}(\mathcal{T}rav(N_k)^\star)$ (by Corollary 2.1) this implies that $t_k^\star = t^\star \upharpoonright V^{(\circledast_k)} \upharpoonright r_k$ which in turn equals $(t \upharpoonright\!\upharpoonright r_k)^\star$ by Lemma 1.17 from Sec. 1.3.6. But since t_k and t do not end with an @-node, this implies equality (11).

We now show that t is indeed a traversal by a case analysis of the rule used to visit the last occurrence of t_k in the tree $\tau(N_k)$:

(a) Suppose the rule used to visit t_k^ω is neither (InputVar) nor (InputVar$^{\mathsf{val}}$). Then by Lemma 2.11, t is a traversal of M.

(b) Suppose t_k^ω is visited with (InputVar). Then t_k is of the form

$$t_k = \ldots \cdot \overset{\curvearrowleft}{z \cdot \ldots \cdot t_k^\omega}$$

for some input-variable $z \in N_{\mathsf{var}}^{\circledast_k \vdash}$ occurring in $\llcorner t_k \lrcorner$ and where $t_k^\omega \in N_\lambda^{\circledast_k \vdash}$.

Thus:

$$u = \ldots \cdot \underset{=m^3}{\psi_{N_k}(z)} \cdot \ldots \cdot \underset{=m}{\psi_{N_k}(t_k^\omega)} \ .$$

The occurrence t_k^ω is hereditarily enabled by the root $\circledast_k$ itself enabled by an application node, thus t_k^ω is not hereditarily enabled by the root $\circledast$. Since only nodes that are hereditarily enabled by the root are mapped to move in A we know that $\psi_{N_k}(t_k^\omega)$ is not played in A and therefore $\psi_{N_k}(t_k^\omega) \in B_k$. Similarly we have $\psi_{N_k}(z) \in B_k$.

Now consider the top-most composition in the interaction strategy $\langle\!\langle M \rangle\!\rangle_{\mathsf{s}}$—that of the interaction strategy $\Sigma : A \to B$ with the evaluation copycat strategy $ev : B \to o$. Consider the sub-sequence $u \upharpoonright A, B, C$ of u consisting only of moves involved in this top-most composition (*i.e.*, the internal moves coming from other compositions at deeper level in the revealed semantics are removed). Since z is a variable node, the move $m^3 = \psi_{N_k}(z) \in B_k$ is a P-move with *respect to the game* $[\![A \to B_k]\!]$, and therefore it is an O-move in the game $[\![B \to o]\!]$. Consequently the strategy ev is responsible to play at $u_{\leqslant m^3} \upharpoonright A, B, C$. Let m^2 denote the move played by ev which immediately follows m^1 in $u \upharpoonright A, B, C$.

We claim that m^3 and m^2 are also consecutive in u. That is to say that no internal moves generated from the other compositions at deeper levels in the interaction strategy can ever be played between m^3 and m^2. Indeed, firstly the strategy ev is a pure standard strategy thus it does not play any (profound) internal move. Furthermore, suppose that the strategy Σ comes from the composition $\Sigma_l \| \Sigma_r$ of two interaction strategies $\Sigma_l : A \to D$ and $\Sigma_r : D \to B$ for some game D, then by the Switching Condition for function-space game [6] the Opponent cannot switch of component, and thus the move following m^3 in the interaction sequence $u \upharpoonright A, D, B$ must belong to B. Hence internal moves from D cannot be played immediately after m^3.

Similarly, we can show that the move m is played by the strategy ev and is the copy of the move m^1 immediately preceding it in $u \upharpoonright A, B, C$ as well as in u.

Hence the sequence u has the following form:

$$u = \ldots \cdot m^3 \cdot m^2 \cdot \ldots \cdot m^1 \cdot m \ .$$

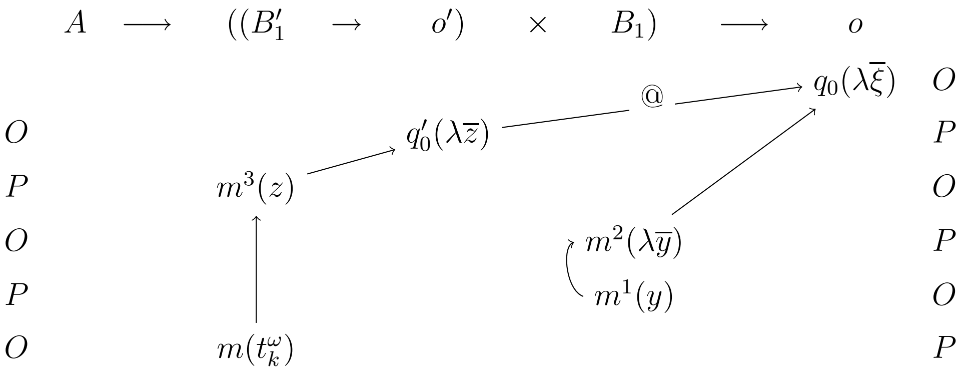
Consequently we have:

$$t_k = \ldots \cdot z \cdot \ldots \cdot t_k^\omega \qquad\qquad t' = \ldots \cdot z \cdot \lambda \overline{y} \cdot \ldots \cdot y \ .$$

The first equation implies that t_k^ω is the i^{th} child of z in the computation tree, thus since $z \notin N^{\circledast\vdash}$, we can apply the (Var) rule to the second equation which produces the traversal of $\tau(M)$:

$$t' \cdot t_k^\omega = \ldots \cdot z \cdot \lambda \overline{y} \cdot \ldots \cdot y \cdot t_k^\omega$$

which is precisely the sequence t. Hence t is indeed a traversal of $\tau(M)$. The diagram on Fig. 3 shows an example of such interaction sequence u.



$A \longrightarrow ((B'_1 \rightarrow o') \times B_1) \longrightarrow o$
$q_0(\lambda\overline{\xi})$
@
$q'_0(\lambda\overline{z})$
$m^3(z)$
$m^2(\lambda\overline{y})$
$m^1(y)$
$m(t^\omega_k)$
O
P
O
P
O
O
P
O
P
O
P

Figure 3: Example of a sequence $u \upharpoonright A, B, C$ for $u \in \langle\!\langle M \rangle\!\rangle_{\mathsf{s}}$ and $l = 1$.

(c) Suppose t_k's last move is visited with the rule ($\mathsf{InputVar}^{\mathsf{val}}$) then the proof is the same as in the previous case but with ($\mathsf{InputVar}^{\mathsf{val}}$) substituted for ($\mathsf{InputVar}$).

$\supseteq$ The converse, $\varphi_M(\mathcal{T}rav(M)^\star) \subseteq \langle\!\langle M \rangle\!\rangle_{\mathsf{s}}$, is the easy part of the proof.

Let u be as sequence of $\varphi_M(\mathcal{T}rav(M)^\star)$. Then $u = \varphi_M(t^\star)$ for some traversal t of $\tau(M)$. To show that u is a position of $\langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}}$ we have to prove that it satisfies the three conditions of (8):

- (a) By definition, φ_M maps justified sequences of nodes to justified sequences of moves from M_T therefore $\varphi_M(t^\star) \in J_T$.
- (b) Take an initial B-move $b \in B_k$, for some $k \in \{0..p\}$, occurring in $\varphi_M(t^\star)$. There is a corresponding occurrence r_k in t of a level-2 lambda node $\circledast_k$ of $\tau(M)$. By definition, the function φ_M maps nodes from the subtree of $\tau(M)$ rooted at $\circledast_{k'}$, for every $k' \in \{0..p\}$, to moves of the game $\langle\!\langle \Gamma \rightarrow B_{k'} \rangle\!\rangle_{\mathsf{s}}$ that are hereditarily justified by some occurrence of $\varphi_M(\circledast_{k'})$. Hence for every $k' \in \{0..p\} \setminus \{k\}$ we clearly have $\varphi_M(t^\star) \upharpoonright T^{0k'} \upharpoonright b = \epsilon$. Moreover:

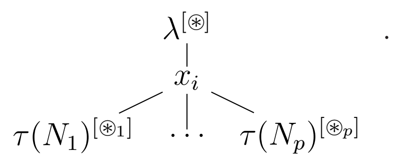
$$\begin{aligned}
u \upharpoonright T^{0k} \upharpoonright b &= \varphi_M(t^\star) \upharpoonright T^{0k} \upharpoonright b \\
&= \varphi_M(t^\star \upharpoonright V^{(\circledast_k)} \upharpoonright r_k) && \text{by Lemma 2.7} \\
&= \varphi_M((t \upharpoonright\!\upharpoonright r_k)^\star) && \text{by Lemma 1.17} \\
&= \varphi_{N_k}((t \upharpoonright\!\upharpoonright r_k)^\star) && \text{since } t \upharpoonright\!\upharpoonright r_k \text{ is a traversal of } N_k \text{ by Prop. 1.5} \\
&\in \varphi_{N_k}(\mathcal{T}rav(N_k)^\star) \\
&\quad = \langle\!\langle N_k \rangle\!\rangle_{\mathsf{s}} && \text{by the induction hypothesis.}
\end{aligned}$$

- (c) We can show that $\varphi_M(t^\star) \upharpoonright B_0 = \varphi_M(t^\star) \upharpoonright B_1, \ldots, B_p, C$ by a trivial induction on the traversal t. (This property holds because of the way the traversal rules mimic the behaviour of the evaluation strategy.)

- (Var-application) $M = x_i N_1 \dots N_p : o$.

