Traces of some primitive recursive schema

René David         Pierre Valarcher

March 2010
TR-LACL-2010-3
Traces of some primitive recursive schema

Traces of some primitive recursive schema

René David
LAMI
University of Chambery
Rene.David@univ-savoie.fr

Pierre Valarcher
LACL
University of Paris Est
pierre.valarcher@univ-paris-est.fr

March 8, 2010

Abstract

This paper focuses on the expressiveness of some programming languages from the point of view of algorithms writable. It considers mainly a restricted class of computable functions, primitive recursive ones, and studies programming languages that compute exclusively this class of total functions. Using the notion of trace developed by the first author in [10], we study, in a fine way, the intensional behaviour of those programming languages. It allows us to compare the expressiveness of those languages and then answer some questions given in [19].

1 Introduction

Almost all the programming devices are equivalent with respect to the class of functions they can compute. This holds both for theoretical models (Turing machines, Lambda-Calculus, ...) or real programming languages (Pascal, C, Lisp, ML, ...). Some restrictions give (in theory) a limitation. For example:

- The primitive recursive functions form a strict subset of the set of the recursive functions and even a strict subset of the total recursive functions.
- The typed Lambda-Calculus (for example the system F) allows to compute only the recursive functions whose totality is provable in Peano second order arithmetic.

We are interested in this paper with the primitive recursive restriction. This one is computably sufficient since all the recursive functions for which the computation time (say, on a Turing machine) is bounded by a "reasonnable" function (an elementary function is, for that purpose, reasonnable) of the size of the data are in these classes.

However, there is an essential difference, for a given model of computation, between the following facts:

- The function can be computed (extensionaly) in the model.
- A particular algorithm can be represented in this model.

First work was done by L. Colson (see [2]) using denotational semantics (the used domain is the lazy natural integers [13]) to prove that, though the function min which computes the minimum of two integers is obviously a primitive recursive function (see [17] for a formal definition), there is no way to represent, in the model of primitive recursive programs (which are called
PR-combinators), the good algorithm, i.e. the one which decreases alternatively both arguments. He proved an ultimate obstination theorem and showed that every PR-combinator must choose one (and only one) of its arguments and thus the alternation between arguments is impossible. A constructive proof of this property can be found in [5].

This work has been followed by the first author (see [10]) who developed a new semantics (the trace of computation) allowing him to prove a new property (the backtracking property) for any primitive recursive program using any kind of data types.

The second author has shown some new results on intensional behaviour of other primitive recursive schemes (in [20]).

In the same framework, L. Colson and D. Fredholm (see [14]) show that call-by-value strategy (with primitive recursion over lists of integers and with primitive recursion in higher types, called system T of Gödel) does not allow to compute the good algorithm of the min function.

Similar questions have been studied by S. Brookes and D. Dancanet in [1] and [7] with non-determinism and CDS languages.

Recently, in [16], Y. Moschovakis has established a linear lower bound for the complexity of non-trivial primitive recursive program from piecewise linear given functions. His main result is that logtime programs for the greatest common divisor from such givens (such as Stein’s) cannot matched in efficiency by primitive recursive programs from the same given functions. He ended by an open problem relative to the classical Euclidean algorithm (L. Van Den Dries gives a partial answer in [12]).

In [6], the min problem is studied in an imperative framework: LOOP language which computes only primitive recursive functions ([15]). But there is no good program for the min function too.

From now on, most results are negative answers to particular problems: the min problem, the gcd problem, . . .

The first author showed in [10] that, even with the use of lists, some “intentional behaviour” cannot be obtained. Intuitively the intentional behaviour of an algorithm is the way it uses the informations on its arguments to produce the result. More precisely it is the function that give the number of output symbols (in the case of unary integers: the number of S) from the number of input symbols (of the arguments) really used. For example, the main result of [10] is (intuitively) that a primitive recursive algorithm using any kind of data can only “memorize” the information on at most one argument (For a precise definition of this “backtracking property”, see section 5 below). Moreover, the notion of trace allows to extend some decidability results about computation over infinite integer (see [11]).

In [19] and [20], the second author studied various extensions of the primitive recursive algorithms from the intentional point of view: either by allowing the use of lists of integers or by allowing new recursion schemata:

- **PRM** (for mutual recursion): \( f_1(Sn) = h_1(n, f_1(n), f_2(n)) \) and \( f_2(Sn) = h_2(n, f_1(n), f_2(n)) \)
- **PRA** (for alternate recursion): \( f(Sn, Sm) = h(n, m, f(n, m)) \)
- **PRV** (for variable recursion): \( f(Sn, m) = h(n, m, f(n, g(n, m))) \).

It is important to note that, from the extensional point of view, the functions we obtain in this way are primitive recursive. In particular he showed that the intentional behaviour of PR-combinators is exactly the extensional behaviour of the PR-combinators for which the base cases always are 0 (they are called PRN-combinators). He also showed how some schemata can be simulated by some others. In [19], many questions remained unsolved. This paper solves some of them and gives some new ones that remain unsolved.
In [2] and [20] the basic tool was the Scott domain of lazy integers (see [13] and [18]). We use here the notion of trace introduced by the first author in [10]. The intuition is the following: a lazy integer is a way of “filling” some “cells” with S or 0. An empty cell means a lack of information. A cell (numbered n) may be filled iff the cell numbered (n - 1) is filled with S.

There are 3 kinds of “integers”:

- $S^0 0$ (the complete integers) are represented by: the cells 0, ..., (n - 1) are filled with S and the cell n is filled with 0.
- $S^n$ (the incomplete integers) are represented by: the cells 0, ..., (n - 1) are filled with S and the cell n is empty.
- $S^\infty$ (the infinite integer) is represented by: all the cells are filled with S.

The “trace” of a computation is a word on the following alphabet $\{x_n/n \in \mathbb{N} \text{ and } x \text{ a letter}\} \cup \{0, S\}$ corresponding to some steps of the computation. If, at some step, the content of the cell numbered n of the argument called x is needed, the corresponding symbol in the trace is $x_n$. If at some step an output symbol (S or 0) is produced, the corresponding symbol in the trace is this output symbol.

The intention of a combinator is fully represented by the trace of the combinator applied to infinite integers. The main point is that traces may be composed (cf the proposition 7) whereas in Scott domains, only the results may be composed loosing then somehow the way this result has been obtained.

**Organization of the paper** The section 2 recalls the definitions about the traces and (without proofs) the main properties. For almost the whole paper, the only data type used is Int, the set of (lazy) integers. We thus recall the (simplified) definitions in this particular case and give (for completeness) the general definitions (used in section 7) in an appendix. In section 3 we extend the main result of [19] ($\text{Int}(PR) = \text{Ext}(PRN)$) to PRM and PRA. (Note that in [10] the proof used the ultimate obstinacy theorem and this theorem fails in PRA). The section 4 shows that for some restriction of PRV called PRS (the only change of parameters allowed is: $m \rightarrow Sm$), the previous result is not true. This is proved by showing that PRSN satisfies the ultimate obstinacy theorem. We donot know what happens for PRV or PRL (PRL-combinators allow the use of lists). In the section 5 we show that PRV has the backtracking property. This was proved in [10] for PRL. Section 6 shows that the expressive power of PRA and PRM are strictly included in the one of PRL0 (the restriction of PRL where only lists of complete integers are allowed). The section 7 shows that, for the intentional point of view, PRS, PRV and PRL0 are equivalent. Note that we donot know whether or not they are equivalent to the full PRL. Finally we conclude and give some open questions.

**2 Definitions and notations**

In this paper, we mainly use the data type of integers (it is called Int to avoid the confusion with the set N of usual integers). This simplifies the definitions and notations of [10] because Int is a data type with no branching. The following definitions and properties are thus the adaptation of those of [10] to this particular case. The main (and in fact, the only) difference is the following: in [10] a trace is a tree with labels attached to the nodes and the edges. The nodes are labelled with constructors of the data type and the edges are labelled with words on the vocabulary $\{x_\alpha/x \text{ is a letter and } \alpha \text{ is an address}\}$. Here, the trace is a single word combining both the labels of the edges and the nodes. These definitions are sufficient for the part of the
paper dealing only with integers, i.e. the whole paper with the exception (in the section 6 and 7) of the definition of $PRL_0$ (see section 6) and the theorem $Int(PRL_0) \subset Int(PRS)$ (see section 7). For a precise definition of traces in the case of a general data type the reader is referred to the appendix and, for proofs, to [10].

2.1 The trace

The trace represents the precise way an algorithm uses its inputs.

Definition 1 $Int = \{S^n/n \in N\} \cup \{S^0/n \in N\} \cup \{S^\infty\}$. $S^n$ is called a complete integer. $S^0$ is called an incomplete integer and $S^\infty$ is called the infinite integer. $Int$ has the following partial ordering: for $n \leq m$, then $S^n \leq S^m \leq S^\infty$ and $S^n \leq S^m0$ (note that $S^m0$ and $S^m0$ are not ordered).

Let $V$ be the alphabet $\{x_n/x$ is a letter and $n \in N\} \cup \{S, 0\}$ in the following definitions:

Definition 2 The set $TR$ of traces is defined by: $w$ is in $TR$ iff $w$ is a finite (non empty) or infinite sequence of symbols in $V$ such that:

- $0$ may only appear as the last symbol in $w$.
- the set of distinct letters appearing in $w$ is finite.

Definition 3 For $w$ in $TR$, $Val(w)$ is the element of $Int$ defined by: if there are infinitely many $S$ in $w$, then $S^\infty$, else let $n$ be the number of $S$ in $w$, if the last symbol of $w$ is $0$, then $S^n0$ else $S^n$.

Definition 4 $TR_f$ is the set of finite sequences in $TR$.

Definition 5 For $v, w$ in $TR$, $w \leq v$ means that $w$ is an initial segment of $v$. The $\oplus$ symbol denotes the concatenation on sequences.

Definition 6 The following traces are called “elements with name $x$” (or “named elements” if we donot care of the name)

$$
S^\infty(x) = x_0Sx_1Sx_2...Sx_n...
$$

$$
S^n(x) = x_0Sx_1Sx_2...Sx_n
$$

$$
S^00(x) = x_0Sx_1Sx_2...Sx_n0
$$

If $w$ is the trace of a computation $Val(w)$ represents the output of this computation.

