Notes on game semantics and safe-lambda calculus

William Blum

March 8, 2006

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

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	$\bot$	tt	ff
$\bot$	$\bot$	tt	$\bot$
tt	tt	tt	tt
ff	$\bot$	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 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, Abramsky et al. [1994].

1.2.1 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;
- $\lambda : M \to \{O, P\} \times \{Q, A\}$ 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 $\lambda^{OP} : M \to \{O, P\}$ and $\lambda^{QA} : M \to \{Q, A\}$.
  - 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:

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

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

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

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

$$s \in P \Rightarrow s \restriction I \in P$$

1.2.2 Game construction

tensor product, implication, product

1.2.3 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.

1.2.4 Categorical interpretation of games

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

1.2.5 Arena of order at most 2

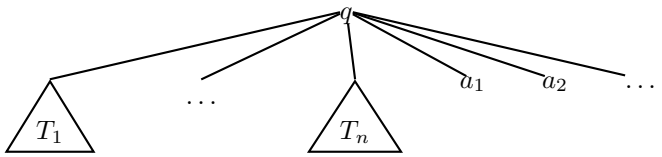
The height of the arena is the length of the longest sequence of moves $m_1 \dots m_h$ in M such that $m_1 \vdash m_2 \vdash \dots \vdash m_h$.

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:



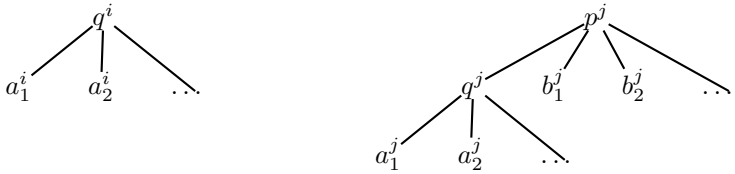
where each triangle T_i represents an arena of order 0 or 1.

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:
  $\hat{s} = q \overset{\frown}{\leftarrow \cdots} m$
  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 $\hat{u} = q \overset{\frown}{\leftarrow \cdots} p^i\ x$ 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 $\llcorner \hat{u} \lrcorner = \llcorner q \overset{\frown}{\leftarrow \cdots} p^i \lrcorner = qp^i$.
  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 $?(\hat{s}) = ?(\hat{u} \cdot a^i_k) = ?(q \overset{\frown}{\leftarrow \cdots} q^i \overset{\frown}{\leftarrow \cdots} a^i_k) \in q \cdot L^*$ 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)? Well 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).

□

1.3 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 $Y_A : (A \to A) \to A$ for any type A.

The types of PCF are given by the following grammar:

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

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 $\Gamma \vdash M : A$.

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

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

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

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

Table 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.

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

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

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

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

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

Table 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.

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

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

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

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

Table 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 $\mathtt{new}\ x := 0\ \mathtt{in}\ M$ 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:

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

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:

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

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

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

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

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

Table 1.4: Big-step operational semantics of IA

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.

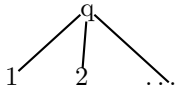
Simply typed λ-calculus

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 $\Gamma = x_1 : A_1, \ldots x_n : A_n$ will be mapped to the arena $[\![\Gamma]\!] = [\![A_1]\!] \times \ldots \times [\![A_n]\!]$ 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:

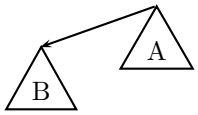
- the empty strategy where P never answer the initial question. This corresponds to a non terminating computation;
- the strategies where P answers by playing a number n. This models the constants of the language.

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:

$$[\![\Gamma : A_1, \ldots, x_n : A_n]\!] = \pi_i : [\![A_i]\!] \times \ldots \times [\![A_i]\!] \times \ldots [\![A_n]\!] \to [\![A_i]\!]$$

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 $ev_{A,B} : (A \Rightarrow B) \times A \to B$:

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

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 $\mathbb{N}^1 \Rightarrow \mathbb{N}^0$:

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

The predecessor arithmetic operator is denoted by the strategy

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

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

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

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

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

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

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

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

The interpretation of the `Y` combinator is a bit more complicated.

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

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

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

It is easy to see that indeed the g_n forms a chain. We define $[\![\texttt{Y}\ M]\!]$ 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.

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 follow:

$M \sqsubseteq N$ 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. $[\![M]\!] \neq \bot$ implies $M \Downarrow$)*

Lemma 1.4.5 (Computational adequacy for IA terms)**.** *All IA terms are computable. (i.e. $[\![M]\!] \neq \bot$ 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 follow:

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:

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

- 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 $A ::= o \mid A \to A$. 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 $A_1 \to \cdots \to A_n \to o$ (by convention, $\to$ 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:

- $\mathsf{order} : Types \longrightarrow (\mathbb{N}, \leq)$ with $\mathsf{order}(o) = 0$ and $\mathsf{order}(A \to B) = \max(\mathsf{order}(A)+1, \mathsf{order}(B))$.
- $\mathsf{height} : Types \longrightarrow (\mathbb{N}, \leq)$ with $\mathsf{height}(o) = 0$ and $\mathsf{height}(A \to B) = 1+\max(\mathsf{height}(A), \mathsf{height}(B))$.
- $\mathsf{nparam} : Types \longrightarrow (\mathbb{N}, \leq)$ 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 $rank(A_1) \geq rank(A_2) \geq \cdots \geq rank(A_n)$ 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, the rank function that we consider is order, the type order. The term "homogeneous" will refer to order-homogeneity.