The revealed denotation is $\langle\!\langle \Gamma \vdash M : o \rangle\!\rangle_{\mathsf{s}} = \underbrace{\langle \pi_i, \langle\!\langle \Gamma \vdash N_1 : B_1 \rangle\!\rangle_{\mathsf{s}}, \dots, \langle\!\langle \Gamma \vdash N_p : B_p \rangle\!\rangle_{\mathsf{s}} \rangle}_{\Sigma} ;^{\emptyset,\{1..p\}} ev$

and the computation tree is


$\lambda^{[\circledast]}$
x_i
$\tau(N_1)^{[\circledast_1]}$
$\dots$
$\tau(N_p)^{[\circledast_p]}$
.

We use the notations of Fig. 1 for names of the games involved in the interaction strategy. The composition of Σ with ev takes place on the following games:

$$\overbrace{X_1 \times \dots \overbrace{((B''_1 \times \dots \times B''_p) \to o'')}^{X_i} \dots \times X_n}^{A} \xrightarrow{\Sigma} \overbrace{\overbrace{((B'_1 \times \dots \times B'_p) \to o')}^{B_0} \times B_1 \times \dots B_p}^{B} \xrightarrow{ev} \overbrace{o}^{C}$$

Let q_0, q'_0 and q''_0 be the initial question of C, B_0 and X_i respectively.

$\langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}} \subseteq \varphi_M(\mathcal{T}rav(M)^\star)$. We show (constructively) by induction that for every $v \in \Sigma \| ev$, there is some traversal t such that the sequence $u = \mathsf{hide}(v, \{0..p\}, \{0\})$ equals $\varphi_M(t^\star)$.

The base case $v = \epsilon$ is trivial. Suppose that $v = v' \cdot m \in \Sigma \| ev$ where $\mathsf{hide}(v', \{0..p\}, \{0\}) = \varphi_M(t'^\star)$ for some traversal t' of $\tau(M)$ and move $m \in M_T$. Unraveling the definition of $v \in \Sigma \| ev$ gives

$$\left.\begin{array}{l}
\text{- } v \in J_T; \\
\text{- for every occurrence } b \text{ in } v \text{ of an initial } B_k\text{-move for some } k \in \{0..p\}: \\
\quad v \upharpoonright T^{00} \upharpoonright b \in \pi_i \text{ if } k = 0 \text{ and } v \upharpoonright T^{0k} \upharpoonright b \in \langle\!\langle N_k \rangle\!\rangle_{\mathsf{s}} \text{ if } k > 0, \\
\quad \text{and } \forall k' \in \{0..p\} \setminus \{k\}.\ v \upharpoonright T^{0k'} \upharpoonright b = \epsilon; \\
\text{- and } v \upharpoonright B_0 = v \upharpoonright B_1, \dots, B_p, C \ .
\end{array}\right\} \qquad (12)$$

We proceed by case analysis on m. It is either played in A, B or C.

1. $m \in C$. The proof is the same as in the @-application case except that the rules $(\mathsf{Value}^{\lambda \mapsto \mathsf{var}})$ and $(\mathsf{Value}^{\mathsf{var} \mapsto \lambda})$ are used instead of $(\mathsf{Value}^{\lambda \mapsto @})$ and $(\mathsf{Value}^{@ \mapsto \lambda})$ respectively.
2. m is a superficial internal B-move. Then $\mathsf{hide}(v, \{0..p\}, \{0\}) = \mathsf{hide}(v', \{0..p\}, \{0\})$ so we can directly conclude from the I.H.
3. m is a profound internal B-move. Then necessarily m belongs to B_k for some $k \in \{1..p\}$ (since π_i does not contain internal moves). Thus m must be hereditarily justified by some $b \in B_k$. The treatment of this case is identical to the @-application case where $m \in T^0$ is not initial in B and $b \in B_k$ for some $k \in \{0..p\}$.

4. $m \in A$. Let b denote the initial B_k-move that hereditarily justifies m for some $k \in \{0..p\}$. If $k > 0$ then the treatment is the same as in case 3. Otherwise $b \in B_0$:
   - Suppose m is an occurrence of the initial o''-move q_0''. Then m is played by π_i and therefore is the copy of q_0' itself the copy of the initial move q_0 of v. Thus $v = q_0 \cdot q_0' \cdot q_0''$ and $u = q_0 \cdot q_0''$. The traversal $t = \lambda^{[\circledast]} \cdot x_i$ formed using the rules (Root) and (Lam) meets the requirement.
   - Otherwise since $v \upharpoonright b \in \pi_i$ we have $v \upharpoonright b \upharpoonright X_i = v \upharpoonright b \upharpoonright B_0$ therefore m must necessarily be hereditarily justified by the *first* occurrence of q_0'' in v.
     * Suppose m is an $\bullet$-question. Then the preceding move in v is necessarily a $\circ$-move also played in A by the strategy π_i and therefore it is also hereditarily justified by the first occurrence of q_0''.

       By definition of φ_M, the last node in t' is a variable node (if the preceding move is a $\circ$-question) or a value-leaf of a lambda node (if the preceding move is a $\circ$-answer) that is hereditarily justified by the node x_i. Hence the rule (InputVar) can be applied at t'.

       Let m' be m's justifier in v' and α' be the corresponding node in t' that φ_M maps to m'. Suppose m is the i^{th} move enabled by m' in the arena and let α be the i^{th} child node of α' in $\tau(M)$. By definition of φ_M we have $\varphi_M(\alpha) = m$. We want to show that we can use the rule (InputVar) to append α to the traversal t'. Since we have $v \upharpoonright A, C \in \llbracket M \rrbracket$, by O-visibility m' appears in $\llcorner v' \upharpoonright A, C \lrcorner$, and by the induction hypothesis we have $v' \upharpoonright A, C = \psi_M(t' \upharpoonright r)$. Hence

       $$\begin{aligned} m' \in \llcorner \psi_M(t' \upharpoonright r) \lrcorner &= \psi_M(\llcorner t' \upharpoonright r \lrcorner) \\ &= \varphi_M(\llcorner t' \upharpoonright r \lrcorner) \quad \text{since } \varphi_M \text{ and } \psi_M \text{ coincide on } V^{\circledast\vdash}, \\ &= \varphi_M(\llcorner t' \lrcorner) \quad \text{by Lemma 1.18.} \end{aligned}$$

       This implies that α' appears in $\llcorner t' \lrcorner$ which allows us to use the rule (InputVar) to form the traversal $t = t' {\cdot} \alpha$ satisfying $\varphi_M(t^\star) = \mathsf{hide}(v, \{0..p\}, \{0\})$.
     * Suppose m is a $\circ$-answer. The same argument as above holds but using (InputValue) instead of (InputVar).
     * Suppose m is an $\bullet$-question. We proceed identically using the rule (Lam) instead of (InputVar). The proof that α' appears in the P-view $\ulcorner t' \urcorner$ goes as follows:

       Let $\ulcorner v \urcorner$ denote the *core* of the interaction sequence v [12]. By P-visibility in $v \upharpoonright A, C$, m occurs in $\ulcorner v' \upharpoonright A, C \urcorner$. Further we have $\ulcorner v' \upharpoonright A, C \urcorner = \ulcorner v' \urcorner \upharpoonright A, C$ [12], and clearly $\ulcorner v' \urcorner \upharpoonright A, C$ equals $\ulcorner \mathsf{hide}(v', \{0..p\}, \{0\}) \urcorner \upharpoonright A, C$. Hence

       $$m' \in \ulcorner \varphi_M(t'^\star) \urcorner \upharpoonright A, C \sqsubseteq \ulcorner \varphi_M(t'^\star) \urcorner \ .$$

       This implies that α' occurs in $\ulcorner t'^\star \urcorner$, which is a subsequence of $\ulcorner t' \urcorner$ by (1). (See Sec. 1.3.5).

- $*$ If m is a $\circ$-answer then we proceed as above but using the rule (Value) instead.

$\varphi_M(\mathcal{T}rav(M)^\star) \subseteq \langle\!\langle M \rangle\!\rangle_{\mathsf{s}}$. Let t be some traversal of $\tau(M)$. To show that $\varphi_M(t^\star)$ is a position of $\langle\!\langle \Gamma \vdash M : T \rangle\!\rangle_{\mathsf{s}}$ we have to prove that $\varphi_M(t^\star) = \mathsf{hide}(v, \{0..p\}, \{0\})$ for some v satisfying condition (12). It suffices to take $v = \Upsilon_{\Sigma,ev}(\varphi_M(t^\star))$ where $\Upsilon_{\Sigma,ev}$ denotes the function defined in Sec. 2.1.4 that transforms plays of the syntactically-revealed semantics to their corresponding plays of the fully-revealed semantics. The rest of the argument is the same as in the @-application case. □

Corollary 2.3. *If M is in β-normal form then for every traversal t, $\varphi_M(t)$ is a maximal play if and only if t is a maximal traversal.*

Proof. If M is in β-normal form then $\mathcal{T}rav(M)^{\upharpoonright\circledast} = \mathcal{T}rav(M)$ therefore φ defines a bijection on $\mathcal{T}rav(M)$. Let t be a traversal such that $\varphi(t)$ is a maximal play. Let t' be a traversal such that $t \leqslant t'$. By monotonicity of φ we have $\varphi(t) \leqslant \varphi(t')$ which implies $\varphi(t) = \varphi(t')$ by maximality of $\varphi(t)$ which in turn implies $t' = t$ by injectivity of φ. The other direction is proved identically using injectivity and monotonicity of φ^{-1}. □

