Notes on game semantics and safe-lambda calculus

William Blum

May 9, 2006

Oxford University Computing Laboratory

CONTENTS

Chapter 1

GAME SEMANTICS

1.1 Semantics of programming languages

Before the introduction of game semantics in the 1990s, there were many approaches to define models for programming languages that we classify into different categories. Among them there is axiomatic, operational and denotational semantics.

Operational semantics gives a meaning to a program by describing the behaviour of a machine executing the program. It is defined formally by giving a state transition system.

Axiomatic semantics defined the behaviour of the program with axioms and is used to prove program correctness by static analysis of the code of the program.

The denotational semantics approach consists in mapping a program to a mathematical structure having good properties such as compositionality. This mapping is achieved by structural induction on the syntax of the program.

In the 1990s, a new kind of semantics called game semantics has been introduced for modeling programming languages. In game semantics, the meaning of a program is given by a strategy in a two-player game. The two players are the Opponent, representing the environment, and the Proponent, representing the system.

1.1.1 Model for PCF

to develop

The problem of the Full Abstraction for PCF goes back to the 1970s.

Scott gave a model for PCF based on domain theory [Scott, 1993].

The Scott domain based model of PCF is not fully abstract, i.e. there exist two PCF terms which are observationally equivalent but their domain denotation is different. This is a consequence of the fact that the parallel-or operator defined by the following truth table is not definable as a PCF term:

p-or	$\perp$	tt	ff
$\perp$	$\perp$	tt	$\perp$
tt	tt	tt	tt
ff	$\perp$	tt	ff

The undefinability of this term can be exploited to prove that the model is not fully abstract. It is possible to create two terms that behave the same except when the parameter is a term computing p-or. Since p-or is not definable in PCF, these two terms will in fact be equivalent.

It is possible to patch PCF by adding the operator $p - or$, the resulting language "PCF+p-or" is fully-abstracted by Scott domain theoretic model [Plotkin, 1977]. However the language we are now dealing with is strictly more powerful than PCF, it has some parallel execution power that PCF has not.

Also, we may want to get rid of the undefinable elements (like p-or) by strengthening the conditions on the function used in the model (a condition stronger than strictness and continuity) but unfortunately this approach did not succeed.

Hence the problem remains: is there any fully abstract model for PCF?

Solutions to the full abstraction problem for PCF have eventually been discovered in the 1990s by three different independent research groups: Ambramsky, Jagadeesan and Malacaria [Abramsky et al., 1994], Hyland and Ong [Hyland and Ong, 2000] and Nickau. There are all based on game semantics.

1.2 Games

We introduce here the notion of game that will be used in the following section to give a model of the programming languages PCF and Idealized Algol. The definitions are taken from Abramsky and McCusker [1997], Hyland and Ong [2000], Abramsky et al. [1994].

1.2.1 Example

TO DO: Introduce the principle with a small example

1.2.2 Basic definition

The games we are interested in are two-players games. The players are named O for Opponent and P for Proponent.

The game played by O and P is constraint by something called *arena*. The arena defines the possible moves of the game. By analogy with board games, the arena represents the board and the rules that tell how players can make their moves on the board[1].

More formally, the arena can be seen as a forest of trees whose nodes are possible questions and leaves are possible answers. The arena is partitioned into two kinds of moves: the moves that can be played by P and the ones that can be played by O. A move is either a question to the other player or an answer to a question previously asked by the other player.

Each move of the game must be justified by another move that has already been played by the other player. This justification relation is induced by the edges of the forest arena. Moreover, an answer must always be justified by the question that it answers and a question is always justified by another question.

Definition 1.2.1 (Arena)**.** An arena is a structure $\langle M, \lambda, \vdash \rangle$ where:

- M is the set of possible moves;
- $(M, \vdash)$ is a forest of trees;
- $\lambda : M \rightarrow \{O, P\} \times \{Q, A\}$ is a labeling functions indicating whether a given move is a question or an answer and whether it can be played by O or by P.
  $\lambda = [\lambda^{OP}, \lambda^{QA}]$ where $\lambda^{OP} : M \rightarrow \{O, P\}$ and $\lambda^{QA} : M \rightarrow \{Q, A\}$.
  - If $\lambda^{OP}(m) = O$, we call m and O-move otherwise m is a P-move. $\lambda^{QA}(m) = Q$ indicates that m is a question otherwise m is an answer.
  - For any leaf l of the tree $(M, \vdash)$, $\lambda^{QA}(l) = A$ and for any node $n \in (M, \vdash)$, $\lambda^{QA}(n) = Q$.
- The forest of tree $(M, \vdash)$ respect the following condition:
  (e1) The roots are O-moves: for any root r of $(M, \vdash)$, $\lambda^{OP}(r) = O$.
  (e2) Answers are enabled by questions: $m \vdash n \wedge \lambda^{QA}(n) = A \Rightarrow \lambda^{QA}(m) = Q$.
  (e3) A player move must be justified by a move played by the other player: $m \vdash n \Rightarrow \lambda^{OP}(m) \neq \lambda^{OP}(n)$.

[1] In fact there is an analogy more appropriate than board games which illustrates well the notion of game that we are exposing here: dialog games. In these games one person (O) interviews another person (P) while P tries to answer the initial O-question by possibly asking O some precisions about its initial question.

For commodity we write the set $\{O, P\} \times \{Q, A\}$ as $\{OQ, OA, PQ, PA\}$. $\overline{\lambda}$ denotes the labeling function λ with the question and answer swapped. For instance:

$$\overline{\lambda(m)} = OQ \iff \lambda(m) = PQ$$

The roots of the forest of tree $(M, \vdash)$ are the *initial moves*.

Once the arena has been defined, the bases of the game are set and the players have something to play with. We now need to describe the state of the game, for that purpose we introduced *justified sequences of moves*. Sequence of moves are used to record the history of all the moves that have been played.

Definition 1.2.2 (Justified sequence of moves)**.** A justified sequence is a sequence of moves s together with an associated sequence of pointers. Any move m in the sequence that is not initial has as pointer that points to a previous move n that justifies it (i.e. $n \vdash m$).

A justified sequence has two particular subsequences which will be of particular interest later on when we introduce strategies. These subsequences are called the P-view and the O-view of the sequence. The idea is that a view describes the local context of the game. Here is the formal definition:

Definition 1.2.3 (View)**.** Given a justified sequence of moves s. We define the proponent view (P-view) noted $\ulcorner s \urcorner$ by induction:

$$\begin{aligned}
\ulcorner \epsilon \urcorner &= \epsilon & & \\
\ulcorner s \cdot m \urcorner &= \ulcorner s \urcorner \cdot\ m & & \text{if } m \text{ is a P-move} \\
\ulcorner s \cdot m \urcorner &= m & & \text{if } m \text{ is initial (O-move)} \\
\ulcorner s \cdot \overset{\curvearrowleft}{m \cdot t \cdot n} \urcorner &= \ulcorner s \urcorner \cdot \overset{\curvearrowleft}{m \cdot n} & & \text{if } n \text{ is a non initial O-move}
\end{aligned}$$

The O-view $\llcorner s \lrcorner$ is defined similarly:

$$\begin{aligned}
\llcorner \epsilon \lrcorner &= \epsilon & & \\
\llcorner s \cdot m \lrcorner &= \llcorner s \lrcorner \cdot\ m & & \text{if } m \text{ is a O-move} \\
\llcorner s \cdot \overset{\curvearrowleft}{m \cdot t \cdot n} \lrcorner &= \ulcorner s \urcorner \cdot \overset{\curvearrowleft}{m \cdot n} & & \text{if } n \text{ is a P-move}
\end{aligned}$$

In fact not all justified sequences will be of interest for the games that we will use. We call *legal position* any justified sequence verifying two additional conditions: alternation and visibility.

Definition 1.2.4 (Legal position)**.** A legal position is a justified sequence of move s respecting the following constraint:

- Alternation: For any subsequence $m \cdot n$ of s, $\lambda^{OP}(m) \neq \lambda^{OP}(n)$.
- Visibility: For any subsequence tm of s, if m is a P-move then m points to a move in $\ulcorner s \urcorner$ and if m is a O-move then m points to a move in $\llcorner s \lrcorner$.

The set of legal position of an arena A is noted L_A.

We say that a move n is hereditarily justified by a move m if there is a sequence of move $m_1, \ldots, m_q$ such that:

$$m \vdash m_1 \vdash m_2 \vdash \ldots m_q \vdash n$$

Suppose that n is an occurrence of a move in the sequence s then we note $s \upharpoonright n$ the subsequence of s containing all the moves hereditarily justified by n. Similarly, $s \upharpoonright I$ denotes the subsequence of s containing all the moves hereditarily justified by the moves in I.

Definition 1.2.5 (Game)**.** A game is a structure $\langle M, \lambda, \vdash, P \rangle$ such that

- $\langle M, \lambda, \vdash \rangle$ is an arena.

- P, called the set of valid positions, is
  - a non-empty prefix closed subset of the set of legal position
  - closed by hereditary filtering: if I is the set of initial moves of s then

$$s \in P \Rightarrow s \upharpoonright I \in P$$

1.2.3 Game construction

tensor product, implication, product

1.2.4 Strategy

During a game, a player may have several choices for his next move. To decide which moves to make, the player refers to the state of the game. This state is given by the position of the game, in other words the history of all the moves already played.

A strategy is therefore based on the history of the game.

Definition 1.2.6 (Strategy)**.** A strategy for player P on a given game $\langle M, \lambda, \vdash, P\rangle$ is a non-empty set of even-length positions from P such that:

1. $sab \in \sigma \Rightarrow s \in \sigma$
2. $sab, sac \in \sigma \Rightarrow b = c$

The idea is that the presence of the even-length sequence sab in σ tells the player P that whenever the game is in position sa it must play the move b. The second condition in the definition requires that this choice of move is deterministic (i.e. there is a function f from the set of odd length position to the set of moves M such that $f(sa) = b$). The first condition ensures that the strategy σ consider only position that the strategy itself could have led to.

Composition of strategy

Strategy constraints:

innocence

well-bracketing We take the definition given in Abramsky and McCusker [1997] where the well-bracketing condition is a condition on P-answers only and where there is no constraint on O-answers. In fact this choice is harmless and the full-abstraction results that we will state in the next section still hold if we assume well-bracketing of O-answers.

1.2.5 Categorical interpretation of games

categories $\mathcal{C}$, $\mathcal{C}_i$, $\mathcal{C}_b$, $\mathcal{C}_{ib}$

1.2.6 Arena of order at most 2

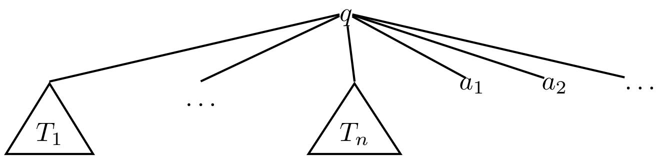
The height of the arena is the length of the longest sequence of moves $m_1 \ldots m_h$ in M such that $m_1 \vdash m_2 \vdash \ldots \vdash m_h$.

The order of an arena $\langle M, \lambda, \vdash\rangle$ is defined to be $h - 2$ where h is the height of the forest of trees $(M, \vdash)$.

Lemma 1.2.7 (Pointers are superfluous up to order 2)**.** *Let A be the arena of order at most 2. Let s be a justified sequence of moves in the arena A satisfying alternation, visibility and well-bracketing then the pointers of the sequence s can be reconstructed uniquely.*

Proof. In the graphic representation of the arena, we display the sub-arena by decreasing order of sub-arena order. It is safe to do so since in the definition of the forest of tree of an arena, the children nodes are not ordered.

Let A be an arena of order 2. We assume that A has only one root. The arena A has therefore the following shape:



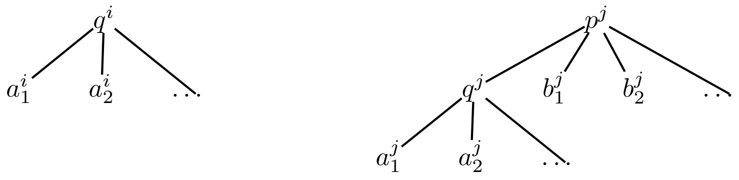
where each triangle T_i represents an arena of order 0 or 1.

We will see that the following proof can easily be adapted to take into account the general case of forest arenas (multiple roots).

We note I_k for $k = 0$ or 1, the set of indices i such that the arena T_i has order k:

$$I_k = \{i \in 1..n \mid \mathsf{order}(T_i) = k\}$$

Here is a graphic representation of the arenas T_i for $i \in I_0$ and T_j for $j \in I_1$:



For any justified sequence of moves u, we write $?(u)$ for the subsequence of u consisting of the questions in the sequence u that are still pending at the end of the sequence.

Let L be the following language $L = \{\ p^i q^i \mid i \in I_1\}$. We consider the following cases:

Case	$\lambda_{OP}(m)$	$?(u) \in$	condition
0	O	$\{\epsilon\}$	
A	P	q	
B	O	$q \cdot L^* \cdot p^i$	$i \in I_1$
C	P	$q \cdot L^* \cdot p^i q^i$	$i \in I_1$
D	O	$q \cdot L^* \cdot q^i$	$i \in I_0$

We use the notation $\hat{s}$ to denote a legal and well-bracketed *justified* sequence of moves and s to denote the same sequence of moves with pointers removed.

Note that the well-bracketing condition already tells us how to uniquely recover the pointers for P answer moves: a P-answers points to the last pending question having the same tag. However for O answers, we will see that the visibility condition already ensures the unique recoverability of the pointer and that the well-bracketing condition is not needed.

We prove by induction on the sequence of moves u that $?(u)$ corresponds to either case 0, A, B, C or D and that the pointers in u can be recovered uniquely.

Base cases:

If u is the empty sequence ϵ then there is no pointer to recover and it corresponds to case 0.

If u is a singleton then it must be the initial question q and there is not pointer to recover. This corresponds to case A.

Step case:

Consider a legal well-bracketed justified sequence $\hat{s}$ where $s = u \cdot m$ and $m \in M_A$. The induction hypothesis tells us that the pointers of u can be recovered (and therefore the P-view or O-view at that point can be computed) and that u corresponds to one of the cases 0,A,B,C or D.

We proceed by case analysis on u:

case 0 This case cannot happen because $?(u) = \epsilon$ (u is a complete play) implies that there cannot be any further move m.

Indeed the visibility condition implies that m must point to a P-question in the O-view at that point. But since u is a complete play, the O-view is $\llcorner \hat{u} \lrcorner = qa$ which does not contain any P-question. Hence the move m cannot be justified and is not valid.

case A $?(u) = q$ and the last move m is played by P. There are several cases:

- m is an answer a_k (to the initial question q) for some k, then m points to q:
  $\hat{s} = \begin{tikzcd}[column sep=small] q & \cdots & m \arrow[ll, bend right] \end{tikzcd}$
  and $?(s) = \epsilon$ therefore s correspond to the case 0 (complete play).
- $m = q^i$ where q^i is an order 0 question ($i \in I_0$). Then q^i points to the initial question q and s falls into category D.
- $m = p^i$, a first order question, then p^i points to q,
  $?(s) = qp^i$ and it is O's turn after s therefore s falls into category B.

case B $?(u) \in q \cdot L^* \cdot p^i$ where $i \in I_1$ and O plays the move m.

We now analyse the different possible O-moves:

- Suppose that O gives the (tagged) answer b^j for some $j \in I_1$ then the visibility condition constraints it to point to a question in the O-view at that point.
  We remark that the last move in $\hat{u}$ must be p^i. Indeed, suppose that there is a move $x \in M_A$ such that $\hat{u} = \begin{tikzcd}[column sep=small] q & \cdots & p^i \arrow[ll, bend right] \end{tikzcd} x$ then by visibility, the O-move x should points to a move in the O-view a that point. The O-view is qp^i, therefore x can only points to p^i. But then, p^i is not a pending question in s which is a contradiction.
  Therefore $\llcorner \hat{u} \lrcorner = \llcorner \begin{tikzcd}[column sep=small] q & \cdots & p^i \arrow[ll, bend right] \end{tikzcd} \lrcorner = qp^i$.
  Hence b^j can only point to p^i (and therefore $i = j$).
  We then have $?(s) = ?(u \cdot b^i) \in q \cdot L^*$ which is covered by case A and C.
- The only other possible O-move is q^i which, again by the visibility condition, points necessarily to the previous move p^i. We then have $?(s) = ?(u \cdot q^i) \in q \cdot L^* \cdot p^i q^i$. This falls into category C.

case C $?(u) \in q \cdot L^* \cdot p^i q^i$ where $i \in I_1$ and the move m is played by P.

Suppose m is an answer, then the well-bracketing condition imposes to answer to q^i first. The move m is therefore an integer a^i pointing to q^i. We then have $?(s) = ?(u \cdot a^i) \in q \cdot L^* \cdot p^i$. This correspond to case B.

Suppose m is a question then there are two cases:

- $m = q^j$ with $j \in I_0$, the pointer goes to the initial question q and s falls into category D.
- $m = p^j$ with $j \in I_1$, the pointer goes to the initial question q and s falls into category B.

case D $?(u) \in q \cdot L^* \cdot q^i$ where $i \in I_0$ and the move m is played by O.

The same argument as in case B holds. However there is now another possible move: the answer $m = a^i_k$ for some k. This moves can only points to q^i (this is the only pending question tagged by $i \in I_0$).

Then $?(\hat{s}) = ?(\hat{u} \cdot a^i_k) = ?(\begin{tikzcd}[column sep=small] q & \cdots & q^i \arrow[ll, bend right] & \cdots & a^i_k \arrow[ll, bend right] \end{tikzcd}) \in q \cdot L^*$ therefore s falls either into category A or C.

This completes the induction.

How to generalize the proof to arenas that have multiple roots (forest arenas)? In fact there is no ambiguity since all the moves are implicitly tagged according to the arena that they belong to. Therefore in the induction, it suffices to ignore the moves that belong to another tree (as if they were part of a different game played in parallel).

□

1.2.7 Pointer-less strategies

Up to order 2, the semantics of PCF terms is entirely defined by pointer-less strategies. In other words, the pointers can be uniquely reconstructed from any non justified sequence of moves satisfying the visibility and well-bracketing condition.

At level 3 however, pointers cannot be omitted. There is an example in Abramsky and McCusker [1997] to illustrate this. Consider the following two terms of type $((\mathbb{N} \Rightarrow \mathbb{N}) \Rightarrow \mathbb{N}) \Rightarrow \mathbb{N}$:

$$M_1 = \lambda f.f(\lambda x.f(\lambda y.y))$$
$$M_2 = \lambda f.f(\lambda x.f(\lambda y.x))$$

We assign tags to the types in order to identify in which arena the questions are asked: $((\mathbb{N}^1 \Rightarrow \mathbb{N}^2) \Rightarrow \mathbb{N}^3) \Rightarrow \mathbb{N}^4$. Consider now the following pointer-less sequence of moves $s = q^4q^3q^2q^3q^2q^1$. It is possible to retrieve the pointers of the first five moves but there is an ambiguity for the last move: does it point to the first or second occurrence of q^2 in the sequence s?

Note that the visibility condition does not eliminate the ambiguity, since the two occurrences of q^2 both appear in the P-view at that point (after recovering the pointers of s up to the second last move we get:

$$\begin{tikzcd}[column sep=tiny] s = & q^4 & q^3 \arrow[l, bend right] & q^2 \arrow[l, bend right] & q^3 \arrow[lll, bend right] & q^2 \arrow[l, bend right] & q^1 \end{tikzcd}$$

therefore the P-view of s is s itself.)

In fact these two different possibilities correspond to two different strategies. Suppose that the link goes to the first occurrence of q^2 then it means that the proponent is requesting the value of the variable x bound in the subterm $\lambda x.f(\lambda y....)$. If P needs to know the value of x, this is because P is in fact following the strategy of the subterm $\lambda y.x$. And the entire play is part of the strategy $[\![M_2]\!]$.

Similarly, if the link points to the second occurrence of q^2 then the play belongs to the strategy $[\![M_1]\!]$.

1.3 PCF

1.3.1 The syntax of the language

PCF is a simply-type λ-calculus with the following additions: integer constants (of ground type), first-order arithmetic operators, if-then-else branching, and the recursion combinator $Y_A : (A \rightarrow A) \rightarrow A$ for any type A.

The types of PCF are given by the following grammar:

$$T ::= \texttt{exp} \mid T \rightarrow T$$

The following grammar gives the structure of terms:

$$\begin{aligned} M ::= {} & x \mid \lambda x : A.M \mid MM \mid \\ & \mid n \mid \texttt{succ}\ M \mid \texttt{pred}\ M \\ & \mid \texttt{cond}\ MMM \mid \texttt{Y}_A\ M \end{aligned}$$

where x ranges over a set of countably many variables and n ranges over the set of natural numbers.