Proposition 1 Every trace is the $Sup$ of an increasing sequence of traces in $TR_f$. Conversely, let $(w_k)$ be an increasing sequence of traces (using a fixed number of distinct letters). Then $Sup w_k$ is in $TR$.

We now define six (functional) programming languages that compute exactly the class $PR$. They correspond to some definitions of the class $PR$.

Definition 7 See [17] for the extensional equivalence:

$PR$ is the least set of combinators containing the constant $0$, the unary combinator $S$, the projections and that is closed by composition and primitive recursion.
PRM is the least set containing PR that is closed by mutual recursion. For example the following scheme is allowed:

\[
\begin{align*}
    f_1(S_n, m) &= h_1(n, m, f_1(n, m), f_2(n, m)) \\
    f_2(S_n, m) &= h_2(n, m, f_1(n, m), f_2(n, m))
\end{align*}
\]

PRV is the least set containing PR that is closed by recursion with a possible variation of the parameters in the recursive call. For example the following scheme is allowed:

\[
\begin{align*}
    f(S_n, m, p) &= h(n, m, p, f(n, j_1(n, m, p), j_2(n, m, p)))).
\end{align*}
\]

PRS is the restriction of PRV where at most one parameter \( m \) may be replaced by \( S m \) in the recursive call. For example the following scheme is allowed:

\[
\begin{align*}
    f(S_n, m, p) &= h(n, m, p, f(n, m, S p)).
\end{align*}
\]

PRA is the least set containing PR that is closed by alternate recursion. For example the following recursion scheme is allowed:

\[
\begin{align*}
    f(0, m, p) &= g_1(m, p) \\
    f(S_n, 0, p) &= g_2(n, p) \\
    f(S_n, S m, p) &= h(n, m, p, f(n, m, p)).
\end{align*}
\]

PRL is the least set of (well typed) combinators using as data types the integers and the list of integers, containing the constant 0, the unary combinator \( S \), the constant \( n i l \), the combinator \( c o n s \) (of type : \( \text{Int, List} \rightarrow \text{List} \)) the projections and that is closed by composition and primitive recursion. Primitive recursion on \( \text{List} \) is as follows:

\[
\begin{align*}
    f(n i l, x) &= g(x) \\
    f(c o n s(n, l), x) &= h(n, l, x, f(l, x))
\end{align*}
\]

Remark In the section 7, we will restrict PRV to the change of at most one parameter in the recursive call.

Definition 8 A function \( F \) from \( \text{TR}^n \) to \( \text{TR} \) is increasing if for every \( v_1, ..., v_n \) and \( w_1, ..., w_n \) such that \( v_i \leq w_i \) for every \( i \), then \( F(v_1, ..., v_n) \leq F(w_1, ..., w_n) \). \( F \) is continuous if it is increasing and it preserves the \( \text{Sup} \).

The next proposition shows how a combinator acts not only on \( \text{Int} \) but also on \( \text{TR} \).

Proposition 2 Every \( n \)-ary combinator \( f \) in PR induces (in an unique way) a continuous function (denoted by \( F \)) from \( \text{TR}^n \) to \( \text{TR} \) such that:

- if \( f \) is the \( i \)-th projection then \( F(w_1, ..., w_n) = w_i \)
- if \( f \) is \( 0 \) (resp \( S \)) then \( F(w_1, ..., w_n) = 0 \) (resp. \( S w_1 \))
- if \( f = g(h_1, ..., h_k) \) and \( r_i = H_i(w_1, ..., w_n) \) then
  \[
  F(w_1, ..., w_n) = G(r_1, ..., r_k)
  \]
- if \( f \) is defined by recursion from \( g \) and \( h \) and the recursive argument is the first one :
- If \( w_1 \) contains neither 0 nor \( S \), then \( F(w_1, ..., w_n) = w_1 \)
- If \( w_1 = v_1 \oplus 0 \) and \( v_1 \) does not contain \( S \), then \( F(w_1, ..., w_n) = v_1 \oplus G(w_2, ..., w_n) \)
- If \( w_1 = v_1 \oplus S \oplus v_2 \) and \( v_1 \) does not contain \( S \), then \( F(w_1, ..., w_n) = v_1 \oplus H(v_2, F(v_2, w_2, ..., w_n), w_2, ..., w_n) \)

**Proposition 3** Similarly every combinator \( f \) in any class in the definition 7 induces a continuous function in the same way.

For example, in \( PRA \), if \( f = \text{rec}(g_1, g_2, h) \)

\[
\begin{align*}
F(w_1, w_2, u) &= w_1 \text{ if } w_1 \text{ contains neither } S \text{ nor } 0 \\
F(r_1 \oplus 0, w_2, u) &= r_1 \oplus G_1(w_2, u) \text{ if } r_1 \text{ does not contain } S \\
F(r_1 \oplus S \oplus r_2, w_2, u) &= r_1 \oplus w_2 \text{ if } r_1, w_2 \text{ contain neither } S \text{ nor } 0 \\
F(r_1 \oplus S \oplus r_2, s_1 \oplus 0, u) &= r_1 \oplus s_1 \oplus G_2(r_2, u) \text{ if } r_1, s_1 \text{ do not contain } S \\
F(r_1 \oplus S \oplus r_2, s_1 \oplus S \oplus s_2, u) &= r_1 \oplus s_1 \oplus H(r_2, s_2, u, F(r_2, s_2, u)) \text{ if } s_1, r_1 \\
& \quad \text{do not contain } S
\end{align*}
\]

**Remark** In particular, \( f \) induces a continuous function from \( Int^n \) into \( Int \). We have, of course, \( Val(F(w_1, ..., w_n)) = f(Val(w_1), ..., Val(w_n)) \).

**Examples** Define \( \text{add} \) by \( \text{add}(0, n) = n \) and \( \text{add}(Sm, n) = S \text{ add}(m, n) \). The following traces represent the indicated computations:

\[
\begin{align*}
ADD(S^n(x), S^n(y)) &= x_0Sx_1Sx_2...Sx_1...
\\
ADD(S^n(x), S^2(y)) &= x_0Sx_1Sx_2
\\
ADD(S^n(x), S^{n}(y)) &= x_0Sx_1Sy_1Sy_2...Sy_1...
\\
ADD(S^n(x), S^{2}(y)) &= x_0Sx_1Sy_1Sy_20
\end{align*}
\]

**Proposition 4** Let \( f \) be a combinator and \( w_1, ..., w_n \) be in \( TR_f \) (the set of finite sequences of \( TR \)). Then \( F(w_1, ..., w_n) \) is in \( TR_f \).

The main point on traces is that they may be composed (see the proposition 7). But first, we need to consider some useful definitions and propositions. In the following, let \( x \) be a letter and \( w \) be in \( TR \):

**Definition 9** \( x \) is regular in \( w \) if for every \( n \leq n' \), if \( x_n \) appears in \( w \), then \( x_n \) also appears in \( w \) and the first occurrence of \( x_n \) appears before the first occurrence of \( x_{n'} \).

**Definition 10** \( w \) is regular if every letter is regular in \( w \).

**Proposition 5** Let \( f \) be a combinator and \( w_1, ..., w_n \) be regular traces. Then \( F(w_1, ..., w_n) \) also is regular.

**Definition 11** Let \( v, w \) be in \( TR \).

1. A letter \( x \) is compatible with \( v \) in \( w \) if
   - \( x \) is regular in \( w \).
   - if \( x_n \) appears in \( w \), then \( Val(v) \geq S^n \).
   - if \( Val(v) = S^n \) and \( x_n \) appears in \( w \), then \( x_n \) appears only in \( w \) as the last symbol.
2. If \( S^n \leq \text {Val}(v) \), we denote by \( v_n \) the subword of \( v \) between the \( n \)-th occurrence of \( S \) and the next output symbol (ie \( S \) or \( 0 \)) or the end of \( v \) if there is no such symbol. (The 0-th occurrence of \( S \) is the first symbol of \( v \)).

3. Let \( (v^i) \) be a finite sequence of traces and \( (x^i) \) be a finite sequence of letters. Assume that, for every \( i \), \( x^i \) is compatible with \( v^i \) in \( w \). Then \( w[x^i := v^i/i = 1, \ldots, k] \) is the trace obtained by simultaneously replacing in \( w \) every \( (x^i) \) by \( (v^i) \).

Proposition 6 The substitution is a continuous function. More precisely, let \( (w_k) \), \( (v_k) \) be increasing sequences of traces. Let \( w = \sup w_k \) and \( v = \sup v_k \) and assume that, for every \( k \), \( x \) is compatible with \( v_k \) in \( w_k \), then \( x \) is compatible with \( v \) in \( w \) and \( w[x := v] = \sup w_k[x := v_k] \).

Proposition 7 Let \( f \) be a combinator and \( w_1, \ldots, w_n \) be in TR. Let \( v_i \) be the named element (with the fresh name \( x_i \)) such that \( \text {Val}(v_i) = \text {Val}(w_i) \). Then \( x_i \) is compatible with \( w_i \) in \( F(v_1, \ldots, v_n) \) and \( F(w_1, \ldots, w_n) = F(v_1, \ldots, v_n)[v_i := w_i/i = 1, \ldots, n] \).

Definition 12 Let \( v \) be a word and \( k \) an integer. \( v|k \) is the restriction of \( v \) to its \( k \) first symbols. (If the length of \( v \) is less than \( k \), \( v|k \) is \( v \)).

Proposition 8 Let \( f \) be a combinator, \( w_1, \ldots, w_n \) be a sequence of named elements with distinct names and \( p \) is an integer. Assume that:

1. \( w_1 \) has name \( x \) and \( \text {Val}(w_1) \geq S^j \).
2. \( v_1 \) is an element with name \( x \) such that \( \text {Val}(v_1) \geq S^j \).
3. \( x_j \) does not appear in \( F(w_1, \ldots, w_n)|p \).

Then \( F(w_1, w_2, \ldots, w_n)|p = F(v_1, w_2, \ldots, w_n)|p \).

2.2 Intentional behaviour

We give here the precise meaning of the intentional and extensional behaviour of an algorithm and their first properties.