2.2 Safe λ-calculus - relaxing the homogeneity condition

In this section, we try to give a general notion of safety for the simply typed λ-calculus. The rules we give here do not assume homogeneity of the types. In section 2.4, we will see another presentation of the safe lambda calculus specialized for the case of homogeneously typed terms.

In this section, we will call safe terms the simply typed lambda terms that are typable within the following system of formation rules:

2.2.1 Rules

We use a set of sequents of the form $\Gamma \vdash^i M : A$ where the meaning is "variables in Γ have orders at least $\mathsf{ord}(A) + i$" where $i \in \mathbb{Z}$. The following set of rules are defined for $i \in \mathbb{Z}$:

$$(\mathbf{seq}^{\mathbf{i}}_{\delta}) \quad \frac{\Gamma \vdash^i M : A}{\Gamma \vdash^{i-\delta} M : A} \quad i \in \mathbb{Z}, \delta > 0$$

$$(\mathbf{var}) \quad \frac{}{x : A \vdash^0 x : A}$$

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

$$(\mathbf{app}^{\mathbf{i}}) \quad \frac{\Gamma \vdash^i M : A \to B \qquad \Gamma \vdash^0 N : A}{\Gamma \vdash^{i+\delta} MN : B} \qquad \delta = \max(0, 1 + \mathsf{ord}(A) - \mathsf{ord}(B))$$

$$(\mathbf{abs}^{\mathbf{i}}) \quad \frac{\Gamma, \overline{x} : \overline{A} \vdash^i M : B}{\Gamma \vdash^0 \lambda \overline{x} : \overline{A}.M : (\overline{A}, B)} \qquad \forall y \in \Gamma : \mathsf{ord}(y) \geq \mathsf{ord}(\overline{A}, B)$$

Note that:

- $(\overline{A}, B)$ 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 in the rule $(\mathbf{app}^{\mathbf{i}})$, the type $A \to B$ is not necessarily homogeneous;
- the environment $\Gamma, \overline{x}$ is not stratified. In particular, variables in $\overline{x}$ do not necessarily have the same order. Also there may be variable in Γ of order smaller than $\mathsf{ord}(x)$ for some variable x in $\overline{x}$.
- The sequents that we really want to prove are those of type $\Gamma \vdash^0 M$. Those terms are the safe terms. Other terms are only used as intermediate steps in a proof.

Remark 2.2.1. This set of rules is equivalent (in term of safe terms that can be generated) to the same set of rules where i is restricted to be a negative integer and where the rule (app^i) becomes:

$$(\mathbf{app}^{\mathbf{i}}) \quad \frac{\Gamma \vdash^i M : A \to B \qquad \Gamma \vdash^0 N : A}{\Gamma \vdash^{\min(i+\delta,0)} MN : B} \qquad \delta = \max(0, 1 + \mathsf{ord}(A) - \mathsf{ord}(B)) \quad i \leq 0$$

With this new set of rules, the sequents of the form $\Gamma \vdash^k M$ with $k > 0$ cannot be derived anymore, however, the set of safe terms that can built remain the same. Indeed, suppose that we derive $\Gamma \vdash^0 M$ using the sequent $\Gamma \vdash^k N$ with $k > 0$ somewhere in the proof. Then an easy induction shows that the sequent $\Gamma \vdash^0 N$ can as well be derived by making use of the rule (seq^i_δ) for $i \leq 0$.

Lemma 2.2.2 (Basic properties)**.** *Suppose $\Gamma \vdash^0 M : B$ is a valid judgment then every variable in Γ has order at least $ord(M)$.*

Proof. An easy induction on the proof tree shows that if $\Gamma \vdash^i M : A$ then the variables in Γ have orders at least $\mathsf{ord}(A) + i$. The induction step for the application is: suppose $\Gamma \vdash^{i+\delta} MN : B$ where $\Gamma \vdash^i M : A \to B$. Then by induction we have $\forall y \in \Gamma : \mathsf{ord}(y) \geq \mathsf{ord}(A \to B) + i = \max(1 + \mathsf{ord}(A), \mathsf{ord}(B)) + i = \delta + \mathsf{ord}(B) + i$. □

2.2.2 Substitution in the safe lambda calculus

The traditional notion of substitution on which the λ-calculus is based on is the following:

Definition 2.2.3 (Substitution)**.**

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

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

Lemma 2.2.4 (No variable capture lemma)**.** *There is no variable capture when performing substitution on a safe term.*

Proof. Suppose that a capture occurs during the substitution $M[N/\varphi]$ where M and N are safe. Then the following conditions must hold:

1. $\varphi : A, \Gamma \vdash^0 M$,
2. $\Gamma \vdash^0 N$,
3. there is a subterm $\lambda\overline{x}.L$ in M (where the abstraction is taken as wide as possible) such that:
4. $\varphi \in fv(\lambda\overline{x}.L)$ (and therefore $\varphi \in fv(L)$),
5. $x \in fv(N)$ for some $x \in \overline{x}$.

