On the asymptotic behaviour of primitive recursive algorithms

René David

On the asymptotic behaviour of primitive recursive algorithms

René DAVID*

Abstract

This paper develops a new semantics (the *trace* of a computation) that is used to study intensional properties of primitive recursive algorithms. It gives a new proof of the "ultimate obstination theorem" of L.Colson and extends it to the case when mutual recursion is permitted. The ultimate obstination theorem fails when other data types (e.g. lists) are used. I define another property (*the backtracking property*) of the same nature but which is weaker than the obstinate obstination. This property is proved for every primitive recursive algorithm using any kind of data types.

Keywords primitive recursive definitions, intensionality, complexity.

1 Introduction

In [3], [7] the denotational semantics of lazy integers is used to prove intensional properties of primitive recursive algorithms. L.Colson proves the *ultimate obstination theorem* and T.Coquand gives a constructive proof of it. An important consequence of the ultimate obstination theorem is that the *inf* of two integers cannot be computed, by a primitive recursive algorithm, neither in the desired way (i.e. by decrementing alternatively the two arguments), nor in the desired time complexity (i.e. $O(inf)$) .

I develop here a new semantics to study the intensional behaviour of algorithms. The intuition is the following. Let $\mathcal{N}$ be the domain of lazy integers. An element e of $\mathcal{N}$ can be seen as a partial function that *fills* some accessible *cells* (in the sense of [2]) with the constructors S and 0. For example in $e_0 = S(0)$ the accessible cells are the ones denoted by their address 0 and 1. The first one is filled with S and the second with 0. In $e_1 = S^2(\perp)$ the accessible cells are the ones denoted $0, 1, 2$. The cells 0 and 1 are filled with S and the third one is *unfilled* (see figure 1).

$e_0 =$

cell number	0	1
constructor	S	0

and $e_1 =$

cell number	0	1	2
constructor	S	S	

fig. 1

The set of traces is defined as follows. Let W be the set of (finite or infinite) words on the alphabet $\{x_n \ / \ n \geq 0, \ x \text{ is a letter}\}$. A trace is a pair (e, λ) where $e \in \mathcal{N}$ and λ is a labelling, i.e. a function from the accessible cells of e to W (see examples in figure 2).

*Laboratoire de Mathématiques. Campus Scientifique. 73376 Le Bourget du Lac Cedex. email david@univ-savoie.fr

To each primitive recursive definition f we associate a function $[[f]]$ from traces to traces which "codes" the way f gets its result : The fact that the *token* x_i occurs in $\lambda(n)$ intuitively means that the cell i of the element named x has been used to get $e(n)$.

An example is given in figure 2 : Define *add* as usual by $add(0,m) = m$ and $add(Sn,m) = S\ add(n,m)$.

- The trace t_2 means that to get S the algorithm has used the cell 0 of t_0 and to get 0 the algorithm has used first the cell 1 of t_0 and next the cell 0 of t_1.

- The trace t_3 means that to get S the algorithm has used first the cell 0 of t_1 and next the cell 0 of t_0 and to get 0 the algorithm has used the cell 1 of t_0.

$t_0 =$

cell number	0	1
constructor	S	0
labelling	x_0	x_1

$t_1 =$

cell number	0
constructor	0
labelling	y_0

$t_2 = [[add]](t_0, t_1) =$

cell number	0	1
constructor	S	0
labelling	x_0	$x_1\ y_0$

$t_3 = [[add]](t_1, t_0) =$

cell number	0	1
constructor	S	0
labelling	$y_0\ x_0$	x_1

fig. 2

This is easily generalized to any data type. In this case, the cells are no more given by integers but by their addresses (i.e. lists of integers) in the tree representing an element of the data type. This notion of trace is related to the sequential algorithms introduced by Berry and Curien ([2] or [1], chapter 14) as follows. In their terminology, a sequential algorithm is a tree. Each branch of this tree corresponds to the computation of the algorithm on particular arguments, that is exactly (with a slight variation on the syntax and the terminology) what I call a trace.

The main advantages of this approach are the following :

(1) There is a notion of modularity (see theorem 34): If e is an element of $\mathcal{N}$, let $e[x]$ be the trace (e,λ) where $\lambda(n) = x_n$ for each n. Then, for $t = (e,\lambda')$, $[[f]](t)$ is obtained by substituting x_i with $\lambda'(i)$ in $[[f]](e[x])$.

(2) A single infinite trace contains the information about each finite computation (see proposition 36). This will be extensively used in the forthcoming papers [8] and [11].

(3) This notion allows to introduce new properties of computations : The backtracking property (see below) cannot be expressed in the usual semantics.

(4) I believe it also makes the proofs easier and, at least, closer to the intuition. In particular, the extension of Coquand's constructive result to the case where mutual recursion is allowed would probably be impossible without the notion of trace.

Say that a trace (e,λ) is ultimately obstinate if, in the word obtained by concatenating the words $\lambda(n)$, there is at most one letter which occurs with unbounded indexes. The intuitive meaning is that, if the trace represents an infinite computation, *at most one argument may be used entirely.* The ultimate obstination theorem follows immediately from the fact (see theorem 13) that, if t_1, ..., t_n are ultimately obstinate, then so is $[[f]](t_1, ..., t_n)$. The main argument in its proof is that, when the first S in an infinite sequence of S is removed, we get the same sequence. This is

of course no more true e.g. for infinite sequences of booleans and thus, the theorem fails when other data types may be used.

Say that the letter x *backtracks* in the word w if, for n large enough, x_n occurs infinitely many times in w. This intuively means that the argument denoted by x may not be "garbage-collected" in the computation represented by w. Say that a trace t has the *backtracking property* if the following holds for any branch b in t (a branch in t is the usual notion on the underlying element) : let w be the word obtained by concatenating the words along b. There is at most one letter x such that : x occurs with unbounded indexes and x does not backtrack . When t represents the computation of an algorithm, this intuitively means that, in the computation of the branch b of the result, at most one argument can be *memorized* (recall that being ultimately obstinate means at most one argument can be *used*). I prove (see theorem 16) that if t_1, ..., t_n have the backtracking property, then so does $[[f]](t_1, ..., t_n)$.

The ultimate obstination theorem is a result about intensionality but it has a consequence in terms of complexity. I believe this is a kind of chance. I introduced (and proved) the backtracking property because it was thought that such a property would give $O(inf^2)$ as a lower bound for the time complexity of the *inf* function but it does not : see the algorithm given in [9]. I thus have no application of this result in terms of complexity (see section 6 for a discussion about this point). However the notion of trace allows to prove some other results. In a forthcoming paper ([8]) I will extend Coquand's constructive result to the case when mutual recursion is allowed. In another paper, in preparation with Valarcher ([11]), we will use the traces to answer open questions in his thesis ([19]).

Warning A primitive recursive definition becomes an algorithm only when a strategy of reduction is given. Even if the strategy does not appear explicitly in this paper, it is hidden in the definition of $[[f]]$ (see proposition 9) and corresponds to call by name. [13], [14] show that, in call by value, the *inf* function cannot be computed in time $O(inf)$ even when lists or mutual recursion is allowed. Note that, in this case, the problems are, at least intuitively, much easier since, when an argument is used, the computation time is, by definition of call by value, at least the value of this argument.

The paper is organized as follows : The section 2 gives the main definitions and results of the paper. In section 3, I prove the main properties of traces, in particular theorem 34 about substitutions. The section 4 and 5 give the proofs of the preservation of the ultimate obstination (as well as its consequences in terms of complexity) and of the backtracking property. The section 6 gives some open questions.

Acknowledgement This paper has a very long story. Many people helped me to transform a very rough draft into this final version. Thanks to all of them and, in particular, T Coquand, C Berline, P L Curien and the anonymous referees.

2 Definitions

2.1 Primitive recursive algorithms

Notations A data type is given by a list of typed constructors. Let cf: $D_1 \times ... \times D_n \to D$ be a constructor of D (n is called the arity of cf). Then :

1. The D_j are either D or previously defined data types.
2. If $D_j = D$, then j is called a recursive argument of cf.
3. cf is recursive if $D_j = D$ for some j.

4. *cf* is terminal if $n = 0$.

Note that, in order to be non-empty, a data type must have at least one non-recursive constructor.