Definition 13 Let \( C \) be a class of combinators as in definition 7 then

\[ \mathcal{E}(C) = \{ \text {ext}(f) / f \in C \} \text { with } \text {ext}(f)(n_1, \ldots, n_k) = m \iff f(S^{n_1}0, \ldots, S^{n_k}0) = S^m0. \]

\[ \mathcal{I}(C) = \{ \text {int}(f) / f \in C \} \text { with } \text {int}(f)(n_1, \ldots, n_k) = m \iff f(S^{n_1}, \ldots, S^{n_k}) = S^m0 \text { or } S^m. \]

Definition 14 Let \( f, g \) be combinators.

- \( f \) and \( g \) are (weakly) intentionally equivalent if and only if \( \text {int}(f) = \text {int}(g) \).
- \( f \) and \( g \) are (strongly) intentionally equivalent if and only if \( F(w_1, \ldots, w_k) = G(w_1, \ldots, w_k) \) for every incomplete named integers \( w_1, \ldots, w_k \).

The following proposition is well known (see [17]).

Proposition 9 Let \( P \) be a class of combinators as in definition 7. \( \text {Ext}(P) = \text {Ext}(\text {PR}) \).

Proposition 10 Let \( P \) be a class of combinators as in definition 7. \( \text {Int}(P) \subset \text {Ext}(\text {PR}) \).
Proof 1 We prove (by induction on the definition of \( f \)) that the following functions \( f_1, f_2 \) are in \( \text{Ext}(PR) \). This is, for simplicity, stated for unary functions using only integers. \( f_1(n) = (0, p) \) if \( f(S^n) = S^p \) and \( (1, p) \) if \( f(S^n) = S^p0 \). \( f_2(n) = (0, p) \) if \( f(S^n0) = S^p \) and \( (1, p) \) if \( f(S^n0) = S^p0 \) where \((n, p)\) is a one-one PR coding of \( N^2 \) onto \( N \). These functions are easily defined by mutual induction.

In the case of \( n \)-ary functions there are about \( 2^n \) functions to consider. In the case of lists we have to consider the lists giving both the value of the elements and the fact that they are complete or not.

Definition 15 Let \( P \) be a class of combinators as in definition 7. \( PN \) is the subset of \( P \) of those combinators for which the recursion scheme has the following restriction: the base cases have to be 0 or \( \text{nil} \) (according to the type of the result). For example for \( f : \text{List}, \text{Int} \rightarrow \text{Int}, f(\text{nil}, x) \) has to be 0.

The following proposition shows that if \( f \) is in \( PN \) (where \( P \) is a class in the definition 7) the complete and the incomplete integers behave the same way.

Proposition 11 Let \( P \) be a class in the definition 7, \( f \) be in \( PN \), \( (k_i)_i \) be a sequence of integers and \( (x^i)_i \) be a sequence of distinct letters. For every \( i \), let \( v_i = S^{k_i}(x^i) \) and \( w_i = S^{k_i}0(x^i) \). Then \( F(v_1, \ldots, v_n) = F(v_1, \ldots, v_n) \oplus 0 \).

Proof 2 By induction on \( f \).

Corollary 1 Let \( P \) be a class of combinators as in definition 7.

\[
\text{Ext}(PN) = \text{Int}(PN) \subset \text{Int}(P)
\]

Proof 3 By the proposition 11.

Proposition 12 Let \( f, g \) be combinators. \( f \) and \( g \) are strongly intentionally equivalent if and only if \( F(v_1, \ldots, v_h) = G(v_1, \ldots, v_h) \) for every infinite named integers \( v_1, \ldots, v_h \).


Computing, in a trace, the number of \( S \) before the least occurrence of an indexed letter allows to show weak equivalence.

Definition 16 Let \( w \) be in \( TR \) and \( x \) be a letter that is regular in \( w \). \( Nb(w, x, n) \) is the number of occurrences of \( S \) in \( w \) before the least occurrence of \( x \) (if \( x \) does not appear in \( w \) \( Nb(w, x, n) \) is undefined).

Proposition 13 Let \( w, v \) be in \( TR \) and \( x, y \) be letters. Assume that \( x \) is compatible with \( v \) in \( w \). Then

\[
Nb(w[x := v], y, n) = \min \{ Nb(w, y, n), Nb(w, x, Nb(v, y, n)) \}
\]

Proposition 14 Let \( f, g \) be combinators. \( f \) and \( g \) are weakly intentionally equivalent if and only if \( Nb(F(w_1, \ldots, w_h), x, n) = Nb(G(w_1, \ldots, w_h), x, n) \) for every \( n, i \) and every infinite named integers \( w_1, \ldots, w_h \) (where \( w_i \) has name \( x_i \)).

Proof 5 Immediate.
3 For \( P \) in \( \{ PR, PRA, PRM \} : Int(P) = Ext(PN) \)

This section is devoted to the proof of the following theorem.

**Theorem 1** For \( P \) in \( \{ PR, PRA, PRM \} : Int(P) = Ext(PN) \).

The result (for \( PR \)) is proved in [?] using the ultime obstination theorem of L.Colson (cf [2]). We give here another proof (with a slightly stronger result: The simulation given is strongly equivalent). This proof does not use the ultime obstination property and its extension to \( PRM \) or \( PRA \) (the latter does not satisfy the ultime obstination theorem) is immediate. We first give the proof for \( PR \).

**Definition 17** \( TRN \) is the set of traces of the form \( F(b_1, ..., b_n) \) where \( f \) is in \( PRN \) and \( b_1, ..., b_n \) are incomplete or infinite named integers.

The crux of the proof is the following proposition.

**Proposition 15** For every \( f \) in \( PR \) and every \( w_1, ..., w_n \) in \( TRN \) such that \( w_i = h_i(b_1, ..., b_p) \) for \( h_i \) in \( PRN \), there is a \( g \) in \( PRN \) such that \( F(w_1, ..., w_n) = G(b_1, ..., b_k) \).

**Remark** The proof is constructive (the one of [?] is not), i.e., we can actually construct \( g \) from a description of \( f \), the \( h_i \) and the \( b_j \). This follows from the fact (see [5] or [9] that we can compute \( f(n_1, ..., n_k) \) (where \( n_1, ..., n_k \) are in \( Int \)) from a description of \( f \).

**Corollary 2**

- For every \( f \) in \( PR \) there is a \( g \) in \( PRN \) such that
  \[
  F(S^\infty(x_1), ..., S^\infty(x_n)) = G(S^\infty(x_1), ..., S^\infty(x_n))
  \]

- \( Int(PR) = Ext(PRN) \).

**Proof 6** Use the proposition with \( w_i = S^\infty(x_i) \).

**Proof of the proposition** By induction on \( f \). The only non trivial case is when \( f \) is defined by recursion, say from \( g \) (the zero case) and \( h \) (the successor case).

1) If \( W_1 \) has no output symbol, \( F(w_1, ..., w_n) = w_1 \) and the result is clear.

2) If \( Val(w_1) = 0 \). Then \( w_1 = r \oplus 0 \) for some finite word \( r \) and \( F(w_1, ..., w_n) = r \oplus G(w_2, ..., w_n) \).

By induction hypothesis, there is a \( g' \) in \( PRN \) such that \( G(w_2, ..., w_n) = G'(b_1, ..., b_p) \). It is thus enough to prove.

**Claim 1** Let \( w \) be a finite word using the letters \( x, y, ... \) and \( g \) be in \( PRN \). Assume \( x, y, ... \) are regular in \( w \) and \( b_1, ..., b_m \) are incomplete or infinite named arguments. There is an \( f \) in \( PRN \) such that, for every large enough \( k \),

\[
F(S^k(x), S^k(y), ..., b_1, ..., b_m) = w \oplus G(b_1, ..., b_m)
\]

**Example** Before giving the general proof, the following example will help understanding. Assume we have to find a function \( f \) in \( PRN \) such that \( F(S^3(x), Y) = x_0, x_0, x_1, x_0, x_1, x_0, x_2, x_1 \oplus G(Y) \) where \( Y \) is any element with name \( y \) distinct from \( x \). The function \( f \) defined below satisfies the desired requirement. In the following definitions, we donot write the base case since the function has to be in \( PRN \) and so the base case always is zero. The names of the variables are not chosen
at random and they should help to understand. The possible confusion with the symbols used in the word w is ... intentional.

\[
\begin{align*}
  f(x_0, y) &= f_0(x_0, x_0, y) \\
  f_0(Sx_1, x_0, y) &= f_1(x_0, x_0, x_1, y) \\
  f_1(Sx_1, x_0, x_1, y) &= f_2(x_1, x_0, x_1, y) \\
  f_2(Sx_2, x_0, x_1, y) &= f_3(x_0, x_0, x_1, x_2, y) \\
  f_3(Sx_2, x_0, x_1, x_2, y) &= f_4(x_1, x_0, x_1, x_2, y) \\
  f_4(Sx_3, x_0, x_1, x_2, y) &= f_5(x_0, x_0, x_1, x_2, y) \\
  f_5(Sx_3, x_0, x_1, x_2, y) &= f_6(x_2, x_0, x_1, x_2, y) \\
  f_6(Sx_3, x_0, x_1, x_2, y) &= f_7(x_1, x_0, x_1, x_2, x_3, y) \\
  f_7(Sx_2, x_0, x_1, x_2, x_3, y) &= g(y)
\end{align*}
\]

**proof of the claim** I may clearly assume without loss of generality that the names of \(b_1, \ldots, b_m\) are distinct from \(x, y, \ldots\). Assume for the simplicity of notations that \(w\) only has two letters \(x\) and \(y\) and \(g\) only has one argument named \(z\). Assume the word \(w\) has length \(n\). I define \(f\) and for \(0 \leq k \leq n - 1\), by induction on \(k\), an auxiliary function \(f_k\) as follows. By the regularity of \(x\) and \(y\) in \(w\), the first symbol in \(w\) is either \(x_0\) or \(y_0\). Say it is \(x_0\). Let \(f(x_0, y_0, z) = f_0(x_0, x_0, y_0, z)\) where \(f_0\) is defined by recursion by \(f_0(Sx_1, x_0, y_0, z) = f_1(a, x_0, x_1, y_0, z)\) where \(a\) is the second symbol in \(w\). (Note that, by the regularity of \(w\), this symbol is either \(x_0\), \(x_1\) or \(y_0\) and thus may be "used" in \(f_1\), not as a symbol but as an argument.) Assume \(f_0, \ldots, f_k\) have been defined. Let \(i\) (resp \(j\)) be the maximum index of \(x\) (resp \(y\)) appearing in \(w[k]\).