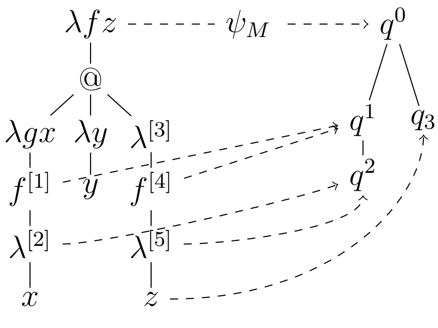
The diagram on Fig. 4 recapitulates the main results of this section.

$$\begin{tikzcd}
& \mathcal{T}rav(M)^\star \arrow[r, shift left, "\varphi_M"] \arrow[dl, bend left, "\_ + @ + \Sigma"] & \langle\!\langle M \rangle\!\rangle_{\mathsf{s}} \arrow[l, shift left, "\varphi_M^{-1}"] \arrow[dd, shift left, "\_ \upharpoonright [\![\Gamma \to T]\!]"] \\
\mathcal{T}rav(M) \arrow[ur, bend left, "\_^\star = \_ - @ - \Sigma"] \arrow[dr, bend right, "\_ \upharpoonright \circledast"'] & & \\
& \mathcal{T}rav(M)^{\upharpoonright\circledast} \arrow[r, shift left, "\varphi_M"] & [\![M]\!] \arrow[l, shift left, "\varphi_M^{-1}"] \arrow[uu, shift left, "\text{full uncovering}"]
\end{tikzcd}$$

where an arrow ‘$A \xrightarrow{f} B$’ indicates that $f(A) = B$.

Figure 4: Transformations involved in the Correspondence Theorem.

Example 2.5. Take $M = \lambda fz.(\lambda gx.fx)(\lambda y.y)(fz) : ((o,o),o,o)$. The figure below represents the computation tree (left tree), the arena $[\![((o,o),o,o)]\!]$ (right tree) and ψ_M (dashed line). (Only question moves are shown for clarity.) The justified sequence of nodes t defined hereunder is an example of traversal:



$$t = \lambda fz \cdot @ \cdot \lambda gx \cdot f^{[1]} \cdot \lambda^{[2]} \cdot x \cdot \lambda^{[3]} \cdot f^{[4]} \cdot \lambda^{[5]} \cdot z$$

$$t \upharpoonright \lambda fz = \lambda fz \cdot f^{[1]} \cdot \lambda^{[2]} \cdot f^{[4]} \cdot \lambda^{[5]} \cdot z$$

$$\varphi_M(t \upharpoonright \lambda fz) = q^0 \cdot q^1 \cdot q^2 \cdot q^1 \cdot q^2 \cdot q^3 \in [\![M]\!] \ .$$

REMARK 2.2 Observe that the way we have defined traversals, the Opponent, contrary to the Proponent, is not required to play deterministically, let alone innocently. It is only required that he plays visibly (*i.e.*, his justifiers must appear in the O-view) and respects well-bracketing. This means that the game-denotation given by the Correspondence Theorem also accounts for contexts that are not simply-typed terms. This indeed corresponds to the standard innocent game model of PCF: the morphisms of the category $\mathcal{C}_{ib}$ are P-innocent strategies but not O-innocent. The addition of O-knowing-plays in the denotations is conservative for observational equivalence because the full-abstraction result holds in the category quotiented by the intrinsic preorder, and in the definition of the preorder, the "test" strategy α ranges over innocent strategies only.

3. Extension to PCF and IA

In this section, we show how to extend the game-semantic correspondence established for the lambda calculus to other languages such as PCF and IA.

3.1. PCF fragment

The Y combinator needs a special treatment. In order to deal with it, we use an idea from Abramsky and McCusker's tutorial on game semantics [11]: we consider the sublanguage PCF_1 of PCF in which the only allowed use of the Y combinator is in terms of the form $Y(\lambda x^A.x)$ for some type A. We will write Ω_A to denote the non-terminating term $Y(\lambda x^A.x)$ for a given type A.

We introduce the *syntactic approximants* to $Y_A M$:

$$\begin{aligned} Y_A^0 M &= \Gamma \vdash \Omega_A : A \\ Y_A^{n+1} M &= M(Y^n M) \ . \end{aligned}$$

For every PCF term M and natural number n, we define M_n to be the PCF_1 term obtained from M by replacing each subterm of the form YN with $Y^n N_n$. We then have $[\![M]\!] = \bigcup_{n\in\omega} [\![M_n]\!]$ [11, lemma 16].

3.1.1. Computation tree

In order to define the notion of computation tree for PCF terms, we first extend the inductive definition of computation tree for simply-typed terms (Def. 1.2) to PCF_1 terms by adding the new inductive case:

$$\tau(\Omega_{(A_1,\ldots,A_n,o)}) = \lambda x_1^{A_1} \ldots x_n^{A_n}.\bot$$

where $\bot$ is a special constant representing the non-terminating computation of ground type Ω_o.

We now introduce a partial order on the set of trees. A ***tree*** t is formally defined by a labelling function $t : T \to L$ where T, called the *domain* of t and written $dom(t)$, is a non-empty prefix-closed subset of some free monoid X^* and L denotes the set of possible labels. Intuitively, T represents the structure of the tree—the set of all paths—and t is the labelling function mapping paths to labels. Trees are ordered using the *approximation ordering* [13, section 1]: we write $t' \sqsubseteq t$ if the tree t' is obtained from t by replacing some of its subtrees by $\bot$. Formally:

$$t' \sqsubseteq t \qquad \Longleftrightarrow \; dom(t') \subseteq dom(t) \wedge \forall w \in dom(t').(t'(w) = t(w) \vee t'(w) = \bot) \ .$$

The set of all trees together with the approximation ordering form a complete partial order.

Here we take L to be the set of labels consisting of the Σ-constants, @, the special constant $\bot$, variables, and abstractions of any sequence of variables. It is easy to check that the sequence of computation trees $(\tau(M_n))_{n\in\omega}$ is a chain. We can therefore define the ***computation tree*** of a PCF term M to be the least upper-bound of the chain of computation trees of its approximants:

$$\tau(M) = \bigcup_{n\in\omega} (\tau(M_n))_{n\in\omega} \ .$$

In other words, we construct the computation tree by expanding ad infinitum any subterm of the form YM. Thus for a term of the form $Y_A F$ with $A = (A_1, \ldots, A_n, o)$, the computation tree is the unique (up to alpha-conversion) infinite tree that is solution of the equation:

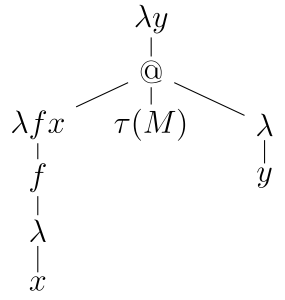
$$\tau(Y_A F) = \lambda \overline{x}^{\overline{A}}.\tau(F)\ \tau(Y_A F)\ \tau(x_1) \ldots \tau(x_n) \tag{13}$$

where $\overline{x} = x_1 \ldots x_n$ are fresh variables.

We will write $(CT, \sqsubseteq)$ to denote the set of computation trees of PCF terms ordered by the approximation ordering $\sqsubseteq$ defined above. Clearly $(CT, \sqsubseteq)$ is also a complete partial order.

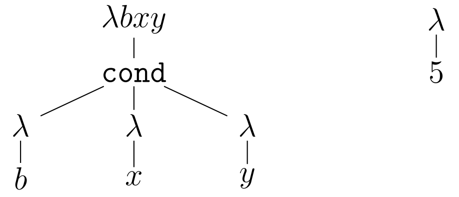
Example 3.1. Take $M = Y(\lambda f x.f x)$ where $f : (o, o)$ and $x : o$. Its computation tree $\tau(M)$, is the tree representation of the η-long nf of the infinite term $(\lambda f x.f x)((\lambda f x.f x)((\lambda f x.f x)(\ldots$.
It is the unique (up to alpha conversion) solution of the following equation on trees:

$\tau(M) =$



The remaining operators of PCF are treated as standard constants and the corresponding computation trees are constructed from the η-long normal form in the standard way. For instance the diagram below shows the computation tree for `cond` $b\ x\ y$ (left) and $\lambda x.5$

(right):



The node labelled 5 has, like any other node, children value-leaves which are not represented on the diagram above for simplicity.

3.1.2. Traversal

New traversal rules are added to interpret PCF constants. The arithmetic constants are traversed as follows:

- (Nat) If $t \cdot n$ is a traversal where n denotes a node labelled with some numeral constant $i \in \mathbb{N}$ then $t \cdot n \cdot i_n$ is also a traversal where i_n denotes the value-leaf of m corresponding to the value $i \in \mathbb{N}$.

- (Succ) If $t \cdot \mathtt{succ}$ is a traversal and λ denotes the only child node of $\mathtt{succ}$ then $t \cdot \mathtt{succ} \cdot \overset{1}{\lambda}$ is also a traversal.

- (Succ′) If $t_1 \cdot \mathtt{succ} \cdot \overset{1}{\lambda} \cdot t_2 \cdot i_\lambda$ is a traversal for $i \in \mathbb{N}$ then $t_1 \cdot \mathtt{succ} \cdot \overset{1}{\lambda} \cdot t_2 \cdot i_\lambda \cdot (i+1)_{\mathtt{succ}}$ is also a traversal.

- The rules for $\mathtt{pred}$ are defined similarly to (Succ) and (Succ′).