Examples

1. The data type of integers is given by $N = \{0 : N,\ S : N \to N\}$. 0 is terminal and S is recursive.
2. The data type of lists of type N is given by $L = \{nil : L,\ cons : N \times L \to L\}$. *cons* has a recursive and a non-recursive argument.
3. The data type of sequences of 0 and 1 is given by $D = \{nil : D,\ s_0 : D \to D,\ s_1 : D \to D\}$.

Definition 1 *1. The sets of n-ary typed prc (primitive recursive combinators) are defined, as usual, as the least sets containing the projections, the constructors and which are closed under composition and primitive recursion.*

2. Primitive recursion is defined as follows (I will assume, without loss of generality, that the recursion always is on the first argument of the prc). There is one equation for each constructor cf of the data type of the first argument. Assume cf has p arguments and (for simplicity of notation), 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 prc associated to cf) :

$$f(cf(x_1, ..., x_p), \overrightarrow{y}) = h(f(x_1, \overrightarrow{y}), ..., f(x_m, \overrightarrow{y}), x_1, ..., x_p, \overrightarrow{y}).$$

Examples

1. The addition is defined, as usual, by : $add(0, n) = n$ and $add(Sm, n) = S\ add(m, n)$.
2. The sum of the elements of a list of integers is defined by : $sum(nil) = 0$ and $sum(cons(n, l)) = add(n, sum(l))$.
3. The number of 0 in a list of 0 and 1 is defined by : $nb(nil) = 0$, $nb(s_0(l)) = S\ nb(l)$, $nb(s_1(l)) = nb(l)$.

Remark
In the section 3 we will also allow the definition of k functions by mutual recursion (for an arbitrary k). For example : $even(0) = true$ and $odd(0) = false$. $even(Sx) = odd(x)$ and $odd(Sx) = even(x)$.

2.2 The trace

In the rest of the paper I will adopt the following conventions (words, traces, ... are defined in this section) :

symbols	range over	symbols	range over
i, j, k, m, n, p, q	integers	u, v, w	words
e	elements of a data type	$r, s, t, \rho, \sigma, \tau$	traces
x, y, z, X	letters	f, g, h	*prc*
$a, b, c, d, \alpha, \beta$	addresses or addressing branches		

Definition 2 *1. An address is a finite list of positive integers. The empty list is denoted by ε.*

2. *If a, a' are addresses, $a \leq a'$ means that a is an initial segment of a'.*

3. *$lg(a)$ represents the length of a and thus, if $lg(a) = n$, a may be written as $[a(0), ..., a(n-1)]$.*

4. *If a is a (finite or infinite) list of integers of length at least m, $a \uparrow m$ is the prefix of length m of a, i.e. $a \uparrow m = [a(0), ..., a(m-1)]$.*

5. *If a is an address and p an integer, $a+p$ denotes the list obtained by concatenating p at the end of a.*

Comment and examples

An address corresponds to a cell in [2]. $[0,1] + 3 = [0,1,3]$

Definition 3 *Let D be a data type.*

1. *An element e of D is a partial function from a prefix closed set of addresses (denoted by $dom(e)$) satisfying the following conditions :*

   (a) *If $\varepsilon \in dom(e)$ then $e(\varepsilon)$ is a constructor of D.*

   (b) *If $a+p \in dom(e)$, $e(a) = cf$ and $cf : D'_1 \times ... \times D'_n \to D'$ then $1 \leq p \leq n$ and $e(a+p)$ is a constructor of D'_p .*

2. *Let e be an element of D and a be an address. Define the accessibility of a in e by the following rules :*

   (a) *ε is accessible in e.*

   (b) *$a+p$ is accessible in e iff $a \in dom(e)$ and $1 \leq p \leq arity(e(a))$.*

3. *Let e be an element of D. Denote by $Acc(e)$ the set of addresses that are accessible in e.*

4. *An element e is finite iff $dom(e)$ is finite.*

5. *Let e, e' be elements of D. $e \leq e'$ means : $dom(e) \subset dom(e')$ and for all $a \in dom(e)$, $e(a) = e'(a)$.*

6. *An address a is maximal in an element e if $a \in Acc(e)$ and no proper extension of a is in $dom(e)$.*

Comment and examples

1. It is easy to see that D is a domain.

2. Usually, an element of a data type is a *finite* tree whose nodes are filled with constructors. Here an element again is a tree but :

   - the tree may have infinite branches. Infinite branches may be seen as "streams".
   - its leaves may be unfilled.

   $a \in dom(e)$ and $cf = e(a)$ means that the cell of address a is *filled* with the constructor cf. An *unfilled* cell a (i.e. $a \in Acc(e) - dom(e)$) corresponds to a lack of information for the content of the cell. The correspondence with, in particular, [2] is the following : I call here accessible (respectively unfilled) what they call enabled (respectively accessible).

3. a is maximal in e if it is accessible in e and either a is unfilled in e or it is filled with a terminal constructor.

4. In the data type of integers the elements are the following (I will write : $1^0 = \varepsilon$ and $1^i = \underbrace{[1, \ ..., \ 1]}_{i}$).

   - $S^n(0)$ stands for : $\{(1^i, S) \ / \ 0 \leq i < n\} \cup \{(1^n, 0)\}$.
   - $S^n(\perp)$ stands for : $\{(1^i, S) \ / \ 0 \leq i < n\}$. Note that here the address 1^n is accessible.
   - S^ω stands for : $\{(1^i, S) \ / \ 0 \leq i\}$.

5. In the data type of lists of type N, the lists $e_0 = [0, 1]$, $e_1 = [0, 0, ...]$ (the infinite list) and $e_2 = cons(0, \perp)$, are given in figure 3. In e_2 the address $[2]$ is accessible but $[2] \notin dom(e_2)$ and is, as usual, labelled by $\perp$.

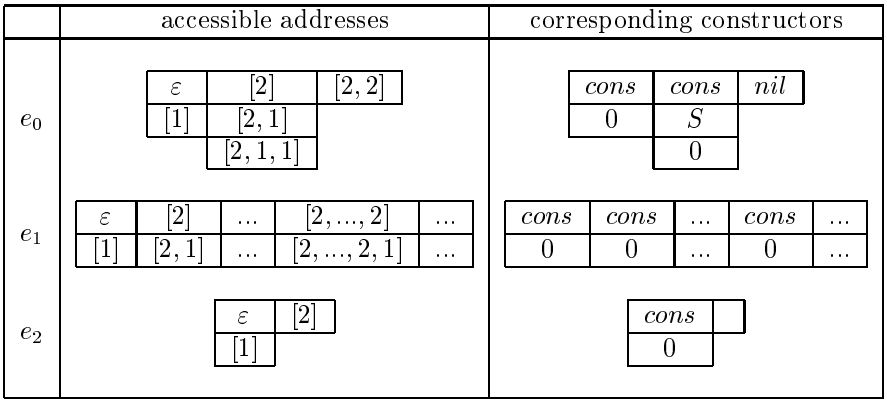

fig. 3

Definition 4 *1. Let $\Sigma = \{x_a \ / \ x$ is a letter and a is an address$\}$. The elements of Σ are called tokens.*

2. A word is a finite (possibly empty) or infinite sequence of tokens. The set of words is thus $W = \Sigma^* \cup \Sigma^\omega$. The empty word is denoted by $\emptyset$.

3. Let u, u' be words. $u \leq u'$ means that u is a prefix of u' and $u \uparrow p$ denotes, for $p \leq lg(u)$, the prefix of u of length p.

4. $u + u'$ is the result of concatenating u' at the end of u. When u is infinite, this is just u again. More generally, if (u_k) is a (finite or infinite) sequence of words $u_0 + u_1 + \ ...$ will be denoted by $\sum u_k$.

Abbreviation In this paper, when x corresponds to an element of N, I will write x_i instead of x_{1^i}.

Definition 5 *1. A trace over the data type D is a pair (e, λ) where e is an element of D and λ is a labelling function $\lambda : Acc(e) \to W$ such that : $\forall a \in dom(e)$, $\lambda(a) \in \Sigma^*$.*

2. A trace (e, λ) is finite if e is finite and all labels are finite, i.e. $rge(\lambda) \subset \Sigma^*$.

3. The ordering on traces is given by : $(e, \lambda) \leq (e', \lambda')$ iff $e \leq e'$ and $\forall a \in Acc(e)$ $\lambda(a) \leq \lambda'(a)$ and $\forall a \in dom(e)$ $\lambda(a) = \lambda'(a)$.