- Assume \(k \leq n - 3\) and let \(a\) be the \((k + 2)\)-th symbol in \(w\). Assume the \((k + 1)\)-th symbol in \(w\) is (say) \(x_p\).
  - If \(p \leq i\), \(f_{k+1}\) is defined by recursion by
    \[f_{k+1}(Su, x_0, \ldots, x_{i+1}, y_0, \ldots, y_{j+1}, z) = f_{k+2}(a, x_0, \ldots, x_{i+1}, y_0, \ldots, y_{j+1}, z)\]
    (Note again that, by the regularity of \(w\), \(a\) is either \(x_0, \ldots, x_{i+1}\) or \(y_0, \ldots, y_{j+1}\) and thus may be used in \(f_{k+2}\).)
  - Otherwise, by the regularity of \(w\), \(p = i + 1\). \(f_{k+1}\) is defined by recursion by
    \[f_{k+1}(Sx_1, x_0, \ldots, x_{i+1}, y_0, \ldots, y_{j+1}, z) = f_{k+2}(a, x_0, \ldots, x_{i+2}, y_0, \ldots, y_{j+1}, z)\]
  - Define \(f_{n-1}\) by recursion by \(f_{n-1}(Su, x_0, \ldots, x_i, y_0, \ldots, y_j, z) = g(z)\).
  - It is immediate to check that \(f\) satisfies the desired property.

3) If \(m \geq 1\) : the proof is similar to the previous case. It is necessary to make a shift of the indexes to be able to use the induction hypothesis. (Use the same trick as in proposition 1-4 in [10].)

**[case 2]** \(\text{Val}(w_1) = S^\infty\). We may assume without loss of generality that \(w_1 = S^\infty(x)\) for some fresh letter \(x\). Indeed, if

\[
F(S^\infty(x), w_2, \ldots, w_n) = F'(S^\infty(x), b_1, \ldots, b_p)
\]

then by the proposition 7, \(F(w_1, w_2, \ldots, w_n) = F'(w_1, b_1, \ldots, b_p)\) and, since \(w_1 = H(b_1, \ldots, b_p)\) for some \(h \in PRN\), the result follows.

- If \(\text{Val}(F(w_1, w_2, \ldots, w_n)) = S^k0\), then, by the continuity of \(F\),

\[
F(w_1, w_2, \ldots, w_n) = F(S^p(x), w_2, \ldots, w_n)
\]

and the result follows from the previous case.

10
• Otherwise, let \( y \) be a fresh letter and \( Y \) be the (infinite or incomplete) element with name \( y \) whose value is \( Val(F(w_1, w_2, ..., w_n)) \). Let \( X = S^\infty(x) \). By the induction hypothesis, there is an \( h' \) in \( PRN \) such that \( H(X, Y, w_2, ..., w_n) = H'(X, Y, b_1, ..., b_p) \). Let \( f' \) (in \( PRN \)) be defined by recursion from \( h' \).

The following claim finishes the proof.

**Claim 2** \( F(X, w_2, ..., w_n) = F'(X, b_1, ..., b_p) \).

**Sketch of proof** For more details see [10] Proposition II-4. Let \( v_i \) be the sequence defined by \( v_0 = x_0 \oplus H(X_1, Y, w_2, ..., w_n) \) and \( v_{i+1} = v_i[y := v_0[x + i + 1]] \). (where \( X_1 = x_1Sx_2S\ldots \) and \( v[x + 1] \) is the word \( v \) where every \( x_k \) has been replaced by \( x_{k+1} \)). Similarly let \( v'_i \) be the sequence defined by \( v'_0 = x_0 \oplus H'(X_1, Y, b_2, ..., b_p) \) and \( v'_{i+1} = v'_i[y := v'_0[x + i + 1]] \). It is easy to check, by induction on \( i \), that for every \( i \), \( v_i = v'_i \) and the result follows from the fact that \( F(X, w_2, ..., w_n) = Lim(v_i) \) and \( F'(X, b_1, ..., b_p) = Lim(v'_i) \). \( \square \)

**Proof of the proposition for PRA and PRM**

The proof is exactly the same with the following modification: Assume \( f \) is defined by recursion, the recursive argument \( w_1 \) is \( S^\infty(x) \) and \( w = F(w_1, w_2, ..., w_n) \). In \( PR \), the conclusion came from the fact that \( w \) satisfies the equation \( w = v[y := w[x + 1]] \). In \( PRA \), \( w \) satisfies an equation as: \( w = v[z := w[x + 1, y + 1]] \). In \( PRM \), \( w_1 \) and \( w_2 \) satisfy equations as: \( w_1 = v_1[z_1 := w_1[x + 1], z_2 := w_2[x + 1]] \) and \( w_2 = v_2[z_1 := w_1[x + 1], z_2 := w_2[x + 1]] \). In both cases the conclusion follows in the same way. \( \square \)

An other characterisation of these classes are given in [5] for \( PR \) and in [10] for \( PR, PRM \) and \( PRA \). For example the following results are proved in [10].

**Theorem 2** \( PR \) the functions that map \( n \) onto \( Nb(F(S^\infty(x_1), ..., S^\infty(x_k)), x^i, n) \) are in the least class \( C \) of increasing functions from \( N \) to \( N \) that contains 0, \( S \), the identity, the predecessor and is closed by:

- composition
- finite change: if \( f \in C \) and for all \( n \), except finitely many, \( f(n) = g(n) \) then \( g \in C \).
- iteration: if \( f \in C \) and \( f(n) > n \) for all \( n \), then \( g \in C \) where \( g \) satisfies: \( g(n + 1) = f(g(n)) \) for \( n \) large enough.

**PRM** by replacing the closure by iteration by the closure by multistep iteration: if \( f \in C \) and \( f(n) > n \) for all \( n \), then \( g \in C \) where \( g \) for (for some \( p \)) \( g \) satisfies: \( g(n + p) = f(g(n)) \) for \( n \) large enough.

**PRA** by adding the closure by minimum : if \( f, g \in C \) then \( h \in C \), where \( h \) is defined by \( h(n) = \min \{ f(n), g(n) \} \) and replacing the closure by iteration by the closure by mixed iteration: if \( f, g \in C \), \( f(n) > n \) for all \( n \) and for \( n \) large enough, \( h(n + 1) = \min \{ g(n + 1), f(h(n)) \} \), then \( h \in C \).

4 **\( Int(PRNS) \) is ultimately obstinated**

This section is devoted to the proof of the theorem 3 below which, in particular, implies that \( Int(PSR) \) is not equal to \( Ext(PRNS) \).

We recall here the main definitions and properties involved in this result. For more details see [10].
Definition 18 Let \( w \) be in \( TR \).

- A letter \( x \) is unbounded (resp bounded) in \( w \) if \( \{ j/x_j \text{ appears in } w \} \) is infinite (resp finite).
- \( w \) is ultimately obstinated if \( w \) has at most one unbounded letter.

Proposition 16 Let \( w \) and \( v \) be in \( TR \) and \( x \) be a letter. Assume that:

- \( x \) is compatible with \( v \) in \( w \).
- \( x \) is bounded in \( w \) and \( w \) does not finish with \( x \).
- \( w \) is ultimately obstinated

then \( w[x := v] \) also is ultimately obstinated.

Proposition 17 Let \( w, v \) be in \( TR \). Let \( x \) be a letter compatible with \( v \) in \( w \). Assume that \( w \) and \( v \) are ultimately obstinated. Then \( w[x := v] \) also is ultimately obstinated.

Theorem 3 Let \( f \) be in \( PRSN \) and \( w_1, \ldots, w_n \) be in \( TR \). Assume that every \( w_i \) is ultimately obstinated, then so is \( F(w_1, \ldots, w_n) \).

Corollary 3 \( \text{Int}(PRS) \neq \text{Ext}(PRSN) \).

proof of the corollary

- It is shown in [2] that the \( \inf \) function is in \( \text{Int}(PRL_0) \) Thus, either by a direct proof or by using the theorem 9 below (\( \text{Int}(PRL_0) \subset \text{Int}(PRS) \)) the \( \inf \) function is in \( \text{Int}(PRS) \).
- \( \inf(S^\infty(x), S^\infty(y)) \) cannot be ultimately obstinated. For more details, see [10] theorem II-6.

Thus \( \text{Int}(PRS) \) donot satisfy the ultime obstination theorem. □

proof of the theorem By the propositions 7 and 16 we may assume that the \( w_i \) are named elements. The proof is by induction on \( f \). The only non trivial case is when \( f \) is defined by recursion. If \( w_1 \) (the recursive argument) is finite the proof is immediate (by induction on the value of \( w_1 \)). Assume then that \( w_1 = S^\infty(x) \). Also assume, for the simplicity of notations, that \( f \) only has 3 arguments and the recursive equation for \( f \) is \( f(Sn, m, p) = h(n, m, p, f(n, Sm, p)) \).

Let \( X_i = x, Sx, x \ldots \). If \( w \) is in \( TR \) and \( x \) is a letter, \( w[x + 1] \) is the trace \( w \) where every \( x_k \) has been replaced by \( x_{k+1} \) and \( w[x - 1] \) is the trace \( w \) where \( x_0 \) is deleted and for \( k \geq 1, x_k \) is replaced by \( x_{k-1} \). For \( i = 0, 1 \) let \( w_i = F(X_i, SY, T) \).

1. Assume first that \( Y = S^\infty(y) \). Then \( w_1 = w_0[x + 1, y - 1] \). Let \( v = x_0 \oplus H(X_1, Y, T, Z) \) where \( T \) is the element with name \( t \) such that \( Val(T) = Val(w_1) \). \( w_0 = v[t := w_0[x + 1, y - 1]] \) and thus the same proof (see [10] theorem II-1) as in case of \( PR \) works.