By lemma 2.2.2 and (v) we have:

$$\mathsf{ord}(x) \geq \mathsf{ord}(N) = \mathsf{ord}(\varphi) \tag{2.1}$$

$\lambda\overline{x}.L$ is a subterm of M, therefore (since the abstraction $\lambda\overline{x}.L$ is taken as large as possible) there is a node $\Sigma \vdash^u \lambda\overline{x}.L$ in the proof tree for some u.

There are only three kind of rules that can derive an abstraction: $(\mathbf{abs^i})$, $(\mathbf{seq^i_\delta})$ and $(\mathbf{wk^i})$. The only rule that can introduce the abstraction is $(\mathbf{abs^i})$. Therefore the proof tree has the following form:

$$\dfrac{\dfrac{\dfrac{\dfrac{\cdots}{\Sigma' \vdash^0 \lambda\overline{x}.L}(\mathbf{abs^i})}{\cdots}r_1}{\vdots}r_2}{\Sigma \vdash^u \lambda\overline{x}.L}r_l \qquad \text{where } r_j \in \{(\mathbf{seq^i_\delta}),\ (\mathbf{wk^i}) \mid i \in \mathbb{Z}, \delta > 0\}, \quad j \in 1..l.$$

Since $\varphi \in fv(L)$ we must have $\varphi \in \Sigma'$ and since $\Sigma' \vdash^0 \lambda\overline{x}.L$, by lemma 2.2.2 we have:

$$\mathsf{ord}(\varphi) \geq \mathsf{ord}(\lambda\overline{x}.L) \geq \max(1 + \mathsf{ord}(x), \mathsf{ord}(L)) > \mathsf{ord}(x)$$

which contradicts equation (2.1). □

Hence, in the safe lambda calculus setting, we can simplify the definition of the substitution and replace the following equation

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

by

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

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

Indeed, after substitution and (therefore after a β-reduction), the property (ii) of lemma 2.4.1 is not preserved!

To fix this problem, we need to perform several consecutive substitutions until we obtain a safe term. We therefore need to use a substitution that can perform several substitution in parallel, that substitution is called "simultaneous substitution" which definition is a generalization of definition 2.2.3.

Definition 2.2.5 (Simultaneous substitution)**.** We use the notation $\left[\overline{N}/\overline{x}\right]$ for $[N_1 \dots N_n/x_1 \dots x_n]$:

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

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 2.2.4 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 by:

$$M\,[N_1 \dots N_n/x_1 \dots x_n] = M\,[N_1/x_1]\,[N_2/x_2] \dots [N_n/x_n]$$

where $M[N/x]$ denotes the single capture permitting substitution.

Equivalently it can be defined by induction as follow:

Definition 2.2.6 (Capture permitting simultaneous substitution)**.** We use the notation $\left[\overline{N}/\overline{x}\right]$ for $[N_1 \dots N_n/x_1 \dots x_n]$:

$$\begin{array}{rcl}
x_i\left[\overline{N}/\overline{x}\right] & = & N_i \\
y\left[\overline{N}/\overline{x}\right] & = & y \quad \text{where } x \neq y_i \text{ for all } i, \\
(M_1M_2)\left[\overline{N}/\overline{x}\right] & = & (M_1\left[\overline{N}/\overline{x}\right])(M_2\left[\overline{N}/\overline{x}\right]) \\
(\lambda x_i.M)\left[\overline{N}/\overline{x}\right] & = & \lambda x_i.M\,[N_1 \dots N_{i-1}N_{i+1} \dots N_n/x_1 \dots x_{i-1}x_{i+1} \dots x_n] \\
(\lambda \mathbf{y}.\mathbf{M})\left[\overline{\mathbf{N}}/\overline{\mathbf{x}}\right] & = & \lambda \mathbf{y}.\mathbf{M}\left[\overline{\mathbf{N}}/\overline{\mathbf{x}}\right] \text{where } y \neq x_i \text{ for all } i
\end{array}$$

In presence of constant symbols, (this is the case in the safe lambda calculus), we add the following definition:

$$f\left[\overline{N}/\overline{x}\right] = f \quad \text{where } f \in \Sigma \text{ is a first-order constant}$$

Proposition 2.2.7.

$$\Gamma \vdash^i M : A \quad \textit{and} \quad \Gamma \vdash^0 N_k : B_k,\ k \in 1..n \qquad \textit{implies} \qquad \Gamma \vdash^i M[\overline{N}/\overline{x}]$$

Proof. Suppose that $\Gamma \vdash^i M : A$ and $\Gamma \vdash^0 N_k : B_k$ for $k \in 1..n$.