Terms are generated according to the formation rules given in table 1.1 where the judgement is of the form $\Gamma \vdash M : A$.

$$(var)\frac{}{x_1 : A_1, x_2 : A_2, \dots x_n : A_n \vdash x_i : A_i}\ i \in 1..n$$

$$(app)\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B} \qquad (abs)\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.M : A \to B}$$

$$(const)\frac{}{\Gamma \vdash n : \mathtt{exp}} \qquad (succ)\frac{\Gamma \vdash M : \mathtt{exp}}{\Gamma \vdash \mathtt{succ}\ M : \mathtt{exp}} \qquad (pred)\frac{\Gamma \vdash M : \mathtt{exp}}{\Gamma \vdash \mathtt{pred}\ M : \mathtt{exp}}$$

$$(cond)\frac{\Gamma \vdash M : exp \qquad \Gamma \vdash N_1 : exp \qquad \Gamma \vdash N_2 : exp}{\Gamma \vdash \mathtt{cond}\ M\ N_1\ N_2} \qquad (rec)\frac{\Gamma \vdash M : A \to A}{\Gamma \vdash Y_A M : A}$$

Tab. 1.1: Formation rules for PCF terms

1.3.2 Operational semantics

We give the big-step operational semantics of PCF. The notation $M \Downarrow V$ means that the closed term M evaluates to the canonical form V. The canonical forms are given by the following grammar:

$$V ::= n \mid \lambda x.M$$

In other word, a canonical form is either a number or a function.

The operational semantics is given for closed terms therefore the context Γ is not present in the evaluation rules.

The full operational semantics is given in table 1.3.2.

$$\frac{}{V \Downarrow V} \quad \text{provided that } V \text{ is in canonical form.}$$

$$\frac{M \Downarrow \lambda x.M' \quad M'[x/N]}{MN \Downarrow V}$$

$$\frac{M \Downarrow n}{\mathtt{succ}\ M \Downarrow n+1} \qquad \frac{M \Downarrow n+1}{\mathtt{pred}\ M \Downarrow n} \qquad \frac{M \Downarrow 0}{\mathtt{pred}\ M \Downarrow 0}$$

$$\frac{M \Downarrow 0 \quad N_1 \Downarrow V}{\mathtt{cond}\ MN_1N_2 \Downarrow V} \qquad \frac{M \Downarrow n+1 \quad N_2 \Downarrow V}{\mathtt{cond}\ MN_1N_2 \Downarrow V}$$

$$\frac{M(\mathrm{Y}M) \Downarrow V}{\mathtt{Y}M \Downarrow V}$$

Tab. 1.2: Big-step operational semantics of PCF

1.4 Idealized Algol (IA)

1.4.1 The syntax of IA

IA is an extension of PCF introduced by J.C. Reynold in Reynolds. It adds imperative features such as local variables and sequential composition.

The description of the language that we give here follows the one of Abramsky and McCusker [1997].

On top of `exp`, PCF has the following two new types: `com` for commands and `var` for variables.

There is a constant `skip` of type `com` which corresponds to the command that do nothing. Commands can be composed using the sequential composition operator `seq`. Local variable are declared using the `new` operator, variable content is written using `assign` and retrieved using `deref`.

The new formations rules are given in table 1.3.

$$\frac{\Gamma \vdash M : \texttt{com} \quad \Gamma \vdash N : A}{\Gamma \vdash \texttt{seq}_A\ M\ N\ : A}\ A \in \{\texttt{com}, \texttt{exp}\}$$

$$\frac{\Gamma \vdash M : \texttt{var} \quad \Gamma \vdash N : \texttt{exp}}{\Gamma \vdash \texttt{assign}\ M\ N\ : \texttt{com}} \qquad \frac{\Gamma \vdash M : \texttt{var}}{\Gamma \vdash \texttt{deref}\ M\ : \texttt{exp}}$$

$$\frac{\Gamma, x : \texttt{var} \vdash M : A}{\Gamma \vdash \texttt{new}\ x\ \texttt{in}\ M}\ A \in \{\texttt{com}, \texttt{exp}\}$$

$$\frac{\Gamma \vdash M_1 : \texttt{exp} \to \texttt{com} \quad \Gamma \vdash M_2 : \texttt{exp}}{\Gamma \vdash \texttt{mkvar}\ M_1\ M_2\ : \texttt{var}}$$

Tab. 1.3: Formation rules for IA terms

If $\vdash M : A$ (i.e. M can be formed with an empty context), we say that M is a close term.

1.4.2 Operational semantics

In IA the semantics is given in a slightly different form from PCF. In PCF, the evaluation rules were given for closed terms only. Suppose that we proceed the same way for IA and consider the evaluation rule for the `new` construct: the conclusion is $\texttt{new}\ x := 0\ \texttt{in}\ M$ and the premise is an evaluation for a certain term constructed from M, more precisely the term M where *some* occurrences of x are replaced by the value 0. Because of the presence of the `assign` operator, we cannot simply replace all the occurrences of x in M (the required substitution is more complicated than the substitution used for beta-reduction).

Therefore, instead of giving the semantics for closed term we consider terms whose free variables are all of type `var`. These free variables are “closed” by mean of stores. A store is a function mapping free variables of type `var` to natural numbers. Suppose Γ is a context containing only variable of type `var`, then we say that Γ is a `var`-context. A store whose domain Γ is called a Γ-store.

The notation $s \mid x \mapsto n$ refers to the store that maps x to n and otherwise maps variables according to the store s.

The canonical forms for IA are given by the grammar:

$$V ::= n \mid \lambda x.M \mid x \mid \texttt{mkvar} M N$$

where $n \in \mathbb{N}$ and $x : var$.

A program is now defined by a term together with a Γ-store such that $\Gamma \vdash M : A$. The evaluation semantics is expressed by the judgment form

$$s, M \Downarrow s', V$$

where s and s' are Γ-stores, $\Gamma \vdash M : A$ and $\Gamma \vdash V : A$ where V is in canonical form.

The operational semantics for IA is given by the rule of PCF (table 1.3.2) together with the rules of table 1.4.2 where the following abbreviation is used:

$$\frac{M_1 \Downarrow V_1 \quad M_2 \Downarrow V_2}{M \Downarrow V} \quad \text{for} \quad \frac{s, M_1 \Downarrow s', V_1 \quad s', M_2 \Downarrow s'', V_2}{s, M \Downarrow s'', V}$$

1.4.3 Game semantics

As we have seen in section 1.2, games and strategies form a cartesian closed category, therefore games can model the simply-typed λ-calculus. Let us first explain how this is achieved before extending the model to PCF and IA.

$$\textbf{Sequencing}\ \frac{M \Downarrow \texttt{skip} \quad N \Downarrow V}{\texttt{seq}\ M\ N \Downarrow V}$$

$$\textbf{Variables}\ \frac{s, N \Downarrow s', n \quad s', M \Downarrow s'', x}{s, \texttt{assign}\ M\ N \Downarrow (s'' \mid x \mapsto n), \texttt{skip}} \qquad \frac{s, M \Downarrow s', x}{s, \texttt{deref}\ M \Downarrow s', s'(x)}$$

$$\texttt{mkvar}\frac{N \Downarrow n \quad M \Downarrow \texttt{mkvar}\ M_1\ M_2 \quad M_1\ n \Downarrow \texttt{skip}}{\texttt{assign}\ M\ N \Downarrow \texttt{skip}} \qquad \frac{N \Downarrow \texttt{mkvar}\ M_1\ M_2 \quad M_2\ \Downarrow n}{\texttt{deref}\ M \Downarrow n}$$

$$\textbf{Block}\frac{(s \mid x \mapsto 0), M \Downarrow (s' \mid x \mapsto n), V}{s, \texttt{new}\ x\ \texttt{in}\ M \Downarrow s', V}$$

Tab. 1.4: Big-step operational semantics of IA

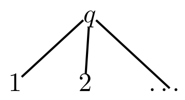
Simply typed λ-calculus

In the cartesian closed category $\mathcal{C}$, the objects are the arenas and the morphisms are the strategies.

In the games that we describe here, the Opponent represents the environment while the Proponent plays according to a strategy imposed by the program itself.

Given a simple type A, we will model it as an arena $[\![A]\!]$. A context $\Gamma = x_1 : A_1, \dots x_n : A_n$ will be mapped to the arena $[\![\Gamma]\!] = [\![A_1]\!] \times \dots \times [\![A_n]\!]$ and a term $\Gamma \vdash M : A$ will be modeled by a strategy on the arena $[\![\Gamma]\!] \to [\![A]\!]$. Since $\mathcal{C}$ is cartesian closed, there is is a terminal object $\mathbf{1}$ (the empty arena) that models the empty context ($[\![\Gamma]\!] = \mathbf{1}$).

The base type `exp` is interpreted by the following flat arena of natural numbers noted $\mathbb{N}$:



In this arena, there is only one question: the initial O-question, P can then answer it by playing a natural number $i \in \mathbb{N}$. There are only two kinds strategy on this arena:

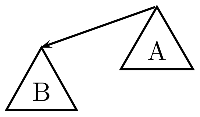
- the empty strategy where P never answer the initial question. This corresponds to a non terminating computation;
- the strategies where P answers by playing a number n. This models the constants of the language.

Given the interpretation of base types, we define the interpretation of $A \to B$ by induction:

$$[\![A \to B]\!] = [\![A]\!] \Rightarrow [\![B]\!]$$

where the operator $\Rightarrow$ denotes the arena construction $!A \multimap B$ which exist because $\mathcal{C}$ is cartesian closed.

Graphically if we represent the arena A and B by two triangles, the arena for $A \to B$ would be represented by:



Variables are interpreted by projection:

$$[\![x_1 : A_1, \dots, x_n : A_n \vdash x_i : A_i]\!] = \pi_i : [\![A_i]\!] \times \dots \times [\![A_i]\!] \times \dots \times [\![A_n]\!] \to [\![A_i]\!]$$

The abstraction $\Gamma \vdash \lambda x : A.M : A \to B$ is modeled by a strategy on the arena $[\![\Gamma]\!] \to ([\![A]\!] \Rightarrow [\![B]\!])$. This strategy is obtain by using the currying operator of the cartesian closed category:

$$[\![\Gamma \vdash \lambda x : A.M : A \to B]\!] = \Lambda([\![\Gamma, x : A \vdash M : B]\!])$$

The application $\Gamma \vdash MN$ is modeled using the evaluation map $ev_{A,B} : (A \Rightarrow B) \times A \rightarrow B$:

$$[\![\Gamma \vdash MN]\!] = \langle [\![\Gamma \vdash M, \Gamma \vdash N]\!] \rangle ; ev_{A,B}$$

PCF

We now show how to model the PCF constructs in the game semantics setting. In the following, the sub-arena of a game are tagged in order to distinguish identical arenas that are present in different components of the game. Moves are also tagged in the exponent in order to identify the sub-arena in which moves are played. We will omit the pointers in the play when they are not essential for the understanding of the model (moreover we will see later on that under certain assumptions up to order 2, pointers can be recovered uniquely).

The successor arithmetic operator is modeled by the following strategy on the arena $\mathbb{N}^1 \Rightarrow \mathbb{N}^0$:

$$[\![\mathtt{succ}]\!] = \{ q^0 \cdot q^1 \cdot n^1 \cdot (n+1)^0 \mid n \in \mathbb{N} \}$$

The predecessor arithmetic operator is denoted by the strategy

$$[\![\mathtt{pred}]\!] = \{ q^0 \cdot q^1 \cdot n^1 \cdot (n-1)^0 \mid n > 0 \} \cup \{ q^0 \cdot q^1 \cdot 0^1 \cdot 0^0 \}$$

Then given a term $\Gamma \vdash \mathtt{succ} M : \mathtt{exp}$ we define:

$$[\![\Gamma \vdash \mathtt{succ}\ \ M : \mathtt{exp}]\!] = [\![\Gamma \vdash M]\!] ; [\![\mathtt{succ}]\!]$$

$$[\![\Gamma \vdash \mathtt{pred}\ \ M : \mathtt{exp}]\!] = [\![\Gamma \vdash M]\!] ; [\![\mathtt{pred}]\!]$$

The conditional operator is denoted by the following strategy on the arena $\mathbb{N}^3 \times \mathbb{N}^2 \times \mathbb{N}^1 \Rightarrow \mathbb{N}^0$:

$$[\![\mathtt{cond}]\!] = \{ q^0 \cdot q^3 \cdot 0 \cdot q^2 \cdot n^2 \cdot n^0 \mid n \in \mathbb{N} \} \cup \{ q^0 \cdot q^3 \cdot m \cdot q^2 \cdot n^2 \cdot n^0 \mid m > 0, n \in \mathbb{N} \}$$

Given a term $\Gamma \vdash \mathtt{cond} M\ N_1\ N_2$ we define:

$$[\![\Gamma \vdash \mathtt{cond} M\ N_1\ N_2]\!] = \langle [\![\Gamma \vdash M]\!], [\![\Gamma \vdash N_1]\!], [\![\Gamma \vdash N_2]\!] \rangle ; [\![\mathtt{cond}]\!]$$

The interpretation of the $\mathtt{Y}$ combinator is a bit more complicated.

Consider the term $\Gamma \vdash M : A \rightarrow A$, its semantics f is a strategy on $[\![\Gamma]\!] \times [\![A]\!] \rightarrow [\![A]\!]$. We define the chain g_n of strategies on the arena $[\![\Gamma]\!] \rightarrow [\![A]\!]$ as follows:

$$\begin{array}{rcl} g_0 & = & \bot \\ g_{n+1} & = & F(g_n) = \langle id_{[\![\Gamma]\!]}, g_n \rangle ; f \end{array}$$

where $\bot$ denotes the empty strategy $\{\epsilon\}$.

It is easy to see that indeed the g_n forms a chain. We define $[\![\mathtt{Y}\ \ M]\!]$ to be the least upper bound of the chain g_n (i.e. the least fixed point of F). Its existence is guaranteed by the fact that the category of games is cpo-enriched.

IA

It is easy to check that all the strategies given until now are well-bracketed and innocent. From now on, we will only require well-bracketing and we will introduce strategies that are not innocent. This is a necessity if we want to give a model of memory cells that correspond to variables. The intuition behind this fact is that a cell needs to remember what was the last value written in it in order to be able to return it when it is read, and this can only be done by looking at the whole history of moves, not only those present in the P-view.

1.4.4 Full-abstraction

In this section we recall the standard full abstraction result proved in Abramsky et al. [1994] and Hyland and Ong [2000].

A context noted $C[-]$ is a term containing a hole denoted by $-$. If $C[-]$ is a context then $C[A]$ denotes the term obtained after replacing the hole by the term A.

Definition 1.4.1 (Observational preorder). Let $\vdash M : A$ and $\vdash N : A$ be two closed terms. We define the relation $\sqsubseteq$ as follows:

$M \sqsubseteq N$ if and only if for all context $C[-]$ such that $C[M]$ and $C[M]$ are well-formed terms if $C[M] \Downarrow$ then $C[N] \Downarrow$.

Lemma 1.4.2 (Soundness for PCF terms). *Let M be a PCF term. If $M \Downarrow V$ then $[\![M]\!] = [\![V]\!]$.*

Lemma 1.4.3 (Soundness for IA terms). *Let $\Gamma \vdash M : A$ be an IA term and a Γ store s. If $s, M \Downarrow s', V$ then the plays of $[\![s, M]\!] : I \multimap A \otimes !\Gamma$ which begin with a move of A are identical to those of $[\![s', V]\!]$.*

Lemma 1.4.4 (Computational adequacy for PCF terms). *All PCF terms are computable. (i.e. $[\![M]\!] \neq \perp$ implies $M \Downarrow$)*

Lemma 1.4.5 (Computational adequacy for IA terms). *All IA terms are computable. (i.e. $[\![M]\!] \neq \perp$ implies $M \Downarrow$)*

The following result follows from soundness and computational adequacy of the model.

Proposition 1.4.6 (Inequational soundness). *Let M and N be two closed terms then*

$$[\![M]\!] \subseteq [\![N]\!] \implies M \sqsubseteq N$$

Proposition 1.4.7 (Definability). *Let σ be a compact well-bracketed on a game A denoting a IA type. Then there is an IA-term M such that $[\![M]\!] = \sigma$.*

The final standard result of game semantics can then be proved using proposition 1.4.6 and 1.4.7:

Theorem 1.4.8 (Full abstraction). *Let M and N be two closed IA-terms.*

$$[\![M]\!] \precsim_b [\![N]\!] \iff M \sqsubseteq N$$

where $\precsim_b$ denotes the intrinsic preorder of the category $\mathcal{C}_b$.

1.4.5 First-order and second-order Idealized Algol

The strategies of second-order IA can be represented by an extended regular language (Dan R. Ghica and Guy McCusker).

1.4.6 Call-by-Value first-order Idealized Algol

Game semantics for call-by-value programming Language.

1.5 Data-abstraction refinement

Recently Dimovski et al. presented [Dimovski et al., 2005] a new technique for data abstraction refinement based on game semantics.

1.5.1 Abstraction refinement

Abstraction refinement is a technique aiming at solving the following problem: does the safety property φ holds for a given infinite model M.

In general the problem is undecidable. The difficulty lies in the non-finiteness of the model. Indeed, the problem becomes decidable for finite models. Abstraction refinement makes use of this remark: it tries to reduces the problem to finite models. The idea is to produce an abstraction of the model which is finite. Using model checking techniques, one can check whether a particular property holds or not for that abstracted model. If it does not hold, then a counter-example can be produced. If this counter-example is not spurious (it is a valid trace in the model M) then we know that M does not verify the safety property. If the counter-example is spurious then we use it to produced a new abstraction, finer that the previous one. The process is then repeated.

The abstraction produced must be a conservative over-abstraction: its safety implies the safety of the original model. Therefore the abstraction refinement algorithm can be stated as follows:

Algorithm 1.5.1 (Abstraction refinement)**.** The input: M an infinite model, φ a safety property. The question: does $M \models \varphi$ hold?

step 1 Build a (finite) abstraction A of model M.

step 2 Check whether $A \models \varphi$ using a model checker. If the answer is yes then **return** $M \models \varphi$ otherwise continue to step 3.

step 3 Check whether the counter-example proving that $A \not\models \varphi$ is not spurious (i.e. is also a counter-example for M). If yes then **return** $M \not\models \varphi$ otherwise continue to step 4

step 4 Use the counter-example to refine A. Goto step 2.

Note that the refinement process may loop forever.

1.5.2 Game semantics and abstraction refinement

In [Dimovski et al., 2005], a data-refinement procedure is derived that is guaranteed to discover an error if it exists.

The target language is Idealized Algol (introduced in section 1.4).

Abstraction is done at the level of data. For that purpose they introduce AIA: Abstract Idealized Algol.

The reduction rules of AIA are similar to those of IA, except that they introduce non determinism and the *abort* operator.

abort reduces to the special value ε and any program involving the evaluation of *abort* reduces to ε.

1.5.3 Game semantics of EIA (Erratic Idealized Algol)

1.5.4 Game semantics of AIA

Abstraction are equivalence classes on $\mathbb{Z}$ noted π. We suppose that the abstractions π are computable.

- Basics types + abstract data types exp_π where π is an abstraction:

$$\tau ::= int_\pi \mid bool$$

The abstraction used in Dimovski et al. [2005] are:

$$[] = \{\mathbb{Z}\} \qquad [n,m] = \{\{i|i<n\},\{n\},\{n+1\},\dots\{m-1\},\{m\},\{i|i>m\}\}\text{where } n \leq 0 \leq m$$

- new operators defined on these new types
- For simplicity we only consider abstraction of the base type *expint*.
- reduction rules: .non determinism
. *abort* operator.

1.5.5 the algorithm

-identify counter-example -analyse it by uncovering the hidden moves.

Strategy represented using CSP process algebra verification done with FDR.

Advantage of the approach:

the game semantics approach gives compositionality for free. small size of the model (due to hiding of unobservable internal moves)

Possible extension: recusion concurrency higher-order fragment

Chapter 2

SAFE λ-CALCULUS

2.1 Background

2.1.1 Homogeneous type

Let $Types$ be the set of simple types generated by the grammar $A ::= o \mid A \rightarrow A$. Any type different from the base type o can be written $(A_1, \cdots, A_n, o)$ for some $n \geq 1$, which is a shorthand for $A_1 \rightarrow \cdots \rightarrow A_n \rightarrow o$ (by convention, $\rightarrow$ associates to the right).

We suppose that a ranking function has been defined: $\mathsf{rank} : Types \longrightarrow (L, \leq)$ where $(L, \leq)$ is any linearly ordered set. Possible candidates for the ranking function are:

- $\mathsf{ord} : Types \longrightarrow (\mathbb{N}, \leq)$ with $\mathsf{ord}(o) = 0$ and $\mathsf{ord}(A \rightarrow B) = \max(\mathsf{ord}(A) + 1, \mathsf{ord}(B))$.
- $\mathsf{height} : Types \longrightarrow (\mathbb{N}, \leq)$ with

$$\begin{array}{rcl} \mathsf{height}(o) & = & 0 \\ \mathsf{height}(A \rightarrow B) & = & 1 + \max(\mathsf{height}(A), \mathsf{height}(B)) \end{array}$$

- $\mathsf{nparam} : Types \longrightarrow (\mathbb{N}, \leq)$ with $\mathsf{nparam}(o) = 0$ and $\mathsf{nparam}(A_1, \cdots, A_n) = n$.
- $\mathsf{ordernp} : Types \longrightarrow (\mathbb{N} \times \mathbb{N}, \leq)$ with $\mathsf{ordernp}(t) = (\mathsf{order}(t), \mathsf{nparam}(t))$ for $t \in Types$.