2. Assume \( Y = S^k(y) \). (By the proposition 11, the case \( Y = S^k_0(y) \) is similar). If \( y_k \) does not appear in \( w_0 \), by the proposition 8, \( w_0 \) and \( w_1 \) coincide up to reindexing and thus again \( w_1 = w_0[x + 1, y - 1] \). If \( y_k \) appears in \( w_0 \), then \( y_k \) is the last symbol in \( w_0 \) which is then finite and thus ultimately obstinated. □

The following result shows that the previous theorem is not valid for \( PRV \) or \( PRA \).

Proposition 18 \( PRV, PRAN \) donot satisfy the ultime obstination theorem.
Proof 7  In PRVN: Define the functions pred, test, and inf by:

\[
\begin{align*}
\text{pred}(0) &= 0 ; \quad \text{pred}(Sx) = x \\
\text{test}(0, y) &= 0 ; \quad \text{test}(Sx, y) = y \\
\text{inf}(0, y) &= 0 ; \quad \text{inf}(Sx, y) = \text{test}(y, \text{inf}(x, \text{pred}(y)))
\end{align*}
\]

It is easy to check that

\[
\text{INF}(S^\infty(x), S^\infty(y)) = x_0 y_0 S x_1 y_0 y_1 S x_2 y_0 y_1 y_2 S...
\]

In PRAN: Define the function inf by:

\[
\begin{align*}
\text{inf}(0, y) &= 0 \\
\text{inf}(Sx, Sy) &= S \text{inf}(x, y)
\end{align*}
\]

It is easy to check that

\[
\text{INF}(S^\infty(x), S^\infty(y)) = x_0 y_0 S x_1 y_1 S x_2 y_2 S...
\]

Note that the \text{inf} of PRVN and the \text{inf} of PRAN are weakly intentionally equivalent but not strongly.

Open question For \( P = \text{PRV} \) or \( \text{PRL} \) we do not know whether \( \text{Int}(P) = \text{Ext}(PN) \) or not.

5 PRV has the backtracking property

\( \text{PRS, PRV, PRL} \) do not have the ultime obstination property. In [10] another property, called the backtracking property, that generalizes the ultime obstination property, is defined and proved for \( \text{PRL} \). It is clear that \( \text{PRA} \) does not satisfy this property. We show here that this property holds for \( \text{PRV} \).

Definition 19 Let \( w \) be in TR.

1. \( x \) is backtracking (abbreviated as \( \text{BT} \)) in \( w \) if for every \( n \) large enough, \( x_n \) appears infinitely many times in \( w \).

2. \( x \) is a \( \text{BT} \)-counterexample for \( w \) if \( x \) is unbounded but not \( \text{BT} \) in \( w \).

3. \( w \) has the backtracking property (abbreviated as \( \text{BTP} \)) if there is at most one \( \text{BT} \)-counterexample for \( w \).

Examples

- \( x_0 y_0 x_1 y_1 x_2 y_2 \ldots \) has not the \( \text{BTP} \) because \( x \) and \( y \) are \( \text{BT} \)-counterexamples.

- \( x_0 y_0 x_1 y_0 y_1 x_2 y_0 y_1 y_2 x_3 \ldots \) has the \( \text{BTP} \) because \( x \) is the only \( \text{BT} \)-counterexample.

The following propositions are the simplified versions (due to the fact that the only data type is \( \text{Int} \)) of the propositions III-2 and III-4 in [10].

Proposition 19 Assume that :

1. \( v, w \) are in TR, \( x \) is a letter compatible with \( v \) in \( w \) and \( y \) is a letter.

2. Either \( x \) is unbounded in \( w \) and \( y \) is \( \text{BT} \) in \( v \) or \( x \) is \( \text{BT} \) in \( w \) and \( y \) is unbounded in \( v \).
Then $y$ is $BT$ in $w[x := v]$.

**Proposition 20** Assume that :

1. $v, w$ are in TR, $x$ is a letter compatible with $v$ in $w$.
2. $v$ and $w$ have the BTP.

Then $w[x := v]$ has the BTP.

**Theorem 4** Let $f$ be a combinator in PRV. If $w_1, ..., w_n$ have the BTP then $F(w_1, ..., w_n)$ also has the BTP.

**Proof 8** By induction on $f$. By the propositions 7 and 20 we may assume that the $w_i$ are named elements. The only non trivial case is when $f$ is defined by recursion and the recursive argument is $S^\infty(x)$. Assume for the simplicity of notations that the recursion scheme is

$$f(Sn, m, p) = h(n, m, p, f(n, a(n, m, p), b(n, m, p))$$

It will be clear that this case is generic. We have to show that $u = F(S^\infty(x), Y, Z)$ has the BTP.

Let $X_i = x_i S x_{i+1} S ...$ and let $(t^n)_n$ be a sequence of fresh variables. Define the sequences $(a_n)_n, (b_n)_n, (u_n)_n$ of traces as follows :

$a_0 = Y, b_0 = Z, a_{n+1} = A(X_n, a_n, b_n)$ and $b_{n+1} = B(X_n, a_n, b_n)$.

$w_n = x_n \oplus H(X_{n+1}, a_n, b_n, T^n)$ where $T^n$ is the element with name $t^n$ whose value is $f(S^\infty, \text{Val}(a_{n+1}), \text{Val}(b_{n+1}))$.

$u_0 = w_0$ and $u_{n+1} = u_n[t^n := w_{n+1}]$. By the induction hypothesis and the proposition 20 every $w_n$ and $u_n$ have the BTP. It is clear (for more details see the corresponding proofs in [10]) that $u = \text{Lim}(u_n)$.

**Claim 3** Assume $y$ is bounded in every $u_n$ and $y$ is unbounded in $u$. Then $y$ is $BT$ in $u$.

(This claim is the simplified version of the claim 7 in the proof of the theorem III-1 in [10].)

**Proof 9** Let $k$ be an integer. We show that $y_k$ appears infinitely many times in $u$. Let $p$ be an integer and $n$ be such that $u[p = u_n[p$. Let $k_0 > k$ be such that if $y_{k'}$ appears in $u_n$, then $k' \leq k_0$.

Since $y$ is unbounded in $u$, there is a $k' > k_0$ such that $y_{k'}$ appears in $u$. $y_{k'}$ appears in $u_{m+1}$ and comes from the substitution of some $(t^m)_j$ by $(w_{m+1})_j$ in $u_n$. By the definition of $k_0$, $m$ is larger than $n$. By the regularity of $t^m$ in $u_{m+1}$ and the regularity of $y$ in $w_{m+1}$, $y_k$ also has an occurrence in $u_{m+1}$ coming from $w_{m+1}$ by a substitution. This occurrence of $y_k$ cannot be in $u[p$.

End of the proof of the theorem Assume that $u$ does not have the BTP and there are two unbounded letters that are not BT.

- Assume first that the unbounded letters are $y$ and $z$. By the previous claim they must be unbounded in some $u_n$ and, since $u_n$ has the BTP, at least one is $BT$ in $u_n$ and thus in $u$. This leads to a contradiction.

- Assume now that the unbounded letters are $x$ and $y$. Again by the claim, $y$ must be unbounded in some $u_n$. If $x$ is unbounded in $u_n$, either $x$ or $y$ is $BT$ in $u_n$ and this leads to a contradiction. Otherwise, since $u = u_n[t^n := F(X_{n+1}, a_n, b_n)]$, then $t^n$ (resp. $x$) is unbounded in $u_n$ (resp. in $F(X_{n+1}, a_n, b_n)$) otherwise, $x$ would be bounded in $u$). Then, since $u_n$ has the BTP, $t^n$ is $BT$ in $u_n$ and thus (by the proposition 19) $x$ is $BT$ in $u$. This leads to a contradiction.
6 \textbf{Int(\textit{PRM}), Int(\textit{PRA}) \subset Int(\textit{PRL})}

\textit{PRL}_0 is the subset of those combinators in \textit{PRL} that use only the so called \textit{L}_0 lists, i.e. lists where “elements” are complete integers with no intentional behaviour. More precisely,

\textbf{Definition 20} \quad • A trace of type \textit{List} is an \textit{L}_0 list if it can be written as \( w = w_0 \oplus \text{cons}(n_0, w_1 \oplus \text{cons}(n_1, \ldots)) \) where the \( n_i \) are complete integers (In particular in the trace \( n_i \) there are no other symbols than \( S \) and \( 0 \)).

• \( w \in \text{TR}_0 \) if \( w \in \text{TR} \) and \( w \) has type \text{Int} or \( w \) is an \textit{L}_0 list.

\textbf{Definition 21} \quad • \( f \in \text{PRL}_0 \) iff (by induction on the definition of \( f \))

• \( f \) is 0, \( S \) or nil.

• or if \( f = g(h_1, \ldots, h_k) \) and \( g \neq \text{cons} \) and \( g, h_1, \ldots, h_k \in \text{PRL}_0 \).

• or if \( f = \text{cons}(h_1, h_2) \) and \( h_1, h_2 \in \text{PRL}_0 \) and, when applied to arguments in \( \text{TR}_0 \), \( h_1 \) gives a complete integer.

• or if \( f = \text{rec}(g, h) \) and \( g \in \text{PRL}_0 \) and

  – Either the type (of the result) of \( f \) is \text{Int} and \( h \in \text{PRL}_0 \).

  – Or the type (of the result) of \( f \) is \text{List} and \( h \neq \text{cons}(h_1, h_2) \) and \( h \in \text{PRL}_0 \).

  – Or \( h = \text{cons}(h_1, h_2) \) and \( h_1, h_2 \in \text{PRL}_0 \) and, when applied to arguments in \( \text{TR}_0 \), \( h_1 \) gives a complete integer and \( h_2 \) gives an \textit{L}_0 list.

\text{Int(PRA)} \subset \text{Int(\textit{PRL})} is already proved in [19]. We show here that both inclusions are consequences of a more general result.