The conditional operator is implemented as follows. (We recall that a $\mathtt{cond}$-node in the computation tree has three children nodes numbered from 1 to 3 corresponding to the three parameters of the conditional operator.)

- (Cond-If) If $t_1 \cdot \mathtt{cond}$ is a traversal and λ denotes the first child of $\mathtt{cond}$ then $t_1 \cdot \mathtt{cond} \cdot \overset{1}{\lambda}$ is also a traversal.

- (Cond-ThenElse) If $t_1 \cdot \mathtt{cond} \cdot \overset{1}{\lambda} \cdot t_2 \cdot i_\lambda$ is a traversal then so is $t_1 \cdot \mathtt{cond} \cdot \overset{1}{\lambda} \cdot t_2 \cdot i_\lambda \cdot \overset{2+[i>0]}{\lambda}$.

- (Cond′) If $t_1 \cdot \mathtt{cond} \cdot t_2 \cdot \overset{k}{\lambda} \cdot t_3 \cdot i_\lambda$ is a traversal for $k = 2$ or $k = 3$ then the sequence $t_1 \cdot \mathtt{cond} \cdot t_2 \cdot \overset{k}{\lambda} \cdot t_3 \cdot i_\lambda \cdot i_{\mathtt{cond}}$ is also a traversal.

It is easy to verify that these traversal rules are all well-behaved. This completes the definition of traversals for PCF.

3.1.3. Revealed semantics

We recall that the definition of the syntactically-revealed semantics (Sec. 2.1, Def. 2.6) accounts for the presence of interpreted constants: For every Σ-constant $f : (A_1, \ldots, A_p, B)$ in the language, the revealed strategy of a term of the form $\lambda\overline{\xi}.f N_1 \ldots N_p$ is defined as:

$$\langle\!\langle \lambda\overline{\xi}.f N_1 \ldots N_p \rangle\!\rangle = \langle \langle\!\langle N_1 \rangle\!\rangle, \ldots, \langle\!\langle N_p \rangle\!\rangle \rangle \,{}_{9}^{0..p-1}\, [\![f]\!]$$

where $[\![f]\!]$ is the standard strategy denotation of f.

3.1.4. Correspondence theorem

We now show how to extend the Correspondence Theorem of the simply-typed lambda calculus (Theorem 2.2) to PCF.

Lemma 3.1. *Let $(S, \subseteq)$ denote the set of sets of justified sequences of nodes ordered by subset inclusion. The function $\mathcal{T}rav(\_)^{\upharpoonright\circledast} : (CT, \sqsubseteq) \rightarrow (S, \subseteq)$ is continuous.*

Proof. - *Monotonicity*: Let T and T' be two computation trees such that $T \sqsubseteq T'$ and let t be some traversal of T. Traversals ending with a node labelled $\perp$ are maximal therefore $\perp$ can only occur at the last position in a traversal. We prove the following properties:

(i) If $t = t \cdot n$ with $n \neq \perp$ then t is a traversal of T';

(ii) if $t = t_1 \cdot \perp$ then $t_1 \in \mathcal{T}rav(T')$.

(i) By induction on the length of t. It is trivial for the empty traversal. Suppose that $t = t_1 \cdot n$ is a traversal where $n \neq \perp$ and t_1 is a traversal of T'. We observe that in all traversal rules, the produced traversal is of the form $t_1 \cdot n$ where n is defined to be a child node or value-leaf of some node m occurring in t_1. Moreover, the choice of the node n only depends on the traversal t_1 (provided that the constant rules are well-behaved).

Since $T \sqsubseteq T'$, any node m occurring in t_1 belongs to T' and the children nodes of m in T also belong to the tree T'. Hence n is also present in T' and the rule used to produce the traversal t of T can be used to produce the traversal t of T'.

(ii) $\perp$ can only occur at the last position in a traversal therefore t_1 does not end with $\perp$ and by (i) we have $t_1 \in \mathcal{T}rav(T')$.

Hence we have:

$$\begin{aligned}
& \mathcal{T}rav(T)^{\upharpoonright\circledast} = \{t \upharpoonright r \mid t \in \mathcal{T}rav(T)\} \\
& \qquad = \{(t \cdot n) \upharpoonright r \mid t \cdot n \in \mathcal{T}rav(T) \wedge n \neq \perp\} \cup \{(t \cdot \perp) \upharpoonright r \mid t \cdot \perp \in \mathcal{T}rav(T)\} \\
\text{(by (i) and (ii))} & \qquad \subseteq \{(t \cdot n) \upharpoonright r \mid t \cdot n \in \mathcal{T}rav(T') \wedge n \neq \perp\} \cup \{t \upharpoonright r \mid t \in \mathcal{T}rav(T')\} \\
& \qquad = \mathcal{T}rav(T')^{\upharpoonright\circledast} \ .
\end{aligned}$$

$\blacksquare$

- *Continuity*: Let $t \in \mathcal{T}rav\left(\bigcup_{n\in\omega} T_n\right)$. We write t_i for the finite prefix of t of length i. The set of traversals is prefix-closed therefore $t_i \in \mathcal{T}rav\left(\bigcup_{n\in\omega} T_n\right)$ for every i. Since t_i has finite length we have $t_i \in \mathcal{T}rav(T_{j_i})$ for some $j_i \in \omega$. Therefore we have:

$$\begin{aligned}
t \upharpoonright r &= \Big(\bigvee_{i\in\omega} t_i\Big) \upharpoonright r && \text{(the sequence } (t_i)_{i\in\omega} \text{ converges to } t) \\
&= \bigcup_{i\in\omega} (t_i \upharpoonright r) && \text{since } \_ \upharpoonright r \text{ is continuous (Lemma 1.1)} \\
&\in \bigcup_{i\in\omega} \mathcal{T}rav(T_{j_i})^{\upharpoonright\circledast} && \text{since } t_i \in \mathcal{T}rav(T_{j_i}) \\
&\subseteq \bigcup_{i\in\omega} \mathcal{T}rav(T_i)^{\upharpoonright\circledast} && \text{since } \{j_i \mid i \in \omega\} \subseteq \omega.
\end{aligned}$$

Hence $\mathcal{T}rav(\bigcup_{n\in\omega} T_n)^{\upharpoonright\circledast} \subseteq \bigcup_{n\in\omega} \mathcal{T}rav(T_n)^{\upharpoonright\circledast}$. □

Proposition 3.1. *Let $\Gamma \vdash M : T$ be a PCF term and r be the root of $\tau(M)$. Then:*

$$\begin{array}{rl} (i) & \varphi_M(\mathcal{T}rav(M)^*) = \langle\!\langle M \rangle\!\rangle \ , \\ (ii) & \varphi_M(\mathcal{T}rav(M)^{\upharpoonright\circledast}) = [\![M]\!] \ . \end{array}$$

Proof. We first show the result for PCF$_1$: For (i), the proof is an induction identical to the proof of Theorem 2.2; we just need to complete it with the new constants cases. The cases `succ`, `pred`, `cond` and numeral constants are straightforward. Case $M = \Omega_o$: We have $\mathcal{T}rav(\Omega_o) = \mathsf{Pref}(\{\lambda \cdot \bot\})$ therefore $\mathcal{T}rav(\Omega_o)^{\upharpoonright\circledast} = \mathsf{Pref}(\{\lambda\})$ and $[\![\Omega_o]\!] = \mathsf{Pref}(\{q\})$ with $\varphi(\lambda) = q$. Hence $[\![\Omega_o]\!] = \varphi(\mathcal{T}rav(\Omega_o)^{\upharpoonright\circledast})$. (ii) is a direct consequence of (i) and the Projection Lemma 2.7.

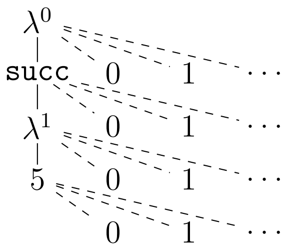
We now extend the result to PCF. Let M be a PCF term, we have:

$$\begin{aligned}
[\![M]\!] &= \bigcup_{n\in\omega} [\![M_n]\!] && \text{[11, lemma 16]} \\
&= \bigcup_{n\in\omega} \mathcal{T}rav(\tau(M_n))^{\upharpoonright\circledast} && \text{since } M_n \text{ is a PCF}_1 \text{ term} \\
&= \mathcal{T}rav(\bigcup_{n\in\omega} \tau(M_n))^{\upharpoonright\circledast} && \text{by continuity of } \mathcal{T}rav(\_)^{\upharpoonright\circledast}\text{, Lemma 3.1} \\
&= \mathcal{T}rav(\tau(M))^{\upharpoonright\circledast} && \text{by definition of } \tau(M) \\
&= \mathcal{T}rav(M)^{\upharpoonright\circledast} \ . \qquad \square
\end{aligned}$$

Hence by Corollary 2.1, φ defines a bijection from $\mathcal{T}rav(M)^{\upharpoonright\circledast}$ to $[\![M]\!]$:

$$\varphi : \mathcal{T}rav(M)^{\upharpoonright\circledast} \xrightarrow{\cong} [\![M]\!] \ .$$

Example 3.2 (Successor operator)**.** Consider the term $M = \texttt{succ}\ 5$ whose computation tree is represented below. Vertices attached to their parent node with a dashed line represent the value-leaves.