4. *The set of traces over D is denoted by T(D). A trace is a trace over some data type. The set of traces is denoted by T.*

5. *The set of finite traces over D is denoted by $T_f(D)$. The set of finite traces is denoted by T_f.*

6. *Let e be an element of D and x be a letter. The trace (e, λ) where $\lambda(a) = x_a$ for all $a \in Acc(e)$ will be denoted as $e[x]$. A trace as $e[x]$ is called an element named x.*

7. *Let $t = (e, \lambda)$ be a trace. e is called the value of t and is denoted by $Val(t)$. λ is called the labelling of t and is denoted by $lab(t)$.*

Comment and notations

1. The labelling of a cell intuitively codes the part of the computation that has been made to get the content of this cell. This computation has to be finite if the constructor is eventually found (i.e. if the cell is filled). Otherwise it may be infinite.

2. Let $t = (e, \lambda)$ be a trace. By extending the function e for $a \in Acc(e) - dom(e)$ by $e(a) = \perp$ one may consider that a trace is a tree whose nodes are labelled by a pair : the first element is either a constructor or $\perp$ and the second element is a word. A trace t has thus one of the two shapes.

   (a) A single accessible address (the empty sequence) which is unfilled and labelled with the word $w \in W$. This will be denoted as : $t = (\perp, w)$.

   (b) A tree whose root is (c, w), where c is an n-ary constructor, $w \in \Sigma^*$ and each of the n branches is another tree. This will be denoted as : $t = \langle (c, w)\ t_1, ...,\ t_n \rangle$. This case has a degenerate situation when c is terminal. Then, the only accessible address is ε and we simply write : $t = (c, w)$.

3. The named elements of N are (since the tree has only one branch, I do not write the "$\langle$" and "$\rangle$") :
   $S^n(0)[x] = (S, x_0)(S, x_1)...(S, x_{n-1})(0, x_n)$.
   $S^n(\perp)[x] = (S, x_0)(S, x_1)...(S, x_{n-1})(\perp, x_n)$
   $S^\omega[x] = (S, x_0)(S, x_1)...(S, x_n)...$

4. Let $t = (e, \lambda)$ be a trace. $Acc(t)$ will denote $Acc(e)$.

5. We often will have to "choose fresh letters" and for that it could be useful to ensure that the alphabet (i.e. the set of letters occurring in the tokens) of a trace is finite. Since this introduces only inessential problems, I will not care here about this.

Proposition 6 *$T(D)$ with its ordering forms a domain. In particular :*

1. *Every trace is a least upper bound (denoted by Sup) of an increasing sequence of finite traces.*

2. *Every increasing sequence has a Sup.*

Proof. Immediate. ■

A primitive recursive definition f induces a function on the domain associated to the corresponding data type. The proposition 9 shows that it also induces a function (denoted by $[[f]]$) on the corresponding traces. It is the study of this function that will allow to understand the intensional behaviour of f.

Definition 7 *Let $t = (e, \lambda)$ be a trace and w be a finite word. $w + t$ is the trace (e, λ') defined by : $\lambda'(\varepsilon) = w + \lambda(\varepsilon)$ and $\lambda'(a) = \lambda(a)$ for $a \neq \varepsilon$.*

Comment and examples

$w + t$ is obtained by prepending w to the word at the root of t. $y_0 + S(0)[x] = (S, y_0\ x_0)\ (0, x_1)$

Definition 8 *Let f be a function from T^n to T.*

1. *f is increasing if for all $t_j \leq t'_j$, $f(t_1,\ ...,\ t_k) \leq f(t'_1,\ ...,\ t'_k)$.*
2. *f is continuous if it is increasing and preserves the Sup of increasing sequences.*

Proposition 9 *Every prc f from $D_1 \times ... \times D_n$ to D induces (in a unique way) a continuous function (denoted by $[[f]]$) from $T(D_1) \times ... \times T(D_n)$ to $T(D)$ such that :*

- *If f is the i-th projection then $[[f]](t_1,\ ...,\ t_n) = t_i$*
- *If f is the n-ary constructor cf then $[[f]](t_1,\ ...,\ t_n) = \langle (cf, \emptyset)\ t_1, ...,\ t_n \rangle$*
- *If $f = g(h_1, ..., h_k)$ then $[[f]](t_1,\ ...,\ t_n) = [[g]](r_1,\ ..., r_k)$ where $r_j = [[h_j]](t_1,\ ...,\ t_n)$*
- *If f is defined by recursion then $[[f]](t, \overrightarrow{s}) =$*
  - *$(\bot, w)$ if $t = (\bot, w)$.*
  - *$w + [[h]](\tau_1, ..., \tau_m, r_1,\ ...,\ r_p, \overrightarrow{s})$ if $t = \langle (cf, w)\ r_1,\ ...,\ r_p \rangle$, $\tau_j = [[f]](r_j, \overrightarrow{s})$ and the recursive equation concerning the constructor cf is $f(cf(x_1, ..., x_p), \overrightarrow{y}) = h(f(x_1, \overrightarrow{y}), ..., f(x_m, \overrightarrow{y}), x_1, ..., x_p, \overrightarrow{y})$.*

Proof. First note that, when f is defined by recursion, the case $t = (c, w)$ is a degenerate special instance of the second clause. $[[f]]$ is defined by induction on f. The only non-trivial case is when f is defined by recursion. It is clear that the desired property defines $[[f]]$ (by induction on the size of $Val(t)$) on $T_f \times T^{n-1}$ and that (on $T_f \times T^{n-1}$) $[[f]]$ is continuous. Otherwise, define $[[f]](t, \overrightarrow{s})$ as follows : Let (τ_k) be an increasing sequence of finite traces such that $t = Sup\ \tau_k$. Since the sequence $[[f]](\tau_k, \overrightarrow{s})$ is increasing we may define $[[f]](t, \overrightarrow{s})$ as $Sup\ [[f]](\tau_k, \overrightarrow{s})$. It is easy to check, because $[[f]]$ is increasing on $T_f \times T^{n-1}$, that this definition does not depend on the chosen sequence and that $[[f]]$ satisfies the desired properties. The uniqueness for T_f (and thus, by continuity, for T) is clear. ■

Examples

$$
\begin{array}{lll}
 & [[add]](S(0)[x], S^\omega[y]) & = x_0 + [[S]]([[add]]((0, x_1), S^\omega[y])) \\
 & & = x_0 + (S, \emptyset)\ [[add]]((0, x_1), S^\omega[y]) \\
 & & = (S, x_0)\ [[add]]((0, x_1), S^\omega[y]) \\
 & & = (S, x_0)\ (x_1 + S^\omega[y]) \\
 & & = (S, x_0)\ (S, x_1 y_0)\ (S, y_1)\ (S, y_2) \cdots \\
1. & [[add]](S(0)[x], S^2(\bot)[y]) & = x_0 + (S, \emptyset)\ [[add]]((0, x_1), S^2(\bot)[y]) \\
 & & = (S, x_0)\ (x_1 + S^2(\bot)[y]) \\
 & & = (S, x_0)\ (S, x_1 y_0)\ (S, y_1)\ (\bot, y_2) \\
 & [[add]](S^2(\bot)[y], S(0)[x]) & = y_0 + (S, \emptyset)\ [[add]]((S, y_1)\ (\bot, y_2), S(0)[x]) \\
 & & = (S, y_0)\ y_1 + (S, \emptyset)[[add]]((\bot, y_2), S(0)[x]) \\
 & & = (S, y_0)\ (S, y_1)\ (\bot, y_2)
\end{array}
$$

2. The "usual" algorithm for the function *inf* is defined by :

   $pred(0) = 0$ and $pred(Sn) = n$.

   $dif\,(0, m) = m$ and $dif\,(Sn, m) = pred(dif\,(n, m))$

   $test(0, p, q) = p$ and $test(Sn, p, q) = q$

   $inf\,(n, m) = test(dif\,(n, m), m, n)$

   Claim $[[inf]](S^\omega[x], S^\omega[y]) = (\perp, \sum_{k\geq 0} x_k)$.