We prove $\Gamma \vdash^i M[\overline{N}/\overline{x}]$ by induction on the size of the proof tree of $\Gamma \vdash^i M : A$ and by case analysis on the last rule used to derive $\Gamma \vdash^i M : A$. 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 $\Gamma \vdash^0 \lambda\overline{y} : \overline{C}.T : (\overline{C}|D)$ where $\Gamma, \overline{y} : \overline{C} \vdash^i T : D$, then by the induction hypothesis $\Gamma, \overline{y} : \overline{C} \vdash^i T\left[\overline{N}/\overline{x}\right] : D$. We can then apply the rule (abs^i) and we get $\Gamma \vdash^0 \lambda\overline{y} : \overline{C}.T\left[\overline{N}/\overline{x}\right]$. □

Corollary 2.2.8 (Simultaneous substitution preserves safety)**.** *If M is safe and N_k is safe for $k \in 1..n$ then $M[\overline{N}/\overline{x}]$ is safe*

Corollary 2.2.9. *Consider a safe multi-redex $(\lambda\overline{x}.M)\overline{N}$ where the N_i are safe, then we can reduced it to $M\left[\overline{N}/\overline{x}\right]$. This reduction is safe and will produce a safe term* ***provided that*** *M* ***is also a safe term****.*

It is important to note that not all multi-β-reduction conserve safety. Only the reductions that peal off enough λ-abstraction will reach a safe term and therefore preserve safety.

To be proved: in the homogeneous case, any reduction strategy performing simultaneous reduction of the variable of the same order preserves safety.

2.2.3 Simultaneous β reduction

We now define a notion of beta reduction that realizes simultaneous substitution. Consider a simply-typed term P. A simultaneous β-redex is a P sub-term of the kind

$$R_1 \equiv (\lambda x_1 x_2 \dots x_n.M)N_1N_2 \dots N_n$$

Reduction is only performed if the simultaneous β-redex encompasses as many lambda abstraction of the same order as possible. Such a redex (which cannot be extended to take into account one more lambda abstraction of the same order) is called a β_s-redex.

Example: consider a term P with a subterm $((\lambda x_1 x_2 \dots x_n.M)N_1N_2 \dots N_n)N_{n+1}$. Suppose that M is the abstraction $M \equiv \lambda x_{n+1}.U$ where $ord(x_{n+1}) = x_1$. Then the redex R_1 will not be considered since it can be enlarged as the redex $(\lambda x_1 x_2 \dots x_n x_{n+1}.M)N_1N_2 \dots N_nN_{n+1}$. Now suppose instead that the term is formed in such a way that there is no N_{n+1} applied on the right of R_1 then the redex R_1 will be considered (whether or not M is an abstraction).

We now give the formal definitions:

The following abbreviations are used $\overline{x} = x_1 \dots x_n$, $\overline{N} = N_1 \dots N_n$, $\overline{x_l} = x_1 \dots x_l$, $\overline{x_r} = x_{l+1} \dots x_n$, $\overline{N_l} = N_1 \dots N_l$ and $\lambda\overline{x} : \overline{A}.T = \lambda x_1^{A_1} \dots x_1^{A_n}.T$.

Definition 2.2.10 (β_s-redex)**.** A safe simply typed lambda term is a redex if it has one of the following forms:

- $(\lambda\overline{x} : \overline{A}.T)\overline{N}$ with $|\overline{x}| = |\overline{N}| = n$, $ord(T) \leq ord(\overline{x}) = ord(x_1) = \dots = ord(x_n)$.
- $(\lambda\overline{x_l} : \overline{A_l}\ \overline{x_r} : \overline{A_r}.T)\overline{N_l}$ with $|\overline{x_l}| = |\overline{N_l}| = l$, $ord(T) \leq ord(\overline{x}) = ord(x_1) = \dots = ord(x_n)$.

These two cases correspond respectively to the formation rules (App) and (App+) of the safe lambda calculus.