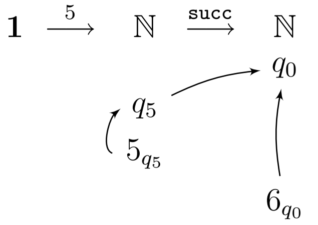
The following sequence of nodes is a traversal of $\tau(M)$:

$$t = \lambda^0 \cdot \texttt{succ} \cdot \lambda^1 \cdot 5 \cdot 5_5 \cdot 5_{\lambda^1} \cdot 6_{\texttt{succ}} \cdot 6_{\lambda^0} \ .$$

The subsequences t^* and $t \upharpoonright r$ are given by:

$$t^* = \lambda^0 \cdot \lambda^1 \cdot 5_{\lambda^1} \cdot 6_{\lambda^0} \qquad \text{and} \qquad t \upharpoonright r = \lambda^0 \cdot 6_{\lambda^0} \ .$$

The sequence $\varphi(t^*) = q_0 \cdot q_5 \cdot 5_{q_5} \cdot 5_{q_0}$ where q_0 and q_5 both denote the root of the flat arena over $\mathbb{N}$, corresponds to a play of the syntactically-revealed semantics. The sequence $\varphi(t \upharpoonright r) = q_0 \cdot 5_{q_0}$ corresponds to a play of the standard semantics. The interaction play $\varphi(t^*)$ is represented below:


$\mathbf{1} \xrightarrow{5} \mathbb{N} \xrightarrow{\texttt{succ}} \mathbb{N}$
q_0
q_5
5_{q_5}
6_{q_0}

Example 3.3 (Conditional)**.**

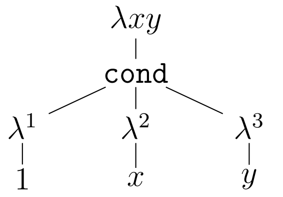


Figure 5: Computation tree of the term $\lambda xy.\texttt{cond}\ 1\ x\ y$.

Take the computation tree represented on the left (value-leaves are not shown). For every value $v \in \mathcal{D}$ we have the following traversal:

$$t = \lambda xy \cdot \texttt{cond} \cdot \lambda^1 \cdot 1 \cdot 1_1 \cdot \lambda^3 \cdot y \cdot v_y \cdot v_{\lambda^3} \cdot v_{\texttt{cond}} \cdot v_{\lambda xy} \ .$$

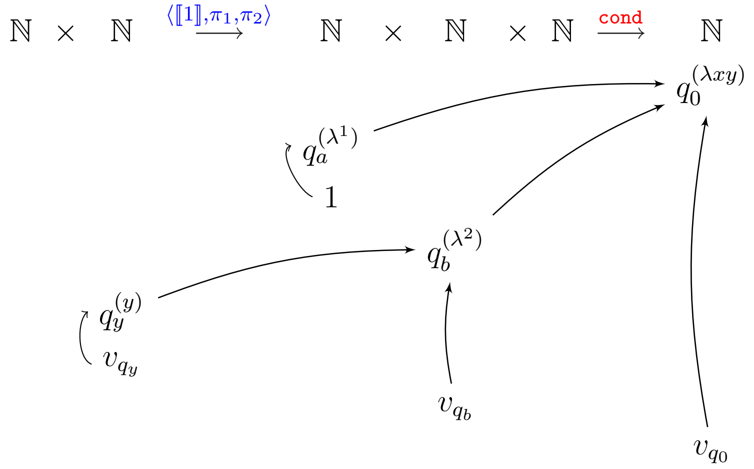
The subsequence t^* is given by:

$$t^* = \lambda xy \cdot \lambda^1 \cdot \lambda^3 \cdot y \cdot v_y \cdot v_{\lambda^3} \cdot v_{\lambda xy}$$

and the core of $t \upharpoonright \circledast$ is given by:

$$t \upharpoonright \circledast = \lambda xy \cdot y \cdot v_y \cdot v_{\lambda xy} \ .$$

By the correspondence theorem, the sequence of moves $\varphi(t^*)$ (represented in the diagram below) is a play of the revealed semantics, and the sequence $\varphi(t \upharpoonright \circledast)$ is the play of the standard semantics obtained by hiding the internal moves from $\varphi(t^*)$.


$\mathbb{N} \times \mathbb{N} \xrightarrow{\langle [\![1]\!], \pi_1, \pi_2 \rangle} \mathbb{N} \times \mathbb{N} \times \mathbb{N} \xrightarrow{\mathtt{cond}} \mathbb{N}$
$q_0^{(\lambda xy)}$
$q_a^{(\lambda^1)}$
1
$q_b^{(\lambda^2)}$
$q_y^{(y)}$
v_{q_y}
v_{q_b}
v_{q_0}

Remark 3.1 (Finite representation of the computation tree) Due to the presence of the Y combinator, computation trees of PCF terms are potentially infinite. It is possible to give an equivalent finite representation using computation *graphs*. We briefly describe here how this can be achieved.

The idea is to replace Y-recursion by μ-recursion: each subterm of the form $Y_A\ M$ is replaced by $\mu f.Mf$ for f fresh. The computation graph is then obtained from the eta-long normal form of the term. The abstraction nodes are generalized to take into account μ binders: an abstraction node is of the form $\lambda\!\!\lambda\overline{x}$ where $\overline{x}$ is a list of μ-bound and λ-bound variables where the μ-bound variables are written in parenthesis to distinguish them from λ-bound variables.

The computation graph of $Y_A(\lambda f^A.M)$ for $A = (A_1, \ldots, A_n, o)$ is then obtained from the syntax representation of $\lambda\!\!\lambda(f)x_1 \ldots x_n.\lceil M \rceil$ by adding a child edge going from each occurrence of the recursion variable f in $\lceil M \rceil$ to the root $\lambda\!\!\lambda(f)x_1 \ldots x_n$.

This presentation also accounts for ground type recursion, for instance the computation graph of the `while` operator of Idealized Algol defined as $\mathtt{while}\ C\ \mathtt{do}\ I \equiv Y(\lambda f.\mathtt{cond}\ C\ \mathtt{skip}\ (\mathtt{seq}\ I f))$ is given by the graph of $\lambda\!\!\lambda(f).\mathtt{cond}\ C\ \mathtt{skip}\ (\mathtt{seq}\ I f)$.

The order of a generalized abstraction node is still defined as the order of the term represented by the subtree rooted at this node. In other word, the order of $\lambda\!\!\lambda\overline{x}$ is defined as the order of $\lambda\!\!\lambda\overline{y}$ where $\overline{y}$ is the sublist of $\overline{x}$ obtained by removing all the recursion variables (those in parenthesis).

Bound variables in a generalized abstraction node $\lambda\!\!\lambda\overline{x}$ are numbered as follows: The i^{th} λ-bound variable in $\overline{x}$ is denoted by i and the i^{th} recursion variable is denoted by (i). The links in a justified sequence of nodes are labelled accordingly.

All the traversal rules are kept unmodified. The recursion variables in the λ-nodes are ignored by the rules since such variables are numbered differently from standard variables. In particular, the (Var) rule only applies to non-recursion variables. We only need to add a rule to handle recursion variable: whenever a traversal meets a recursion variable f in the subgraph $\tau(F)$, the traversal jumps to the root of the graph:

($\mathsf{Var}_{\mathsf{rec}}$) If $t' \cdot n \cdot \overset{\overset{(i)}{\curvearrowleft}}{\lambda\!\!\lambda\overline{x} \ldots f_i}$ is a traversal for some *recursion* variable f_i bound by $\lambda\!\!\lambda\overline{x}$ then so is $t' \cdot n \cdot \overset{\overset{(i)}{\curvearrowleft}}{\lambda\!\!\lambda\overline{x} \ldots f_i} \overset{\curvearrowleft}{\cdot} \lambda\!\!\lambda\overline{x}$.

The enabling relation $\vdash$ needs to be adapted to allow the root to be justified by a recursion variable (as if it was a child of the recursion variable). Since a traversal can now visit the root multiple times, the definition of the traversal core also needs to be adapted: instead of keeping all the nodes hereditarily enabled by the root, it keeps the nodes that are hereditarily justified by an occurrence of the root with no justifier. The definition of the mapping ψ from nodes to moves remains consistent with this notion of computation tree, and the game-semantic correspondence follows.

3.2. Idealized algol

We now consider the language Idealized Algol. The general idea is the same as for PCF, however there are some difficulties caused by the presence of the two base types `var` and `com`. We briefly sketch how our framework can be adapted to IA without going into the details of the proof of the Correspondence theorem.

Computation hypertree

The languages that we have considered up to now (lambda calculus and PCF) do not have product types. Consequently, the arenas involved in their game model only have a single initial move at most, and can therefore be regarded as trees. This property permitted us to represent the game denotation of term directly on some representation of its abstract syntax tree—the computation tree. This cannot be done in IA because the base type `var` is given by the product $\texttt{com}^{\omega} \times \texttt{exp}$ which corresponding game has infinitely many initial moves, whereas the AST of the term is a tree and therefore has a single root.

The overcome this mismatch, we use hypertrees instead of trees. These hypertrees provide an abstract representation of the syntax of the term in which some nodes, called *generalized lambda nodes*, are themselves constituted of (possibly infinitely many) subnodes. Furthermore each individual subnode can have its own children nodes.

