Notes on game semantics and safe-lambda calculus

William Blum

February 22, 2006

Chapter 1

Game semantics

1.1 Semantics of programming languages

Before the introduction of game semantics in the 1990s, there were principally two kinds of models for programming languages: operational semantics and denotational semantics. Operational semantics defines a program using rewriting rules. whereas denotational semantics defines a program using mathematical function.

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 program itself.

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 ([AMJ93]).

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 from 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. 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 strong 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 [AMJ94], Hyland and Ong [AMJ00] and Nickau. There are all based on game semantics.

1.1.2 Remark

1. Well-bracketing condition is a condition on P-answers only, it does not constraint O-answers (see [AMJ97]).

1.1.3 Second order

Lemma 1.1.1 (Pointers are superfluous up to order 2)**.** *Let A be the arena corresponding to a PCF type 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. The only base type here is `exp`. The interpretation of it is the flat game of natural numbers $\mathbf{N}$. The arena is:

q

1 2 ...

Let T be a PCF type of order at most 2. $T = T_1 \to T_2 \to \ldots T_n \to$ `exp`. Let $I_k = \{1..n \mid \mathsf{order}(T_i) = k\}$ for $k = 0..1$.

The arenas T_i for $i \in I_0$ and T_j for $j \in I_1$ are given below:

q_i q'_j

1_i 2_i $\ldots$ q_j $1'_j$ $2'_j$ $\ldots$

1_j 2_j $\ldots$

The entire arena A is

q

1 2 $\ldots$

T_1 T_i T_n

where each triangle represents one of the two possible sub arena.

First note that the well-bracketing condition allows us to recover the pointers for all the P answer moves (simply make the answers point to the last pending question with the same tag).

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 $L = \{q'_i q_i \mid i \in I_1\}$. We consider the following cases:

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

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 depending on the case u corresponds to:

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:

- $m = a \in \mathbf{N}$ is an answer to the initial question q, then m points to q:
  $\hat{s} = q \cdots m$
  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.
- $m = q'_i$, a first order question, then q'_i points to q,
  $?(s) = qq'_i$ and it is O's turn after s therefore s falls into category B.

case B $?(u) \in q \cdot L^* \cdot q'_i$ 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 a'_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 q'_i. Indeed, suppose that there is a move $x \in M_A$ such that $\hat{u} = q \cdots q'_i \; x$ then by visibility, the O-move x should points to a move in the O-view a that point. The O-view is qq'_i, therefore x can only points to q'_i. But then, q'_i is not a pending question in s which is a contradiction.
  Therefore $\llcorner \hat{u} \lrcorner = \llcorner q \cdots q'_i \lrcorner = qq'_i$.
  Hence a'_j can only point to q'_i (and therefore $i = j$).
  We then have $?(s) = ?(u \cdot a'_i) \in q \cdot L^*$ 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 q'_i. We then have $?(s) = ?(u \cdot q_i) \in q \cdot L^* \cdot q'_i q_i$. This falls into category C.

case C $?(u) \in q \cdot L^* \cdot q'_i q_i$ 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 $?(s) = ?(u \cdot a_i) \in q \cdot L^* \cdot q'_i$. 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.
- $m = q'_j$ 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 $m = a_i \in \mathbf{N}$. This moves can only points to q_i (this is the only pending question tagged by $i \in I_0$).

Then $?(\hat{s}) = ?(\hat{u} \cdot a_i) = ?(q \cdots q_i \cdots a_i) \in q \cdot L^*$ therefore s falls either into category A or C.

$\square$

1.1.4 Idealized Algol

1.1.5 First-order Idealized Algol

Dan R. Ghica and Guy McCusker
regular language Semantics for first order

1.1.6 Call-by-Value first-order Idealized Algol

Call-by-value Programming Language, Ghica

Chapter 2

Safe λ-calculus

We recall the definition of the safe λ-calculus given in [Ong05].

2.0.7 Homogenous 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: $\mathsf{rank} : \mathit{Types} \longrightarrow (L, \leq)$ where $(L, \leq)$ is any linearly ordered set. Possible candidates for the ranking function are:

- $\mathsf{order} : \mathit{Types} \longrightarrow (\mathbb{N}, \leq)$ with $\mathsf{order}(o) = 0$ and $\mathsf{order}(A \rightarrow B) = \max(\mathsf{order}(A)+1, \mathsf{order}(B))$.
- $\mathsf{height} : \mathit{Types} \longrightarrow (\mathbb{N}, \leq)$ with $\mathsf{height}(o) = 0$ and $\mathsf{height}(A \rightarrow B) = 1+\max(\mathsf{height}(A), \mathsf{height}(B))$.
- $\mathsf{nparam} : \mathit{Types} \longrightarrow (\mathbb{N}, \leq)$ with $\mathsf{nparam}(o) = 0$ and $\mathsf{nparam}(A_1, \cdots, A_n) = n$.
- $\mathsf{ordernp} : \mathit{Types} \longrightarrow (\mathbb{N} \times \mathbb{N}, \leq)$ with $\mathsf{ordernp}(t) = (\mathsf{order}(t), \mathsf{nparam}(t))$ for $t \in \mathit{Types}$.