\textbf{Theorem 5} \quad Let \( f_1, \ldots, f_k \) be increasing primitive recursive functions. There is a combinator \( f \) in \text{PRL}_0 such that for every \( i, n \):

\[ \text{Nb}(F(S^\infty(x^1), \ldots, S^\infty(x^k)), x^i, n) = f_i(n) \]

\textbf{Proof 10} \quad For the simplicity of notations assume that \( k = 2 \). Assume that \( f_1 \) and \( f_2 \) are increasing and primitive recursive. We are looking for \( h \) in \text{PRL}_0 such that \( w = H(S^\infty(x), S^\infty(y)) \) can be written as \( a_0 S a_1 S a_2 S \ldots \) where every \( a_j \) is a finite word on the vocabulary \( \{x_n/n \in N\} \cup \{y_n/n \in N\} \) and \( w \) is correct with \( (f_1, x) \) and \( (f_2, y) \) where \( w \) is correct with \( (f, x) \) means : for every \( n \), \( \text{Max}\{j/f(j) \leq n\} = \text{Max}\{j/x_j \text{ appears in } a_0 \oplus a_1 \oplus \ldots \oplus a_n\} \). We assume that \( f_1 \) and \( f_2 \) are unbounded, since it is easy to make \( h \) correct with \( (f, x) \) when \( f \) is bounded.

• Define \text{incr} and \( L \) by:

\[ \text{incr(nil)} = \text{nil} \quad ; \quad \text{incr(cons}(a, l)) = \text{cons}(S a, \text{incr}(l)) \]

\[ L(0) = \text{cons}(0, \text{nil}) \quad ; \quad L(n + 1) = \text{cons}(0, \text{incr}(L(n))) \]

Then \( L(S^\infty) \) is the infinite list \([0, 1, 2, \ldots]\) and the intentional behaviour is the same as \( S^\infty(x) \), i.e \( L(S^\infty(x)) = x_0 \text{cons}(0, x_1 \text{cons}(S 0, x_2 \text{cons}(S S 0, \ldots)) \ldots) \)

• Define \text{pred}, \text{dif} and \text{test} by:

\[ \text{pred}(0) = 0 \quad ; \quad \text{pred}(S x) = x \]

\[ \text{dif}(x, 0) = x \quad ; \quad \text{dif}(x, S y) = \text{pred}(\text{dif}(x, y)) \]

\[ \text{test}(0, a, b) = a \quad ; \quad \text{test}(S x, a, b) = b \]
• Define \( h_i \) and \( r_i \) by:

\[
\begin{align*}
    h_i(\text{nil}) & = 0 \\
    h_i(\text{cons}(m, ll)) & = \text{let } p = \text{dif}(f_i(m + 1), f_i(m)) \\
                        & \quad \text{in } \text{test}(p, h_i(ll), S^0 h_i(ll))
\end{align*}
\]

Let \( p_i = f_i(0) \) and \( r_i(x) = S^p h_i(L(x)) \). It is easy to check that \( r_i \) is correct with \( (f_i, x) \).

• Define \( g \) and \( g' \) by:

\[
\begin{align*}
    g(\text{nil}, p) & = 0 \quad ; \quad g(\text{cons}(m, ll), p) = \text{test}(\text{dif}(m, p), 0, S g(ll, p)) \\
    g'(n, m) & = g(L(n), m)
\end{align*}
\]

It is easy to check that

\[
G'(S^\infty(x), S^\infty(y)) = x_0 y_0 S x_0 y_1 S x_2 y_0 y_2 S ...
\]

• It follows immediately that \( h \) defined by: \( h(n, m) = g'(r_1(n), r_2(m)) \) satisfies the desired properties.

**Theorem 6** \( \text{Int(PRM)}, \text{Int(PRA)} \subset \text{Int(PRL)} \).

**Proof 11** Let \( f \) be a combinator in PRA or PRM. By the theorem 5 and the proposition 12 it is enough to show that the functions that map \( n \) onto \( \text{Nb}(S^\infty(x^1), ..., S^\infty(x^k)), x^i, n) \) are increasing (this is trivial) and primitive recursive.

Assume (for the simplicity of notations) that \( k = 2 \). The result is proved by induction on the definition of \( f \). The only non trivial case is when \( f \) is defined by recursion.

**PRA** If \( f \) is given by \( f(Sx, Sy) = h(x, y, f(x, y)) \). Let \( w = F(S^\infty(x), S^\infty(y)) \) and \( v = x_0 y_0 \oplus H(X_1, Y_1, Z) \) where \( X_1 = x_1 S x_2 S ... \), \( Y_1 = y_1 S y_2 S ... \) and \( Z \) is the element with name \( z \) such that \( \text{Val}(Z) = \text{Val}(w) \). It is clear that \( w = x_0 y_0 \oplus v[z := w[x + 1, y + 1]] \). It follows (from the proposition 13) that for \( n \geq 1, \text{Nb}(w, x, n + 1) = \text{Min}\{\text{Nb}(v, x, n + 1), \text{Nb}(v, z, \text{Nb}(w, x, n))\} \) and similarly for \( y \). The result follows easily.

**PRM** Assume, for the simplicity of notations, that there are only two functions defined by mutual recursion: \( f_i(Sx, y) = h_i(x, y, f_i(x, y), f_2(x, y)) \). Let \( w_i = F_i(S^\infty(x), Y) \) and \( v_i = x_0 \oplus H_i(X_1, Y, Z, T) \) where \( X_1 = x_1 S x_2 S ... \) and \( Z \) (resp \( T \)) are named elements with value \( \text{Val}(v_i) \) (resp \( \text{Val}(w_i) \)). \( w_i = v_i[z := v_i[x + 1, t := w_2[x + 1]] \). It follows that \( \text{Nb}(w, x, n + 1) = \text{Min}\{\text{Nb}(v, x, n + 1), \text{Nb}(v, z, \text{Nb}(w, x, n)), \text{Nb}(v, t, \text{Nb}(w_2, x, n))\} \)

Since we know that extensionally PRM is included in PR, the result follows.

**Remark.** We donot know if the following stronger result is true: For \( f \) in \{PRA, PRM\} there is \( g \) in PRL such that

\[
F(S^\infty(x^1), ..., S^\infty(x^k)) = G(S^\infty(x^1), ..., S^\infty(x^k))
\]

**Proposition 21** \( \text{Int(PRA)} \neq \text{Int(PRL)} \) and \( \text{Int(PRM)} \neq \text{Int(PRL)} \).

**Proof 12**

1. It is shown in \[10\] that the square function cannot be (intentionaly) represented in PRA and, by the theorem 6 it can be represented in PRL.

2. It is shown in \[10\] that for every \( f \) in PRM, \( F(S^\infty(x^1), ..., S^\infty(x^k)) \) is ultimately obstinated and this is not true for PRL.
Open question We have not been able to prove that for \( f \) in \( PRL \) (and in fact, even not for \( PRL_0 \)) the function \( n \rightarrow Nb(F(S^n(x), S^n(y), x, n) \) is primitive recursive.

**Definition 22** Let \( f \) be a function from \( N^k \) into \( N \)
- \( f \) is increasing if for every \( n_1, \ldots, n_k, m_1, \ldots, m_k \) such that \( n_i \leq m_i \) for every \( i \), \( f(n_1, \ldots, n_k) \leq f(m_1, \ldots, m_k) \)
- \( f \) is intentional if \( f \) is increasing and there are primitive recursive functions \( f_1, \ldots, f_k \) such that \( f(n_1, \ldots, n_k) = f_i(n_i) \) for every \( i \), \( n_i \) and every large enough \( n_1, \ldots, n_i-1, n_i+1, \ldots, n_k \).

An increasing function is intentional iff (in the terminology of [19], [20]) it is "en forme de plateau" (in the shape of plate) and the "fonctions de blocs" (block’s functions) are primitive recursive (this terminology comes from a graphical representation of intentional behaviour functions). The following theorem is proved in [19] (theorem 6.2.1 of the section 4) with a small restriction (some "fonction de blocs" had to be strictly increasing). In fact, this restriction is not necessary.

**Theorem 7** Let \( \theta \) and \( \rho \) be primitive recursive functions from \( N^n \) into \( N \) such that \( \theta \geq \rho \) and \( \rho \) is intentional. Then there is a combinator \( f \) in \( PRL_0 \) such that for every \( p_1, \ldots, p_n \):

\[
\begin{align*}
f(S^{p_1}(0), \ldots, S^{p_n}(0)) &= S^{\theta(p_1, \ldots, p_n)}(0) \\
f(S^{p_1}, \ldots, S^{p_n}) &= S^{\rho(p_1, \ldots, p_n)}
\end{align*}
\]

That is, we may compute in \( PRL_0 \) the function \( \theta \) with the behaviour \( \rho \) (this result is an extension of a result of Conson [4] with unary function and system \( T \) of Gödel).

**Proof 13** The existence of \( f \) satisfying the second condition is exactly the theorem 5. It is immediate to get the full result by using the same trick as in [19], [20].

**7** \( \text{Int}(PRS) = \text{Int}(PRV) = \text{Int}(PRL_0) \)

As already mentioned we restrict, in this section, \( PRV \) to the possible change for at most one parameter. We do not know if the theorem 8 below remains true without this restriction. \( \text{Int}(PRS) \subseteq \text{Int}(PRV) \) and \( \text{Int}(PRL_0) \subseteq \text{Int}(PRL) \) are trivial. We thus have to show : \( \text{Int}(PRL_0) \subseteq \text{Int}(PRS) \) and \( \text{Int}(PRV) \subseteq \text{Int}(PRL_0) \). The second point is proved in [19]. We recall here the proof for self completeness.

**Theorem 8** Let \( f \) be a combinator in \( PRV \). There is a combinator \( fl \) in \( PRL_0 \) such that \( f \) and \( fl \) are (weakly) intentionally equivalent.

**Proof 14** Without loss of generality we prove the proposition for a binary \( f \) defined by :

\[
\begin{align*}
f(0, y) &= g(y) \\
f(Sx, y) &= h(x, f(x, j(x, y)), y)
\end{align*}
\]

Let \( jl \) be the \( PRL_0 \) combinator (weakly) equivalent to \( j \). Define \( \text{iter} \) by :

\[
\begin{align*}
\text{iter}(0, x, y) &= y \\
\text{iter}(Sa, x, y) &= jl(x - Sa, \text{iter}(a, x, y))
\end{align*}
\]

**Claim 4** Let \( n \) be a complete integer. Then,

\[
\text{iter}(Sn, Sx, y) = \text{iter}(n, x, jl(x, y))
\]
**Proof 15** By induction on $n$.