Following Knapik et al. [2002], a type is rank-homogeneous if it is o or if it is $(A_1, \cdots, A_n, o)$ with the condition that $\mathsf{rank}(A_1) \geq \mathsf{rank}(A_2) \geq \cdots \geq \mathsf{rank}(A_n)$ and each $A_1, \ldots, A_n$ is rank-homogeneous.

Suppose that $\overline{A_1}, \overline{A_2}, \ldots, \overline{A_n}$ are n lists of types, where A_{ij} denotes the j^{th} type of list $\overline{A_i}$ and l_i the size of $\overline{A_i}$. Then the notation $A = (\overline{A_1} \mid \cdots \mid \overline{A_r} \mid o)$ means that

- A is the type $(A_{11}, A_{12}, \cdots, A_{1l_1}, A_{21}, \cdots, A_{2l_2}, \cdots A_{n1}, \cdots, A_{nl_n}, o)$
- $\forall i : \forall u, v \in A_i : \mathsf{rank}(u) = \mathsf{rank}(v)$
- $\forall i, j. \forall u \in A_i . \forall v \in A_j . i < j \implies \mathsf{rank}(u) > \mathsf{rank}(v)$

Consequently, A is rank-homogenous. This notation organises the A_{ij}s into partitions according to their ranks. Suppose $B = (\overline{B_1} \mid \cdots \mid \overline{B_m} \mid o)$. We write $(\overline{A_1} \mid \cdots \mid \overline{A_n} \mid B)$ to mean

$$(\overline{A_1} \mid \cdots \mid \overline{A_n} \mid \overline{B_1} \mid \cdots \mid \overline{B_m} \mid o).$$

From now on, we only consider the rank function ord. The term "homogeneous" will be used to refer to ord-homogeneity.

2.2 Homogeneous safe λ-calculus

We recall the definition of the safe λ-calculus given in Ong [2005].

2.2.1 Rules

These rules are a corrected version of Aehlig et al. [2005]

In the following we shall consider terms-in-context $\Gamma \vdash M : A$ of the simply-typed λ-calculus. Let Δ be a simply-typed alphabet i.e., each symbol in Δ has a simple type. We write $\mathcal{T}^A(\Delta)$ for the set of terms of type A built up from the set Δ understood as constant symbols, *without* using λ-abstraction.

The ***Safe λ-Calculus*** is a sub-system of the simply-typed λ-calculus. Typing judgements (or terms-in-context) are of the form

$$\overline{x_1} : \overline{A_1} \,|\, \cdots \,|\, \overline{x_n} : \overline{A_n} \vdash M : B$$

which is shorthand for $x_{11} : A_{11}, \cdots , x_{1r} : A_{1r}, A_{21}, \ldots \vdash M : B$ such that the context variables are listed in decreasing type order and with the condition that $\mathsf{ord}(x_{ik}) < \mathsf{ord}(x_{jl})$ for any k, l and $i < j$.

Valid typing judgements of the system are defined by induction over the following rules, where Δ is a given homogeneously-typed alphabet:

$$(\mathbf{wk})\frac{\Sigma \vdash M : B \qquad \Sigma \subset \Delta}{\Delta \vdash M : B} \qquad (\mathbf{perm})\frac{\Gamma \vdash M : B \qquad \sigma(\Gamma) \text{ homogeneous}}{\sigma(\Gamma) \vdash M : B}$$

$$(\mathbf{\Sigma\text{-const}})\frac{b : o^r \to o \in \Sigma}{\vdash b : o^r \to o} \qquad (\mathbf{var})\frac{}{x_{ij} : A_{ij} \vdash x_{ij} : A_{ij}}$$

$$(\lambda\text{-}\mathbf{abs})\frac{\overline{x_1} : \overline{A_1} \,|\, \cdots \,|\, \overline{x_{n+1}} : \overline{A_{n+1}} \vdash M : B \qquad \mathsf{ord}(\overline{A_{n+1}}) \geq \mathsf{ord}(B) - 1}{\overline{x_1} : \overline{A_1} \,|\, \cdots \,|\, \overline{x_n} : \overline{A_n} \vdash \boldsymbol{\lambda}\overline{x_{n+1}} : \overline{A_{n+1}}.M : (\overline{A_{n+1}} \,|\, B)}$$

$$(\mathbf{app})\frac{\Gamma \vdash M : (\overline{B_1} \,|\, \cdots \,|\, \overline{B_m} \,|\, o) \qquad \Gamma \vdash N_1 : B_{11} \quad \cdots \quad \Gamma \vdash N_l : B_{1l} \qquad l = |\overline{B_1}|}{\Gamma \vdash MN_1 \cdots N_l : (\overline{B_2} \,|\, \cdots \,|\, \overline{B_m} \,|\, o)}$$

$$(\mathbf{app}^+)\frac{\Gamma \vdash M : (\overline{B_1} \,|\, \cdots \,|\, \overline{B_m} \,|\, o) \qquad \Gamma \vdash N_1 : B_{11} \quad \cdots \quad \Gamma \vdash N_l : B_{1l} \qquad l < |\overline{B_1}|}{\Gamma \vdash MN_1 \cdots N_l : (\overline{B} \,|\, \cdots \,|\, \overline{B_m} \,|\, o)}$$

where $\overline{B_1} = B_{11}, \ldots , B_{1l}, \overline{B}$ with the condition that every variable in Γ has an order strictly greater than $\mathsf{ord}(\overline{B_1})$.

Property 2.2.1 (Basic properties). Suppose $\Gamma \vdash M : B$ is a valid judgment then

(i) B is homogeneous

(ii) Every free variables of M has order at least $\mathsf{ord}(M)$

(iii) $fv(M) \vdash M : B$

where $fv(M) \subseteq \Gamma$ denotes the context constituted of the variables in Γ occurring free in M.

Proof. (i) and (ii) An easy proof by structural induction. (iii) Because the weakening rule is the only rule which can introduce in the context some variable not occurring freely in M. ☐

We now define a special kind of substitution that performs simultaneous substitution and that permits variable capture (i.e. that does not rename variables when the substitution is performed on an abstraction).

Definition 2.2.2 (Capture permitting simultaneous substitution (for homogeneous safe terms))**.** We use the notation $\left[\overline{N}/\overline{x}\right]$ for $[N_1 \ldots N_n/x_1 \ldots x_n]$ and $\overline{y} : \overline{A}$ for $y_1 : A_1, \ldots, y_p : A_p$. A safe term must have one of the forms occurring on the left-hand side of the following equations, where the terms M, $N_1, \ldots N_l$ are safe terms:

$$\begin{array}{rcl}
c\left[\overline{N}/\overline{x}\right] & = & c \quad \text{where } c \text{ is a } \Sigma\text{-constant} \\
x_i\left[\overline{N}/\overline{x}\right] & = & N_i \\
y\left[\overline{N}/\overline{x}\right] & = & y \quad \text{if } y \neq x_i \text{ for all } i, \\
(MN_1 \ldots N_l)\left[\overline{N}/\overline{x}\right] & = & (M\left[\overline{N}/\overline{x}\right])(N_1\left[\overline{N}/\overline{x}\right]) \ldots (N_l\left[\overline{N}/\overline{x}\right]) \\
(\lambda\overline{y} : \overline{A}.M)\left[\overline{N}/\overline{x}\right] & = & \lambda\overline{y}.M\left[\overline{N} \upharpoonright I/\overline{x} \upharpoonright I\right] \\
 & & \text{where } I = \{i \in 1..n \mid x_i \notin \overline{y}\}
\end{array}$$

where $\upharpoonright$ is the index filtering operator: if s is a sequence and I a set of indices then $s \upharpoonright I$ is the subsequence of s obtained by removing from s all the elements at a position that is not in I.

This substitution is well-defined for safe terms in the sense that safety is preserved by substitution:

Lemma 2.2.3 (Capture-permitting simultaneous substitution preserves safety)**.** *Let $\Gamma \cup \overline{x} \vdash M$ be a safe term where $\overline{x}$ denotes a list of variables (not necessarily belonging to the same partition).*

Then for any safe terms safe terms $\Gamma \vdash N_1, \cdots, \Gamma \vdash N_n$, the capture permitting simultaneous substitution $M[N_1/x_1, \cdots, N_n/x_n]$ is safe. i.e. the following judgment is valid :

$$\Gamma \vdash M[N_1/x_1, \cdots, N_n/x_n]$$

Proof. An easy proof by an induction on the structure of the safe term. □

With the traditional substitution, it is necessary to rename variables when performing substitution on an abstraction in order to avoid possible captures of variables. Hence in general implementing substitution requires to have access to an unbound number of variables names. An interesting property of the homogeneous safe λ-calculus is that variable capture never occurs when performing substitution. Consequently, the capture permitting substitution will produce the same terms as the traditional substitution:

Lemma 2.2.4 (No variable capture lemma)**.** *In the safe λ-calculus, there is no capture of variable when performing the following capture permitting simultaneous substitution:*

$$M[N_1/x_1, \cdots, N_n/x_n]$$

provided that $\Gamma \cup \overline{x} \vdash M$, $\Gamma \vdash N_1, \cdots, \Gamma \vdash N_n$ are valid judgments.

Proof. We prove the result by induction. The variable, constant and application cases are trivial. For the abstraction case, suppose $M = \lambda\overline{y} : \overline{A}.P$ where $\overline{y} = y_1 \ldots y_p$. The capture permitting simultaneous substitution gives:

$$M\left[\overline{N}/\overline{x}\right] = \lambda\overline{y}.P\left[\overline{N} \upharpoonright I/\overline{x} \upharpoonright I\right]$$

where $I = \{i \in 1..n \mid x_i \notin \overline{y}\}$.

By the induction hypothesis there is no variable capture in $P\left[\overline{N} \upharpoonright I/\overline{x} \upharpoonright I\right]$.

Hence the only possible case of variable capture happens when for some $i \in I$ and $j \in 1..p$ the variable y_j occurs freely in N_i and x_i occurs freely in P. In that case, property 2.2.1 (ii) gives:

$$\mathsf{ord}(y_j) \geq \mathsf{ord}(N_i) = \mathsf{ord}(x_i)$$

Moreover $i \in I$ therefore $x_i \notin \overline{y}$ and since x_i occurs freely in P, x_i must also occur freely in the safe term $\lambda\overline{y}.P$. Thus, property 2.2.1 (ii) gives:

$$\mathsf{ord}(x_i) \geq \mathsf{ord}(\lambda y_1 \ldots y_p.T) \geq 1 + \mathsf{ord}(y_j) > \mathsf{ord}(y_j)$$

Hence we reach a contradiction. □

2.2.2 Safe β-reduction

We now introduce the notion of safe β-redex and show how such redex can be reduced using the capture-permitting simultaneous substitution. We will then show that a safe β-reduction reduces to a safe term.

In the simply-typed lambda calculus a redex is a term of the form $(\lambda x.M)N$. We generalize this notion to the safe lambda calculus. We call multi-redex a term of the form $(\lambda x_1 \dots x_n.M)N_1 \dots N_l$ (it is not required to have $n = l$).

We say that a multi-redex is safe if it respects the formation rules of the safe λ-calculus. More precisely, the multi-redex $(\lambda x_1 \dots x_n.M)N_1 \dots N_l$ is a safe redex if the variable $x_1, \dots, x_n$ are abstracted altogether at once using the abstraction rule and if the terms $N_1 \dots N_l$ are applied to the term $\lambda x_1 \dots x_n.M$ at once using either the application rule (**app**$^+$) or (**app**).

Note that there exist safe terms of the form $(\lambda x_1 \dots x_n.M)N_1 \dots N_l$ such that $l > n$. For instance the following term:

$$(\lambda fg.((\lambda hi.i)a))(\lambda x.x)(\lambda x.x)(\lambda x.x)$$

where is a constant, $x : o$ and $f, g, h, i, a : o \rightarrow o$ can be formed using the (**app**) rule as follows:

$$\frac{\emptyset \vdash (\lambda fg.((\lambda hi.i)a)) \quad \emptyset \vdash (\lambda x.x) \quad \emptyset \vdash (\lambda x.x) \quad \emptyset \vdash (\lambda x.x)}{\emptyset \vdash (\lambda fg.((\lambda hi.i)a))(\lambda x.x)(\lambda x.x)(\lambda x.x)}(\mathbf{app})$$

The formal definition follows:

Definition 2.2.5 (Safe redex)**.** A safe redex is a term of the form:

$$(\lambda \overline{x}.M)N_1 \dots N_l$$

such that the variables $\overline{x} = x_1 \dots x_n$ are abstracted altogether by one occurrence of the rule (**abs**) in the proof tree (possibly followed by the weakening rule) which implies that:

$$\mathsf{ord}(M) - 1 \leq \mathsf{ord}(\overline{x}) = \mathsf{ord}(x_1) = \ldots = \mathsf{ord}(x_n)$$

and such that the terms $(\lambda \overline{x}.M)$, N_1, N_l are applied together at once using either:

- the rule (**app**) with

  $$\frac{\Sigma \vdash \lambda \overline{x}.M : (\overline{B_1}| \dots |\overline{B_m}|o) \quad \Sigma \vdash N_1 \quad \dots \quad \Sigma \vdash N_l \quad l = |\overline{B_1}|}{\Sigma \vdash (\lambda \overline{x}.M)N_1 \dots N_l}(\mathbf{app})$$

  In which case $n \leq |\overline{B_1}| = l$.

- or the rule (**app**$^+$) with:

  $$\frac{\Sigma \vdash \lambda \overline{x}.M : (\overline{B_1}| \dots |\overline{B_m}|o) \quad \Sigma \vdash N_1 \quad \dots \quad \Sigma \vdash N_l \quad l < |\overline{B_1}|}{\Sigma \vdash (\lambda \overline{x}.L)N_1 \dots N_l}(\mathbf{app}^+)$$

  In which case $n \leq |\overline{B_1}|$ and no relation holds between n and l.

Note that we do not necessarily have $n = |\overline{B_1}|$.

Definition 2.2.6 (Safe reduction β_s)**.** For concision the following abbreviations are used $\overline{x} = x_1 \dots x_n$, $\overline{N} = N_1 \dots N_l$, and when $n \geq l$, $\overline{x_L} = x_1 \dots x_l$, $\overline{x_R} = x_{l+1} \dots x_n$.

- The relation β_s is defined on the set of safe redex as follows:

  $$\begin{array}{rcl} \beta_s & = & \{ \ (\lambda \overline{x} : \overline{A}.T)N_1 \dots N_l \mapsto \lambda \overline{x_R}.T \left[\overline{N}/\overline{x_L}\right] \\ & & \quad \text{where } (\lambda \overline{x} : \overline{A}.T)N_1 \dots N_l \text{ is a safe redex such that } n > l\} \\ & \cup & \{ \ (\lambda \overline{x} : \overline{A}.T)N_1 \dots N_l \mapsto T \left[\overline{N}/\overline{x}\right] N_{n+1} \dots N_l \\ & & \quad \text{where } (\lambda \overline{x} : \overline{A}.T)N_1 \dots N_l \text{ is a safe redex such that } n \leq l\} \end{array}$$

  where the notation $\left[\overline{N}/\overline{x}\right]$ denotes the capture-permitting simultaneous substitution.

- The safe β-reduction noted $\rightarrow_{\beta_s}$ is the closure of the relation β_s by compatibility with the formation rules of the safe λ-calculus.

We observe that safe β-reduction is a certain kind of multi-steps β-reduction.

Property 2.2.7. $\rightarrow_{\beta_s} \subset \twoheadrightarrow_\beta$, i.e. the safe β-reduction relation is included in the transitive closure of the β-reduction relation.

Proof. Suppose that $(M \mapsto N) \in \beta_s$. We show that $M \rightarrow^*_\beta N$.

- Suppose that the safe-redex is $M \equiv (\lambda \overline{x} : \overline{A}.T)N_1 \ldots N_l$ such that $n \leq l$ then:

$$\begin{array}{rcl}
M & =_\alpha & (\lambda z_1 \ldots z_n.T[z_1, \ldots z_n/x_1, \ldots x_n])\ N_1 N_2 \ldots N_l \\
& & \text{where the } z_i \text{ are fresh variables} \\
& \rightarrow_\beta & (\lambda z_2 \ldots z_n.T[z_1, \ldots z_n/x_1, \ldots x_n]\,[N_1/z_1])\ N_2 \ldots N_l \\
& & \ (\text{because the } z_i \text{ do not occur freely in } N_1) \\
& \rightarrow_\beta & \ldots \\
& \rightarrow_\beta & (T[z_1, \ldots z_n/x_1, \ldots x_n]\,[N_1/z_1] \ldots [N_n/z_n])\ N_{n+1} \ldots N_l \\
& \rightarrow_\beta & (T[N_1 \ldots N_l/x_1, \ldots x_l])\ N_{n+1} \ldots N_l
\end{array}$$

  And since T is safe, the substitution $T[N_1 \ldots N_l/x_1, \ldots x_l]$ in the last equation can be performed using the capture-permitting substitution. Hence $M \rightarrow^*_\beta N$.

- Suppose that $M \equiv (\lambda \overline{x} : \overline{A}.T)N_1 \ldots N_l$ such that $n > l$, then necessarily the redex must be formed using the $(\mathbf{app}^+)$ rule. The side-condition of this rules says that the free variables of the terms $N_1, \ldots N_l$ have all order strictly greater than $\mathsf{ord}(\overline{x})$, hence the x_i do not occur freely in $N_1, \ldots N_l$. Therefore:

$$\begin{array}{rcl}
M & = & (\lambda x_1 \ldots x_n.T)\ N_1 N_2 \ldots N_l \\
& \rightarrow_\beta & (\lambda x_2 \ldots x_n.T\,[N_1/x_1])\ N_2 \ldots N_l \\
& & (\text{for } i \in 2..n,\ x_i \text{ does not occur freely in } N_1) \\
& \rightarrow_\beta & \ldots \\
& \rightarrow_\beta & \lambda x_{l+1} \ldots x_n.T\,[N_1/x_1] \ldots [N_l/x_l] \\
& & (\text{for } i \in (l+1)..n,\ x_i \text{ does not occur freely in } N_l) \\
& \rightarrow_\beta & \lambda x_{l+1} \ldots x_n.T[N_1 \ldots, N_l/\ x_1, \ldots, x_l] \\
& & (\text{the } x_i \text{ do not occur freely in } N_1, \ldots N_l)
\end{array}$$

  And since T is safe, the substitution $T[N_1 \ldots N_l/x_1, \ldots x_l]$ in the last equation can be performed using the capture-permitting substitution. Hence $M \rightarrow^*_\beta N$.

□

Property 2.2.8. In the simply typed λ-calculus setting:

1. $\rightarrow_{\beta_s}$ is strongly normalizing.
2. β_s has the unique normal form property.
3. β_s has the Church-Rosser property.

Proof. 1. This is because $\rightarrow_{\beta_s} \subset \twoheadrightarrow_\beta$ and $\rightarrow_\beta$ is strongly normalizing (in the simply typed lambda calculus). 2. A term has a safe redex iff it has a β-redex therefore the set of β_s normal form is equal to the set of β_s normal form. Hence, the unicity of β normal form implies the unicity of β_s normal form. 3. is a consequence of 1 and 2. □

Since capture-permitting simultaneous substitution preserves safety (lemma 2.2.3), consequently any safe redex reduces to a safe term:

Lemma 2.2.9 (The safe reduction β_s preserves safety)**.** *If M is safe and $M \rightarrow_{\beta_s} N$ then N is safe.*

Proof. It suffices to show that the relation β_s preserves safety. Consider the safe-redex $(s \mapsto t) \in \beta_s$ where $s \equiv (\lambda x_1 \ldots x_n.M)N_1 \ldots N_l$. We proceed by case analysis on the the last rule used to form the redex.

- Suppose the last rules used is (**app**), then necessarily $n \leq l$ and the reduction is:

$$(\lambda x_1 \ldots x_n.M)N_1 \ldots N_l \qquad \mapsto \qquad t \equiv M[N_1/x_1, \cdots, N_n/x_n]\ N_{n+1} \ldots N_l$$

  The first premise of the rule (**app**) tells us that M is safe therefore using lemma 2.2.3 and the application rule we obtain that t is safe.