   Proof. The following facts are easily verified :

   - $[[pred]](\perp, w) = (\perp, w)$. $[[pred]](0, w) = (0, w)$. $[[pred]]((S, w)\ t) = w + t$

   - $[[dif]]((\perp, w), t') = (\perp, w)$. $[[dif]]((0, w), t') = w + t'$. $[[dif]]((S, w)\ t, t') = w + [[pred]]([[dif]](t, t'))$

   - $[[dif]](S^n(\perp)[x], S^w[y]) = (\perp, \sum_{k\leq n} x_k)$ (immediate induction)

   - $[[dif]](S^\omega[x], S^w[y]) = (\perp, \sum_{k\geq 0} x_k)$ (by continuity)

   - $[[inf]](S^\omega[x], S^\omega[y]) = [[test]]((\perp, \sum_{k\geq 0} w_k), S^\omega[x], S^w[y]) = (\perp, \sum_{k\geq 0} w_k)$.

3. Colson introduces (see [4]) an algorithm, called inf_with_lists, to compute the *inf* of two integers in time $inf(n, m)^2$. This algorithm is defined as follows :

   $incr(nil) = nil$ and $incr(cons(n, l)) = cons(Sn, incr(l))$.

   $L(0) = nil$ and $L(Sn) = cons(0, incr(L(n))$.

   $v(n, m, p, q) = test(dif\,(m, n), p, q)$

   $h(nil, m) = 0$ and $h(cons(n, l), m) = v(n, m, S\ h(l, m), 0)$

   $inf\_with\_lists(n, m) = h(L(n), m)$

   Claim $[[inf\_with\_lists]](S^\omega[x], S^\omega[y]) = (\perp, \sum_{k\geq 0} w_k)$ where $w_k = x_k + \sum_{i\leq k} y_i$.

   Proof. Let $a_i = \underbrace{[2, ..., 2]}_{i}$ and $b_{i,p} = [\underbrace{2, ..., 2}_{i}, \underbrace{1, ..., 1}_{p}]$. The result follows easily from the following facts.

   $L(S^n(0)) = [0, S(0),\ ...,\ S^n(0)]$.

   $L(S^n(0)[x]) = (L(S^n(0), \lambda_n)$ where $\lambda_n(a_i) = x_i$ for $i \leq\ n$ and $\lambda_n(b_{i,p}) = \emptyset$ for $i \leq n$ and $p \leq i$.

4. I introduced another algorithm *Good_inf* (see [9]), also using lists, that computes the *inf* of two integers in time $O(inf)$. This algorithm satisfies :

   $[[Good\_inf]](S^\omega[x], S^\omega[y]) = (\perp, \sum_{k\geq 0} w_k)$ where $w_k = x_k + \sum_{i\leq 2^k} x_i + \sum_{i\leq 2^k} y_i$

Definition 10 *Let $t = (e, \lambda)$ be a trace.*

1. *An addressing branch for t is a maximal path through the tree representing t, i.e. it is either a maximal address in t or a function a from N to N* (the set of positive integers) such that for every m, $a \uparrow m \in dom(e)$.*

2. *Let a be an addressing branch for t. Br(t,a) is the word built by concatenating the labels along the path, i.e. $Br(t, a) = \sum_{k\leq lg(a)} \lambda(a \uparrow k)$.*

3. *A branch in t is a word of the form Br(t,a) for some addressing branch a.*

Examples

1. A trace over N has only one branch. For example the branch of $S^\omega[x]$ is $\sum_{k\geq 0} x_k$.
2. Let t be the list $[0, S0, SS0, ...]$. The branches of $t[x]$ are :
   - For each k, the branch w_k corresponding to the k-th element of the list : $w_k = \sum_{i\leq k} x_{a_i} + \sum_{i\leq k} x_{b_{k,i}}$ where $a_i = \underbrace{[2, 2, ..., 2]}_{i}$ and $b_{k,i} = [\underbrace{2, 2, ..., 2}_{k}, \underbrace{1, ..., 1}_{i}]$
   - The branch corresponding to the list itself : $w = \sum_{i\geq 0} x_{a_i}$

In the proofs of theorem 13 and 16, I will need the following notion of *limit*.

Definition 11 *1. Let (w_n) be a sequence of words. I will write $w = Lim\ (w_n)$ if the following holds : $\forall p\ \exists n_0\ \forall n \geq n_0\ w \uparrow p = w_n \uparrow p$.*

2. Let (t_n) be a sequence of traces. I will write $t = Lim\ (t_n)$ if :

(a) For each n, $Val(t_n) = Val(t)$.

(b) For each branch b of t, $Lim(Br(t_n, b)) = Br(t, b)$.

Remark

1. Note that, for the limit of traces, the first condition is very strong and, in particular, $t = Sup\ t_n$ does not imply $t = Lim(t_n)$: the second condition is satisfied but the first one is not. It would be easy to define a weaker notion of limit to ensure that $t = Sup\ t_n$ implies $t = Lim(t_n)$ but I don't need it in this paper.
2. In the definition of the limit of a sequence of traces, the convergence actually is uniform with respect to the branches : n_0 depends only on p and does not depend on the chosen branch. This simply comes from the fact that, for a given length, t has only a finite number of nodes.

2.3 The ultimate obstination

In this subsection I am only concerned with the data type N. Recall I write x_i instead of x_{1^i}.

Definition 12 *1. Let t be a trace. A letter x is unbounded (respectively bounded) in t if $\{j \ / \ x_j$ occurs in the branch of $t\}$ is infinite (respectively finite).*

2. A trace over N is ultimately obstinate if it has at most one unbounded letter.

Examples

Every named element over N is ultimately obstinate but $(\perp, \sum_{k\geq 0} x_k\ y_k)$ is not.

Theorem 13 *Let f be a prc and $t_1, ...,\ t_n$ be ultimately obstinate traces. Then $[[f]](t_1, ...,\ t_n)$ also is ultimately obstinate.*

Corollary 14 *1. Let f be a prc and $t_1, ...,\ t_n$ be named elements. Then $[[f]](t_1,$ $...,\ t_n)$ is ultimately obstinate.*

2. There is no prc f such that $[[f]](S^\omega[x], S^\omega[y]) = (\perp, w + \sum_{k\geq n} x_k\ y_k)$ where w is a finite word.

The clause (2) of the corollary means that there is no way to make a computation which *ultimately alternates* between two arguments.

2.4 The backtracking property

In this subsection we come back to the general case with all possible data types. The ultimate obstination intuitively means that, in a computation, at most one infinite argument may be *used* entirely. The backtracking property intuitively means that, among the infinite branches of all the arguments at most one may be *memorized*.

Definition 15 *1. Let w be a word, a be a function from N to N* and x be a letter.*

- *x is a-unbounded (respectively a-bounded) in w if* $\{n \ / \ x_{a\uparrow n}$ *occurs in* $w\}$ *is infinite (respectively finite).*
- *x is a-backtracking (abbreviated as a-BT) in w if for every n large enough,* $x_{a\uparrow n}$ *occurs infinitely many times in w.*
- *(x,a) is a BT-counterexample for w if x is a-unbounded but not a-BT in w.*
- *w has the backtracking property (abbreviated as BTP) if there is at most one BT-counterexample for w.*

2. Let t be a trace. t has the BTP if every branch in t has the BTP.

Comment and examples

1. x is unbounded in the sense of the definition 12 is the particular case of being a-unbounded with $a = 1^\omega$.
2. Every named element has the BTP.

   In the examples 3 and 4 below, I again write x_n instead of x_{1^n}.
3. Let $w = \sum_{k\geq 0} x_k \ y_k$. Then, w has not the BTP because $(x, 1^\omega)$ and $(y, 1^\omega)$ are BT-counterexamples.
4. Let $w = \sum_{k\geq 0} w_k$ where $w_k = x_k \ y_0 \ y_1 ... \ y_k$. Then, w has the BTP because $(x, 1^\omega)$ is the only BT-counterexample.
5. Let $w = \sum_{k\geq 0} x_{a_k} \ x_{b_k}$ where $a_k = 1^k$ and $b_k = 2^k$ (w could be a computation using as argument $e[x]$ where e is the list $[S^\omega, 0, 0, ..., 0, ...]$. Note that e has two infinite branches). Since $(x, 1^\omega)$ and $(x, 2^\omega)$ are BT-counterexamples, w has not the BTP .