Notations 3.1 For every type μ, we write $\mathcal{D}_{\mu}$ to denote the set of values of type μ. We have $\mathcal{D}_{\texttt{exp}} = \mathbb{N}$, $\mathcal{D}_{\texttt{com}} = \{\texttt{done}\}$ and $\mathcal{D}_{\texttt{var}} = \mathcal{D}_{\texttt{exp}} \cup \mathcal{D}_{\texttt{com}}$. For every node n, if $\kappa(n)$ is of type $(A_1, \ldots A_n, B)$, we call B the *return type of* n. The set of value-leaves of a node n is given by $\mathcal{D}_{\mu}$ where μ is the return type of n. For conciseness, when representing value-leaves in the hypertree, we merge all the value-leaves of a given node of type μ into a single leaf labelled $\mathcal{D}_{\mu}$. For instance we use the tree notation

$$\begin{array}{c} n \\ | \\ \mathcal{D}_{\texttt{exp}} \end{array} \quad \text{to mean} \quad \begin{array}{c} n \\ / \ / \ \backslash \ \backslash \\ 0 \ \ 1 \ \ 2 \ \cdots \end{array} \qquad \text{and} \quad \begin{array}{c} n \\ | \\ \mathcal{D}_{\texttt{com}} \end{array} \quad \text{for} \quad \begin{array}{c} n \\ | \\ \texttt{done} \end{array} .$$

The computation hypertree of a term with return type `var` has infinitely many root lambda-nodes which are merged all-together into a single node called a ***generalized lambda-node***. The subnodes of a generalized lambda nodes are labelled λ^r, λ^{w_0}, λ^{w_1},

λ^{w_2}, ... Suppose that M is a term of type `var`, then the computation hypertree for $\lambda\overline{\xi}.M$ is obtained by relabelling the root λ-nodes λ^r, λ^{w_0}, λ^{w_1}, λ^{w_2}, ... into $\lambda^r\overline{\xi}$, $\lambda^{w_0}\overline{\xi}$, $\lambda^{w_1}\overline{\xi}$, $\lambda^{w_2}\overline{\xi}$, ... For a term M of type `exp` or `com`, the computation hypertree for $\lambda\overline{\xi}.M$ is computed the same way as for computation trees of lambda-terms.

Table 4 defines the computation hypertree for each term-construct of IA. A generalized lambda node is represented by a frame surrounding its subnodes (2^{nd} and 6^{th} row in the table).

Enabling relation, justified sequence

The notion of binder is redefined as follows: Given a variable node x, the binder of x is the first node occurring in the path to the root that is a lambda node $\lambda\overline{x}$ with $x \in \overline{x}$ or a block-declaration node `new` x.

The enabling relation and the definition of justified sequence is modified so that occurrences of block-allocated variables are justified by nodes of type `new` x instead of lambda nodes.

Children numbering convention

Let p be a node and suppose that its i^{th} child n has return type `var`. Then n is a generalized lambda-node with subnodes $\lambda^r\overline{\xi}$, $\lambda^{w_0}\overline{\xi}$, ... From the point of view of the parent node p, these subnodes are referenced as "$i.\alpha$" where $0 \leq i \leq arity(p)$ and $\alpha \in \{r\} \cup \{w_k \mid k \in \mathbb{N}\}$. For instance $i.r$ refers to the root labelled $\lambda^r\overline{\xi}$ of the i^{th} child of p, and $i.w_k$ refers to the root labelled $\lambda^{w_k}\overline{\xi}$.

Traversals

The following new rules are added on top of those defined in Sec. 1.3:

- *Application rules*

  The rule (app) is now split up in three rules $(\mathsf{app}_{\mathtt{exp}})$, $(\mathsf{app}_{\mathtt{com}})$ and $(\mathsf{app}_{\mathtt{var}})$ corresponding to traversals ending with an @-node of return type `exp`, `com` and `var` respectively. The rules $(\mathsf{app}_{\mathtt{exp}})$, $(\mathsf{app}_{\mathtt{com}})$ are defined identically to (app) (see Sec. 1.3). The rule $(\mathsf{app}_{\mathtt{var}})$ is

  $$(\mathsf{app}_{\mathtt{var}})\ t \cdot \lambda^k\overline{\xi} \overset{0}{\curvearrowleft} @ \in \mathcal{T}rav \text{ and } k \in \{r, w_0, w_1, \ldots\} \implies t \cdot \lambda^k\overline{\xi} \overset{0}{\curvearrowleft} @ \overset{0.k}{\curvearrowleft} \lambda^k\overline{\eta} \in \mathcal{T}rav \ .$$

- *Input-variable rules*

  We define the rules $(\mathsf{InputVal}^{\$})$ for \$ ranging in $\{\mathtt{com}, \mathtt{var}, \mathtt{exp}\}$. For `com` and `exp`, the rules are defined identically to $(\mathsf{InputVal})$ of Sec. 1.3. The `var` case is implemented by two rules:

  $$(\mathsf{InputValue}_{\mathsf{r}}^{\mathsf{var}})\ \frac{t_1 \cdot \lambda^r\overline{\xi} \cdot x \cdot t_2 \in \mathcal{T}rav}{t_1 \cdot x \cdot t_2 \overset{v}{\curvearrowleft} v_x \in \mathcal{T}rav}\ x \text{ pending node } \wedge\ x \in N^{\circledast\vdash}_{\mathsf{var}} \wedge x : \mathtt{var},\ v \in \mathcal{D}\ .$$

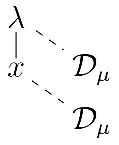
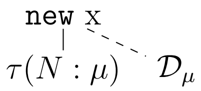
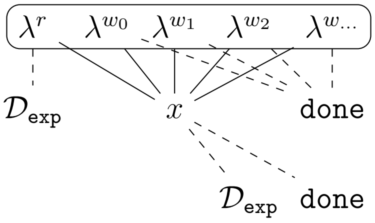
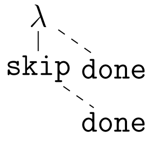
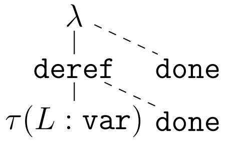
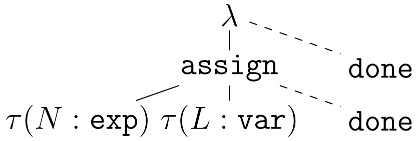
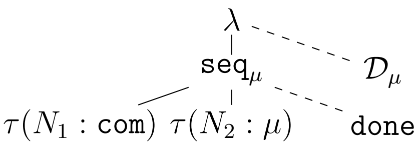
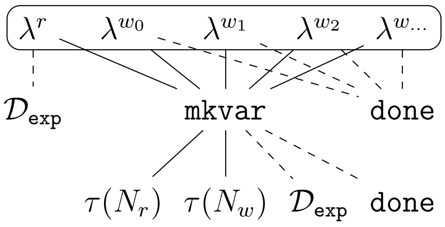
M	$\tau(M)$
$\mathtt{x} : \mu$ $\mu \in \{\mathtt{com}, \mathtt{exp}\}$	
$\mathtt{new}\ x\ \mathtt{in}\ N : \mu$ $\mu \in \{\mathtt{com}, \mathtt{exp}\}$	
$\mathtt{x} : \mathtt{var}$	 λ^r λ^{w_0} λ^{w_1} λ^{w_2} $\lambda^{w_{\dots}}$ $\mathcal{D}_{\mathtt{exp}}$ x done $\mathcal{D}_{\mathtt{exp}}$ done
$\mathtt{skip}$: $\mathtt{com}$	
$\mathtt{deref}\ L : \mathtt{exp}$	 λ deref done $\tau(L : \mathtt{var})$ done
$\mathtt{assign}\ L\ N : \mathtt{com}$	 λ assign done $\tau(N : \mathtt{exp})$ $\tau(L : \mathtt{var})$ done
$\mathtt{seq}_\mu\ \ N_1\ \ N_2\ \ :$ $\mathtt{com}$ $\mu \in \{\mathtt{exp}, \mathtt{com}\}$	 λ $\mathtt{seq}_\mu$ $\mathcal{D}_\mu$ $\tau(N_1 : \mathtt{com})$ $\tau(N_2 : \mu)$ done
$\mathtt{mkvar}\ N_w\ N_r : \mathtt{var}$	 λ^r λ^{w_0} λ^{w_1} λ^{w_2} $\lambda^{w_{\dots}}$ $\mathcal{D}_{\mathtt{exp}}$ mkvar done $\tau(N_r)$ $\tau(N_w)$ $\mathcal{D}_{\mathtt{exp}}$ done

Table 4: Computation hypertrees of IA constructs.

$$(\mathsf{InputValue}^{\mathtt{var}}_{\mathsf{w}})\ \frac{t_1 \cdot \lambda^w \overline{\xi} \cdot x \cdot t_2 \in \mathcal{T}rav}{t_1 \cdot x \cdot t_2 \cdot \overset{v}{\mathtt{done}_x} \in \mathcal{T}rav}\ x \text{ pending node } \wedge\ x \in N^{\circledast\vdash}_{\mathsf{var}} \wedge x : \mathtt{var}\ .$$

- *IA constants rules*

  The rules for the constants of IA are given in Table 5. These rules for `new` are purely structural, they are defined similarly to $(\mathsf{app}_{\mathsf{exp}})$, $(\mathsf{app}_{\mathsf{com}})$ and $(\mathsf{app}_{\mathsf{done}})$.

  The rules from Table 5 do not suffice to model `mkvar` however. We need to specify what happens when reaching a variable node that is hereditarily justified by the constant `mkvar`. Take for instance the term `assign` (`mkvar` $(\lambda x.M) N)7$. The rule (mkvar''_w) permits one to pass the node `mkvar` and to continue with the traversal of the computation tree of $\lambda x.M$, which may subsequently lead to some occurrence of x. The behaviour of the traversal at this point is specified by the traversal rules defined in the next paragraph.