Definition 2.2.11 (Simultaneous β-reduction)**.**

- The relation β_s is defined on the set of β_s-redex.

$$\begin{aligned}\beta_s \quad = \quad & \{\left((\lambda\overline{x} : \overline{A}.T)\overline{N}, T\left[\overline{x}/\overline{N}\right]\right) \\ & \text{where } |\overline{x}| = |\overline{N}| = n \text{ and } ord(T) \leq ord(\overline{x}) = ord(x_1) = \dots = ord(x_n)\} \\ & \cup \\ & \{\left((\lambda\overline{x_l} : \overline{A_l}\ \overline{x_r} : \overline{A_r}.T)\overline{N_l}, \lambda\overline{x_r} : \overline{A_r}.T\left[\overline{x_l}/\overline{N_l}\right]\right) \\ & \text{where } |\overline{x}| = |\overline{N}| = n \text{ and } ord(T) \leq ord(\overline{x}) = ord(x_1) = \dots = ord(x_n)\}\end{aligned}$$

Note that in the second case, the substitution is done under the $\lambda\overline{x_r}$. The side condition of the formation rule (App+) guarantees that there will not be any variable capture.

- The simultaneous 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.

Note that β_s-redex are the only redex that can be reduced by $\rightarrow_{\beta_s}$.

2.2.4 Some properties of β_s reduction

We remark that $\rightarrow_{\beta_s} \subset \twoheadrightarrow_\beta$ (i.e. the simultaneous β-reduction relation) is included in the transitive closure of the β-reduction relation. More precisely, if $M \rightarrow_{\beta_s} N$ then $M \twoheadrightarrow_\beta N$. Simultaneous β-reduction is a certain kind of multi-steps β-reduction.

Lemma 2.2.12. *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 β_s-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 (i) and (ii).

□

Lemma 2.2.13. *β_s-reduction preserves safety. (i.e. M safe term and $M\beta_s N$ implies N safe)*

Proof. Simultaneous substitution preserves safety (property 2.2.8), therefore we just need to prove that the relation β_s preserves safety and the result will follow:

Suppose $s\ \beta_s\ t$ then s is a β_s-redex. There are two kinds of them depending on which rule has been used last to form the redex.

- Suppose the last rules used is (App), then the redex is

$$s \equiv (\lambda x_1 \dots x_n.M)N_1 \dots N_n \qquad \rightarrow_{\beta_s} \qquad M[N_1/x_1, \cdots, N_n/x_n] \equiv t$$

  where $ord(M) \leq ord(x_1) = \dots = ord(x_n)$

  The first premise of the rule (App) tells us that M is safe, therefore since substitution preserves safety, (property **??**), t is safe.

- Suppose the last rules used is (App+), then the redex is

$$s \equiv (\lambda\overline{x_l} : \overline{A_l}\ \overline{x_r} : \overline{A_r}.T)\overline{N_l} \qquad \rightarrow_{\beta_s} \qquad \lambda\overline{x_r} : \overline{A_r}.T\left[\overline{x_l}/\overline{N_l}\right] \equiv t$$

  where $ord(T) \leq ord(x_1) = \dots = ord(x_n)$

  $T\left[\overline{x_l}/\overline{N_l}\right]$ is safe for the same reason as in the first case. We can then apply the rule (Abs) and that prove the safety of t.

□

Remark 2.2.14. While $\rightarrow_{\beta_s}$ preserves safety it does not however preserves un-safety: given two terms of the same type, one safe $\Gamma \vdash_s S : A$ and the other unsafe $\Gamma \vdash U : A$, the term $(\lambda xy.y)US$ is unsafe but it β_s-reduces to S which is safe.

2.2.5 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^3 in the sequence s?

Note that the visibility condition does not eliminate the ambiguity, since the two occurrences of q^3 both appear in the P-view at that point (after recovering the pointers of s up to the second last move we get $s = q^4 \; q^3 \; q^2 \; q^3 \; q^2 \quad q^1$, 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^3 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^3 then the play belongs to the strategy $[\![M_1]\!]$.

2.2.6 Game semantics of safe λ terms

We would like to find out whether the safety condition defined in Ong [2005] leads to a pointer economy in the corresponding game semantics.

The example of section 2.2.5 is a good example to start with. We observe that for this particular example and in the safe λ-calculus setting, the ambiguity that led us to the addition of pointers to strategies disappear. More precisely, M_1 is a safe term whereas M_2 is not. Indeed, there is a free occurrence of the variable x of type o in the subterm $f(\lambda y.x)$ which is not abstracted together with y of type o.

1. Is it the case that in general, the pointers from the semantics of safe λ-terms can be reconstructed uniquely from the moves of the play?

2. Is there any unsafe term whose game semantics is a strategy where pointers can be recovered?

   The answer is yes: take the term $T_i = (\lambda xy.y)M_iS$ where $i = 1..2$ and $\Gamma \vdash_s S : A$. T_1 and T_2 both β-reduce to the safe term S, therefore $[\![T_1]\!] = [\![T_2]\!] = [\![S]\!]$. But T_1 is safe whereas T_2 is unsafe. Since it is possible to recover the pointer from the game semantics of S, it is as well possible to recover the pointer from the semantics of T_2 which is unsafe.

3. Is there any unsafe β-normal form whose game semantics is a strategy where pointers can be recovered?

2.2.7 η-extension

Let η-normal form of a term is the term obtained after hereditarily η-expanding every subterm.