Theorem 16 *Let f be a prc and* $t_1, ..., \ t_n$ *be traces that have the BTP. Then* $[[f]](t_1, ..., \ t_n)$ *has the BTP.*

Corollary 17 *1. Let f be a prc and* $t_1, ..., \ t_n$ *be named elements. Then* $[[f]](t_1, ..., \ t_n)$ *has the BTP.*

2. There is no prc such that $[[f]](S^\omega[x], (S^\omega[y]) = (\perp, w + \sum_{k\geq n} x_k \ y_k)$ *where w is a finite word.*

3 Some useful properties of traces

In this section, I prove that various properties of traces are preserved by $[[f]]$ for every *prc* f. This is used in sections 4 and 5. I also prove the following key property (see theorem 34) : To compute $[[f]](t)$ it is enough to compute $[[f]](e[x])$ where x is a fresh letter and $e = Val(t)$ and then substitute in the result each x_a by $\lambda(a)$ where λ is the labelling function of t. This implies, in particular, that, in the proofs of theorems 13 and 16, we may assume that the arguments are named elements.

3.1 Finiteness

Proposition 18 *Let f be a prc and $t_1, ..., t_n$ be finite traces. Then $[[f]](t_1, ..., t_n)$ also is finite.*

Proof. By an immediate induction on f and, when f is defined by recursion, by induction on the first argument. ■

3.2 Restrictions

The notion of restriction as defined below plays somehow the role of sequentiality index of [3].

Definition 19 *1. Let w be a word. $w \downarrow x_a = w$ if x_a does not occur in w and otherwise $w' + x_a$ where w' is the longest prefix of w that does not contain an occurrence of x_a.*

2. Let t be a trace. $t \downarrow x_a$ is defined by : $(\perp, w) \downarrow x_a = (\perp, w \downarrow x_a)$.

$\langle (c, w)\ t_1 ... t_n \rangle \downarrow x_a = (\perp, w \downarrow x_a)$ if x_a occurs in w and otherwise $\langle (c, w)\ t_1 \downarrow x_a, ..., t_n \downarrow x_a \rangle$.

Comment and examples

1. $w \downarrow x_a$ is the word obtained by truncating w *after* the first occurrence (if any) of x_a. $t \downarrow x_a$ is the trace obtained by truncating each branch at the first node where x_a occurs.
2. $S^\omega[x] \downarrow x_n = S^n(\perp)[x]$.

Lemma 20 *1. Let w be a word and t be a trace. Then $(w + t) \downarrow x_a = (\perp, w \downarrow x_a)$ if x_a occurs in w and $w + (t \downarrow x_a)$ otherwise.*

2. Let (t_k) be an increasing sequence of traces. Then $Sup\ (t_k) \downarrow x_a = Sup\ (t_k \downarrow x_a)$.

Proof. Immediate. ■

Proposition 21 *Let f be a prc, x be a letter, a be an address and $t_1, ..., t_n$ be traces. Then $[[f]](t_1, ..., t_n) \downarrow x_a = [[f]](t_1 \downarrow x_a, ..., t_n \downarrow x_a)$*

Proof. By induction on f. The only non-trivial case is when f is defined by recursion. For the simplicity of notations let t be the first argument and $\overrightarrow{s}$ be the sequence of parameters.

1. When $Val(t)$ is finite, the result is proved by induction on $Val(t)$. For $t = (\perp, w)$ the result is clear. Otherwise it follows immediately from the induction hypothesis and lemma 20 (1).

2. Otherwise, the result follows by continuity (use the lemma 20 (2)) : Let (r_k) be an increasing sequence of finite traces such that $t = Sup\ r_k$. Then :

$$\begin{array}{rcl} [[f]](t \downarrow x_a,\ \overrightarrow{s} \downarrow x_a) & = & [[f]](Sup(r_k) \downarrow x_a,\ \overrightarrow{s} \downarrow x_a) \\ & = & [[f]](Sup(r_k \downarrow x_a),\ \overrightarrow{s} \downarrow x_a) \\ & = & Sup([[f]](r_k \downarrow x_a,\ \overrightarrow{s} \downarrow x_a)) \\ & = & Sup([[f]](r_k,\ \overrightarrow{s}) \downarrow x_a) \\ & = & Sup([[f]](r_k,\ \overrightarrow{s})) \downarrow x_a) \\ & = & [[f]](t, \overrightarrow{s}) \downarrow x_a.\ \blacksquare \end{array}$$

3.3 Regularity

The regularity intuitively means that, in a computation, a cell may not be accessed before the previous cells have been accessed.

Definition 22 *1. Let w be a word. A letter x is regular in w if for all addresses $a \leq a'$ such that $x_{a'}$ occurs in w, x_a also occurs in w and the first occurrence of x_a is earlier than the first occurrence of $x_{a'}$.*

2. Let t be a trace. A letter is regular in t if it is regular in each branch of t.

3. A word w (respectively a trace t) is regular if every letter is regular in w (respectively in t).

Comment and examples

- Regularity is called safety in [2].
- x is regular in $e[x]$ for every element e. x is not regular neither in $x_{[1]}\ x_\varepsilon$ nor in $x_\varepsilon\ x_{[1,1]}$.

Proposition 23 *A letter x is regular in a trace t iff it is regular in each finite approximation of t, i.e. in each finite $t' \leq t$.*

Proof. Immediate. ■

Proposition 24 *Let f be a prc and $t_1, ...,\ t_n$ be traces.*

1. *Assume that a letter x is regular in each of the t_i, then x also is regular in $[[f]](t_1, ...,\ t_n)$.*
2. *Assume that each of the t_i is regular, then $[[f]](t_1, ...,\ t_n)$ also is regular.*

Proof. (2) follows immediately from (1) which is proved by induction on f. By proposition 23 and the continuity of $[[f]]$, it is enough to prove the result for finite traces. The only non-trivial case is when f is defined by recursion. For the simplicity of notations let t be the first argument and $\overrightarrow{s}$ be the sequence of parameters. The result is proved by induction on the size of $Val(t)$.

If $t = (\perp, w)$ the result is clear. Otherwise, let $t = \langle (c, w)\ r_1 ... r_p \rangle$. $[[f]](t, \overrightarrow{s}) = w + [[h]]([[f]](r_1, \overrightarrow{s}), ..., [[f]](r_m, \overrightarrow{s}), r_1, ..., r_p, \overrightarrow{s})$. We cannot use immediately the induction hypothesis because x is not necessarily regular in $r_1, ...,\ r_m$. For example $S^\omega[x] = \langle (S, x_0)\ r_1 \rangle$ and x is not regular in $r_1 = \langle (S, x_1)\ ... \rangle$. We thus have to change the name of x and make a "lift" on the addresses to make the r_i regular. Let y, z be fresh letters. Get $\overrightarrow{s}'$ from $\overrightarrow{s}$ by replacing x_a by y_a for each a.

For $j \in [1,\ ...,\ m]$, get r'_j from r_j in the following way : Assume x_a occurs in r_j and $lg(a) > q$ where q is the largest number such that $x_{a\uparrow q}$ occurs in w, then replace x_a by $z_{a'}$ where $a' = [a(q), ...,\ a(lg(a) - 1)]$.

It is clear that y and z are regular in $\overrightarrow{s'}$, $r'_1, ..., r'_m$. By induction hypothesis (since $Val(r_j) < Val(t)$) y and z are regular in $t'_j = [[f]](r'_j, \overrightarrow{s'})$ for $j \in [1, ..., m]$. Thus, by induction hypothesis, x, y and z are regular in $[[h]](t'_1, ..., t'_m, r_1, ..., r_p, \overrightarrow{s})$. It is then easy to check (by replacing the letters the other way round, i.e. by replacing y_a by x_a and $z_{a'}$ by x_a) that x is regular in $[[f]](\ t, \overrightarrow{s})$.

Note that the induction hypothesis has been used with a simultaneous substitution of several letters and the proposition is stated ... only for one letter. The general case is of course the same but the notations would be more complicated. ■

3.4 Compatibility

When t represents a computation, x is compatible with s means that t may be seen as a computation using an argument $e[x]$ where $e = Val(s)$, i.e. the "use" of x in t is compatible with e. The intuition for the clause (3) in the definition below is the following : if an address a is unfilled in s (this means a lack of information) and the information at this address is needed in a computation (this means that x_a occurs), then the computation has to stop.