- Suppose the last rules used is (**app**$^+$) and $n > l$ then the reduction is

$$(\lambda \overline{x_L} : \overline{A_L}\ \overline{x_R} : \overline{A_R}.T)\overline{N_L} \qquad \mapsto \qquad t \equiv \lambda \overline{x_R} : \overline{A_R}.T\left[\overline{x_L}/\overline{N_L}\right]$$

  By lemma 2.2.3, $T\left[\overline{x_L}/\overline{N_L}\right]$ is a safe term. Using the rule (**abs**) we derive that t is safe.

- Suppose the last rules used is (**app**$^+$) and $n \leq l$ then the reduction is

$$(\lambda x_1 \ldots x_n.M)N_1 \ldots N_l \qquad \mapsto \qquad t \equiv M[N_1/x_1, \cdots, N_n/x_n]\ N_{n+1} \ldots N_l$$

  Similarly to case (**app**), we conclude that t is safe.

- Rule (**wk**) (**seq**): these cases reduce to one of the previous cases.

$\square$

Remark 2.2.10. $\rightarrow_{\beta_s}$ *does not* preserves un-safety: given two terms of the same type S safe and U unsafe, the term $(\lambda xy.y)US$ is also unsafe but it β_s-reduces to S which is safe.

2.2.3 An alternative system of rules

In this section, we will refine the formation rules given in the previous section. We say that $\Gamma \vdash M : A$ verifies P_i for $i \in \mathbb{Z}$ if the variables in Γ all have orders at least $\mathsf{ord}(A) + i$. We introduce the notation $\Gamma \vdash^i M : A$ for $i \in \mathbb{Z}$ to mean that $\Gamma \vdash M : A$ is a valid judgment satisfying P_i.

We remark that if $\Gamma \vdash M : A$ then the variables in Γ with order strictly smaller than M cannot occur freely in M and therefore it is possible to restrict the context to a smaller number of variables:

Lemma 2.2.11 (Context reduction)**.** *If $\Gamma \vdash^i M : A$ then $\Gamma' \vdash^0 M : A$ where*

$$\Gamma' = \{z \in \Gamma \mid \mathsf{ord}(z) \geq \mathsf{ord}(M)\} = \Gamma \setminus \{z \in \Gamma \mid \mathsf{ord}(M) + i \leq \mathsf{ord}(z) < \mathsf{ord}(M)\}$$

Proof. If $i \geq 0$ then the result is trivial. Suppose $i < 0$. We proceed by structural induction and by case analysis. We only give the details for the application cases (**app**) and (**app**$^+$):

- Case of the rule (**app**):

$$(\mathbf{app})\frac{\Gamma \vdash M : (\overline{B_1} \mid \cdots \mid \overline{B_m} \mid o) \qquad \Gamma \vdash N_1 : B_{11} \quad \cdots \quad \Gamma \vdash N_l : B_{1l} \qquad l = |\overline{B_1}|}{\Gamma \vdash M N_1 \cdots N_l : (\overline{B_2} \mid \cdots \mid \overline{B_m} \mid o)}$$

If the conclusion verifies P_i then, for all $z \in \Gamma$:

$$\begin{aligned}\mathsf{ord}(z) \geq 1 + \mathsf{ord}(\overline{B_2}) + i &= 1 + \mathsf{ord}(\overline{B_1}) + \mathsf{ord}(\overline{B_2}) - \mathsf{ord}(\overline{B_1}) + i \\ &= \mathsf{ord}(M) + (\mathsf{ord}(\overline{B_2}) - \mathsf{ord}(\overline{B_1}) + i)\end{aligned}$$

Therefore the first premise satisfies P_j where $j = \mathsf{ord}(\overline{B_2}) - \mathsf{ord}(\overline{B_1}) + i$. Hence by the induction hypothesis,

$$\Gamma' \vdash^0 M : (\overline{B_1} \,|\, \cdots \,|\, \overline{B_m} \,|\, o)$$

where $\Gamma' = \Gamma \setminus \{z \in \Gamma \mid \mathsf{ord}(M) + j \leq \mathsf{ord}(z) < \mathsf{ord}(M)\}$.

Similarly, for all $z \in \Sigma$:

$$\begin{aligned}\mathsf{ord}(z) \geq 1 + \mathsf{ord}(\overline{B_2}) + i &= \mathsf{ord}(\overline{B_1}) + (1 + \mathsf{ord}(\overline{B_2}) - \mathsf{ord}(\overline{B_1}) + i) \\ &= \mathsf{ord}(\overline{B_1}) + j + 1\end{aligned}$$

Hence by the induction hypothesis:

$$\Gamma'' \vdash^0 N_k : B_{1k} \text{ for } k \in 1..l$$

where $\Gamma'' = \Gamma \setminus \{z \in \Gamma \mid \mathsf{ord}(M) + j + 1 \leq \mathsf{ord}(z) < \mathsf{ord}(M)\}$.

Furthermore, $\Gamma'' = \Gamma' \cup \{z \in \Gamma \mid \mathsf{ord}(M) + j = \mathsf{ord}(z)\}$ therefore the weakening rule gives:

$$\Gamma'' \vdash^{-1} M : (\overline{B_1} \,|\, \cdots \,|\, \overline{B_m} \,|\, o)$$

Finally the (**app**) rule gives:

$$\frac{\Gamma'' \vdash^{-1} M : (\overline{B_1} \,|\, \cdots \,|\, \overline{B_m} \,|\, o) \quad \Gamma'' \vdash^0 N_1 : B_{11} \quad \ldots \quad \Gamma'' \vdash^0 N_1 : B_{1l}}{\Gamma'' \vdash M N_1 \ldots N_l : (\overline{B_2} \,|\, \cdots \,|\, \overline{B_m} \,|\, o)}$$

such that for all $z \in \Gamma''$:

$$\mathsf{ord}(z) \geq \mathsf{ord}(\overline{B_1}) \quad \geq \quad 1 + \mathsf{ord}(\overline{B_2}) = \mathsf{ord}(M N_1 \ldots N_l)$$

Therefore:

$$\Gamma'' \vdash^0 M N_1 \ldots N_l : (\overline{B_2} \,|\, \cdots \,|\, \overline{B_m} \,|\, o)$$

- (**app**$^+$) The side-condition of the rule (**app**$^+$) ensures that the first premise verify P_0. The conclusion of the rule has the same order as the first premise therefore the conclusion also verifies P_0.

□

Lemma 2.2.12. *If $\Gamma \vdash^0 M : T$ or $\Gamma \vdash^{-1} M : T$ then there is valid proof tree showing $\Gamma \vdash M : T$ such that all the judgments appearing in the proof tree verify either P_0 or P_{-1}.*

Proof. Since P_{-1} implies P_0, we assume that the judgment $\Gamma \vdash M : T$ satisfies P_{-1}.

We show that there is a proof tree for $\Gamma \vdash M : T$ where all the nodes of the tree verify P_0 or P_{-1}. We proceed by structural induction and case analysis on the last rule used to show $\Gamma \vdash M : T$.

- Axiom (**Σ-const**): the context is empty therefore the sequent verifies P_{-1}.
- Axiom (**var**): the context contains only the variable itself therefore the sequent verifies P_0.
- Rule (**wk**): If $\Delta \vdash M : T$ verifies P_{-1} then in particular $\Gamma \vdash M : T$ verifies P_{-1} for any $\Gamma \subseteq \Delta$.
- Rule (**perm**): By the induction hypothesis.

- Rule (**abs**): the second premise of the rule guarantees that the first premise verifies P_{-1}.
- Rule (**app**$^+$): The first premise has the same order has the conclusion of the rule therefore the first premise verifies P_0. The side-condition of the rule (**app**$^+$) ensures that all the other premises verify P_0.
- Rule (**app**):

$$(\mathbf{app})\frac{\Gamma \vdash M : (\overline{A}\,|B) \qquad \Gamma \vdash N_1 : A_1 \quad \cdots \quad \Gamma \vdash N_l : A_l \qquad l = |\overline{A}|}{\Gamma \vdash^0 M N_1 \cdots N_l : B}$$

Applying lemma 2.2.11 to the first premise we obtain:

$$\Sigma \vdash^0 M : (\overline{A}\,|B) \tag{2.1}$$

where $\Sigma = \{z \in \Gamma \mid \mathsf{ord}(z) \geq \mathsf{ord}((\overline{A}\,|B))\} = \{z \in \Gamma \mid \mathsf{ord}(z) \geq 1 + \mathsf{ord}(\overline{A})\}$

Since $\mathsf{ord}(\overline{A}) = \mathsf{ord}(A_1) = \ldots = \mathsf{ord}(A_l)$ by applying lemma 2.2.11 to all the premises but the first we have for all $i \in 1..p$:

$$\Sigma' \vdash^0 N_i : A_i$$

where $\Sigma' = \{z \in \Gamma \mid \mathsf{ord}(z) \geq \mathsf{ord}(\overline{A})\} \supseteq \Sigma$

If the inclusion $\Sigma \subseteq \Sigma'$ is strict then we apply the weakening rule to sequent (2.1):

$$\frac{\Sigma \vdash^0 M : (\overline{A}\,|B)}{\Sigma' \vdash^{-1} M : (\overline{A}\,|B)}(\mathbf{wk})$$

We obtain the following proof tree:

$$\cfrac{\cfrac{\Sigma' \vdash^{-1} M : (\overline{A}\,|B) \qquad \Sigma' \vdash^0 N_1 : A_1 \quad \cdots \quad \Sigma' \vdash^0 N_l : A_l \qquad l = |\overline{A}|}{\Sigma' \vdash^0 M N_1 \cdots N_l : B}(\mathbf{app})}{\Gamma \vdash^0 M N_1 \cdots N_l : B}(\mathbf{wk})$$

where the last weakening rules is applied only if the inclusion $\Sigma' \subseteq \Gamma$ is strict.

We can now conclude using the induction hypothesis on the sequents $\Sigma' \vdash^{-1} M$, $\Sigma' \vdash^0 N_1$, $\ldots$, $\Sigma' \vdash^0 N_l$.

□

Refining the rules of the homogeneous safe λ-calculus

Using the observations that we have just made, we will now derive the rules of the safe λ-calculus with homogeneous type. We want a system of rules generating sequents that verify P_0. Also, it must be able to generate intermediate sequents that do not necessarily satisfy P_0 provided that they can be used to produce *in fine* terms satisfying P_0.

Because of the lemma 2.2.12, we know that the only necessary intermediate sequents are those that either satisfy P_0 or P_{-1}. Hence, we will assume by default that premises of the rules all satisfy P_{-1}. We will see that in some cases, some premise will actually satisfy P_0.

First we define an additional rules expressing the fact that P_0 implies P_{-1}:

$$(\mathbf{seq}) \quad \frac{\Gamma \vdash^0 M : A}{\Gamma \vdash^{-1} M : A}$$

The weakening rule specializes into two rules:

$$(\mathbf{wk^0}) \quad \frac{\Gamma \vdash^0 M : A}{\Gamma, x : B \vdash^0 M : A} \quad \mathsf{ord}(B) \geq \mathsf{ord}(A)$$

$$(\mathbf{wk^{-1}}) \quad \frac{\Gamma \vdash^{-1} M : A}{\Gamma, x : B \vdash^{-1} M : A} \quad \mathsf{ord}(B) \geq \mathsf{ord}(A) - 1$$

Because of the context reduction lemma, any sequent verifying P_{-1} can be obtained by applying the weakening rule $(\mathbf{wk^{-1}})$ or the rule $(\mathbf{seq})$ to another sequent verifying P_0. Therefore, with the exception of these two rules, we only need to use rules whose conclusion sequents verify P_0:

- For the rules $(\mathbf{perm})$, $(\mathbf{const})$ and $(\mathbf{var})$, only the tagging of the sequents changes:

$$(\mathbf{var}) \quad \frac{}{x : A \vdash^0 x : A} \qquad (\mathbf{perm}) \frac{\Gamma \vdash^0 M : B \qquad \sigma(\Gamma) \text{ homogeneous}}{\sigma(\Gamma) \vdash^0 M : B}$$

$$(\mathbf{const}) \frac{}{\vdash^0 b : o^r \to o} \quad b : o^r \to o \in \Sigma$$

- $(\mathbf{abs})$ The definition of the abstraction rule has a side condition expressing the fact that the premise verifies P_0 or P_{-1}. Since this is always true for sequents generated by our new system of rules, we can drop the side condition:

$$(\mathbf{abs}) \quad \frac{\Gamma | \overline{x} : \overline{A} \vdash^{-1} M : B}{\Gamma \vdash^0 \lambda \overline{x} : \overline{A}.M : (\overline{A}, B)}$$

- $(\mathbf{app})$ The application rule has the following form:

$$(\mathbf{app}) \frac{\Gamma \vdash^{-1} M : (\overline{A} | B) \qquad \Gamma \vdash^{-1} N_1 : A_1 \quad \cdots \quad \Gamma \vdash^{-1} N_l : A_l \qquad l = |\overline{A}|}{\Gamma \vdash^0 M N_1 \cdots N_l : B}$$

  Since the first premise verifies P_{-1}, by property 2.2.1(ii) we have:

$$\forall z \in \Gamma : \mathsf{ord}(z) \geq 1 + \mathsf{ord}(\overline{A}) - 1 = \mathsf{ord}(\overline{A}) = \mathsf{ord}(\overline{N})$$

  Hence, all the sequents of the premises but the first must verify P_0. The rule (app) is therefore given by:

$$(\mathbf{app}) \frac{\Gamma \vdash^{-1} M : (\overline{A} | B) \qquad \Gamma \vdash^{0} N_1 : A_1 \quad \cdots \quad \Gamma \vdash^{0} N_l : A_l \qquad l = |\overline{A}|}{\Gamma \vdash^0 M N_1 \cdots N_l : B}$$

- For the application rule $(\mathbf{app^+})$, the type of the sequent in the first premise has the same order as the type of the conclusion premises, therefore since the conclusion verifies P_0, the first premise also verifies P_0. The side-condition implies that that all the other sequents in the premise verify P_0. Moreover the fact that the first premise verifies P_0 ensure that the side-condition always holds. Hence the rule becomes:

$$(\mathbf{app^+}) \frac{\Gamma \vdash^0 M : (\overline{B_1} | \cdots | \overline{B_m} | o) \qquad \Gamma \vdash^0 N_1 : B_{11} \quad \cdots \quad \Gamma \vdash^0 N_l : B_{1l} \qquad l < |\overline{B_1}|}{\Gamma \vdash^0 M N_1 \cdots N_l : (\overline{B} | \cdots | \overline{B_m} | o)}$$

  where $\overline{B_1} = B_{11}, \ldots, B_{1l}, \overline{B}$. Clearly, this rule can be equivalently stated as:

$$\frac{\Gamma \vdash^0 M : A \to B \qquad \Gamma \vdash^0 N : A}{\Gamma \vdash^0 MN : B}$$

The full set of rules is given in table 2.1.

$$(\mathbf{perm})\frac{\Gamma \vdash^0 M : B \qquad \sigma(\Gamma) \text{ homogeneous}}{\sigma(\Gamma) \vdash^0 M : B} \qquad (\mathbf{seq})\ \frac{\Gamma \vdash^0 M : A}{\Gamma \vdash^{-1} M : A}$$

$$(\mathbf{const})\frac{}{\vdash^0 b : o^r \to o}\quad b : o^r \to o \in \Sigma \qquad (\mathbf{var})\ \frac{}{x : A \vdash^0 x : A}$$

$$(\mathbf{wk^0})\ \frac{\Gamma \vdash^0 M : A}{\Gamma, x : B \vdash^0 M : A}\quad \mathsf{ord}(B) \geq \mathsf{ord}(A)$$

$$(\mathbf{wk^{-1}})\ \frac{\Gamma \vdash^{-1} M : A}{\Gamma, x : B \vdash^{-1} M : A}\quad \mathsf{ord}(B) \geq \mathsf{ord}(A) - 1$$

$$(\mathbf{app})\frac{\Gamma \vdash^{-1} M : (\overline{A}|B) \qquad \Gamma \vdash^0 N_1 : A_1 \quad \cdots \quad \Gamma \vdash^0 N_l : A_l \qquad l = |\overline{A}|}{\Gamma \vdash^0 M N_1 \cdots N_l : B}$$

$$(\mathbf{app^+})\ \frac{\Gamma \vdash^0 M : A \to B \qquad \Gamma \vdash^0 N : A}{\Gamma \vdash^0 MN : B}$$

$$(\mathbf{abs})\ \frac{\Gamma|\overline{x} : \overline{A} \vdash^{-1} M : B}{\Gamma \vdash^0 \lambda\overline{x} : \overline{A}.M : (\overline{A}|B)}$$

where $\Gamma|\overline{x} : \overline{A}$ means that the lowest type-partition of the context is $\overline{x} : \overline{A}$.

Tab. 2.1: Alternative rules for the homogeneous safe lambda calculus

2.3 Non homogeneous safe λ-calculus - VERSION B

In section 2.2, we have presented a safe lambda calculus in the setting of homogeneous types. In this section, we give a general notion of safety for the simply typed λ-calculus. The rules we give here do not assume homogeneity of the types.

We call safe terms the simply typed lambda terms that are typable within the following system of formation rules:

2.3.1 Rules

We use a set of sequents of the form $\Gamma \vdash M : A$ where Γ is the context of the term and A is its type. The following system of rules defined the non homogeneous safe λ-calculus:

$$(\mathbf{var})\ \frac{}{x : A \vdash x : A} \qquad (\mathbf{wk})\ \frac{\Gamma \vdash M : A}{\Delta \vdash M : A}\quad \Sigma \subseteq \Gamma$$

$$(\mathbf{app})\frac{\Gamma \vdash M : (A, \ldots, A_l, B) \qquad \Gamma \vdash N_1 : A_1 \quad \ldots \quad \Gamma \vdash N_l : A_l}{\Gamma \vdash M N_1 \ldots N_l : B}\quad \forall y \in \Gamma : \mathsf{ord}(y) \geq \mathsf{ord}(B)$$

$$(\mathbf{abs})\ \frac{\Gamma \cup \overline{x} : \overline{A} \vdash M : B}{\Gamma \vdash \lambda \overline{x} : \overline{A}.M : (\overline{A}, B)}\qquad \forall y \in \Gamma : \mathsf{ord}(y) \geq \mathsf{ord}(\overline{A}, B)$$

Remark:

- $(\overline{A}, B)$ denotes the type $(A_1, A_2, \ldots, A_n, B)$;
- all the types appearing in the rule are not required to be homogeneous. For instance the type $(A, \ldots, A_l, B)$ in the rule $(\mathbf{app})$ may be such that $\mathsf{ord}(A_l) < \mathsf{ord}(B)$;
- the environment $\Gamma \cup \overline{x} : \overline{A}$ is not stratified. In particular, variables in $\overline{x}$ do not necessarily have the same order;
- In the abstraction rule, the side-condition imposes that at least all the variable of the lowest order in the context are abstracted. However other variables can also be abstracted together with the lowest order variables. Moreover there is not constraint on the order on which the variables are abstracted (this contrasts with the homogeneous safe lambda calculus);

Example 2.3.1. Suppose $x : o$, $f : o \to o$ and $\varphi : (o \to o) \to o$ then the term

$$\vdash \lambda x f \varphi.\varphi : o \to (o \to o) \to ((o \to o) \to o) \to (o \to o) \to o$$

is a valid non homogeneous safe term.

Side-remark: safety is preserved by η-expansion: consider the safe term $\Gamma \vdash M : (A_1, \ldots, A_l, o)$ where $(A_1, \ldots, A_l, o)$ is not homogeneous. Then for all $i \in 1..l$ we have $\Gamma \cup \Sigma \vdash x_i : A_i$ for some context $\Sigma = \{x_1 : A_1, \cdots x_l : A_l\}$. Applying $(\mathbf{app})$ we then get that $\Gamma, \Sigma \vdash M x_1 \ldots x_l$ is valid. Hence the η-expansion of M is safe:

$$\Gamma \vdash \lambda x_1 : A_1, \cdots x_l : A_l.\ M\ x_1 \ldots x_l$$

Lemma 2.3.2 (Context reduction). *If* $\Gamma \vdash M : B$ *is a valid judgment then*

1. $fv(M) \vdash M : B$
2. *every variable in* Γ occuring free in M *has order at least ord*(M).

where $fv(M)$ *denotes the context constituted of the free variables occurring in* M.