Following [KNU02], 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

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

2.0.8 Rules

These rules are a corrected version of [AdMO05]

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

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

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:

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

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

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

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

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

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

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

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

- B *is homogeneous*
- *Every free variables of* M *has order at least* $ord(M)$

2.0.9 Simultaneous substitution

The first interesting property that we will prove for the safe λ-calculus is that when performing substitution on safe λ term, there is no need to rename bound variables provided that the substitution is performed *simultaneously* on all free variables of the same order.

The requirement that the substitution is performed simultaneously is quite strong: implementing simultaneously substitution requires to have access to an unbound number of fresh variables. Therefore in safe lambda calculus the fact that there is no variable capture during substitution does not really lead to a complete economy of variable names.

Definition 2.0.3 (Simultaneous substitution)**.** Substitution for simply typed lambda term is defined as follow:

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

Simultaneous substitution is defined as follow:

$$M\left[N_1 \dots N_n/x_1 \dots x_n\right] = M\left[z_2/x_2\right]\dots\left[z_n/x_n\right]\left[N_1/x_1\right]\left[N_2/z_2\right]\dots\left[N_n/z_n\right]$$

where $z_2 \dots z_n$ are fresh variables.

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

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

In fact, we can define the simultaneous substitution inductively without relying on the definition of the standard substitution. Here is the definition specialized to the safe lambda calculus case:

Definition 2.0.4 (Simultaneous substitution in the safe-lambda calculus)**.** We use the notation $\left[\overline{N}/\overline{x}\right]$ for $\left[N_1 \dots N_n/x_1 \dots x_n\right]$:

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

This alternative definition permits us to observe the following two properties:

Property 2.0.5 (Simultaneous substitution on safe terms).

1. Performing simultaneous substitution on a safe term can be achieved by inductively applying the simultaneous substitution on other *safe* sub-terms only.

2. Simultaneous substitution of safe terms preserves safety.

Proof. 1. By analysing the inductive definition 2.0.4, we observe that each substitution is performed on a safe term provided that the original term is safe. For the abstraction case, we remark that the substitution $\left[\overline{z}/\overline{y}\right]$ is just a renaming of variable that preserve safety.

2. Consider the safe terms $\Gamma \vdash_s S : A$ and $\Gamma \vdash_s N_i : B_i$ for $i = 1..n$.

   We prove that $S\left[\overline{x}/\overline{N}\right]$ is safe by induction on the size of the proof tree of $\Gamma \vdash_s S : A$. We just give the detail for the abstraction case:

   Assume that we proved the property for all term whose proof tree is smaller than S. Suppose $S = \lambda\overline{y} : \overline{A}.T$ where T is a safe term, then $T\left[\overline{z}/\overline{y}\right]$ is just the term T with its variable $\overline{y}$ renamed to fresh names therefore it is safe. By the induction hypothesis, $T\left[\overline{z}/\overline{y}\right]\left[\overline{N}/\overline{x}\right]$ is also safe. We can apply the rule (abs) of the safe-lambda calculus and we get that $\lambda\overline{z}.T\left[\overline{z}/\overline{y}\right]\left[\overline{N}/\overline{x}\right]$ is safe.

□

2.0.10 Simultaneous substitution does not involve renaming

Lemma 2.0.6 (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 substitution is not simultaneous ($M[N_1/x_1]\dots[N_n/x_n]$), then a variable capture arises 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: $M[N_1,\dots N_n/x_1,\dots x_n]$.

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. By Property 2.0.5, 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 2.0.2 (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 2.0.2 (ii) gives:

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

Hence we reach a contradiction. □

2.0.11 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

$$R_1 \equiv (\lambda x_1 x_2 \dots x_n.M) N_1 N_2 \dots N_n$$

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 $((\lambda x_1 x_2 \dots x_n.M) N_1 N_2 \dots N_n) N_{n+1}$. 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 $(\lambda x_1 x_2 \dots x_n x_{n+1}.M) N_1 N_2 \dots N_n N_{n+1}$. 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 abbreviation 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 2.0.7 (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 2.0.8 (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 $\rightarrow_{\beta_s}$.