Definition 25 *Let s, t be traces. A letter x is compatible with s in t if*

1. *x is regular in t.*
2. *If x_a occurs in t (i.e. x_a occurs in some branch of t), then $a \in Acc(s)$.*
3. *If a is unfilled in s, then $t = t \downarrow x_a$.*

Comment and examples

1. Note that x is compatible with s in t iff the following conditions are satisfied for every branch $w = Br(t, b)$:
   - x is regular in w
   - If x_a occurs in w, then $a \in Acc(s)$.
   - If x_a occurs in w then b is finite (and thus $b \in Acc(t)$), unfilled in t and x_a occurs only in w as the final token of $\lambda(b)$ (where $\lambda = lab(t)$).
2. Let $t = (e, \lambda)$. Then x is compatible with t in $e[x]$.
3. Let $t = (\perp, \sum_{k \geq 0} x_k)$. Then x compatible with (S^ω, λ) in t but x is not compatible with $(S^n(0), \lambda)$ in t for any n.
4. x is not compatible with $s = (0, \lambda)$ in $t = (\perp, x_\varepsilon\ x_{[1]})$ because $[1] \notin Acc(s)$. x is not compatible with $s = (S(\perp), \lambda)$ neither in $t = (0, x_\varepsilon\ x_{[1]})$ nor in $t' = (\perp, x_0\ x_{[1]}\ x_{[1]})$ because $t \neq t \downarrow x_{[1]}$ and $t' \neq t' \downarrow x_{[1]}$.

Proposition 26 *Let e be an element, λ_1 and λ_2 be labelling functions for e. A letter x is compatible with (e, λ_1) in t iff x is compatible with (e, λ_2) in t.*

Proof. Immediate. ■

Proposition 27 *1. x is compatible with s in t iff x is compatible with s in every finite approximation of t.*

2. Let $(t_k), (s_k)$ be increasing sequences of traces. Let $t = Sup\ t_k$ and $s = Sup\ s_k$. Assume that, for each k, x is compatible with s_k in t_k. Then x is compatible with s in t.

Proof. (1) is immediate. (2) Let $w = Br(t,b)$ be a branch in t and $b_k = Sup\{b' \ / \ b' \leq b \text{ and } b' \in Acc(t_k)\}$. It is clear that b_k is an addressing branch in t_k and $w = Sup\ w_k$ where $w_k = Br(t_k, b_k)$. The regularity of each letter in w follows immediately. Assume x_a occurs in w. Then, for some k, it occurs in w_k. Thus a is accessible in s_k and then in s. Assume that a is unfilled in s. Then, for some k_0, a is unfilled in s_k for each $k \geq k_0$. Since x is compatible with s_k in t_k, $t_k = t_k \downarrow x_a$. Thus, by lemma 20, $t = t \downarrow x_a$. ■

Proposition 28 *Assume x is compatible with s in each of the $t_1, ..., t_n$ and f is a prc. Then x is compatible with s in $[[f]](t_1, ..., t_n)$.*

Proof. By induction on f. The only non-trivial case is when f is defined by recursion. For the simplicity of notations let t be the first argument and $\overrightarrow{\rho}$ be the sequence of parameters. The regularity of x has been proved in proposition 24. The other conditions are proved as follows.

1. Assume first t is finite. The clauses (2) and (3) in definition 25 are proved by induction on $Val(t)$.

   For $t = (\perp, w)$, this is clear. Otherwise, assume $t = \langle (c,w)\ r_1, ..., r_p \rangle$. Then $[[f]](t, \overrightarrow{\rho}) = w + [[h]]([[f]](r_1, \overrightarrow{\rho}), ..., [[f]](r_m, \overrightarrow{\rho}), r_1, ..., r_p, \overrightarrow{\rho})$.

   - Assume x_a occurs in w. Since x is compatible with s in t, clearly $a \in Acc(s)$. The cell a may not be unfilled in s, because otherwise clause (3) in definition 25 (for the compatibility of x with s in t) would imply $t = (\perp, w)$.
   - Otherwise the result follows immediately from the induction hypothesis and proposition 21.

2. Otherwise, the result follows from proposition 27 and the continuity of $[[f]]$. ■

Corollary 29 *For each i, x^i is compatible with $e_i[x^i]$ in $[[f]](e_1[x^1], ..., e_n[x^n])$.*

3.5 Substitutions

The notion of composition is crucial when functions are studied but, usually, only the *results* are, in some sense, composed. The notion of traces allows to compose also the *computations*. The precise meaning of this is given in theorem 34 which, as already mentionned, is the key point of this section. It needs the notion of substitutions.

Definition 30 *Let t be a trace, $(s_i) = (e_i, \lambda_i)$ be a sequence of traces and (x^i) be a sequence of distinct letters. Assume that, for each i, x^i is compatible with s_i in t. Then $t[x^i := s_i / i = 1, ..., n]$ is the trace obtained by simultaneously replacing each $(x^i)_a$ by $\lambda_i(a)$ in all the words $\lambda(c)$ for $c \in Acc(t)$.*

Comment and examples

1. Note that, to be able to make substitutions, the clause (2) in the definition of compatibility would be enough but, since in the sections 4 and 5, I will make substitutions only when the other clauses are also satisfied I consider only this restrictive situation.

2. I may define in the same way $w[x^i := s_i / i = 1, ..., n]$ if w is a branch of a trace.

3. Let $t = (e, \lambda)$. Then $t = e[x][x := t]$.

Proposition 31 *Let x be compatible with s in t.*

1. $Val(t[x := s]) = Val(t)$.
2. *b is an addressing branch for t iff b is an addressing branch for $t[x := s]$. Moreover $Br(t[x := s], b) = Br(t, b)[x := s]$*

Proof. Immediate. ■

Proposition 32 *Let (t_k), (s_k) be increasing sequence of traces. Let $t = Sup\ t_k$ and $s = Sup\ s_k$. Assume that, for each k, x is compatible with s_k in t_k. Then $t[x := s] = Sup\ t_k[x := s_k]$.*

Proof. Remember that, by proposition 27, x is compatible with s in t. Assume $a \in Acc(t_{k_0})$. I must prove that $\lambda_{t[x:=s]}(a) = Sup\{\lambda_{t_k[x:=s_k]}(a) \ / \ k \geq k_0\}$. It is clearly enough to prove that, for $k \geq k_0$, if x_b occurs in $\lambda_{t_k}(a)$ and x_b is not the final token in $\lambda_{t_k}(a)$, then $\lambda_{s_k}(b) = \lambda_s(b)$. The cell b is filled in s_k because otherwise, since x is compatible with s_k in t_k, x_b would be the final token in $\lambda_{t_k}(a)$ and thus the result follows from the definition of the ordering on traces. ■

Proposition 33 *Let r, s, t be traces. Assume x is compatible with s in t. Then*

1. *If y is regular both in t and s, then y is regular in $t[x := s]$.*
2. *If y is compatible with r both in t and s, then y is compatible with r in $t[x := s]$.*

Proof.

(1) Let $t' = t[x := s]$. Let b be an addressing branch for t, $u = Br(t, b)$ and $u' = Br(t', b)$. Let $a < c$ and assume y_c occurs in u'. When the least occurrence of y_c comes from u the result is clear. Otherwise, it comes from the substitution of some $x_{a'}$ by $\lambda(a')$ where $\lambda = lab(s)$. Since y is regular in s, y_a occurs in $\lambda(b')$ for some $b' \leq a'$. Since x is regular in u, $x_{b'}$ occurs in u before $x_{a'}$ and so y_a occurs in u' before y_c.

(2) The regularity of y is proved in (1). Assume y_a occurs in t'. Then it comes either from t or from the substitution of some x_c. Since y is compatible with r both in s and t, a is accessible in s. Assume finally that a is unfilled in r. Thus, $t = t \downarrow y_a$ and $s = s \downarrow y_a$. By proposition 32 and lemma 20 it is enough to prove that $t' = t' \downarrow y_a$ for finite t. This is proved by induction on $Val(t)$.