2.2.8 Pointers in the game semantics of safe terms are recoverable

We claim that the pointers in the game semantics of a safe term are uniquely recoverable.

Consider a term M safe, we can assume that M is in η normal form (provided that safety is preserved by η-expansion.

The term can be represented by a computation tree: nodes at even depth (starting at level 0) correspond to λ and nodes at odd length corresponds to either application @, variable x or variable followed by an application $f@$. A λ node represented consecutive abstraction of variables.

There justification pointers going upward from variable occurrences to their bindings.

In the game semantics of the term M, the pointers for O and P answers can be recovered by using the well-bracketing condition.

For O-question, the justification pointer always points to its parent node in the computation tree.

For P-question, suppose P ask for the value of variable x. Then there may be several choices for the destination of the pointer but we claim that in the case of safe terms, it should point to the closest parent node (in the path from the root to P-question) whose order is greater than the order of x.

2.3 Particular case of homogeneously-safe lambda terms

We look at a particular sub-class of lambda terms. The types of these terms respect a property call homogeneity as defined in section 2.1.1. A type $(A_1, A_2, \dots A_n, o)$ is said to be homogeneous whenever $\mathsf{order}(A_1) \geq \mathsf{order}(A_2) \geq \dots \geq \mathsf{order}(A_n)$ and each of the A_i are homogeneous. A term is homogeneous if its type is homogeneous.

In their definition of safety (Knapik et al. [2002]), Knapik et al. require that all the recursion equations of a safe recursion scheme have a homogeneous type.

In the rules defining safety for the non-homogeneous case, the only rule that can potentially introduce a non-homogeneous term from a homogeneous one is the abstraction rule. But such a term (a lambda abstraction) corresponds exactly to a recursion equation in the recursion scheme setting of Knapik et al. Therefore requiring that recursions equation have homogeneous type is the same as requiring that all sequents appearing in the proof tree of a safe term are 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!

Lemma 2.3.1. *If a term is homogeneously-safe then there is valid proof tree showing that it is safe containing only judgments of the form* $\Gamma \vdash^k M : T$ *with* $k \in \{-1, 0\}$.

Proof. Assume that $\Gamma \vdash^0 S : T_S$ with T_S homogeneous.

Because of remark 2.2.1 we just need to show that there is a proof tree where there is no sequent of the form $\Gamma \vdash^k M$ with $k < -1$.

Suppose that the proof tree of $\Gamma \vdash^0 S : T_S$ contains $\Gamma \vdash^{-k} M : T$ for $k > 0$ and T a homogeneous type.

The term M is unsafe but we hope that eventually we will form a safe term with it. Since M is unsafe, its order must be strictly greater than 1: we assume that $T = \overline{A}|B$. The homogeneity of $\overline{A}|B$ implies $ord(M) = 1 + ord(\overline{A})$.

We observe that the only two possible ways to make a safe term is to use the rule (app^i) or (abs^i) for some i (they are the only rules which can decrease k):

- Suppose that we want to form a homogeneously-safe term by abstracting a variable. Respecting type homogeneity requires $ord(x) \geq ord(A)$.

  Then it is easy to see that the sequent $\Gamma \vdash^{-k} M : A \to B$ was too strong and that we could have derived the sequent $\Gamma \vdash^0 M : A \to B$ instead!

- Suppose that we want to form a safe term by applying another term safe term $\Gamma \vdash^0 N : A$ to $\Gamma \vdash^{-k} M : A \to B$ (that way the unsafe term M does not appear at an operand position).

  Using the application rules once may not be enough to get a safe term, it may be necessary to perform several consecutive applications until the order of the term becomes low enough. We now consider the very last such application, the one that turns the non safe term into a safe one. This consideration allows us to assume that in the type $A \to B$, A is the last type of its partition, i.e. $\mathsf{ord}(A) \geq \mathsf{ord}(B)$ and $\mathsf{ord}(M) = 1 + \mathsf{ord}(A)$.

  We observe that in the rule (app^{-i}), the environments of the two premises (Γ) are the same. The second premise is $\Gamma \vdash^0 N : A$ therefore by lemma 2.2.2 we have:

$$\forall x \in \Gamma : \mathsf{ord}(x) \geq \mathsf{ord}(N) = \mathsf{ord}(A) = \mathsf{ord}(M) - 1 \tag{2.2}$$

  Again the sequent $\Gamma \vdash^{-k} M : A \to B$ was too strong and we could have derived the sequent $\Gamma \vdash^{-1} M : A \to B$ instead!

□

From this lemma we can derive rules for the homogeneously-safe lambda calculus.

2.3.1 The example of the application rule

We are now about to derive the application rules specialized for the case of homogeneous types. We recall the rule $(\mathbf{app^i})$:

$$(\mathbf{app^i}) \quad \frac{\Gamma \vdash^i M : A \to B \qquad \Gamma \vdash^0 N : A}{\Gamma \vdash^u MN : B} \qquad u = \min(i + \max(0, 1 + \mathsf{ord}(A) - \mathsf{ord}(B)), 0) \quad i \in \{-1, 0\}$$

Type homogeneity implies that $\mathsf{ord}(A) \geq \mathsf{ord}(B) - 1$.