Proof. (i) Suppose some variable x in Γ does not occur free in M, then the x must have been introduced in the context by the weakening rule. Hence M must also be typable with the rules of the safe lambda calculus but with the variable x removed from the context. (ii) An easy structural induction. □

2.3.2 Substitution in the safe lambda calculus

The traditional notion of substitution, on which the λ-calculus is based on, is defined as follows:

Definition 2.3.3 (Substitution)**.**

$$\begin{array}{rcl}
c\,[t/x] & = & c \quad \text{where } c \text{ is a } \Sigma\text{-constant} \\
x\,[t/x] & = & t \\
y\,[t/x] & = & y \quad \text{for } x \neq y, \\
(M_1 M_2)\,[t/x] & = & (M_1\,[t/x])(M_2\,[t/x]) \\
(\lambda x.M)\,[t/x] & = & \lambda x.M \\
(\lambda y.M)\,[t/x] & = & \lambda z.M\,[z/y]\,[t/x] \,\text{where } z \text{ is a fresh variable and } x \neq y
\end{array}$$

In the setting of the safe lambda calculus, the notion of substitution can be simplified. Indeed, similarly to what we observe in the homogeneous safe λ-calculus, we remark that for safe λ-terms there is no need to rename variables when performing substitution:

Lemma 2.3.4 (No variable capture lemma)**.** *There is no variable capture when performing substitution on a safe term.*

This is the equivalent of lemma 2.2.4 for the non-homogeneous case. The proof (which does not rely on homogeneity) is identical. Consequently, in the safe lambda calculus setting, we can omit to to rename variable when performing substitution and the equation

$$(\lambda x.M)\,[t/y] = \lambda z.M\,[z/x]\,[t/y] \,\text{where } z \text{ is a fresh variable}$$

becomes

$$(\lambda x.M)\,[t/y] = \lambda x.M\,[t/y]\,.$$

Unfortunately, this notion of substitution is still not adequate for the purpose of the safe simply-typed lambda calculus. The problem is that performing a single β-reduction on a safe term will not necessarily produce another safe term.

The solution consist in reducing several consecutive β-redex at the same time until we obtain a safe term. To achieve this, we introduce the *simultaneous substitution*, a generalization of the standard substitution given in definition 2.3.3.

Definition 2.3.5 (Simultaneous substitution)**.** We note $\left[\overline{N}/\overline{x}\right]$ for $[N_1 \ldots N_n / x_1 \ldots x_n]$:

$$\begin{array}{rcl}
c\left[\overline{N}/\overline{x}\right] & = & c \quad \text{where } c \text{ is a } \Sigma\text{-constant} \\
x_i\left[\overline{N}/\overline{x}\right] & = & N_i \\
y\left[\overline{N}/\overline{x}\right] & = & y \quad \text{if } y \neq x_i \text{ for all } i, \\
(MN)\left[\overline{N}/\overline{x}\right] & = & (M\left[\overline{N}/\overline{x}\right])(N\left[\overline{N}/\overline{x}\right]) \\
(\lambda x_i.M)\left[\overline{N}/\overline{x}\right] & = & \lambda x_i.M\,[N_1 \ldots N_{i-1}N_{i+1} \ldots N_n / x_1 \ldots x_{i-1}x_{i+1} \ldots x_n] \\
(\lambda y.M)\left[\overline{N}/\overline{x}\right] & = & \lambda z.M\,[z/y]\left[\overline{N}/\overline{x}\right] \\
 & & \text{where } z \text{ is a fresh variables and } y \neq x_i \text{ for all } i
\end{array}$$

In general, variable captures should be avoided, this explains why the definition of simultaneous substitution uses auxiliary fresh variables. However in the current setting, lemma 2.3.4 can clearly be transposed to the simultaneous substitution therefore there is no need to rename variable.

The notion of substitution that we need is therefore the *capture permitting simultaneous substitution* defined as follow:

Definition 2.3.6 (Capture permitting simultaneous substitution)**.** We use the notation $\left[\overline{N}/\overline{x}\right]$ for $[N_1 \ldots N_n/x_1 \ldots x_n]$:

$$\begin{array}{rcl}
c\left[\overline{N}/\overline{x}\right] & = & c \quad \text{where } c \text{ is a } \Sigma\text{-constant} \\
x_i\left[\overline{N}/\overline{x}\right] & = & N_i \\
y\left[\overline{N}/\overline{x}\right] & = & y \quad \text{where } x \neq y_i \text{ for all } i, \\
(M_1 M_2)\left[\overline{N}/\overline{x}\right] & = & (M_1\left[\overline{N}/\overline{x}\right])(M_2\left[\overline{N}/\overline{x}\right]) \\
(\lambda x_i.M)\left[\overline{N}/\overline{x}\right] & = & \lambda x_i.M\left[N_1 \ldots N_{i-1} N_{i+1} \ldots N_n/x_1 \ldots x_{i-1} x_{i+1} \ldots x_n\right] \\
(\lambda y.M)\left[\overline{N}/\overline{x}\right] & = & \lambda y.M\left[\overline{N}/\overline{x}\right] \text{ where } y \neq x_i \text{ for all } i \qquad (\star)
\end{array}$$

The symbol $(\star)$ identifies the equation that changed compared to the previous definition.

Lemma 2.3.7 (Substitution preserves safety)**.**

$$\Gamma \cup \overline{x} : \overline{A} \vdash M : T \quad \textit{and} \quad \Gamma \vdash N_k : B_k,\ k \in 1..n \qquad \textit{implies} \qquad \Gamma \vdash M[\overline{N}/\overline{x}] : T$$

Proof. Suppose that $\Gamma \cup \overline{x} : \overline{A} \vdash M : T$ and $\Gamma \vdash N_k : B_k$ for $k \in 1..n$.

We prove $\Gamma \vdash M[\overline{N}/\overline{x}]$ by induction on the size of the proof tree of $\Gamma \cup \overline{x} : \overline{A} \vdash M : T$ and by case analysis on the last rule used. We just give the detail for the abstraction case. Suppose that the property is verified for terms whose proof tree is smaller than M. Suppose $\Gamma \cup \overline{x} : \overline{A} \vdash \lambda\overline{y} : \overline{C}.P : (\overline{C}|D)$ where $\Gamma \cup \overline{x} : \overline{A} \cup \overline{y} : \overline{C} \vdash P : D$, then by the induction hypothesis $\Gamma \cup \overline{y} : \overline{C} \vdash P\left[\overline{N}/\overline{x}\right] : D$. Applying the rule (**abs**) gives $\Gamma \vdash \lambda\overline{y} : \overline{C}.P\left[\overline{N}/\overline{x}\right]$. □

2.3.3 Safe-redex

In the simply-typed lambda calculus a redex is a term of the form $(\lambda x.M)N$. We generalize this definition to the safe lambda calculus:

Definition 2.3.8 (Safe redex)**.** We call safe redex a term of the form $(\lambda\overline{x}.M)N_1 \ldots N_l$ such that:

- $\Gamma \vdash (\lambda\overline{x}.M)N_1 \ldots N_l$
- the variable $\overline{x} = x_1 \ldots x_n$ are abstracted altogether by one occurrence of the rule (**abs**) in the proof tree.
- The terms $(\lambda\overline{x}.M)$, N_1, N_l are applied together at once using the (**app**) rule :

$$\frac{\Sigma \vdash \lambda\overline{x}.M \quad \Sigma \vdash N_1 \quad \ldots \quad \Sigma \vdash N_l}{\Sigma \vdash (\lambda\overline{x}.L)N_1 \ldots N_l}(\mathbf{app})$$

  Consequently each N_i is safe.
- $l \leq n$

Note that the condition $l \leq n$ in the definition is not too restrictive because if $l > n$ then the application rule is in fact "wider" than the abstration and it can be replaced by an application of exactly n terms followed by another application for the remaining terms $N_{n+1}, \ldots, N_l$.

Define the safe reduction: Consider the safe-redex $(\lambda\overline{x}.M)N_1 \ldots N_l$, it reduces to $\lambda x_l \ldots x_n.M\left[N_1 \ldots N_l/x_1 \ldots x_l\right]$. The relation β_s is defined on safe-redex: $(s \mapsto t) \in \beta_s$ iff $s \equiv (\lambda\overline{x}.M)N_1 \ldots N_l$ is a safe redex and $t \equiv \lambda x_l \ldots x_n.M\left[N_1 \ldots N_l/x_1 \ldots x_l\right]$

Show that $\rightarrow_{\beta_s} \subseteq \rightarrow_\beta^*$.

Lemma 2.3.9. *A safe redex reduces to a safe term.*

This lemma, which is a consequence of lemma 2.3.7, is the counterpart of lemma 2.2.9 in the homogeneous safe lambda calculus. Their proofs are identical.

2.3.4 Examples

Example 1

Let $f, g : o \to o$, $x, y : o \to o$, $\Gamma = g : o \to o$ and $\Gamma' = g : o \to o, y : o$. The term $(\lambda fx.x)gy$ is safe. One possible proof tree is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash \lambda fx.x} \qquad \dfrac{}{\Gamma \vdash g}}{\Gamma \vdash (\lambda fx.x)g}(\mathbf{app})}{\Gamma' \vdash (\lambda fx.x)g}(\mathbf{wk}) \qquad \dfrac{}{\Gamma' \vdash y}}{\Gamma' \vdash (\lambda fx.x)gy}(\mathbf{app})$$

Here is another proof for the same judgment:

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash \lambda fx.x}}{\Gamma' \vdash \lambda fx.x}(\mathbf{wk}) \qquad \dfrac{}{\Gamma' \vdash g} \qquad \dfrac{}{\Gamma' \vdash y}}{\Gamma' \vdash (\lambda fx.x)gy}(\mathbf{app})$$

We see on this particular example that there may exist different proof tree deriving the same judgment.

Example 2 - Damien Sereni SCT counter-example

In Sereni [2005], the following counter-example is given to show that not all simply-typed terms are size-change terminating (see Lee et al. [2001] for a definition of size-change termination):

$$E = (\lambda a.a(\lambda b.a(\lambda cd.d)))(\lambda e.e(\lambda f.f))$$

where:

$$\begin{array}{lcl} a & : & ((\tau \to \tau) \to \mu \to \mu) \to \mu \to \mu \\ b & : & \tau \to \tau \\ c & : & \tau \to \tau \\ d & : & \mu \\ e & : & (\tau \to \tau) \to \mu \to \mu \\ f & : & \tau \end{array}$$

2.3.5 Particular case of homogeneously-safe lambda terms

In this section, we will refine the rules of the non-homogenous safe lambda calculus to the homogeneous case.

We recall the definition of type homogeneity given in section 2.1.1: a type $(A_1, A_2, \ldots A_n, o)$ is said to be homogeneous whenever $\mathsf{ord}(A_1) \geq \mathsf{ord}(A_2) \geq \ldots \geq \mathsf{ord}(A_n)$ and each of the A_i are homogeneous. A term is said homogeneous if its type is homogeneous.

Safety in the sense of Knapik et al. (Knapik et al. [2002]) is defined for homogeneous recursion scheme. A recursion scheme is homogeneous if

(i) the Σ constants are of order 1 at most and therefore have homogeneous types;

(ii) the variables in the alphabet Δ have homogeneous types,

(iii) and every recursion equations of a safe recursion scheme has a homogeneous type.

We now impose the same restrictions on the terms generated by our system rules. The first and second restriction imposes that every applicative terms in $\mathcal{T}(\Delta)$ must be homogeneous.

Abstractions in simply-typed terms correspond to recursion equations of recursion schemes, therefore by imposing restriction (iii) to our calculus, all the sequents generated by the rules must be of homogeneous type. We say that a term is *homogeneously-safe* if its type is homogeneous and there is a proof tree showing its safety where all the sequents of the proof tree are of homogenous type.

We say that $\Gamma \vdash M : A$ verifies P_i for $i \in \mathbb{Z}$ if all the variables in Γ have orders at least $\mathsf{ord}(A) + i$. Lemma 2.3.2 can then be restated as follows:

Lemma 2.3.10 (Context reduction). *If $\Gamma \vdash M : A$ then the sequent $fv(M) \vdash M : A$ is valid and satisfies P_0.*

We now prove that if we impose the homogeneity of types, the set of rules of the non-homogenous safe λ-calculus and the rules of table 2.1 are equivalent. We recall that in the system of rules of table 2.1, if the sequent $\Gamma \vdash^i M : A$ is valid for some $i \in \mathbb{Z}$ then all the variables in Γ have orders at least $\mathsf{ord}(A) + i$.

Proposition 2.3.11 (Homogeneity restriction). *Let $k \in \{0, -1\}$. The sequent $\Gamma \vdash M : A$ is valid, homogeneously-safe and satisfies P_k if and only if the sequent $\Gamma \vdash^k M : A$ is valid in the system of rules of table 2.1.*

Proof. If: An easy induction by case analysis on the last rule used to derive $\Gamma \vdash^0 M : A$.

Only if: Consider an homogeneously-safe term $\Gamma \vdash S : T$ satisfying P_0. We proceed by induction and case analysis on the last rule used to derive $\Gamma \vdash S : T$. We only give the details for the application and abstraction case:

- **Abstraction.** We recall the abstraction rule:

$$(\mathbf{abs}) \quad \frac{\Gamma \cup \overline{x} : \overline{A} \vdash M : B}{\Gamma \vdash \lambda \overline{x} : \overline{A}.M : (\overline{A}, B)} \qquad \forall y \in \Gamma : \mathsf{ord}(y) \geq \mathsf{ord}(\overline{A}, B)$$

  Type homogeneity requires that for all i: $\mathsf{ord}(x_i) = \mathsf{ord}(A_i) \geq \mathsf{ord}(B) - 1$. Therefore the premise of the rule verifies P_{-1}. Using the induction hypothesis we have:

$$\Gamma \cup \overline{x} : \overline{A} \vdash^{-1} M : B. \tag{2.2}$$

  We now partition the context Γ according to the order of the variables. The partitions are written in decreasing order of type order. The notation $\Gamma | \overline{x} : \overline{A}$ means that $\overline{x} : \overline{A}$ is the lowest partition of the context. We also use the notation $(\overline{A}|B)$ to denote the homogeneous type $(A_1, A_2, \ldots A_n, B)$ where $\mathsf{ord}(A_1) = \mathsf{ord}(A_2) = \ldots \mathsf{ord}(A_n) \geq \mathsf{ord}(B) - 1$.

  Suppose that we abstract a single variable x, then in order to respect the side condition, we need to abstract all variables of order lower or equal to $\mathsf{ord}(x)$. In particular we need to abstract the partition of the order of x. Moreover to respect type homogeneity, we need to abstract variables of the lowest order first.

  Hence $\overline{x}$ must contain at least the lowest variable partition (all the variables of the lowest order). If $\overline{x}$ contains variables of different order, then the instance of the abstraction rule can be replaced by consecutive instances of the abstraction rule, one for each of the different order of variable present in $\overline{x}$. Therefore, without loose of generality, we can assume that $\overline{x}$ only contains the lowest partition, that is to say, $\overline{x}$ *is* the lowest partition.

  The sequent 2.2 therefore becomes:

$$\Gamma | \overline{x} : \overline{A} \vdash^{-1} M : B.$$

  And we can conclude by applying the abstraction rule of table 2.1:

$$(\mathbf{abs}) \quad \frac{\Gamma | \overline{x} : \overline{A} \vdash^{-1} M : B}{\Gamma \vdash^0 \lambda \overline{x} : \overline{A}.M : (\overline{A}|B)}$$

- **Application.** We recall the application rule:

$$(\mathbf{app})\ \frac{\Gamma \vdash M : (A, \ldots, A_l, B) \qquad \Gamma \vdash N_1 : A_1 \quad \ldots \quad \Gamma \vdash N_l : A_l}{\Gamma \vdash M N_1 \ldots N_l : B} \quad \forall y \in \Gamma : \mathsf{ord}(y) \geq \mathsf{ord}(B)$$

  The term in the conclusion is homogeneously safe therefore the term in the first premise must be of homogeneous type. This implies that $\mathsf{ord}(A_1) \geq \ldots \geq \mathsf{ord}(A_l) \geq \mathsf{ord}(B) - 1$. Furthermore, we can make the assumption that $\mathsf{ord}(A_1) = \ldots = \mathsf{ord}(A_l) = \mathsf{ord}(\overline{A})$ (it is always possible to replace an instance of the application rule by several consecutive instances of this kind).

  By lemma 2.3.10, we have for all $i \in 1..l$:

$$fv(N_i) \vdash N_i : A_i \text{ is valid and satisfies } P_0.$$

  Let $\Sigma = \bigcup_{i=1..p} fv(N_i)$. Since $\mathsf{ord}(A_1) = \ldots = \mathsf{ord}(A_l)$, by applying the weakening rule we get for all $i \in 1..p$:

$$\Sigma \vdash N_i : A_i \text{ is valid and satisfies } P_0.$$

  Applying lemma 2.3.10 to the term M we have:

$$fv(M) \vdash M : (A_1, \ldots, A_l, B) \text{ is valid and satisfies } P_0.$$

  The weakening rule (**wk**) then gives: $fv(M) \cup \Sigma \vdash M : (A_1, \ldots, A_l, B)$. Since $\Sigma \vdash N_i : A_i$ satisfies P_0, for any $z \in \Sigma$ we have $\mathsf{ord}(z) \geq \mathsf{ord}(A_i) = \mathsf{ord}((A_1, \ldots, A_l, B)) - 1$. Hence:

$$fv(M) \cup \Sigma \vdash M : (A_1, \ldots, A_l, B) \text{ is valid and satisfies } P_{-1} \tag{2.3}$$

  Similarly, for all $i \in 1..p$, the weakening rule gives $fv(M) \cup \Sigma \vdash N_i : A_i$. Since $fv(M) \vdash M : (A_1, \ldots, A_l, B)$ satisfies P_0, for any $z \in fv(M)$ we have $\mathsf{ord}(z) \geq \mathsf{ord}(M) \geq \mathsf{ord}(A_i)$. Hence:

$$fv(M) \cup \Sigma \vdash N_i : A_i \text{ is valid and satisfies } P_0 \tag{2.4}$$

  Let us define the context $\Sigma' = fv(M) \cup \Sigma$. Using the induction hypthesis on equation 2.3 and 2.4 we have:

$$\Sigma' \vdash^{-1} M : (A_1, \ldots, A_l, B) \qquad \text{and} \qquad \Sigma' \vdash^{0} N_i : A_i \text{ for all } i \in 1..l$$

  We consider the following two sub-cases:

  - If $A_1, \ldots A_l$ forms a type partition then we can apply rule (**app**) of table 2.1:

$$\frac{\Sigma' \vdash^{-1} M : \overline{A}|B \qquad \Sigma' \vdash^{0} N_1 : A_1 \quad \ldots \quad \Sigma' \vdash^{0} N_l : A_l \quad l = |\overline{A}|}{\Sigma' \vdash^{0} M N_1 \ldots N_l : B} \ (\mathbf{app})$$

    where $\overline{A} = A_1, \ldots A_l$.
  - Suppose that $A_1, \ldots A_l$ do not form a type partition, then we have

$$\mathsf{ord}(A_1) = \ldots = \mathsf{ord}(A_l) = \mathsf{ord}(B) - 1.$$

    The side condition in the original instance of the application rule says:

$$\forall y \in \Gamma : \mathsf{ord}(y) \geq \mathsf{ord}(B) = 1 + \mathsf{ord}(A_l) = \mathsf{ord}((A_1, \ldots, A_l, B)) = \mathsf{ord}(M)$$

    In particular the variables in $\Sigma' \subseteq \Gamma$ are of order greater than $\mathsf{ord}(M)$ and consequently the sequent $\Sigma' \vdash M : (A, \ldots, A_l, B)$ verifies P_0. The induction hypothesis then gives:

$$\Sigma' \vdash^{0} M : (A, \ldots, A_l, B)$$

By using l consecutive instances of the rules $(\mathbf{app}^+)$ from table 2.1 we get:

$$\cfrac{\cfrac{\cfrac{\Sigma' \vdash^0 M : (A_1, \ldots, A_l, B) \qquad \Sigma' \vdash^0 N_1 : A_1}{\Sigma' \vdash^0 MN_1 : (A_2, \ldots, A_l, B)}\ (\mathbf{app}^+)}{\vdots}\ (\mathbf{app}^+)}{\Sigma' \vdash^0 MN_1 \ldots N_l : B}\ (\mathbf{app}^+)$$

In both cases we have proved that $\Sigma' \vdash^0 MN_1 \ldots N_l : B$ is a valid sequent.

Clearly $\Sigma' \subseteq \Gamma$ since $fv(M) \subseteq \Gamma$ and $\Sigma' = \bigcup_{i \in 1..l} fv(N_i) \subseteq \Gamma$. Suppose that $\Sigma' = \Gamma$ then the proof is done. Suppose that $\Sigma' \subset \Gamma$, then the side condition in the original instance of the application rule says that all the variables in Γ have order greater or equal to $\mathsf{ord}(B)$, we can therefore apply the weakening rule $(\mathbf{wk^0})$ of table 2.1 exactly $|\Gamma \setminus \Sigma'|$ times and get:

$$\frac{\Sigma' \vdash^0 MN_1 \ldots N_l : B}{\Gamma \vdash^0 MN_1 \ldots N_l : B}\ (\mathbf{wk^0})$$

$\square$

2.4 Game semantics of safe λ-terms

In this section we will prove that the safety condition of section 2.2 leads to am economy of pointer in the game semantics: for safe λ-terms the pointers from the game semantics can be reconstructed uniquely from the moves of the play.