- $t = (\bot, w)$. Then $t' = (\bot, w[x := s])$.
  - Assume y_a does not occur in $w[x := s]$. The result is clear.
  - Assume the least occurrence of y_a in $w[x := s]$ comes from w. Since $t = t \downarrow y_a$, $w = w \downarrow y_a$ and the result follows.
  - Assume the least occurrence of y_a in $w[x := s]$ comes from the substitution of x_b by $\lambda_s(b)$. Since y is compatible with r in s, b is unfilled in s. Since x is compatible with s in $t, w = w \downarrow x_b$ and again the result follows.
- $t = \langle (c, w)\ t_1,\ ...,\ t_n \rangle$. Then $t' = \langle (c, w[x := s])\ t_1[x := s],\ ...,\ t_n[x := s] \rangle$. *Claim* : y_a does not occur in $w[x := s]$. *Proof* : Since $t = t \downarrow y_a$, y_a does not occur in w. Let b be an address and assume x_b occurs in w. Then y_a does not occur in $\lambda_s(b)$: Otherwise, since y is compatible with r in s, b would be unfilled in s. Since x is compatible with s in t, this contradicts the fact that ε is filled in t. *End of proof.*

  Thus $t' \downarrow y_a = \langle (c, w[x := s])\ t_1[x := s] \downarrow y_a,\ ...,\ t_n[x := s] \downarrow y_a \rangle$. The result follows then from the induction hypothesis. ■

Theorem 34 *Let f be a prc, $t_1, ..., t_n$ be traces and, for each i, let r_i be the named element (with the fresh name x^i) such that $Val(t_i) = Val(r_i)$. Then x^i is compatible with t_i in $[[f]](r_1, ..., r_n)$ and $[[f]](t_1, ..., t_n) = [[f]](r_1, ..., r_n)[x^i := t_i \ / \ i = 1, ..., n]$.*

Proof. The compatibility comes from corollary 29. The second point is proved by induction on f. The only non-trivial case is when f is defined by recursion. When $Val(t_1)$ is finite, this is done by an immediate induction on its size. Otherwise, this follows by continuity (cf. proposition 32). ■

3.6 Some other results

The results in this subsection are not used in sections 4 and 5 and may be skipped. For the same reason, I do not give a proof of propositions 38, 40 and 41. These proofs are very similar to the ones in the previous subsections.

Proposition 36 is the basic tool in [11] to study the intensional properties of algorithms. Propositions 38 and 40 show that other properties of traces can be considered.

3.6.1 Intensionality

In the next definition and proposition I assume that the type of f and g are $N^k \to N$ but they may use other auxiliary data types in their definition.

Definition 35 *The prc f and g are (strongly) intensionally equivalent iff $[[f]](S^{n_1}(\bot)[x^1], ..., S^{n_k}(\bot)[x^k]) = [[g]](S^{n_1}(\bot)[x^1], ..., S^{n_k}(\bot)[x^k])$ for every sequence $n_1, ..., n_k$ of integers and distinct letters $x^1, ..., x^n$.*

Proposition 36 *The prc f and g are (strongly) intensionally equivalent iff $[[f]](S^\omega[x^1], ..., S^\omega[x^k]) = [[g]](S^\omega[x^1], ..., S^\omega[x^k])$.*

Proof. (if) This follows from the fact that $S^\omega[x] \downarrow x_n = S^n(\bot)[x]$ and proposition 21. (only if) This follows from the fact that $S^\omega[x] = Sup\ S^n(\bot)[x]$ and the continuity of $[[f]]$ and $[[g]]$. ■

3.6.2 Normal traces

The "useful" traces (i.e. the image by some *prc* of named elements) have additional properties. Here are two examples :

- Let $t = (e, \lambda)$. For $a \in Acc(e) - dom(e)$, the word $\lambda(a)$ is non-empty. This means that, if the algorithm cannot find the content of the node a in e, this is only because of a lack of information on some input and thus $\lambda(a)$ must contain some x_b where b is an unfilled cell in some argument.
- A computation may not be infinite if, intuitively speaking, it does not examine "from time to time" some cells.

The next definition formalizes these properties and proposition 38 states the desired result.

Definition 37 *Let T^* be the set of* normal *traces $t = (e, \lambda)$, i.e. such that for all addresses a:*

1. *If a is unfilled in t, then $\lambda(a)$ is non-empty.*

2. *If a is an infinite addressing branch for t, then there are infinitely many places along a at which the word is non-empty, i.e. the set $\{n \ / \ \lambda(a \uparrow n)$ is non-empty$\}$ is infinite.*

Comment and examples

Note that a trace may have infinitely many cells a such that $\lambda(a) = \emptyset$. For example, define *double* by : $double(0) = 0$ and $double(Sx) = SSdouble(x)$. It is easy to check that $[[double]](S^\omega[x]) = (S, x_0)(S, \emptyset)(S, x_1)(S, \emptyset)(S, x_2)(S, \emptyset) \ldots$.

Proposition 38 *Let f be a prc and $t_1, ..., t_n$ be in T^*. Then, $[[f]](t_1, ..., t_n) \in T^*$.*

3.6.3 Index of sequentiality

Using the traces, there is no need to express the sequentiality in the usual way, since the trace itself codes, in some sense, the sequentiality. This can however be done.

Definition 39 *Let f be a prc and $t_j = (e_j, \lambda_j)$ be normal traces. Assume the address a is unfilled in $(e, \lambda) = [[f]](t_1, \ ..., \ t_n)$ and $\lambda(a)$ is finite. A pair (i, b) is an index of sequentiality of (e, λ) at the address a if*

1. *b is unfilled in t_i.*
2. *The final token of $\lambda(a)$ is the final token of $\lambda_i(b)$.*
3. *For all sequences of normal traces $t'_j = (e'_j, \lambda'_j)$ such that, for each j, $t'_j \geq t_j$ and letting $(e', \lambda') = [[f]](t'_1, \ ..., \ t'_n)$:*
   (a) *Assume $\lambda'_i(b) > \lambda_i(b)$. Then $\lambda'(a) > \lambda(a)$.*
   (b) *Assume $\lambda'_i(b) = \lambda_i(b)$. Then $\lambda'(a) = \lambda(a)$.*

Proposition 40 *Let f be a prc and $t_1, ..., t_n$ be in T^*. Assume a is unfilled in $(e, \lambda) = [[f]](t_1, \ ..., \ t_n)$ and $\lambda(a)$ is finite. Then, there is an index of sequentiality for (e, λ) at the address a.*

3.6.4 A stronger notion of continuity

The next proposition shows that $[[f]]$ is continuous in a stronger sense than preserving the Sup : it also preserves the *limit*.

Proposition 41 *Let f be a prc and (t^1_n), ..., (t^k_n) be sequences of traces. Assume that $Lim(t^i_n) = t^i$ for $i = 1, \ ..., k$. Then $Lim([[f]](t^1_n, \ ..., t^k_n)) = [[f]](t^1, ..., t^k)$.*

4 The ultimate obstination theorem

This section is devoted to the proof of theorem 13 and its consequences in terms of complexity (theorems 48 and 49). Recall that the only data type to be used is N and that, in this case, x_{1^i} is written for simplicity x_i. Also recall that a trace t over N has only one branch that I denote by $Br(t)$.

In this section I allow the use of mutual recursion in the definition of a *prc*. The extension of the definitions and the properties given in section 2 and 3 are immediate.

The proof of theorem 13 is by induction on f. The only non-trivial case is when f is defined by recursion. The idea is the following : Let $r = S^\omega[x] = \langle (S, x_0) \ r_1 \rangle$ and $t = [[f]](r, \overrightarrow{\sigma}) = x_0 + [[h]](r_1, t_1, \overrightarrow{\sigma})$ where $t_1 = [[f]](r_1, \overrightarrow{\sigma})$. Let $s = x_0 + [[h]](r_1, e[y], \overrightarrow{\sigma})$ where $e = Val(t_1)$. By theorem 34, $t = s[y := t_1]$. Since r_1 is the same as r where x is lifted (i.e. x_j is replaced by x_{j+1}), t_1 is the same as t where x

is lifted. By the induction hypothesis s is ultimately obstinate. The difficult case is when the (unique) unbounded letter in s is y. Since the other letters are used in t and t_1 in the same way, it is not difficult to show that the only possibly unbounded letter is x. Proposition 45 makes this argument precise.