- *Variable rules*

  Let x be an internal variable node. Then by definition it is either hereditarily justified by an @-node or by a Σ-constant node.

  - Suppose that x's binder is a lambda-node $\lambda\overline{x}$ and $x \in N^{@\vdash}$.

    This case is a generalization of the rule (Var) (Sec. 1.3). The only difference is that for variables of type `var`, the lambda nodes preceding x in the traversal determines the lambda-node that is visited next:

    $$(\mathsf{Var}_{\mathtt{var}})\frac{t \cdot n \cdot \lambda\overline{x} \ldots \lambda^{\alpha} x_i \cdot \overset{i}{x_i} \in \mathcal{T}rav}{t \cdot n \cdot \lambda\overline{x} \ldots \lambda^{\alpha} x_i \cdot \overset{i}{x_i} \cdot \overset{i.\alpha}{\lambda\overline{\eta_i}} \in \mathcal{T}rav}\quad x_i \in N^{@\vdash}_{\mathsf{var}} \wedge \alpha \in \{r\} \cup \{w_i \mid i \in \mathbb{N}\}\ .$$

  - Suppose that x's binder is a lambda-node and $x \in N^{N_\Sigma\vdash}$. Then x's binder is necessarily the second child of a `mkvar`-node (since `mkvar` is the only constant of order greater than 0).

    $$(\mathtt{mkvar}\text{-}\mathsf{Var})\frac{t \cdot \lambda^{w_k}\overline{\xi} \cdot \mathtt{mkvar} \cdot \lambda x \cdot t_2 \cdot x \in \mathcal{T}rav}{t \cdot \lambda^{w_k}\overline{\xi} \cdot \mathtt{mkvar} \cdot \lambda x \cdot t_2 \cdot x \cdot k_x \in \mathcal{T}rav}\ .$$

  - Suppose that x is a block-allocated variable.

    Given a block-declaration `new` x, we call *assignment of* x any segment of traversal of the form $\lambda^{w_k}\overline{\xi} \cdot x$ for some $k \in \mathcal{D}_{\mathsf{exp}}$ and occurrence x of a node bound by `new` x. We call k the *value assigned* to x.

    $$(\mathtt{new}\text{-}\mathsf{Var}_{\mathsf{w}})\ \frac{t \cdot \lambda^{w_k}\overline{\xi} \cdot x \in \mathcal{T}rav}{t \cdot \lambda^{w_k}\overline{\xi} \cdot x \cdot \mathtt{done}_x \in \mathcal{T}rav}\ x \in N^{\mathtt{new}\vdash}_{\mathsf{var}}\ .$$

$$(\mathsf{deref})\,\frac{t \cdot \mathtt{deref} \in \mathcal{T}rav}{t \cdot \mathtt{deref} \cdot \overset{1.r}{n} \in \mathcal{T}rav} \qquad (\mathsf{deref}')\,\frac{t \cdot \mathtt{deref} \cdot n \cdot t_2 \cdot v_n \in \mathcal{T}rav}{t \cdot \mathtt{deref} \cdot n \cdot t_2 \cdot v_n \cdot v_{\mathtt{deref}} \in \mathcal{T}rav}$$

$$(\mathsf{assign})\,\frac{t \cdot \mathtt{assign} \in \mathcal{T}rav}{t \cdot \mathtt{assign} \cdot \overset{1}{\lambda} \in \mathcal{T}rav} \qquad (\mathsf{assign}')\,\frac{t \cdot \mathtt{assign} \cdot \lambda \cdot t_2 \cdot v_\lambda \in \mathcal{T}rav}{t \cdot \mathtt{assign} \cdot \lambda \cdot t_2 \cdot v_\lambda \cdot \overset{2.w_n}{\lambda\overline{\eta}} \in \mathcal{T}rav}$$

$$(\mathsf{assign}'')\,\frac{t \cdot \mathtt{assign} \cdot t_2 \cdot \overset{2.w_k}{\lambda\overline{\eta}} \cdot t_3 \cdot \mathtt{done}_{\lambda\overline{\eta}} \in \mathcal{T}rav}{t \cdot \mathtt{assign} \cdot t_2 \cdot \lambda\overline{\eta} \cdot t_3 \cdot \mathtt{done}_{\lambda\overline{\eta}} \cdot \mathtt{done}_{\mathtt{assign}} \in \mathcal{T}rav}$$

$$(\mathsf{seq})\,\frac{t \cdot \mathtt{seq} \in \mathcal{T}rav}{t \cdot \mathtt{seq} \cdot \overset{1}{n} \in \mathcal{T}rav} \qquad (\mathsf{seq}')\,\frac{t \cdot \mathtt{seq} \cdot n \cdot t_2 \cdot v_n \in \mathcal{T}rav}{t \cdot \mathtt{seq} \cdot n \cdot t_2 \cdot v_n \cdot \overset{2}{m} \in \mathcal{T}rav}$$

$$(\mathsf{seq}'')\,\frac{t \cdot \mathtt{seq} \cdot t_2 \cdot \overset{2}{m} \cdot t_3 \cdot v_m \in \mathcal{T}rav}{t \cdot \mathtt{seq} \cdot t_2 \cdot m \cdot t_3 \cdot v_m \cdot v_{\mathtt{seq}} \in \mathcal{T}rav}$$

$$(\mathsf{mkvar_r})\,\frac{t \cdot \lambda^r\overline{\xi} \cdot \mathtt{mkvar} \in \mathcal{T}rav}{t \cdot \lambda^r\overline{\xi} \cdot \mathtt{mkvar} \cdot \overset{1}{\lambda} \in \mathcal{T}rav} \qquad (\mathsf{mkvar_r'})\,\frac{t \cdot \mathtt{mkvar} \cdot \lambda \cdot t_2 \cdot v_\lambda \in \mathcal{T}rav}{t \cdot \mathtt{mkvar} \cdot \lambda \cdot t_2 \cdot v_\lambda \cdot v_{\mathtt{mkvar}} \in \mathcal{T}rav}$$

$$(\mathsf{mkvar_w})\,\frac{t \cdot \lambda^{w_k}\overline{\xi} \cdot \mathtt{mkvar} \in \mathcal{T}rav}{t \cdot \lambda^{w_k}\overline{\xi} \cdot \mathtt{mkvar} \cdot \overset{2}{\lambda\overline{\eta}} \in \mathcal{T}rav}$$

$$(\mathsf{mkvar_w''})\,\frac{t \cdot \lambda^{w_k}\overline{\xi} \cdot \mathtt{mkvar} \cdot \lambda\overline{\eta} \cdot t_2 \cdot \mathtt{done}_{\lambda\overline{\eta}} \in \mathcal{T}rav}{t \cdot \lambda^{w_k}\overline{\xi} \cdot \mathtt{mkvar} \cdot \lambda\overline{\eta} \cdot t_2 \cdot \mathtt{done}_{\lambda\overline{\eta}} \cdot \mathtt{done}_{\mathtt{mkvar}} \in \mathcal{T}rav}$$

where v denotes some value from $\mathcal{D}$.

Table 5: Traversal rules for IA constants.

$$(\mathsf{new\text{-}Var_r})\ \frac{t_1 \cdot \texttt{new}\ x \cdot t_2 \cdot \lambda^r \overline{\xi} \cdot x \in \mathcal{T}rav}{t_1 \cdot \texttt{new}\ x \cdot t_2 \cdot \lambda^r \overline{\xi} \cdot x \cdot k_x \in \mathcal{T}rav}$$

where $k \in \mathbb{N}$ is the last value assigned to x in t_2, or 0 if there is no such assignment.

3.2.1. Game semantics correspondence

The properties that we proved for computation trees and traversals of the lambda calculus with constants can easily be lifted to computation hypertrees of IA. In particular:

- Constant traversal rules are well-behaved (for order-0 and order-1 constants, this is a consequence of Lemma 1.3; for `mkvar` and `new` this can be easily verified);
- P-view of traversals are paths in the computation hypertrees;
- For beta-normal terms, the P-view of a traversal core is the core of the P-view (Lemma 1.20, and the O-view of a traversal is the O-view of its core (Lemma 1.18);
- There is a mapping from vertices of the computation hypertrees to moves in the interaction game semantics;
- There is a correspondence between traversals of the computation tree and plays in interaction game semantics;
- Consequently, there is a correspondence between the standard game semantics and the set of justified sequences of nodes $\mathcal{T}rav(M)^{\upharpoonright \circledast}$.

4. Conclusion and related works

We have given a new presentation of game semantics based on the theory of traversals. This presentation is concrete in the sense that the traversal denotation carries syntactic information about the term. We established the connection with the Hyland-Ong game semantics by means of a Correspondence Theorem: The set of traversals of a term is isomorphic to the revealed game denotation of the term.

One advantage of the traversal theory lies in its ability to compute beta-reduction locally without having to perform term substitution. As observed by Danos et al. [14], “the interaction processes at work in game semantics are implementations of *linear head reduction*”. In that regards, the traversals theory can be viewed as a rule-based implementation of the *head linear reduction strategy* [15]. Although the idea of evaluating a term using this strategy is not new, our presentation has several advantages and novelties. Firstly, the Correspondence theorem establishes a clear correspondence with game semantics, namely that traversals gives you a way to compute precisely the revealed game denotation of a term. To our knowledge, although the notion of revealed game semantics was mentioned in previous works [9], it was never formally defined. Secondly, our presentation highlights more clearly the algorithmic aspect of game semantics. The rule-based definition of traversals lends itself well to automaton characterization. An example is the characterization of higher-order recursion schemes by *collapsible higher-order pushdown automata* [16].