The example of section 1.2.7 gives the intuition. Remember that in order to distinguish the terms M_1 and M_2, we introduced pointers in strategies. In the safe λ-calculus this ambiguity disappears because M_2 is not a safe term. Indeed, in the sub-term $f(\lambda y.x)$, the free variable x has the same order as y but x is not abstracted together with y.

We fix Σ, a set of constants of order at most 1.

2.4.1 η-long normal form and computation tree

The η-expansion of $M : A \rightarrow B$ is defined to be the term $\lambda x.Mx : A \rightarrow B$ where $x : A$ is a fresh variable. It is easy to check that if M is safe then $\lambda x.Mx$ is also safe.

A term $M : (A_1, \ldots, A_n, o)$ can be expanded in several steps into $\lambda\varphi_1 \ldots \varphi_l.M\varphi_1 \ldots \varphi_l$ where the $\varphi_i : A_i$ are fresh variables.

The η-normal form of a term is obtained by hereditarily η-expanding every sub-term occurring at an operand position:

Definition 2.4.1 (η-long normal form)**.** A term is either an abstraction or it can be written uniquely as $s_0 s_1 \ldots s_m$ where $m \geq 0$ and s_0 is a variable, a constant or an abstraction.

The η-long normal form of a term M is denoted $\lceil M \rceil$ and is defined as follows:

$$\begin{aligned}
\lceil x s_1 \ldots s_m : (A_1, \ldots, A_n, o) \rceil &= \lambda\overline{\varphi}.x \lceil s_1 \rceil \lceil s_2 \rceil \ldots \lceil s_m \rceil \lceil \varphi_1 \rceil \ldots \lceil \varphi_n \rceil \\
\lceil (\lambda x.s_0) s_1 \ldots s_m \rceil &= (\lambda x.\lceil s_0 \rceil) \lceil s_1 \rceil \lceil s_2 \rceil \ldots \lceil s_m \rceil
\end{aligned}$$

where $m, n \geq 0$ and x is either a variable or a constant.

For $n = 0$, the first clause in the definition gives:

$$\lceil x s_1 \ldots s_m : o \rceil = \lambda.x \lceil s_1 \rceil \lceil s_2 \rceil \ldots \lceil s_m \rceil$$

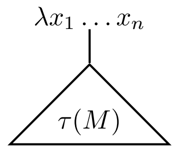
and in that case, the symbol λ is called *dummy* lambda. We do not omit it since it plays an important role in the correspondence with game semantics.

The η-long normal form appeared in [Jensen and Pietrzykowski, 1976] under the name *long reduced form* and in [Huet, 1975] under the name *η-normal form.* It was then investigated in [Huet, 1976] under the name *extensional form.*

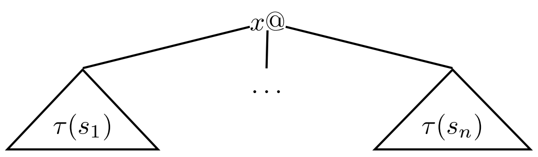
A term can be represented by a tree defined formally by induction on the structure of its η-long normal form as follows:

Definition 2.4.2 (Computation tree)**.** The computation tree associated to the term s is noted $\tau(s)$. It is obtained by applying the following rules inductively *on the η-long normal form* of s. In the following x is either a variable or a constant.

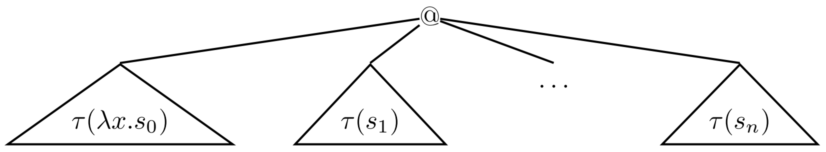
- the tree for x is the single leaf x.
- for $n \geq 0$, the tree for $\lambda x_1 \ldots x_n.M$ where M is not an abstraction is:

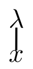
$\tau(\lambda x_1 \ldots x_n.M) =$ 

- for $n \geq 1$, the tree for $xs_1 \dots s_n$ is:

$\tau(xs_1 \dots s_n) =$


- the tree for $(\lambda x.s_0)s_1 \dots s_n$ is:

$\tau((\lambda x.s_0)s_1 \dots s_n) =$


Example: if x is a variable or a constant then $\tau(\lambda.x) =$ 

The nodes (and leaves) of the tree are of three kinds:

- λ-node labeled $\lambda\overline{x}$. A λ-node represents several consecutive abstractions of variables.
- application node labeled @
- operator-application nodes labeled x@ where the operator x is either a variable or a constant.

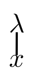
A sub-tree of the computation tree represents a λ-term. We define the map $\kappa : N \rightarrow \mathcal{T}$ where N denotes the set of nodes and leaves of the computation tree $\tau(s)$ and $\mathcal{T}$ denotes the set of λ-terms. κ associates to any node n of the tree the λ-term $\kappa(n)$ that is represented by the sub-tree of $\tau(s)$ rooted at n. In particular if r is the root of the tree $\tau(s)$ then $\kappa(r) = \lceil s \rceil$.

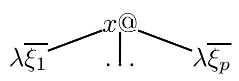
Definition 2.4.3 (Node order)**.** The order of a lambda-node n labelled $\lambda\overline{\xi}$ is defined as follows:

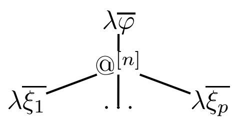
$$\mathsf{ord}(n) = \max_{i=1..n} \mathsf{ord}(\xi_i)$$

The order of n labelled x or x@ is the order of the variable or constant x: $\mathsf{ord}(n) = \mathsf{ord}(x)$. The order of node labelled @ is 0.

Consider the computation tree $\tau(s)$ of a term s in η-long normal form. Then:

- One can check that nodes at even level are abstraction node and nodes at odd level are either application nodes, operator-application nodes, variable or constant nodes (the root level being numbered 0).
- Suppose that a variable x occurs in s. The corresponding node in the tree has of one of the two following forms:
  -  where $\mathsf{ord}(x) = 0$

  -  where $\mathsf{ord}(x) > 0$ and $x : (A_1, \dots, A_p, o)$

- Moreover for any abstraction node  we have $\mathsf{ord}(\kappa(@^{[n]})) = 0$

$\lambda\overline{\varphi}$
$@^{[n]}$
$\lambda\overline{\xi_1}$
$\lambda\overline{\xi_p}$

Pointers and justified sequence of nodes

Definition 2.4.4 (Binder)**.** Let n be a node of the computation tree labelled x or $x@$. We say a node n is bound by the node m if m is the closest node in the path from n to the root of the tree such that m is labelled $\lambda\overline{\xi}$ with $x \in \overline{\xi}$. m is called the binder of n

Definition 2.4.5 (Enabling)**.** The enabling relation noted $\vdash$ over the set of nodes of the computation tree is defined as follows:

We write $m \vdash n$ and we say that the node n is enabled by m or that m enables n if and only if

- the node n is labelled x or $x@$ and m is the binder of n;
- or n is a lambda node labelled $\lambda\overline{\xi}$ and m is the parent node of n (labelled $@$ or $x@$ for some variable or Σ-constant x).

Definition 2.4.6 (Justified sequence of nodes)**.** A *justified sequence of nodes* is an alternating sequence of lambda and non lambda nodes from the computation tree $\tau(M)$ together with pointers. Each node n of the sequence that is either a lambda-node, a ξ-node or a $\xi@$-node where ξ is a variable bound in M, and that is not the root of the computation tree has a pointer to a previous node m in the sequence such that $m \vdash n$.

If n points to m we say that m justifies n and we note it:

$$\begin{tikzcd} m & \cdot \ldots \cdot & n \arrow[ll, bend right] \end{tikzcd}$$

We sometimes add a label to the edge to specify that either node n is labelled by the i^{th} variable abstracted in the lambda node m or that n is the i^{th} child of the non-lambda node m. Hence the pointers in a justified sequence of nodes must be of one of the following five forms:

$$\begin{tikzcd} \lambda\overline{\xi} & \cdot \ldots \cdot & \xi_i@ \arrow[ll, bend right, "i"'] \end{tikzcd} \qquad \begin{tikzcd} \lambda\overline{\xi} & \cdot \ldots \cdot & \xi_i \arrow[ll, bend right, "i"'] \end{tikzcd}$$

$$\begin{tikzcd} @ & \cdot \ldots \cdot & \lambda\overline{\xi} \arrow[ll, bend right, "i"'] \end{tikzcd} \qquad \begin{tikzcd} x@ & \cdot \ldots \cdot & \lambda\overline{\xi} \arrow[ll, bend right, "i"'] \end{tikzcd} \qquad \begin{tikzcd} f@ & \cdot \ldots \cdot & \lambda\overline{\xi} \arrow[ll, bend right, "i"'] \end{tikzcd}$$

for some integer i, variable x and Σ-constant f.

With the following conventions:

- the first child of a @ node is numbered 0,
- the first child of an application nodes of type $x@$ is numbered 1,
- the first variable in $\overline{\xi}$ is numbered 1.

Note that justified sequences are also defined for non closed terms (a nodes occurring in a justified sequence and labelled by a variable free in M has no pointer).

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 for all $i \in 0..p-1$, n_i points to n_{i+1}.

If N is a set of nodes and s a justified sequence of nodes then we note $s \upharpoonright N$ the subsequence of s obtained after removing the nodes that are not hereditarily justified by nodes in N. This subsequence is also a justified sequence of nodes.

Let r be the root of the tree $\tau(M)$ then no node in the justified sequence $s \upharpoonright \{r\}$ is labelled by a free variable in M.

Definition 2.4.7 (Equality of justified sequence)**.** We note $s = t$ to denotes that the two justified sequences t and s are the equal (having same nodes and same pointers).

Definition 2.4.8 (P-view of justified sequence of nodes)**.** The P-view of a justified sequence of nodes t of $\tau(M)$ noted $\ulcorner t \urcorner$ is defined as follows:

$$\begin{array}{rcl}
\ulcorner \epsilon \urcorner & = & \epsilon \\
\ulcorner s \cdot n \urcorner & = & \ulcorner s \urcorner \cdot n \\
\ulcorner s \cdot \overset{\curvearrowleft}{m \cdot \ldots \cdot \lambda\overline{\xi}} \urcorner & = & \ulcorner s \urcorner \cdot \overset{\curvearrowleft}{m \cdot \lambda\overline{\xi}} \\
\ulcorner s \cdot r \urcorner & = & r
\end{array}$$

where r is the root of the tree $\tau(M)$ and n ranges over non-lambda nodes (either @, f@, f, x or x@ for some Σ-constant f or variable x).

In the second clause, the pointer associated to n is preserved from the left-hand side to the right-hand side: 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 corresponding occurrence of the node in $\ulcorner s \urcorner$.

In the third equality, the pointer associated to m is also preserved.

Similarly we define the O-view, the dual of the P-view:

Definition 2.4.9 (O-view of justified sequence of nodes)**.** The O-view of a justified sequence of nodes t of $\tau(M)$ noted $\llcorner t \lrcorner$ is defined as follows:

$$\begin{array}{rcl}
\llcorner \epsilon \lrcorner & = & \epsilon \\
\llcorner s \cdot \lambda\overline{\xi} \lrcorner & = & \llcorner s \lrcorner \cdot \lambda\overline{\xi} \\
\llcorner s \cdot \overset{\curvearrowleft}{m \cdot \ldots \cdot n} \lrcorner & = & \llcorner s \lrcorner \cdot \overset{\curvearrowleft}{m \cdot n} \\
\llcorner s \cdot n' \lrcorner & = & n'
\end{array}$$

where n ranges over non-lambda nodes x or x@ where x is a variable bound in M. and n' ranges over non-lambda nodes with no pointer (either @, f@, f, x or x@ for some Σ-constant f or variable x free in M).

The pointer associated to $\lambda\overline{\xi}$ in the second equality as well as the pointer associated to m in the third equality are preserved from the left-hand side to the right-hand side of the equalities.

Definition 2.4.10 (Visibility)**.**
A justified sequence of nodes s satisfies *P-visibility* if every variable node with pointer points to a node occurring in the P-view a that point.

Dually, s satisfies *O-visibility* if every lambda node occurring in it that has a pointer points to a node in the O-view a that point.

Property 2.4.11. The P-view (resp. O-view) of a justified sequence verifying P-visibility (resp. O-visibility) is a well-formed justified sequence verifying P-visibility (resp. P-visibility).

This is proved by an easy induction.

Traversal of the computation tree

The evaluation of term is performed by traversing the computation tree in a particular way called *traversal*. Intuitively, a *traversal* is a justified sequence of nodes of the computation tree such that each node of the sequence indicates which step is taken during the evaluation of the term.

A *traversal* is defined to be the prefix of a maximal-traversal. A maximal-traversal is a traversal that cannot be extended by visiting another node. It is formally defined as follows:

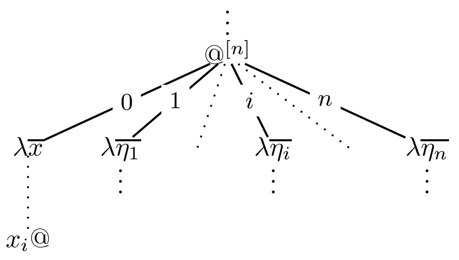
Definition 2.4.12 (Maximal-traversal)**.** A maximal-traversal always starts with the root of the tree. This marks the beginning of the computation. The exploration of the tree then progresses as follows:

- (Lam) After visiting a lambda-node, a maximal-traversal must visit its child node.

- (App) After a node labelled @, a maximal-traversal must visit the node corresponding to the operator of the application: the child node number 0.
- (Sig^0) After a node labelled f where f is an order 0 Σ-constant a maximal-traversal stops.
- (Sig) After a node labelled f@ where f is an order 1 Σ-constant, a maximal-traversal must visit one of the children nodes of f@ (which are all labelled λ).
- (Var) After a node labelled x or x@ where x is either a variable occuring free in M or bound by the root of the tree $\tau(M)$, a maximal-traversal must visit one of the children nodes.
- (Var') If the last node visited is x_i (or x_i@) where the node x_i is bound by a node $\lambda\overline{x}$ different from the root of $\tau(M)$, then the next node visited in a maximal-traversal corresponds to the term that would be substituted for x_i if the β-redex in term M were reduced.

  The binding node $\lambda\overline{x}$ must have been visited previously in the traversal. Since $\lambda\overline{x}$ is not the root of the tree, it must be justified by some previous node in the traversal, therefore there must be an application node n preceding $\lambda\overline{x}$ in the traversal. We do a case analysis on n:

  - Suppose n is labelled by a constant f@ or a ground type variable y. This implies that the node visited after n is labelled λ. However here the following node is $\lambda\overline{x}$, therefore this case is impossible.
  - Suppose n is an application node @ then $\lambda\overline{x}$ is the first child node of n and n has exactly $|\overline{x}|+1$ children nodes:


$@^{[n]}$
0
1
i
n
$\lambda\overline{x}$
$\lambda\overline{\eta_1}$
$\lambda\overline{\eta_i}$
$\lambda\overline{\eta_n}$
$x_i@$

    The next step of the maximal-traversal is a jump to the i^{th} child of @ (the node corresponding to the term that would be substituted for x_i if the β-reduction was performed). The traversal has the following form:

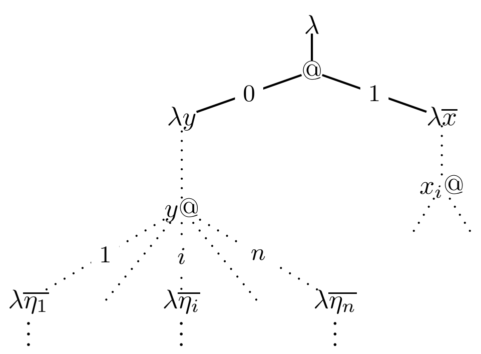
$$\begin{tikzcd}[column sep=tiny] \ldots @^{[n]} & \cdot\, \lambda\overline{x} & \cdot \ldots \cdot & x_i@ \arrow[ll, bend right, "i"'] & \cdot\, \lambda\overline{\eta_i} \arrow[llll, bend right, "i"'] & \cdot \ldots \end{tikzcd}$$

  - Suppose n is labelled y@. This means that this occurrence of the variable y is substituted by the term $\kappa(\lambda\overline{x})$ during the evaluation of the term. Hence y must be a non-free variable bound by a node different from the root (otherwise y would not be substituted during the evaluation of the term).

    Therefore during reduction, the occurrence of the variable x_i will be substituted by the term represented by the i^{th} child node of node y@. Hence the next node in the traversal is the i^{th} child of y@ noted $\lambda\overline{\eta_i}$:

$$\begin{tikzcd}[column sep=tiny] \ldots \cdot y@^{[n]} & \cdot\, \lambda\overline{x} & \cdot \ldots \cdot & x_i@ \arrow[ll, bend right, "i"'] & \cdot\, \lambda\overline{\eta_i} \arrow[llll, bend right, "i"'] & \cdot \ldots \end{tikzcd}$$

Example 2.4.13. Consider the following tree:



The following justified sequence is a traversal of the tree:

$$\begin{tikzcd}[column sep=tiny]
\lambda \cdot & @ \cdot & \lambda y \cdot & \ldots \cdot & y@ \cdot \arrow[ll, bend right, "1"'] & \lambda\overline{x} \cdot \arrow[llll, bend right, "1"'] & \ldots \cdot & x_i@ \cdot \arrow[ll, bend right, "i"'] & \lambda\overline{\eta_i} \cdot \arrow[llll, bend right, "i"'] & \ldots
\end{tikzcd}$$

Property 2.4.14.

(i) Traversals are justified sequence verifying P-visibility and O-visibility.

(ii) The P-view of a traversal is a path from the root to some node in the computation tree.

(i) and (ii) are proved simultaneously by induction on the rule. The proof is given in [Ong, 2006].

Definition 2.4.15 (Reduced-traversal)**.** A *reduced-traversal* is a justified sequence of nodes s such that for some traversal t:

$$s = t \upharpoonright \{r\}$$

where r denotes the root of the computation tree.

We say that s is the reduction of the traversal t.

In a reduced-traversal, non-lambda nodes are all labelled $x@$ or x for some variable x *bound* in M. There cannot be nodes labelled @, $u@$ or u, where u is a free variable in M or a Σ-constant, since such nodes are pointer-less.

Remark 2.4.16. If s is a reduced-traversal s then $s = t \upharpoonright \{r\}$ for some traversal t whose last node does not belong to $N_\Sigma \cup Succ(N_\Sigma)$.

Definition 2.4.17 (Set substraction operator)**.** Suppose s is a justified sequence of nodes and N is a set of nodes then $s \setminus N$ is the justified sequence of nodes obtained by removing from s all the nodes that belongs to N. The links of $s \setminus N$ are the links of s minus all the links pointing from or to nodes in N.

Property 2.4.18 (Traversal of closed term in $\beta\eta$-long normal form). Suppose M is a closed term in $\beta\eta$-long normal form (i.e. the η-long normal form of a β-normal form), then for any traversal t of $\tau(M)$ and any node n in t:

$$n \text{ not hereditarily justified by } r \qquad \Longleftrightarrow \qquad n \in N_\Sigma \cup Succ(N_\Sigma)$$

where N_Σ denotes the set of nodes labelled u or $u@$ for some Σ-constant u, $Succ(N)$ denotes the set of nodes of $\tau(M)$ whose parent nodes are in N and r denotes the root of the tree $\tau(M)$.

Proof. For the right-to-left implication: node in N_Σ do not have justified pointer, nodes in $Succ(N_\Sigma)$ are justified by nodes in N_Σ. We can conclude by observing that $r \notin N_\Sigma \cup Succ(N_\Sigma)$.

For the left-to-right implication: we proved the contraposition by induction on the length of the traversal. It is trivially true for the empty traversal. Suppose n is not in $N_\Sigma \cup Succ(N_\Sigma)$.

There are two cases:

- n a non-lambda node: We observe that in the computation tree of a term in $\beta\eta$-long normal form there is no node of type @. Moreover M is closed therefore there is no node of type x@ or x for some free variable x .

  Consequently, only the nodes in N_Σ have no pointer. Since $n \not\in N_\Sigma \cup Succ(N_\Sigma)$, n must have a pointer to a previous lambda node m in the traversal.

  m is not in $Succ(N_\Sigma)$ because these nodes are all labelled λ and no other node can point to such node. Hence $m \not\in N_\Sigma \cup Succ(N_\Sigma)$.