- Suppose that $\mathsf{ord}(A) \geq \mathsf{ord}(B)$ then the condition $i \in \{-1, 0\}$ implies $u = 0$ and we obtain the following rule:

$$(\mathbf{app^i_1}) \quad \frac{\Gamma \vdash^i M : A \to B \qquad \Gamma \vdash^0 N : A}{\Gamma \vdash^0 MN : B} \qquad \mathsf{ord}(A) \geq \mathsf{ord}(B), \quad i \in \{-1, 0\}$$

- Suppose that $\mathsf{ord}(A) = \mathsf{ord}(B) - 1$ then $u = \min(i, 0) = i$ (since $i \in \{-1, 0\}$). We obtain the following rule:

$$(\mathbf{app^i_2}) \quad \frac{\Gamma \vdash^i M : A \to B \qquad \Gamma \vdash^0 N : A,}{\Gamma \vdash^i MN : B} \qquad \mathsf{ord}(A) = \mathsf{ord}(B) - 1, \quad i \in \{-1, 0\}$$

In fact $(\mathbf{app^0_1})$ is redundant since we can derive it from $(\mathbf{app^{-1}_1})$ and $(\mathbf{seq^0_1})$. The rules $(\mathbf{app^i_1})$ and $(\mathbf{app^i_2})$ can be restated as follow:

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

$$(\mathbf{app^{-1}}) \quad \frac{\Gamma \vdash^{-1} M : A \to B \qquad \Gamma \vdash^0 N : A}{\Gamma \vdash^0 MN : B} \qquad \mathsf{ord}(A) \geq \mathsf{ord}(B)$$

$$(\mathbf{app'^{-1}}) \quad \frac{\Gamma \vdash^{-1} M : A \to B \qquad \Gamma \vdash^0 N : A}{\Gamma \vdash^{-1} MN : B} \qquad \mathsf{ord}(A) = \mathsf{ord}(B) - 1$$

2.3.2 The abstraction rule

Let us derive the abstraction rule specialized for the case of homogeneous types. We recall the rule (**abs**):

$$(\mathbf{abs^i}) \quad \frac{\Gamma, \overline{x} : \overline{A} \vdash^i M : B}{\Gamma \vdash^0 \lambda \overline{x} : \overline{A}.M : (\overline{A}, B)} \qquad \forall y \in \Gamma : \mathsf{ord}(y) \geq \mathsf{ord}(\overline{A}, B)$$

We now partitionned the context Γ according to the order of the variables. The partition 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 the single variable $\overline{x} = x$, 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 we can change the abstraction rule so that it only allows abstraction of the lowest variable partition. The rule can then be used repeatedely if further partitions need to be abstracted. We obtained the following rule where the side-condition has disappeared:

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

2.3.3 The entire set of rules

Table 2.1 gives the entire set of rules.

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

$$(\mathbf{var}) \quad \frac{}{x : A \vdash^0 x : A}$$

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

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

$$(\mathbf{app^{-1}}) \quad \frac{\Gamma \vdash^{-1} M : A \to B \qquad \Gamma \vdash^0 N : A,}{\Gamma \vdash^0 MN : B} \qquad \mathsf{ord}(A) \geq \mathsf{ord}(B)$$

$$(\mathbf{app'^{-1}}) \quad \frac{\Gamma \vdash^{-1} M : A \to B \qquad \Gamma \vdash^0 N : A,}{\Gamma \vdash^{-1} MN : B} \qquad \mathsf{ord}(A) = \mathsf{ord}(B) - 1$$

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

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

Table 2.1: Rules of the homogeneous safe lambda calculus

If we rename the sequents $\vdash^0$ and $\vdash^{-1}$ into $\vdash^+$ and $\vdash^-$ respectively we observe that the rules are similar to the ones given in Ong [2005] except that the rule $(\mathbf{app'^{-1}})$ is missing in Ong [2005].