Another advantage of the traversal theory is its efficiency for effectively computing the game-semantic denotation of a term. The traditional approach is to proceed bottom-up by appealing to compositionality. Although the compositional nature of game semantics is very attractive from a theoretical point of view, in practice it is not efficient to compute a denotation in that way. Indeed, for every subterm one has to compute all the possible ways to interact with the environment for that subterm. But this denotation is then immediately composed with another subterm, which determines part of the environment's behaviour, thus it was wasteful in the first place to consider all the possible behaviours of the environment for the first term.

The traversal theory follows a top-down approach which means that we only consider possible behaviour of the outermost environment. Moreover contrary to the compositional method, there is no need to implement any composition mechanism: the set of traversals is just obtained by following the traversal rules; the hiding of internal nodes is postponed until the end.

The lazy nature of the traversal evaluation provides a further source of efficiency: the beta-redexes are computed "on-demand" instead of performing a global substitution.

Last but not least, we believe that the syntactic correspondence between game semantics and its syntax is of pedagogical interest. Game semantics is often found hard to understand due to some obscure technical definitions. A concrete presentation such as the one given by the traversal theory, allows one to explain game-semantic concepts (such as P-view, innocence, visibility) from a programmer point of view. I have implemented a prototype tool using the F# programming language, which among other things, illustrates the theory of traversals [17]. The tool lets the user "play" the game induced by a simply-typed term (or a higher-order grammar) just by choosing nodes from the computation tree. As the game unfolds the corresponding traversal is shown. A calculator mode allows the user to perform various operations on justified sequences. (All the examples from this chapter were generated using this tool.)

Further correspondences

The traversal theory that we have presented here captures the lambda calculus fragment of the game model of call-by-name programming languages such as PCF and Idealized Algol. A natural way to extend this work would be to define the appropriate notion of traversal corresponding to the call-by-value games [18, 19].

Applications

The theory of traversal has applications in several domains of research:

Verification

The theory of traversal was originally introduced by Ong to study the decidability of MSO theories of infinite trees generated by higher-order recursion schemes. This result was recently used by Kobayashi to develop a novel framework for verification of temporal properties of higher-order functional programs [20].

Another promising application of the traversal theory concerns the study of reachability problems. In its most general form, the reachability problem for programming languages can informally be stated as: *Given a term M and coloured subterm N, is there a context $C[-]$ such that evaluating $C[M]$ involves the evaluation of N?*. In an ongoing research project, Luke Ong and Nikos Tzevelekos make use of the traversal theory to study several variations of the reachability problem for finitary PCF [21].

Automata theory

The traversal theory has led to an equi-expressivity result between a certain type of automaton device called *collapsible pushdown automaton* (CPDA) and higher-order recursion schemes (HORS) [16]. One direction of this proof relies on the traversal theory: for a given HORS, a CPDA is constructed that computes precisely the set of traversals over the computation tree of the HORS.

A crucial point in this encoding is that structures generated by recursion schemes are of ground type. Because such structures do not interact with the environment, their game-semantic denotation is relatively simple. In particular, the O-view of the traversal does not play any role in the traversal rules and therefore the automaton does not need to calculate or remember it. A natural extension would be a similar automata-characterization for *higher-order* structures such as simply-typed terms.

Pattern matching

Higher-order matching is the following problem: Given an equation $M = N$ where M is an open simply-typed term and N is a closed simply-typed term, is there a solution substitution θ such that $M\theta$ and N have the same $\beta\eta$-normal form? Huet conjectured in 1976 that this problem is decidable [22]. It was proved only recently by Colin Stirling that it is indeed the case [23].

Stirling's argument is based on a game-theoretic argument, namely the concept of tree-checking games. As pointed out by Luke Ong, Stirling's games are closely related to the innocent game semantics framework provided by the theory of traversals. The concept of traversals is implicitly present in Stirling's proof (though the notion of justification pointers is replaced by "iteratively defined look-up tables").

Analyzing syntactic constraints

The connection between syntax and semantics provided by the traversal theory enables us to analyze the effect of a given syntactic constraint on the game model. The next chapter is an example of such an application: By making simple observations about the computation tree of safe terms, the Correspondence Theorem allows us to show that their strategy denotations are of a particular kind: Their plays satisfy a certain property called *incremental justification.*

References

[1] C.-H. L. Ong, On model-checking trees generated by higher-order recursion schemes, in: Proceedings of IEEE Symposium on Logic in Computer Science., Computer Society Press, 2006, pp. 81–90, extended abstract.

[2] V. Danos, L. Regnier, Local and asynchronous beta-reduction (an analysis of girard's execution formula), in: M. Vardi (Ed.), Proceedings of the Eighth Annual IEEE Symp. on Logic in Computer Science, LICS 1993, IEEE Computer Society Press, 1993, pp. 296–306.

[3] A. Asperti, V. Danos, C. Laneve, L. Regnier, Paths in the lambda-calculus, in: LICS, IEEE Computer Society, 1994, pp. 426–436.

[4] J. Lamping, An algorithm for optimal lambda calculus reduction, in: POPL '90: Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, ACM Press, New York, NY, USA, 1990, pp. 16–30. `doi:http://doi.acm.org/10.1145/96709.96711`.

[5] C.-H. L. Ong, On model-checking trees generated by higher-order recursion schemes (technical report), preprint, 42 pp (2006).
URL http://users.comlab.ox.ac.uk/luke.ong/publications/ntrees.pdf

[6] J. M. E. Hyland, C.-H. L. Ong, On full abstraction for PCF: I, II, and III, Information and Computation 163 (2) (2000) 285–408. `doi:http://dx.doi.org/10.1006/inco.2000.2917`.

[7] R. Harmer, Innocent game semantics, course notes (November 2005).

[8] S. Abramsky, P. Malacaria, R. Jagadeesan, Full abstraction for PCF, in: Theoretical Aspects of Computer Software, 1994, pp. 1–15.
URL citeseer.ist.psu.edu/abramsky95full.html

[9] W. Greenland, Game semantics for region analysis, Ph.D. thesis, University of Oxford (2004).

[10] A. Dimovski, D. R. Ghica, R. Lazic, Data-abstraction refinement: A game semantic approach, in: C. Hankin, I. Siveroni (Eds.), SAS, Vol. 3672 of Lecture Notes in Computer Science, Springer, 2005, pp. 102–117.
URL http://dblp.uni-trier.de/db/conf/sas/sas2005.html#DimovskiGL05

[11] S. Abramsky, G. McCusker, Game semantics, in: H. Schwichtenberg, U. Berger (Eds.), Logic and Computation: Proceedings of the 1997 Marktoberdorf Summer School, Springer-Verlag, 1998, pp. 1–56, lecture notes.

[12] G. McCusker, Games and full abstraction for FPC, in: E. M. Clarke (Ed.), Proceedings of the Eleventh Annual IEEE Symp. on Logic in Computer Science, LICS 1996, IEEE Computer Society Press, 1996, pp. 174–183.

[13] T. Knapik, D. Niwiński, P. Urzyczyn, Higher-order pushdown trees are easy, in: FOSSACS'02, Springer, 2002, pp. 205–222, lNCS Vol. 2303.

[14] V. Danos, H. Herbelin, L. Regnier, Game semantics and abstract machines, in: Logic in Computer Science, 1996. LICS '96. Proceedings., Eleventh Annual IEEE Symposium on, 1996, pp. 394–405. doi:10.1109/LICS.1996.561456.

[15] V. Danos, L. Regnier, Head linear reduction, submitted for publication (2004).
URL citeseer.ist.psu.edu/danos04head.html

[16] M. Hague, A. S. Murawski, C.-H. L. Ong, O. Serre, Collapsible pushdown automata and recursive schemes, LICS (2008) 452–461.

[17] W. Blum, A tool for constructing structures generated by higher-order recursion schemes and collapsible pushdown automata, http://william.famille-blum.org/research/tools.html (2008).
URL web.comlab.ox.ac.uk/oucl/work/william.blum/

[18] G. D. Plotkin, Call-by-name, call-by-value and the lambda-calculus, Theoretical Computer Science 1 (2) (1975) 125–159. doi:10.1016/0304-3975(75)90017-1.
URL http://dx.doi.org/10.1016/0304-3975(75)90017-1

[19] S. Abramsky, G. McCusker, Call-by-value games, in: M. Nielsen, W. Thomas (Eds.), Computer Science Logic: 11th International Workshop Proceedings, Springer-Verlag, 1998.
URL citeseer.ist.psu.edu/abramsky97callbyvalue.html

[20] N. Kobayashi, Types and higher-order recursion schemes for verification of higher-order programs, submitted to the Symposium on Principles of Programming Languages (2009).

[21] C.-H. L. Ong, N. Tzevelekos, Functional reachability, work in progress.

[22] G. P. Huet, Résolution d'équations dans des langages d'ordre 1,2,...,ω, Thèse de doctorat es sciences mathématiques, Université Paris VII (Septembre 1976).

[23] C. Stirling, A game-theoretic approach to deciding higher-order matching, in: M. Bugliesi, B. Preneel, V. Sassone, I. Wegener (Eds.), ICALP (2), Vol. 4052 of Lecture Notes in Computer Science, Springer, 2006, pp. 348–359.