- n is a lambda node: n does not point to a node in N_Σ otherwise n would be in $Succ(N_\Sigma)$. Hence n points to some previous non-lambda node m in the traversal such that $m \not\in N_\Sigma \cup Succ(N_\Sigma)$

In both case, we use the induction hypothesis to conclude that m is hereditarily justified by r and therefore n is hereditarily justified by r. □

Lemma 2.4.19 (Reduced-traversal and traversal)**.** *Suppose M is a closed term in $\beta\eta$-long normal form (i.e. the η-long normal form of a β-normal form) then:*

(i) for any traversal t, the reduced-traversal $s = t \upharpoonright \{r\}$ verifies

$$\ulcorner s \urcorner = \ulcorner t \urcorner \setminus (N_\Sigma \cup Succ(N_\Sigma))$$

(ii) for any traversal t whose last node does not belong to $N_\Sigma \cup Succ(N_\Sigma)$, the reduced-traversal $s = t \upharpoonright \{r\}$ verifies

$$\llcorner s \lrcorner = \llcorner t \lrcorner$$

where N_Σ denotes the set of nodes labelled u or u@ for some Σ-constant u, $Succ(N)$ denotes the set of nodes of $\tau(M)$ whose parent nodes are in N and r denotes the root of the tree $\tau(M)$.

Proof. (i) By induction: it is trivially true for the empty traversal $t = \epsilon$. We note s the reduced-traversal $s = t \upharpoonright \{r\}$

- Suppose n is r the root of the tree. Then $t = r$ because the root node occurs only once in a traversal. Hence:

$$\begin{aligned}
\ulcorner t \urcorner \setminus (N_\Sigma \cup Succ(N_\Sigma)) &= r \setminus (N_\Sigma \cup Succ(N_\Sigma)) \\
&= r & (r \not\in N_\Sigma \cup Succ(N_\Sigma)) \\
&= \ulcorner r \urcorner \\
&= \ulcorner r \upharpoonright \{r\} \urcorner = \ulcorner t \upharpoonright \{r\} \urcorner = \ulcorner s \urcorner
\end{aligned}$$

- Consider a traversal $t = t' \cdot n$ where n is a non-lambda move. Suppose that property (i) is verified for traversal t'. We note s' the reduced-traversal $s' = t' \upharpoonright \{r\}$:

$$\ulcorner t \urcorner = \ulcorner t' \cdot n \urcorner = \ulcorner t' \urcorner \cdot n \tag{2.5}$$

  We consider the following two sub-cases:

  – n is not hereditarily justified by r. Then

$$s = (t' \cdot n) \upharpoonright \{r\} = t' \upharpoonright \{r\} = s' \tag{2.6}$$

  From property 2.4.18, $n \in N_\Sigma \cup Succ(N_\Sigma)$ hence:

$$\begin{aligned}
\ulcorner t \urcorner \setminus (N_\Sigma \cup Succ(N_\Sigma)) &= (\ulcorner t' \urcorner \cdot n) \setminus (N_\Sigma \cup Succ(N_\Sigma)) & \text{(equation 2.5)} \\
&= \ulcorner t' \urcorner \setminus (N_\Sigma \cup Succ(N_\Sigma)) & (n \in N_\Sigma \cup Succ(N_\Sigma)) \\
&= \ulcorner s' \urcorner & \text{(induction hypothesis)} \\
&= \ulcorner s \urcorner & \text{(equation 2.6)}
\end{aligned}$$

– n is hereditarily justified by r. Then

$$s = (t' \cdot n) \upharpoonright \{r\} = (t' \upharpoonright \{r\}) \cdot n = s' \cdot n \tag{2.7}$$

From property 2.4.18, $n \notin N_\Sigma \cup Succ(N_\Sigma)$ hence:

$$\begin{array}{rcll}
\ulcorner t \urcorner \setminus (N_\Sigma \cup Succ(N_\Sigma)) & = & (\ulcorner t' \urcorner \cdot n) \setminus (N_\Sigma \cup Succ(N_\Sigma)) & \text{(equation 2.5)} \\
& = & (\ulcorner t' \urcorner \setminus (N_\Sigma \cup Succ(N_\Sigma))) \cdot n & (n \notin N_\Sigma \cup Succ(N_\Sigma)) \\
& = & \ulcorner s' \urcorner \cdot n & \text{(induction hypothesis)} \\
& = & \ulcorner s' \cdot n \urcorner & \text{(def. P-view)} \\
& = & \ulcorner s \urcorner & \text{(equation 2.7)}
\end{array}$$

- Consider the traversal:

$$t = t'' \cdot \overparen{m \cdot u \cdot n}$$

and suppose that t'' verifies property (i). We note s'' the reduced-traversal $s'' = t'' \upharpoonright \{r\}$. We have:

$$\ulcorner t \urcorner = \ulcorner t' \cdot \overparen{m \cdot u \cdot n} \urcorner = \ulcorner t'' \urcorner \cdot \overparen{m \cdot n} \tag{2.8}$$

– Suppose $n = \lambda$, then $n \in Succ(N_\Sigma)$ and $m \in N_\Sigma$. Therefore by property 2.4.18, m and r are not hereditarily justified by r.

$$\begin{array}{rll}
\ulcorner t \urcorner \setminus (N_\Sigma \cup Succ(N_\Sigma)) & = (\ulcorner t'' \urcorner \cdot \overparen{m \cdot n}) \setminus (N_\Sigma \cup Succ(N_\Sigma)) & \text{(equation 2.8)} \\
& = \ulcorner t'' \urcorner \setminus (N_\Sigma \cup Succ(N_\Sigma)) & (m, n \in N_\Sigma \cup Succ(N_\Sigma)) \\
& = \ulcorner s'' \urcorner & \text{(induction hypothesis)} \\
& = \ulcorner s'' \cdot \overparen{m \cdot (u \upharpoonright \{r\}) \cdot n} \urcorner & \text{(def. P-view)} \\
& = \ulcorner (t'' \upharpoonright \{r\}) \cdot \overparen{m \cdot (u \upharpoonright \{r\}) \cdot n} \urcorner & \text{(def. s”)} \\
& = \ulcorner (t'' \cdot \overparen{m \cdot u \cdot n}) \upharpoonright \{r\} \urcorner & \text{(def. operator \\)} \\
& = \ulcorner t \upharpoonright \{r\} \urcorner & \text{(def. of } t) \\
& = \ulcorner s \urcorner & \text{(def. of } s)
\end{array}$$

– Suppose $n = \lambda\overline{\xi}$ then $n \notin Succ(N_\Sigma)$ and $m \notin N_\Sigma$ which implies by property 2.4.18 that m and n are hereditarily justified by r.

$$\begin{array}{rll}
\ulcorner t \urcorner \setminus (N_\Sigma \cup Succ(N_\Sigma)) & = (\ulcorner t'' \urcorner \cdot \overparen{m \cdot n}) \setminus (N_\Sigma \cup Succ(N_\Sigma)) & \text{(equation 2.8)} \\
& = \ulcorner t'' \urcorner \setminus (N_\Sigma \cup Succ(N_\Sigma)) \cdot \overparen{m \cdot n} & (m, n \notin N_\Sigma \cup Succ(N_\Sigma)) \\
& = \ulcorner s'' \urcorner \cdot \overparen{m \cdot n} & \text{(induction hypothesis)} \\
& = \ulcorner s'' \cdot \overparen{m \cdot (u \upharpoonright \{r\}) \cdot n} \urcorner & \text{(def. P-view)} \\
& = \ulcorner (t'' \upharpoonright \{r\}) \cdot \overparen{m \cdot (u \upharpoonright \{r\}) \cdot n} \urcorner & \text{(def. s”)} \\
& = \ulcorner (t'' \cdot \overparen{m \cdot u \cdot n}) \upharpoonright \{r\} \urcorner & \text{(def. operator \\)} \\
& = \ulcorner t \upharpoonright \{r\} \urcorner & \text{(def. of } t) \\
& = \ulcorner s \urcorner & \text{(def. of } s)
\end{array}$$

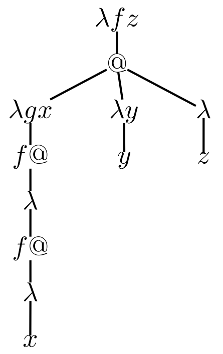
(ii)

□ proof of ii

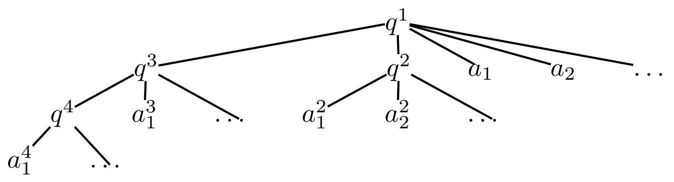
2.4.2 Correspondence with game semantics

By representing side-by-side the computation tree and the type arena of a term in η-normal form we observe that for each question move of the arena there are corresponding nodes in the computation tree.

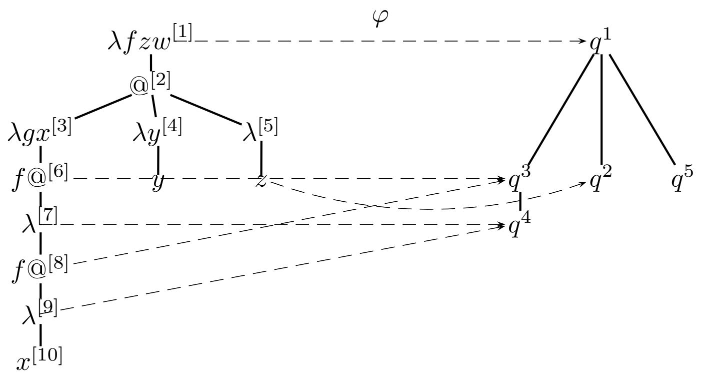
Example 2.4.20. Consider the following term $M \equiv \lambda fz.(\lambda gx.f(fx))(\lambda y.y)z$ of type $(o \to o) \to o \to o$. Its η-long normal form is $\lambda fz.(\lambda gx.f(fx))(\lambda y.y)(\lambda.z)$. The computation tree is:



The arena for the type $(o \to o) \to o \to o$ is:



We now omit the answers moves when we represent the arena. The arena and the computation tree are represented on the figure below (right and left respectively). The dashed line defines a partial function φ from the set of nodes in the computation tree to the set of question moves:



Consider the justified sequence of moves $s \in [\![M]\!]$:

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

There is a corresponding justified sequence of nodes in the computation tree:

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

such that $s_i = \varphi(r_i)$ for all $i < |s|$.

The sequence r is in fact a reduced-traversal, it is the reduction of the following traversal:

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

We see on this example that the game semantics and the computation tree are somehow related to each other.

Let us now analyze precisely the relationship between the game semantics and the computation tree.

Let $\Gamma \vdash M : A$ be a term in η-long normal form. Suppose $(M^{QA}, \vdash)$ is the arena $[\![A]\!]$ where M^{QA} is the set of moves and $\vdash$ is the enabling relation. $M^Q \subseteq M^{QA}$ denotes the set of question-moves. We note (N, E) the computation tree $\tau(M)$ where N is the set of nodes and leaves of the tree and E is the parent-child relation.

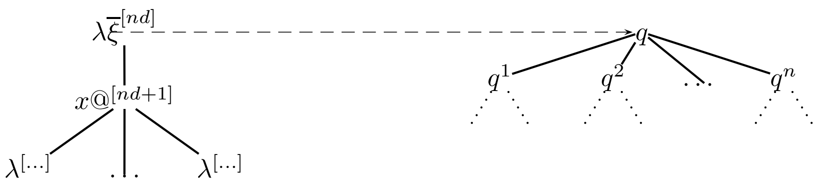
We give an algorithm computing the partial function $\varphi : N \rightarrow M^Q$ which maps some question-moves to set of nodes in the computation tree.

Definition 2.4.21 (Relation between question-moves and nodes of the computation tree)**.** We start by defining two preliminary procedures. The procedure f takes two parameters: n is the index number of a λ-node in the computation tree and q is a question move of the arena of $\Gamma \vdash M : A$ such that q and $\kappa(n)$ have the same type.

Similarly we define the procedure g that takes two parameters: n is the index number of a x@-labelled node or a x-labelled node in the computation tree and q is a question move of the arena of $\Gamma \vdash M : A$ such that q and x have the same type.

Procedure $f(nd, q)$ where nd is a λ-node.

- If $\mathsf{ord}(\kappa(nd)) = 0$ then the term is of ground type therefore the game for M is played on the flat arena with only one question q. Moreover the node nd of the computation tree is labelled with λ.
  return $\{nd \mapsto q\}$.
- $\mathsf{ord}(\kappa(nd)) > 0$. The computation tree and the arena have the following form:



  such that $\Gamma, \overline{\xi} \vdash \kappa(x@^{[nd+1]}) : o$.
  For each of the abstracted variable ξ_i there is a corresponding question move q_i of the same order in the arena. Each free occurrence of the variable ξ_i is mapped to the move q_i by the procedure g.

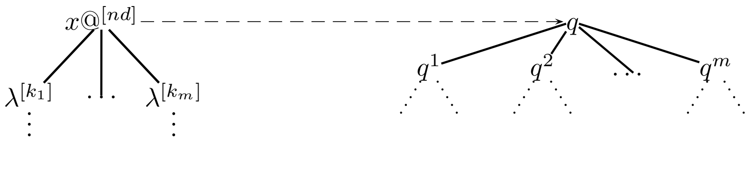
$$\textbf{return } \{nd \mapsto q\} \cup \bigcup_{\substack{i=1..n \\ \xi_i^{[k]} \in Desc(nd)}} g(k, q_i) \quad \cup \bigcup_{\substack{i=1..n \\ @\xi_i^{[k]} \in Desc(nd)}} g(k, q_i)$$

  where $Desc(nd)$ is the set of descendants of node nd (nodes m such that there is a path from node nd to node m in the computation tree).

Procedure $g(nd, q)$
The procedure is not defined on λ-nodes or @ nodes. This is ok since all the calls to g in f are of the type $g(nd, q)$ where nd denotes a @x-node or a x-node.

case 1 Suppose that nd is labelled with x then we must have $x : 0$. **return** $\{nd \mapsto q\}$.

case 2 If nd is labelled with x@ then $x : (A_1 | \dots | A_m | o)$. The computation tree and the arena have the following form:



such that $\Gamma, \overline{\xi} \vdash \kappa(x@^{[nd]}) : o$.
For each of the children node of nd there is a corresponding question move q_i of the same type in the arena.

$$\textbf{return } \{nd \mapsto q\} \cup \bigcup_{i=1..m} f(k_i, q_i)$$

We define φ as follows:

$$\varphi = f(0, q^0)$$

where the index 0 denotes the root of the computation tree and q^0 is the root of the arena (in the game semantics of simply-typed lambda calculus the arenas have a single root).

The function $\varphi : N \to M^Q$ and its inverse $\varphi^{-1} : M^Q \to \mathcal{P}(N)$ are partial function. Moreover φ relates player O questions to λ-nodes and player P questions to $x@$-nodes or x-nodes where x is a variable.

Property 2.4.22 (φ conserves order). φ maps nodes of a given order to moves of the same order.

The function φ is also defined on justified sequences: suppose t is a justified sequences of nodes $t = t_0 t_1 \ldots$ then $\varphi(t)$ denotes the following justified sequence of question-moves of the arena $[\![A]\!]$:

$$\varphi(t) = \varphi(t_0)\ \varphi(t_1)\ \varphi(t_2) \ldots$$

where the pointers of the justified sequence of move $\varphi(t)$ are defined to be exactly those of the justified sequences of nodes t.

Definition 2.4.23 (Question-move filtering)**.** Suppose s is a justified sequence of moves then we note $\tilde{s}$ the subsequence of s consisting of question-moves only:

$$\begin{array}{rcl} \tilde{} \ : L_A & \longrightarrow & (M^Q)^* \\ s & \longmapsto & \tilde{s} = s \upharpoonright M^Q \end{array}$$

where M^Q denotes the set of question-moves. $\tilde{s}$ is also a justified sequence of move (there is no dangling pointer since questions-moves points to other question-moves). If $s = u \upharpoonright A, B$ then clearly $\tilde{s} = \tilde{u} \upharpoonright A, B$.

Proposition 2.4.24 (Relation between game semantics and reduced-traversals)**.** *Let $\emptyset \vdash M : A$ be a closed term. Let r denotes the root of $\tau(M)$. If s is a justified sequence of moves such that $s \in [\![\Gamma \vdash M : A]\!]$ then there is a reduced-traversal of nodes of $\tau(M)$ $t = n_0 n_1 \ldots$ such that:*

$$\tilde{s} = \varphi_M(t)$$

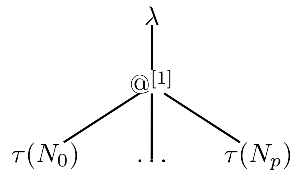
Proof. Let us assume that M is already in η-long normal form. Let s be as justified sequence of moves in $[\![M]\!]$.

We proceed by induction on the height of of the tree $\tau(M)$ and by case analysis on the structure of the term.

- (variable) $M = x$. This case does not happen since M is a closed term.
- (application) The term M is of the form $\emptyset \vdash N_0 N_1 \ldots N_p : o$ where N_0 is not a variable (since M is closed) and:

$$\begin{array}{rcl} \emptyset & \vdash & N_0 : (A_1, \ldots, A_p, o) \\ \emptyset & \vdash & N_i : A_i \text{ for } i \in 1..p \end{array}$$

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



We have:

$$[\![M]\!] = [\![\emptyset \vdash N_0 N_1 \ldots N_p : o]\!] = \overbrace{\langle [\![\emptyset \vdash N_0]\!], \ldots, [\![\emptyset \vdash N_p]\!]\rangle}^{\sigma} ; ev_{A_1,\ldots,A_p,o}$$

with $\sigma : \mathbf{1} \longrightarrow B$ and $ev_{A_1,\ldots,A_p,o} : B \longrightarrow [\![o]\!]$ where

$$B = [\![(A_1,\ldots,A_p,o)]\!] \times [\![A_1]\!] \times \ldots \times [\![A_p]\!]$$

Since $s \in [\![M]\!] = [\![\emptyset \vdash N_0 N_1 \ldots N_p : o]\!]$ we have:

$$s = u \upharpoonright \mathbf{1}, [\![o]\!] = u \upharpoonright [\![o]\!]$$

for some $u \in \sigma \parallel ev_{A_1,\ldots,A_p,o}$.

Let q_C denotes the only question of the arena $[\![o]\!]$ then $s \in \{q_C\}^*$. We construct the reduced-traversal t by replacing the moves q_C in $\tilde{s}$ by the root node $\lambda\overline{\xi}$:

$$t = \tilde{s}[\lambda\overline{\xi}/q_C].$$

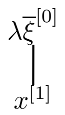
Since $\tilde{s} \in \{q_C\}^*$ and $\varphi_M(\lambda\overline{\xi}) = q_C$, we have:

$$\varphi_M(t) = \varphi_M(\tilde{s}[\lambda\overline{\xi}/q_C]) = \tilde{s}[\lambda\overline{\xi}/q_C][q_C/\lambda\overline{\xi}] = \tilde{s}$$

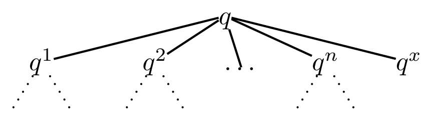
Since all the moves in $\tilde{s}$ are initial, they do not have pointers. Hence

$$\varphi_M(t) = \tilde{s}$$

- (variable-abstraction) $M = \lambda\overline{\xi}.x$. Then x must be of ground type o and since M is closed $x = \xi_i \in \overline{\xi}$. Then $\tau(M)$ has the following shape:



The arena is of the following form (only question moves are represented):



where q^x denotes the root of the flat arena $[\![o]\!]$. We have:

$$[\![M]\!] = [\![\emptyset \vdash \lambda\overline{\xi}.\xi_i]\!] = \Lambda^n([\![\overline{\xi} \vdash \xi_i]\!]) = \Lambda^n(\pi_i)$$

where π_i denotes the i-th projection

Therefore the only question-moves that can occur in $s \in [\![M]\!]$ are the initial question q and the question q_x. We take $t = \tilde{s}[0/q, 1/q^x]$ which is a valid justified sequence of nodes since node 1 (x) is bound by node 0 ($\lambda\overline{\xi}$) and a valid reduced-traversal. Clearly we have $\varphi(t) = \tilde{s}$.

- (application-abstraction) The term M is of the form $M = \lambda\overline{\xi}.xN_1 \ldots N_p$ where

$$\begin{array}{lcl}
\emptyset & \vdash & M : (X_1,\ldots,X_n,o) \\
\Gamma & \vdash & xN_1 \ldots N_p : o \\
\Gamma & \vdash & x : (A_1,\ldots,A_p,o) \\
\Gamma & \vdash & N_i : A_i \text{ for } i \in 1..p \\
\Gamma & = & \overline{\xi} : \overline{X}
\end{array}$$

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

$$\begin{tikzcd}
 & \lambda\overline{\xi}^{[0]} \arrow[d, no head] & \\
 & x@^{[1]} \arrow[dl, no head] \arrow[d, no head] \arrow[dr, no head] & \\
\tau(N_0) & \ldots & \tau(N_p)
\end{tikzcd}$$

where $x \in \overline{\xi}$ (since M is a closed term).