Let $gl, hl$ be respectively the $PRL_0$ combinators (weakly) equivalent to $g$ and $h$. Define $\text{inrcr}$, $L, f_0$ and $fl$ by:

- $\text{inrcr}(\text{nil}) = \text{nil}$
- $\text{inrcr}(\text{cons}(a, l)) = \text{cons}(\text{Suc}(a), \text{inrcr}(l))$.
- $L(0) = \text{cons}(0, \text{nil})$
- $L(Sn) = \text{cons}(0, \text{inrcr}(L(n)))$
- $fl_0(\text{nil}, x, y) = x$
- $fl_0(\text{cons}(a, l), x, y) = hl(x - Sna, fl_0(l, x, y, z), \text{iter}(Sa, x, y, z))$
- $fl(x, y) = fl_0(L(x), x, y)$

**Claim 5** Let $l$ be an $L_0$ list. Then,

$$fl_0(\text{inrcr}(l), Sx, y) = fl_0(l, x, j(x, y))$$

**Proof 16** By induction on $l$ (using claim 4).

The following claim finishes the proof of the theorem.

**Claim 6** Let $x, y$ be elements in $\text{Int}$. Then, $f(x, y) = fl(x, y)$

**Proof** By induction on the value of $x$. This easily follows from the following computation. Note that in this part of the computation the simulation is not strong since $x_k$ is replaced by $x_0x_1...x_k$

$$fl(Sx, y) = fl_0(L(Sx), Sx, y)$$

$$= fl_0(\text{cons}(0, \text{inrcr}(L(x))), Sx, y)$$

$$= hl(x, fl_0(\text{inrcr}(L(x)), Sx, y), \text{iter}(0, x, y))$$

$$= hl(x, fl_0(L(x), x, j(x, y)), y)$$

By the claim 5

$$= hl(x, f(x, j(x, y), y)$$

By induction hypothesis

$$= f(Sx, j(x, y))$$

**Remark** The previous proof does not work for the general case in PRV, i.e. when we allow any change of parameters. Let $f$ be the following term:

$$f(0, y, z) = g(y, z)$$

$$f(Sx, y, z) = hl(x, f(x, j_1(x, y, z), j_2(x, y, z)), y, z)$$

During the computation of $f(Sx, y, z)$, we need to compute first $y$ and $z$, then $j_1(Sx, y, z)$ and $j_2(Sx, y, z)$, and finally $j_1(x, j_1(Sx, y, z), j_2(Sx, y, z))$ and $j_2(x, j_1(Sx, y, z), j_2(Sx, y, z))$. Thus, $fl$ must compute these terms. The terms $\text{iter}_1$ and $\text{iter}_2$ should be defined by mutual recursion. We know that $\text{Int}(\text{PRM})$ is a subset of $\text{Int}(\text{PRL}_0)$ but we need more since $j_1$ and $j_2$ are in $\text{PRL}_0$. We thus have to know that $\text{Int}(\text{PRM} + L_0)$ (ie $PR$ where mutual recursion and $L_0$ lists are allowed) is included in $\text{Int}(\text{PRL}_0)$ and we do not know how to prove that.

**Theorem 9** Let $f$ be a combinator in $PRL_0$, there is a combinator $fs$ in $PRS$ such that $f$ and $fs$ are (weakly) intentionally equivalent.

**Proof 17** In this proof we donot use the notion of trace and we only use the finite (complete or incomplete) elements of Int and List. The finite $L_0$ lists are coded in the following way.

**Definition 23** Let $w = w_0 \oplus \text{cons}(w_0, w_1 \oplus \text{cons}(w_1, ...))$ be a $L_0$ list. $w$ is coded by the pair $[v, h]$ where $v$ is the trace (of type Int) defined by $v = w_0Sw_1S...$ and $h$ is a complete integer coding the list $[n_0, n_1, ...]$
Remark The length of $w$ may be incomplete and thus $v$ may be an incomplete integer. The fact that, in this case, $h$ also is an incomplete list does not cause any trouble. The coding takes either a complete or incomplete list and always gives a complete integer $h$.

Let $\phi, \psi : N, N \to N$ be p.r. functions such that:

- $\phi(n, h)$ returns the $n$-th element of the list encoded by $h$.
- $\psi(n, h)$ returns the encoded list representing $h$ without its $n$ first elements.

Let $\lg : N \to N$ be the function such that $\lg(h)$ is the length of the list encoded by $h$. Denote by $\perp$ the incomplete list of length 0.

Note again that, since when applied to complete integers, $\phi$ and $\psi$ always give complete integers, the precise way the coding is done has no importance.

The theorem follows immediately from the next proposition (stated for $f$ with only two arguments for the simplicity of notations).

**Proposition 22**

1. Let $f : \text{List}, \text{Int} \to \text{Int}$ be a combinator in $\text{PRL}_0$. There is a combinator $f_1$ in $\text{PRS}$ such that $\langle f \rangle$ is defined by induction from $g$.

2. Let $f : \text{List}, \text{Int} \to \text{List}$ be a combinator in $\text{PRL}_0$. There are combinators $f_{s1}$ in $\text{PRS}$ and $f_{s2}$ in $\text{PR}$ such that $\langle f \rangle$ is defined by induction from $g$.

**Proof** By induction on $f$. We only look the case when $f$ is defined by induction from $g, h$, the type of $\langle f \rangle$ is $\text{List}$, $h$ is not a cons and the recursive argument has type $\text{List}$. The other cases are similar or simpler. Assume $f$ is defined by

$$f(\text{nil}, y) = g(y) ; f(\text{cons}(a, l), y) = h(a, l, f(l, y), y)$$

Let $g_{s1}, g_{s2}$ (resp $h_{s1}, h_{s2}$) be the transformations of $g$ (resp $h$).

Since $\text{Ext}(\text{PR}) = \text{Ext}(\text{PRL})$, there is $f_2$ in $\text{PR}$ such that:

$$f_2(0, h, y) = \text{If } h \text{ is the code of } \perp \text{ then } h \text{ else } g_{s2}(y)$$

$$f_2(Sn, h, y) = h_{s2}(\phi(n, h), \psi(n, h), f_2(n, \psi(n, h), y), y)$$

Define $f_1$ by:

$$f_1(0, h, y, n) = g_{s1}(y)$$

$$f_1(Sx, h, y, n) = h_{s1}(\phi(n, h), x, \psi(n, h), f_1(x, h, y, Sn), y)$$

Let $f_{s2}(h, y) = f_2(\lg(h), h, y)$ and $f_{s1}(x, h, y) = f_1(x, h, y, 0)$. Then it is easy to check that $(f_{s1}, f_{s2})$ satisfy the desired properties.

8 Conclusion and open questions

We have investigated some (functional) programing languages that compute exactly the class of primitive recursive functions. We studied the behaviour of programs on incomplete inputs by looking at the trace (a word representing the step by step computation) of an algorithm. This work solved many questions on the relationship between these languages from the intentional point of view.

Some questions remain unsolved.
1. The main one is: \( \text{Int}(PRL) = \text{Int}(PRL_0) \) ?
   Similarly, \( \text{Int}(PRV) = \text{Int}(PRL_0) \) ? Note that we mean here the full \( PRV \), ie without the restriction given in the section 7.

2. The following question is important for (time) complexity reasons. Some simulations given in this paper are not strong. Most of the time, this is because in the trace \( F(S^\infty(x)) \) \( x_k \) has been replaced by \( x_0, ..., x_k \) and this increases the time complexity of the computation. Is it possible to make all the simulations strong ?

3. May be less importantly : Is it true that for \( P \in \{PRV, PRL\} \text{Int}(P) = \text{Ext}(PN) \) ?

4. Finally the following question is very challenging.

   - [3] gives a combinator \( f \) in \( PRL_0 \) that computes the \( \inf \) function with the good intentional behaviour (ie \( \text{int}(f) = \text{ext}(f) = \inf \)) but a bad time complexity since the computation time of \( f(n,m) \) is \( O(\inf(n,m)^2) \).
   - [8] gives a combinator \( f \) in \( PRL_0 \) that computes the \( \inf \) function with a good time complexity (the computation time of \( f(n,m) \) is \( O(\inf(f(n,m))) \)) but a bad intentional behaviour (we can only get \( f(S^n, S^m) = S^p \) where \( p \) is the largest such that \( 2^p \leq \inf(f(n,m)) \)).
   - Is there a combinator in \( PRL_0 \) that computes the \( \inf \) function both with a good time complexity and a good intentional behaviour. We conjecture the answer is No.

**Appendix**

In this section we recall (for self completeness) the main definitions and results about the traces (for any data types) as given in [10].

**Definition 24** Data types are recursively defined in the following way: a data type \( D \) is given by a list of typed constructors. This is written as:

\[
D = \{ cf_i : D_1 \to ... \to D_n \to D / i = 1, \cdots, p \}
\]

Let \( cf : D_1 \to \cdots \to D_n \to D \) be a constructor of \( D \), then

- the \( D_j \) are either \( D \) or previously defined data types.
- if \( D_j = D \), then \( j \) is called a recursive argument of \( cf \).
- if some \( D_j \) is \( D \), then \( cf \) is said to be recursive.
- if \( n = 0 \), then \( cf \) is said to be terminal.

A data type must have at least one non recursive constructor.

**Definition 25** An address (also called a cell) is a finite list of positive integers.

- The empty list is denoted by \( \epsilon \).
- If \( a, a' \) are addresses, \( a \leq a' \) means that \( a \) is an initial segment of \( a' \).
- If \( a \) is a (finite or infinite) list of integers of length at least \( m \), \( a|m \) is the list \( [a(0), ..., a(m-1)] \).
• If \( a \) is an address and \( p \) an integer, \([a::p]\) denotes the list obtained by adding \( p \) at the end of \( a \).