2.0.12 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 2.0.9. *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).

$\square$

Lemma 2.0.10. *β_s-reduction preserves safety. (i.e. M safe term and $M \beta_s N$ implies N safe)*

Proof. Simultaneous substitution preserves safety (property 2.0.5), 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

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

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

  The first premise of the rule (App) tells us that M is safe, therefore since substitution preserves safety, (property 2.0.5), 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 $ord(T) \leq ord(x_1) = \ldots = ord(x_n)$

  $T \left[\overline{x_l} / \overline{N_l} \right]$ 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.

$\square$

Remark 2.0.11. 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.

2.0.13 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 [AMJ97] 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 $s = q^4 \quad q^3 \quad q^2 \quad q^3 \quad q^2 \quad q^1$, 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 $[\![M_1]\!]$.

2.0.14 Game semantics of safe λ terms

We would like to find out whether the safety condition defined in [Ong05] leads to a pointer economy in the corresponding game semantics.

The example of section 2.0.13 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?

2.0.15 η-extension

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

2.0.16 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.

2.0.17 Safe lambda calculus without homogeneous types

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 $i \in \mathbb{Z}$:

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

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

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

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

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

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 $A \rightarrow B$ 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 2.0.12. This set of rules is equivalent (in term of safe terms that can be generated) to the same set of rule where i is restricted to be a negative integer and where the rule (app^i) becomes:

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

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 2.0.13 (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 $\forall y \in \Gamma : \mathsf{ord}(y) \geq \mathsf{ord}(A \to B) + i = \max(1 + \mathsf{ord}(A), \mathsf{ord}(B)) + i = \delta + \mathsf{ord}(B) + i$. ☐

Lemma 2.0.14 (No variable capture lemma)**.** *Provided that substitution is done simultaneously (even for variable of different order), there is not variable capture when performing substitution on a safe (non homogeneous) 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 2.0.13 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 2.0.13 we have:

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

which contradicts equation (2.1). ☐

2.1 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 2.0.7. A type $(A_1, A_2, \dots A_n, o)$ is said to be homogeneous whenever $\mathsf{order}(A_1) \geq \mathsf{order}(A_2) \geq \dots \geq \mathsf{order}(A_n)$. A term is homogeneous if its type is homogeneous.

In their definition of safety ([KNU02]), 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 2.1.1. *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 2.0.12 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 $\Gamma \vdash^{-k} M : A \rightarrow B$ was too strong and that we could have derived the sequent $\Gamma \vdash^0 M : A \rightarrow B$ instead!

- Suppose that we want to form a safe term by applying another term safe term $\Gamma \vdash^0 N : A$ to $\Gamma \vdash^{-k} M : A \rightarrow B$ (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 $A \rightarrow B$, 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 2.0.13 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 $\Gamma \vdash^{-k} M : A \rightarrow B$ was too strong and we could have derived the sequent $\Gamma \vdash^{-1} M : A \rightarrow B$ instead!

□

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

2.1.1 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:

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

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

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

2.1.2 The abstraction rule

Let us derive the abstraction rule specialized for the case of homogeneous types. We recall the rule $(\mathbf{abs})$:

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

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:

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

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

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

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

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

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

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

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

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

Table 2.1: Rules of the homogeneous safe lambda calculus

2.1.3 The entire set of rules

Table 2.1 gives the entire set of rules.

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 [Ong05] except that the rule ($\mathbf{app'^{-1}}$) is missing in [Ong05].

Bibliography

[AdMO05] 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.

[AMJ93] Samson Abramsky, Pasquale Malacaria, and Radha Jagadeesan. A type-theoretical alternative to iswim, cuch, owhy. *Theor. Comput. Sci.*, 121(1-2):411–440, 1993.

[AMJ94] Samson Abramsky, Pasquale Malacaria, and Radha Jagadeesan. Full abstraction for PCF. In *Theoretical Aspects of Computer Software*, pages 1–15, 1994.

[AMJ97] Samson Abramsky, Pasquale Malacaria, and Radha Jagadeesan. Game semantics. In *Proceedings of Marktorberdorf '97 Summerschool*, 1997. Lecture notes.

[AMJ00] Samson Abramsky, Pasquale Malacaria, and Radha Jagadeesan. On full abstraction for pcf: I, ii, and iii. *Inf. Comput.*, 163(2):285–408, 2000.

[KNU02] T. Knapik, D. Niwiński, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FOSSACS'02*, pages 205–222. Springer, 2002. LNCS Vol. 2303.

[Ong05] C.-H. L. Ong. Safe lambda calculus: Some questions. Note on the safe lambda calculus., December 2005.