2.4 Safe λ-calculus - Another presentation

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

2.4.1 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 $x_{11} : A_{11}, \cdots, x_{1r} : A_{1r}, \cdots \vdash M : B$. *Valid typing judgements* of the system are defined by induction over the following rules, where Δ is a given homogeneously-typed alphabet:

$$\frac{\Sigma \vdash M : B \qquad \Sigma \subset \Delta}{\Delta \vdash M : B}(\text{wk})$$

$$\frac{\Gamma \vdash M : B \qquad \sigma(\Gamma) \text{ homogeneous}}{\sigma(\Gamma) \vdash M : B}(\text{perm})$$

$$\frac{b : o^r \to o \in \Sigma}{\vdash b : o^r \to o}(\Sigma\text{-const})$$

$$\frac{}{\overline{x_{ij}} : \overline{A_{ij}} \vdash x_{ij} : A_{ij}}(\text{var})$$

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

$$\frac{\Gamma \vdash M : (\overline{B_1} \,|\, \cdots \,|\, \overline{B_m} \,|\, o) \qquad \Gamma \vdash N_1 : B_{11} \quad \cdots \quad \Gamma \vdash N_l : B_{1l} \qquad l = |\overline{B_1}|}{\Gamma \vdash MN_1 \cdots N_{l_1} : (\overline{B_2} \,|\, \cdots \,|\, \overline{B_m} \,|\, o)}(\text{app})$$

$$\frac{\Gamma \vdash M : (\overline{B_1} \,|\, \cdots \,|\, \overline{B_m} \,|\, o) \qquad \Gamma \vdash N_1 : B_{11} \quad \cdots \quad \Gamma \vdash N_l : B_{1l} \qquad l < |\overline{B_1}|}{\Gamma \vdash MN_1 \cdots N_{l_1} : (\overline{B_2} \,|\, \cdots \,|\, \overline{B_m} \,|\, o)}(\text{app}+)$$

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

Lemma 2.4.1 (Basic properties)**.** *Suppose $\Gamma \vdash_s M : B$ is a valid judgment then*

(i) B is homogeneous

(ii) Every free variables of M has order at least $ord(M)$

Definition 2.4.2 (Simultaneous substitution for safe terms)**.** We use the notation $\left[\overline{N}/\overline{x}\right]$ for $[N_1 \dots N_n/x_1 \dots x_n]$:

$$\begin{array}{rcl}
x_i \left[\overline{N}/\overline{x}\right] & = & N_i \\
y \left[\overline{N}/\overline{x}\right] & = & y \quad \text{if } y \neq x_i \text{ for all } i, \\
(MN_1 \dots N_l) \left[\overline{N}/\overline{x}\right] & = & (M \left[\overline{N}/\overline{x}\right])(N_1 \left[\overline{N}/\overline{x}\right]) \dots (N_l \left[\overline{N}/\overline{x}\right]) \\
(\lambda x_i.M) \left[\overline{N}/\overline{x}\right] & = & \lambda x_i.M \left[N_1 \dots N_{i-1} N_{i+1} \dots N_n / x_1 \dots x_{i-1} x_{i+1} \dots x_n\right] \\
(\lambda \overline{y} : \overline{A}.T) \left[\overline{N}/\overline{x}\right] & = & \lambda \overline{z}.T \left[\overline{z}/\overline{y}\right] \left[\overline{N}/\overline{x}\right] \\
 & & \text{where } T \text{ is a safe term and } \overline{z} = z_1, \dots z_p \text{ are all fresh variables}
\end{array}$$

Remark: On safe terms, simultaneous substitution can be achieved inductively by only performing simultaneous substitution on smaller sub-terms that are safe.

We now prove that the "no variable clash lemma" also hold with this new definition of the homogeneous safe λ-calculus.

Lemma 2.4.3 (No variable clash lemma)**.** *In the safe λ-calculus, there is no clash of variable name when performing substitution:*

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

provided the substitution is performed simultaneously on all *free variables of the same order in M i.e. $\{ x_1, \cdots, x_n \}$ is the set variables of the same order as x_1 that occur free in M.*

Proof. First we note that if the substitutions were consecutive ($M \left[N_1/x_1\right] \dots \left[N_n/x_n\right]$) instead of being simultaneous then a variable capture would arise if some N_i has a free occurrence of a variable x_j with $j > i$. However this capture does not happen when performing the substitutions simultaneously as follow: $M \left[N_1, \dots N_n/x_1, \dots x_n\right]$.

Suppose that a variable capture occurs in the term M: M has a subterm $\lambda y_1 \dots y_p.T$ such that some x_i appears freely in T and some y_k appears freely in N_i. Because of the previous remark, we can assume that the subterm $\lambda y_1 \dots y_p.T$ is safe.

Since x_i appears freely in the safe term $\lambda y_1 \dots y_p.T$, by Lemma 2.4.1 (ii) we get:

$$ord(x_i) \geq ord(\lambda y_1 \dots y_p.T) \geq 1 + ord(y_k) > ord(y_k)$$

Since y_k appears freely in the safe term N_i, Lemma 2.4.1 (ii) gives:

$$ord(y_k) \geq ord(N_i) = ord(x_i)$$

Hence we reach a contradiction. □

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.

J. M. E. Hyland and C.-H. L. Ong. On full abstraction for pcf: I, II, and III, journal = Inf. Comput., year = 2000, volume = 163, pages = 285–408, number = 2, address = Duluth, MN, USA, doi = http://dx.doi.org/10.1006/inco.2000.2917, issn = 0890-5401, publisher = Academic Press, Inc.,.

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

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

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.