**Definition 26**  
1. An element of the data type \( D \) is a partial function from some accessible addresses to the set of constructors (of \( D \) or -recursively- of data types occurring in the definition of \( D \)). Let \( a \) be accessible: If the function is defined at the address \( a \) and the image of \( a \) is \( cf \), we say \( a \) is filled with \( cf \). Otherwise, we say \( a \) is empty. The function has to satisfy the following properties:
   - \( \varepsilon \) is always accessible. It may be filled with a constructor of \( D \).
   - \([a::p]\) is accessible if \( a \) is accessible, \( a \) is filled with a non terminal constructor \( cf \) of type \( D_1 \to \cdots \to D_n \to D' \) and \( 1 \leq p \leq n \). \([a::p]\) may be filled with a constructor of \( D_p \).
2. \( t \) is a finite element of \( D \) if \( t \) has a finite number of accessible cells.
3. Let \( t, t' \) be elements of \( D \). \( t \leq t' \) means the following: If \( a \) is an address accessible in \( t \), then \( a \) also is accessible in \( t' \). If \( a \) is filled (by \( t \)) with \( cf \), then \( a \) also is filled (by \( t' \)) with the same \( cf \).
4. Let \( a \) be an address accessible in \( t \). \( a \) is maximal in \( t \) if \([a::p]\) is not accessible in \( t \) for any \( p \). This means that in \( t \) either \( a \) is empty or filled with a terminal constructor.

**Definition 27**  The sets of typed combinators are defined by recursion as the least sets containing the projections, the constructors and closed by composition and primitive recursion. Primitive recursion is defined as follows. I will assume -without loss of generality- that the recursion always is on the first argument of the combinator. There is one equation for each constructor \( cf \) of the data type of the first argument. Assume \( cf \) has \( p \) arguments and (for the simplicity of notations), the recursive arguments of \( cf \) are \([j/1 \leq j \leq m]\). Note that \( p \) or \( m \) may be 0. Then, the recursive equation for \( cf \) is (\( h \) is a previously defined combinator associated to \( cf \)):
   \[ f(cf(t_1, \ldots, t_p, u_2, \ldots, u_n)) = h(f(t_1, u_2, \ldots, u_n), \ldots, f(t_m, u_2, \ldots, u_n), t_1, \ldots, t_p, u_2, \ldots, u_n). \]

**Definition 28**  
1. Let \( V \) be \([x_a/x \text{ is a letter and } a \text{ is an address}]\). A word is a finite (possibly empty) or infinite sequence of symbols in \( V \). If \( u, u' \) are words, \( u \leq u' \) means that \( u \) is an initial segment of \( u' \). \( \oplus \) denotes the concatenation on words.
2. The set \( TR(D) \) of traces of type \( D \) is defined as follows. \( w \) is in \( TR(D) \) iff \( w \) is a pair \((Val(w), \{w_a/a \text{ accessible in } Val(w)\})\) where \( Val(w) \) is an element of \( D \) and for every address \( a \) accessible in \( Val(w) \), \( w_a \) is a word. \( w_a \) must be finite if \( a \) is non empty. We also assume that the set of distinct letters appearing in the \( w_a \) is finite.
3. \( TR_f(D) \) is the set of traces \( w \) such that \( Val(w) \) is finite and every \( w_a \) is a finite word.
4. For \( v, w \) in \( TR(D) \), \( w \leq v \) means the following:
   - \( Val(w) \leq Val(v) \)
   - For every address \( a \) accessible in \( Val(w) \), \( w_a \leq v_a \) and, if \( a \) is filled in \( Val(w) \), \( w_a = v_a \).
5. \( w \) is in \( TR \) (resp \( TR_f \)) iff \( w \) is in \( TR(D) \) (resp. \( TR_f(D) \)) for some \( D \).

**Abbreviations**
1. Let $t$ be an element of $D$ and $x$ be a letter. Define $w = t(x)$ in the following way: $Val(w) = t$ and, for every address $a$ accessible in $t$, $w_a = x_a$. $t(x)$ is in $TR(D)$ and is called an "element with name $x$".

2. In the case of the type $Int$ (where the trees have no branching) a trace may be written as a list of symbols either in $V$ (representing the labels of the edges) or in $\{S, 0\}$ (representing the labels of the nodes). It is also convenient to write $x^k$ instead of $x_{a}^k$ where $a$ is the infinite list $[1, 1, ...]$. For example $S^\infty(x)$ may be written as follows:

$$x_0 S x_1 S x_2 S x_3 S ...$$

**Proposition 23** Every trace is the Sup of an increasing sequence of traces in $TR_f$. Conversely, let $(w_k)$ be an increasing sequence of traces (using a fixed number of distinct letters). Then $\text{Sup}(w_k)$ is in $TR$.

**Definition 29** A function $F$ from $TR^n$ to $TR$ is increasing if for every $v_1, ..., v_n$ and $w_1, ..., w_n$ such that $v_i \leq w_i$ for every $i$, then $F(v_1, ..., v_n) \leq F(w_1, ..., w_n)$. $F$ is continuous if it is increasing and it preserves the Sup.

**Proposition 24** Every combinator $f$ of type $D_1 \to ... \to D_n \to D$ induces (in an unique way) a continuous function (denoted by $F$) from $TR(D_1) \times ... \times TR(D_n)$ to $TR(D)$ such that:

- if $f$ is the $i$-th projection then $F(w_1, ..., w_n) = w_i$
- if $f$ is the constructor $cf$ then $F(w_1, ..., w_n) = cf(w_1, ..., w_n)$
- if $f = g(h_1, ..., h_k)$ and $r_i = H_i(w_1, ..., w_n)$ then $F(w_1, ..., w_n) = G(r_1, ..., r_k)$
- if $f$ is defined by recursion:
  - If the cell $c$ is empty in $Val(w_1)$, then $F(w_1, ..., w_n) = (w_1)_c$
  - If the cell $c$ is filled in $Val(w_1)$ with $cf$, $w_1 = u \oplus cf(v_1, ..., v_p)$, the recursive arguments of $cf$ are $\{j/1 \leq j \leq m\}$ and $cf$ is associated with $h$, then: $F(w_1, ..., w_n) = u \oplus H(F(v_1, w_2, ..., w_n), ..., F(v_m, w_2, ..., w_n), v_1, ..., v_p, w_2, ..., w_n)$

**Remark** In particular, $f$ induces a continuous function from the set of elements (the usual finite ones, but also the incomplete or the infinite ones) of $D_1 \times ... \times D_n$ into $D$. We of course have $\text{Sup}(F(w_1, ..., w_n)) = f(\text{Sup}(w_1), ..., \text{Sup}(w_n))$.

**Proposition 25** Let $f$ be a combinator and $w_1, ..., w_n$ be in $TR_f$. Then, $F(w_1, ..., w_n)$ is in $TR_f$.

**Definition 30** Let $w$ be in $TR$.

1. An addressing branch for $w$ is either a maximal address in $w$ or a function $a$ from $N$ to $N^*$ such that for every $m$, $a(m)$ is accessible in $Val(w)$. If $a$ is an addressing branch for $w$, $a(w)$ is the word obtained by concatenating the words $w_{a(n)}$ for $n$ less than the length of $a$.
2. A branch in $w$ is a word of the form $a(w)$ for some addressing branch $a$.

**Definition 31** Let $x$ be a letter.

1. Let $u$ be a word. $x$ is regular in $u$ if for every addresses $a \leq a'$, if $x_{a'}$ appears in $u$, then $x_a$ also appears in $u$ and the first occurence of $x_a$ appears before the first occurrence of $x_{a'}$.  

22
2. Let \( w \) be in \( TR \). \( x \) is regular in \( w \) if \( x \) is regular in every branch of \( w \).

A word \( u \) (resp a trace \( w \)) is regular if every letter is regular in \( u \) (resp in \( w \)).

**Proposition 26** Let \( f \) be a combinator and \( w_1, \ldots, w_n \) be in \( TR \).

1. Assume that \( x \) is regular in every \( w_i \), then \( x \) also is regular in \( F(w_1, \ldots, w_n) \).

2. Assume that every \( w_i \) is regular, then \( F(w_1, \ldots, w_n) \) also is regular.

**Definition 32** Let \( x \) be a letter. Let \( v, w \) be in \( TR \) and \( u = b(w) \) be a branch in \( w \).

1. \( x \) is compatible with \( v \) in \( u \) if
   
   • \( x \) is regular in \( u \).
   
   • if \( x_a \) appears in \( u \), then \( a \) is accessible in \( Val(v) \).
   
   • assume \( a \) is empty in \( Val(v) \) and \( x_a \) appears in \( u \). Then \( b \) is empty (and thus finite and maximal) in \( Val(w) \) and \( x_a \) appears only in \( u \) as the last symbol.

2. \( x \) is compatible with \( v \) in \( w \) if \( x \) is compatible with \( v \) in every branch of \( w \).

3. Let \( (v_i) \) be a finite sequence of traces and \( (x_i) \) be a finite sequence of letters. Assume that, for every \( i \), \( x_i \) is compatible with \( v_i \) in \( u \) (resp in \( w \)). Then \( u[x_i := v_i/i = 1, \ldots, n] \) (resp \( w[x_i := v_i/i = 1, \ldots, n] \)) is the word (resp the trace) obtained by simultaneously replacing every \( (x_i)_a \) by \( (v_i)_a \) in \( u \) (resp in every word \( w_k \), for \( b \) accessible in \( Val(w) \)).

**Proposition 27** The substitution is a continuous function. More precisely, let \( (w_k), (v_k) \) be increasing sequence of traces. Let \( w = \text{Sup}(w_k) \) and \( v = \text{Sup}(v_k) \). Assume that, for every \( k \), \( x \) is compatible with \( v_k \) in \( w_k \). Then \( x \) is compatible with \( v \) in \( w \) and \( w[x := v] = \text{Sup}(w_k[x := v_k]) \).

**Proposition 28** Let \( f \) be a combinator and \( w_1, \ldots, w_n \) be in \( TR \). Let \( v_i \) be the named element (with the fresh name \( x_i \)) such that \( Val(v_i) = Val(w_i) \). Then \( x_i \) is compatible with \( w_i \) in \( F(v_1, \ldots, v_n) \) and \( F(w_1, \ldots, w_n) = F(v_1, \ldots, v_n)[x_i := w_i/i = 1, \ldots, n] \).

**References**