We have:

$$\begin{aligned}
[\![\emptyset \vdash M]\!] &= \Lambda^n([\![\Gamma \vdash xN_1 \ldots N_p : o]\!]) \\
[\![\Gamma \vdash xN_1 \ldots N_p : o]\!] &= \overbrace{\langle [\![\Gamma \vdash N_0]\!], \ldots, [\![\Gamma \vdash N_p]\!] \rangle}^{\sigma} ; ev_{A_1,\ldots,A_p,o}
\end{aligned}$$

with $\sigma : A \longrightarrow B$ and $ev_{A_1,\ldots,A_p,o} : B \longrightarrow C$, the arena being defined as follows:

$$\begin{aligned}
A &= [\![\Gamma]\!] = [\![X_1]\!] \times \ldots \times [\![X_n]\!] \\
B &= H_0 \times H_1 \times \ldots \times H_p \\
C &= [\![o]\!] \\
H_0 &= [\![(A_1, \ldots, A_p, o)]\!] \\
H_j &= [\![A_j]\!] \text{ for } j \in 1..p
\end{aligned}$$

Since $s \in [\![M]\!]$ we must have $s \in [\![\Gamma \vdash xN_1 \ldots N_p : o]\!]$ therefore:

$$s = u \upharpoonright A, C \tag{2.9}$$

for some $u \in \sigma \parallel ev_{A_1,\ldots,A_p,o}$, more precisely for some u such that:

$$\begin{aligned}
u &\in int(A, B, C) \\
w = u \upharpoonright A, B &\in \sigma \\
u \upharpoonright B, C &\in ev_{A_1,\ldots,A_p,o}
\end{aligned} \tag{2.10}$$

$w \in \sigma$ implies that for some $j \in 1..p$:

$$z = w \upharpoonright A, H_j \quad \in \quad [\![\Gamma \vdash N_j]\!] \tag{2.11}$$

$$\text{and for every } k \neq j \quad : \quad w \upharpoonright H_k = \epsilon \tag{2.12}$$

We cannot use the induction hypothesis on $\Gamma \vdash N_j : A_j$ because it is not a closed terms! We therefore consider the closed term $N'_j = \lambda\overline{\xi}.N_j$:

$$\emptyset \vdash N'_j : (X_1, \ldots, X_n, A_j)$$

We have $z \in [\![N_j]\!] = \Lambda^n([\![\Gamma \vdash N_j]\!])$. But $\Lambda^n([\![\Gamma \vdash N_j]\!])$ and $[\![\Gamma \vdash N_j]\!]$ are the same strategies up to an isomorphism that retaggs the moves. Hence the equation 2.11 gives $z \in [\![N'_j]\!]$ where N'_j is a closed term. Since the term N'_j has the same height as the term N_j which is strictly smaller than the height of the term M, we can use the induction hypothesis on N'_j: there is a reduced-traversal t' of the tree $\tau(N'_j)$ such that $\varphi_{N'_j}(t') = \tilde{z}$.

From equation 2.10 we have:

$$\tilde{w} = \tilde{u} \upharpoonright A, B \tag{2.13}$$

From equation 2.12, for $k \neq j : w \upharpoonright H_k = \epsilon$. Since $B = H_0 \times H_1 \times \ldots \times H_p$ this implies that moves in w hereditarily justified by moves in B are in fact all hereditarily justified by moves in H_j. Hence $z = w \upharpoonright A, H_j = w \upharpoonright A, B$ and:

$$\begin{array}{rcl} \tilde{z} = & \tilde{w} \upharpoonright A, B & \\ = & (\tilde{u} \upharpoonright A, B) \upharpoonright A, B & \text{(by eq 2.13)} \\ = & \tilde{u} \upharpoonright A, B & \\ = & \tilde{w} & \text{(by eq 2.13)} \end{array}$$

Hence $\varphi_{N'_j}(t') = \tilde{w}$.

For $i \in 1..p$, the tree $\tau(N_i)$ (left) and $\tau(N'_i)$ (right) have the following form:

$\lambda \overline{y_i}^{[n_i]}$
$\lambda \overline{\xi}\ \overline{y_i}^{[n_i]}$

where only the label of the root differs between the two trees.

From the justified sequence t' of $\tau(N'_j)$ we construct the justified sequence t of $\tau(M)$ by mapping the nodes of $\tau(N'_j)$ to the corresponding node in the subtree $\tau(N_j)$ of $\tau(M)$. Moreover we change all the pointers going from a node ξ_i or $\xi_i@$ to the root node of $\tau(N'_j)$ into a pointer starting from the corresponding node ξ_i or $\xi_i@$ to the root of the tree $\tau(M)$. t is a valid justified sequence of nodes of $\tau(M)$ since the nodes labelled ξ_i and $\xi_i@$ are bound by the root $\lambda\overline{\xi}$.

The function $\varphi : Nodes \to M^Q$ has been defined by an inductive procedure guaranteeing that φ_{N_j} is equal to the function φ_M restricted to the set of nodes of $\tau(N_j)$. Hence we have:

$$\varphi_M(t) = \tilde{w}. \tag{2.14}$$

Let q_C denotes the only question of the arena $C = [\![o]\!]$. We can construct from t the justified sequence t^*. The construction is in three steps.

- First we describe the elements of the sequence t^* (the pointers are ignored at the moment).

  We insert the root node $\lambda\overline{\xi}$ at the positions where the move q_C occurs in $\tilde{u}$. Since $\varphi_M(\lambda\overline{\xi}) = q_C$, we have the following equality of *pointer-less* sequences:

  $$\varphi_M(t^*) = \tilde{u}$$

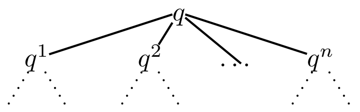
- The next step is to add the pointers to the sequence t^*. These pointers are transferred from $\tilde{u}$ to t^*: for each move $\tilde{u}_k$ pointing to $\tilde{u}_i = q_C$ in $\tilde{u}$ for some $0 \leq i \leq k$, we add a pointer in t^* going from t^*_k to $t^*_i = \lambda\overline{\xi}$.

  Then the pointers of the traversal t^* and the sequence of move $\tilde{u}$ are the same and we obtain the equality of the *justified* sequences:

  $$\varphi_M(t^*) = \tilde{u}. \tag{2.15}$$

- Finally, we need to ensure that the sequence t^* is a valid justified sequence of $\tau(M)$. This requires us to check the validity of the the pointers introduced in t^*. Suppose u_k points to q_C in $\tilde{u}$ then u_k belongs to the arena A. More precisely it is one of the root $q^1 \ldots q^n$ of the arena A. Therefore u_k must belong to the sequence $\tilde{w}$ and for some $r \geq 0$, $u_k = \phi^{-1}_{N'_j}(t'_r)$ and $t^*_k = t'_r$.

The arena of the term $\emptyset \vdash N'_j = \lambda\overline{\xi}.N_j$ is of the following form (only question moves are represented):



We observe from the definition of $\varphi_{N'_j}$ that the only node that are mapped to the question $q^1 \ldots q^n$ are labelled ξ_i or $\xi_i@$ for some $i \in 1..n$. Consequently $t^*_k = \xi_i$ and the node t^*_k is bound by the node $\lambda\overline{\xi}$. Hence the added pointer is indeed valid.

For any justified sequence of nodes t we define the justified sequence of nodes $t \upharpoonright A, C$ to be the subsequence of t constituted of nodes n verifying $\varphi_M(n) \in A \cup C$ together with the same pointers as the justified sequence of move $\varphi_M(t) \upharpoonright A, C$ (here the operator $\upharpoonright$ denotes the filtering operator defined on justified sequence of moves).

The effect of this transformation is to remove from t^* all the elements at positions i such that $\tilde{u}_i \in B$. We obtain a justified sequence of nodes $t^\dagger = t^* \upharpoonright A, C$ verifying

$$\varphi_M(t^\dagger) = \tilde{s}$$

$t^\dagger$ is also a reduced-traversal of $\tau(M)$.

Example: Let us illustrate each step of the demonstration on a short example. For clarity only some relevant pointers are specified in the following justified sequences.

$$\begin{array}{rccccccccccccc}
 & \tilde{u} & = & q_C & m_1 & q_C & m_3 & m_4 & q_C & m_6 & m_7 & q_C & \cdots \\
 & & \in & C & B & C & A & A & C & B & B & C & \cdots \\
\text{(equation 2.9)} & \tilde{s} & = & q_C & & q_C & m_3 & m_4 & q_C & & & q_C & \cdots \\
\text{(equation 2.10)} & \tilde{w} & = & & m_1 & & m_3 & m_4 & & m_6 & m_7 & & \cdots \\
 & & & & \varphi_{N'_j}\uparrow & & \uparrow & \uparrow & & \uparrow & \uparrow & & \\
\text{(equation 2.14)} & t & = & & n_1 & & n_3 & n_4 & & n_6 & n_7 & & \cdots
\end{array}$$

$$\begin{array}{rccccccccccccc}
 & \tilde{u} & = & q_C & m_1 & q_C & m_3 & m_4 & q_C & m_6 & m_7 & q_C & \cdots \\
 & & & \varphi_M\Uparrow & \varphi_{N'_j}\uparrow & \Uparrow & \uparrow & \uparrow & \Uparrow & \uparrow & \uparrow & \Uparrow & \\
\text{(equation 2.15)} & t^* & = & \lambda & n_1 & \lambda & n_3 & n_4 & \lambda & n_6 & n_7 & \lambda & \cdots \\
t^* \upharpoonright A, C = & t^\dagger & = & \lambda & & \lambda & n_3 & n_4 & \lambda & & & \lambda & \cdots
\end{array}$$

Note that there is only one question q_C in the arena C therefore $m_2 = m_5 = m_8 = q_C$. It is easy to check that $\varphi(t') = \tilde{s}$

$\square$

The previous proposition has an equivalent for open terms. Its proof is omitted here. It follows the same step as for the case of closed terms.

Proposition 2.4.25 (Relation between game semantics and reduced-traversals of open terms). *Suppose $\Gamma \vdash M : A$. Let r denotes the root of $\tau(M)$. If s is a justified sequence of moves such that $s \in [\![\Gamma \vdash M : A]\!]$ then there is a reduced-traversal of nodes of $\tau(M)$ $t = n_0 n_1 \ldots$ such that:*

$$\tilde{s} \upharpoonright [\![A]\!] = \varphi_M(t \upharpoonright \{r\})$$

and

$$\forall i \geq 0. \tilde{s}_i = \varphi_M(t_i) \notin [\![A]\!] \implies \begin{cases} t_i = x \text{ or } t_i = x@, \text{where } x \in \Gamma \\ \varphi_M(t \upharpoonright \{t_i\}) = \tilde{s} \upharpoonright \{\varphi_M(t_i)\} \end{cases}$$

Property 2.4.26. Suppose $\varphi(t) = s$ where s is a justified sequence of moves and t is a justified sequence of nodes then

- (i) s and t have the same pointers.
- (ii) the P-view of $\tilde{s}$ and the P-view of t are computed identically: the set of indices of elements that must be removed from both sequences in order to obtain their P-view is the same.
- (iii) the O-view of $\tilde{s}$ and the O-view of t are identically.

Proof. (i): By definition of φ, t and $\varphi(t)$ have the same pointers.

(ii) and (iii): φ maps lambda nodes to O-question and non-lambda nodes to P-question. Therefore since t and s have the same pointers, the computations of the P-view (resp. O-view) of the sequence of moves and the P-view (resp. O-view) of the sequence of nodes follow the same steps. □

2.4.3 Pointers in the game semantics of safe terms are recoverable

The computation tree of safe terms verifies a property called regularity:

Property 2.4.27 (Computation tree of safe terms are regular). If M is a safe term then any node of the computation tree $\tau(M)$ labelled x or $x@$ where x is a variable bound in M is bound by the first λ-node in the path to the root that has order greater or equal to $\mathsf{ord}(x)$.

Proof. (a) In the safe λ-calculus when applying the abstraction rule the variables in the lowest partition of the context (smallest order) must all be abstracted together. Translating this to the computation tree point of view (where consecutive abstractions are merged into a single node), it gives : for each lambda node $\lambda\overline{\xi}$, any variable x occurring free in $\kappa(\lambda\overline{\xi})$ has order strictly greater than $\mathsf{ord}(\lambda\overline{\xi})$.

(b) If x is bound by a node $\lambda\overline{\xi}$ then $\mathsf{ord}(\lambda\overline{\xi}) \geq \mathsf{ord}(x)$.

(c) t satisfies P-visibility (property 2.4.14) therefore x points to a lambda-node in the P-view at that point. By property 2.4.14 the P-view $\ulcorner t \urcorner$ is the path in $\tau(M)$ from the root to the node n.

Consider the lambda nodes occurring in this path: $\lambda\overline{\eta_p}, \ldots, \lambda\overline{\eta_1}$ (the bigger the index is the closer the nodes is to the root).

Since x is not free in M, it must be bound by one of the nodes $\lambda\overline{\eta_1}, \ldots, \lambda\overline{\eta_p}$. Take i the smallest index such that $\mathsf{ord}(\lambda\overline{\eta_i}) \geq \mathsf{ord}(x)$ ((b) guarantees the existence of i).

The nodes $\lambda\overline{\eta_j}$ for $j < i$ are of order smaller than $\mathsf{ord}(x)$ therefore by (b) for all $j < i$, $x \notin \overline{\eta_j}$. Suppose that $x \notin \overline{\eta_i}$ then x occurs free in $\kappa(\lambda\overline{\eta_i})$. By (a) this implies $\mathsf{ord}(x) > \mathsf{ord}(\lambda\overline{\eta_i})$ which is a contradiction. Hence $x \in \overline{\eta_i}$.

Hence the first λ-node in the path to the root whose order is greater or equal to $\mathsf{ord}(x)$ is also the first λ-node $\lambda\overline{\xi}$ in the path to the root such that $x \in \overline{\xi}$. □

Proposition 2.4.28. *The pointers in the game semantics of safe terms are uniquely recoverable.*

Proof. Let $\Gamma \vdash M : A$ be an opened safe term where $\Gamma = y_1 : Y_1, \ldots y_n : Y_n$. We assume that M is in $\beta\eta$-long normal form : the η-long normal form of the β-normal form. It is a safe assumption since safety is preserved by η-expansion and β-reduction.

Consider a justified sequence of move $s \in [\![\Gamma \vdash M]\!]$. Firstly, we remark that the pointers for O and P answer moves in s can all be recovered thanks to the well-bracketing condition.

Let us prove by induction that the pointers for question moves can be recovered.

Base case: if $s \in [\![M]\!]$ where $|s| \leq 1$ then there is no pointer to recover.

Step case: suppose $s \in [\![M]\!]$ with $|s| > 1$. We note q the last move in s.

Consider the closed term $M' = \lambda\overline{y}.M$. Up to a retagging of the moves, the justified sequence of moves s belongs to the strategy $[\![\vdash \lambda\overline{y}.M]\!] = \Lambda^n([\![\Gamma \vdash M]\!])$.

Proposition 2.4.24 tells us that there is a reduced-traversal r of $\tau(M')$ such that:

$$\varphi_{M'}(r) = \tilde{s}.$$

Let us note n the last node in the sequence r and *root* the root of $\tau(M')$. r is a reduced-traversal therefore by remark 2.4.16:

$$r = t \upharpoonright \{root\}$$

for some traversal t whose last node n is not in $N_\Sigma \cup Succ(N_\Sigma)$.

- Suppose q is a P-move.

  Since r is a reduced-traversal of $\tau(M')$ and M' is in $\beta\eta$-long normal form, lemma 2.4.19 (i) gives:

  $$\ulcorner r \urcorner = \ulcorner t \urcorner \setminus (N_\Sigma \cup Succ(N_\Sigma)) \tag{2.16}$$

  Property 2.4.14 gives us that $\ulcorner t \urcorner$ is the path in $\tau(M)$ from the root to the node n.

  Hence the regularity of $\tau(M)$ (property 2.4.27) tells us that n points to the the last λ-node in $\ulcorner t \urcorner$ (the path from the root to n) that has order greater or equal to $\mathsf{ord}(n)$.

  The lambda nodes in $N_\Sigma \cup Succ(N_\Sigma)$ are the nodes in $Succ(N_\Sigma)$; they are labelled λ and therefore no node points to them. Therefore, because of equation 2.16, n also points to the the last λ-node in $\ulcorner r \urcorner$ that has order greater or equal to $\mathsf{ord}(n)$.

  By property 2.4.26 (ii), the P-view of $\tilde{s}$ and the P-view of r are computed similarly and have the same pointers. This means that node n and move q point to the same position in the justified sequence $\ulcorner r \urcorner$ and $\ulcorner \tilde{s} \urcorner$ respectively.

  Finally, by the induction hypothesis, the pointers of s can been recovered up to the last move q: this ensures that the P-view $\ulcorner \tilde{s} \urcorner$ can be computed.

  Since φ maps nodes of a given order to moves of the same order (property 2.4.22), q also points to the last O-move in $\ulcorner \tilde{s} \urcorner$ whose order is greater or equal to $\mathsf{ord}(q)$.

- Suppose q is an O-move.

  M' is in $\beta\eta$-long normal form and t is a traversal of $\tau(M')$ whose last node n is not in $N_\Sigma \cup Succ(N_\Sigma)$. Therefore by lemma 2.4.19 (ii):

  $$\llcorner r \lrcorner = \llcorner t \lrcorner \tag{2.17}$$

  A lambda-nodes n always points to its parent node p, therefore verifying the relation $\mathsf{ord}(p) > \mathsf{ord}(n)$. By inspecting the formation rules of traversals (definition 2.4.12) one can check that a lambda-node n occurring in a traversal always points to the last node in the O-view at that point whose order is strictly greater that $\mathsf{ord}(n)$ (there are just two cases, n points either to the preceding node or to the third previous node in its O-view).

  Similarly as in the P-move case, we conclude that q points to the last move in the O-view of $\tilde{s}$ having an order strictly greater than $\mathsf{ord}(q)$, the induction hypothesis guaranteeing that the O-view $\llcorner \tilde{s} \lrcorner$ is computable.

$\square$

BIBLIOGRAPHY

Samson Abramsky and Guy McCusker. Game semantics. In *Proceedings of Marktorberdorf '97 Summerschool*, 1997. URL `citeseer.ist.psu.edu/abramsky99game.html`. Lecture notes.

Samson Abramsky, Pasquale Malacaria, and Radha Jagadeesan. Full abstraction for PCF. In *Theoretical Aspects of Computer Software*, pages 1–15, 1994. URL `citeseer.ist.psu.edu/abramsky95full.html`.

Klaus Aehlig, Jolie G. de Miranda, and C.-H. Luke Ong. Safety is not a restriction at level 2 for string languages. In Vladimiro Sassone, editor, *FoSSaCS*, volume 3441 of *Lecture Notes in Computer Science*, pages 490–504. Springer, 2005. ISBN 3-540-25388-2.

Aleksandar Dimovski, Dan R. Ghica, and Ranko Lazic. Data-abstraction refinement: A game semantic approach. In Chris Hankin and Igor Siveroni, editors, *SAS*, volume 3672 of *Lecture Notes in Computer Science*, pages 102–117. Springer, 2005. ISBN 3-540-28584-9.

Gérard P. Huet. A unification algorithm for typed lambda-calculus. *Theor. Comput. Sci.*, 1(1): 27–57, 1975.

Gérard P. Huet. *Résolution d'équations dans des langages d'ordre 1,2,...,ω*. Thèse de doctorat es sciences mathématiques, Université Paris VII, Septembre 1976.

J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II, and III. *Inf. Comput.*, 163 (2):285–408, 2000. ISSN 0890-5401. doi: http://dx.doi.org/10.1006/inco.2000.2917.

D. C. Jensen and Tomasz Pietrzykowski. Mechanizing *mega*-order type theory through unification. *Theor. Comput. Sci.*, 3(2):123–171, 1976.

T. Knapik, D. Niwiński, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FOSSACS'02*, pages 205–222. Springer, 2002. LNCS Vol. 2303.

Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The size-change principle for program termination. In *ACM Symposium on Principles of Programming Languages*, volume 28, pages 81–92. ACM press, january 2001.

C.-H. L. Ong. Safe lambda calculus: Some questions. Note on the safe lambda calculus., December 2005.

C.-H. L. Ong. On model-checking trees generating by higher-order recursion schemes. Preprint. To appear in Proceedings of LICS 2006., 2006.

Gordon D. Plotkin. Lcf considered as a programming language. *Theor. Comput. Sci.*, 5(3):225–255, 1977.

John C. Reynolds. The essence of algol. In J. W. de Bakker and J. C. van Vliet, editors, *Algorithmic Languages*, pages 345–372. IFIP, North-Holland, Amsterdam, 1981.

Dana S. Scott. A type-theoretical alternative to iswim, cuch, owhy. *Theor. Comput. Sci.*, 121 (1-2):411–440, 1993. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/0304-3975(93)90095-B.

Damien Sereni. Simply typed λ-calculus and sct. 2005.