4.1 Some preliminary results

Propositions 43 and 44 show that the ultimate obstination is preserved by substitution.

Definition 42 *Let t be a trace.*

1. *t finishes with the letter x if the final token (if any) of $Br(t)$ is some x_k.*
2. *$t\langle x+k\rangle$ is the trace obtained from t by replacing, in the labelling of t, x_j by x_{j+k} for each j.*

Comment and examples

Let $t = S^\omega[x]$, then $t = \langle (S, x_0)\ t\langle x+1\rangle\rangle$. Note that, for all traces, $Val(t\langle x+n\rangle) = Val(t)$.

Proposition 43 *Let t and s be traces and x be a letter. Assume that :*

1. *x is compatible with s in t.*
2. *x is bounded in t and t does not finish with x.*
3. *t is ultimately obstinate.*

Then $t[x := s]$ also is ultimately obstinate.

Proof. Since x is bounded in t, the tokens introduced by the substitution come from the finite set of words $\{\lambda_s(c)\ /\ x_c$ occurs in $Br(t)\}$. But $\lambda_s(c)$ may be infinite only if c is unfilled in s. Since x is compatible with s and t does not finish with x, if x_c occurs in $Br(t)$, c may not be unfilled in s and thus $\lambda_s(c)$ is finite. Thus, the substitution introduces only a finite set of new tokens. ■

Proposition 44 *Let t, s be traces and x be a letter. Assume that :*

1. *x is compatible with s in t.*
2. *s and t are ultimately obstinate.*

then $t[x := s]$ also is ultimately obstinate.

Proof. By case analysis.
- $Br(t)$ is finite and does not finish with x : By proposition 43.
- t finishes with x and $Br(t[x := s])$ is infinite : There is a final segment of $Br(s)$ which is a final segment of $Br(t[x := s])$ and the result follows from the fact that s is ultimately obstinate.
- x is unbounded in t : The other letters are bounded in t. So, the unbounded letters in $t[x := s]$ come from s and thus there is at most one such letter since s is ultimately obstinate.
- Otherwise : By proposition 43. ■

Proposition 45 *Let t, s be traces, x, y be letters and n be an integer. Assume that :*

1. *y does not occur in t.*

2. *y is compatible with t in s and $t = s[y := t\langle x + n\rangle]$.*

3. *the first token in $Br(s)$ is x_0.*

Define the sequence (s_i) by : $s_0 = s$, $s_{i+1} = s_i[y := s\langle x + n(i+1)\rangle]$. Then

1. *$t = Lim\ (s_i)$.*

2. *Assume moreover that s is ultimately obstinate. Then t also is ultimately obstinate.*

Proof. Since $t = s[y := t\langle x + n\rangle]$, by proposition 31, $Val(t) = Val(s)$ and thus, by proposition 26, the compatibility of y with t and s are equivalent. Similarly, since $Val(t\langle x + n\rangle) = Val(t)$ and $Val(s\langle x + n\rangle) = Val(s)$, the compatibility of y with t and $t\langle x + n\rangle$ (respectively with $s\langle x + n\rangle$) are equivalent.

Claim 1 For all i, $Val(s_i) = Val(s) = Val(t)$. The letter y is compatible with $s\langle x + n(i+1)\rangle$ and t in s_i. In particular, the sequence (s_i) is well-defined.

Proof By induction on i. Use proposition 33.

Claim 2 For all i, $t = s_i[y := t\langle x + n(i+1)\rangle]$.

Proof By induction on i. The case $i = 0$ is trivial . The case $i + 1$ is given below (where $p = n(i+1)$).

$$\begin{aligned}
s_{i+1}[y := t\langle x + p + n\rangle] &= s_i[y := s\langle x + p\rangle][y := t\langle x + p + n\rangle] \\
&= s_i[y := s\langle x + p\rangle[y := t\langle x + p + n\rangle]] \ (*) \\
&= s_i[y := s\langle x + p\rangle[y := t\langle x + n\rangle\langle x + p\rangle]] \\
&= s_i[y := s[y := t\langle x + n\rangle]\langle x + p\rangle] \\
&= s_i[y := t\langle x + p\rangle] \\
&= t
\end{aligned}$$

(*) because the only occurrences of y in $s_i[y := s\langle x+p\rangle]$ are coming from $s\langle x+p\rangle$.

Claim 3 The first token of $Br(s\langle x + j\rangle)$ is x_j.

Proof Immediate.

Claim 4 For all i, k, if y does not occur in $Br(s_i) \uparrow k$ then $Br(s_i) \uparrow k = Br(t) \uparrow k$.

Proof Immediate from claim 2. Recall that $w \uparrow k$ is the prefix of w of length p.

Proof of (1). By the claim 4, I have to prove that for each k, y does not occur in $Br(s_i) \uparrow k$ for i large enough. This is done by an easy induction on k : since y is regular in s_i (because y is compatible with s_i), the first occurrence of y_0 in s_i is substituted (to get s_{i+1}) by an initial segment of $s\langle x + n(i+1)\rangle$, i.e. a word whose first token is $x_{n(i+1)}$.

Proof of (2).

- Assume every letter $z \neq x, y$ is bounded in s. Let m be a bound for z in s. It is easy to check, by induction on i, that z is also bounded by m in each s_i and thus also in t. Thus, the only letter that may be unbounded in t is x.

- Assume some letter $z \neq x, y$ is unbounded in s. Since s is ultimately obstinate, y is bounded in s and (since it follows from the assumption that s is infinite) s does not finish with y. The result follows then from proposition 43. ■

4.2 Proof of theorem 13

By induction on f. The only non-trivial case is when f is defined by recursion. By theorem 34 and proposition 44, I may assume that the arguments are named elements. Let r be the recursive argument and $\overrightarrow{\sigma}$ be the sequence of parameters. For finite r, the result is easily proved by induction on its size. Assume then that $r = S^{\omega}[x]$ and let $r_i = r\langle x+i\rangle$.

For a better understanding, I first give the proof when mutual recursion is not allowed and then, the general case.

(1) Assume the recursive equation for f is $f(Sn, \overrightarrow{m}) = h(n, f(n, \overrightarrow{m}), \overrightarrow{m})$. Then $t = [[f]](r, \overrightarrow{\sigma}) = x_0 + [[h]](r_1, [[f]](r_1, \overrightarrow{\sigma}), \overrightarrow{\sigma})$. Let $s = x_0 + [[h]](r_1, e[y], \overrightarrow{\sigma})$ where $e = Val([[f]](r_1, \overrightarrow{\sigma}))$ and y is a fresh letter. Then, by theorem 34, $t = s[y := [[f]](r_1, \overrightarrow{\sigma})]$. Clearly $[[f]](r_1, \overrightarrow{\sigma}) = t\langle x+1\rangle$ and thus $t = s[y := t\langle x+1\rangle]$. By the induction hypothesis s is ultimately obstinate. The result follows then from proposition 45.

(2) Assume $f_1, ..., f_k$ are defined by mutual recursion and $f_j(Sn, \overrightarrow{m}) = h_j(n, f_1(n, \overrightarrow{m}), ..., f_k(n, \overrightarrow{m}), \overrightarrow{m})$. Let $v_j = e_j[z^j]$ where $e_j = Val([[f_j]](r_1, \overrightarrow{\sigma}))$ and z^j is a fresh letter. Let $\tau_j = x_0 + [[h_j]](r_1, \overrightarrow{v}, \overrightarrow{\sigma})$ and $t_j = [[f_j]](r, \overrightarrow{\sigma})$. By the induction hypothesis, the τ_j are ultimately obstinate. By theorem 34, $t_j = \tau_j[z^i := [[f_i]](r_1, \overrightarrow{\sigma}) \ / \ i = 1, \ ..., \ k]$. By proposition 43, the only cases where it is not clear that t_j is ultimately obstinate are those where τ_j is infinite and the only unbounded letter is one of the z^i, or when it is finite and it finishes by some z^i. In such a case say that f_j recursively calls f_i.

We have to prove that t_1 is ultimately obstinate.

(2.1) Assume first that f_1 recursively calls f_1. Let $\rho_1 = \tau_1[z^i := [[f_i]](r_1, \overrightarrow{\sigma}) \ / \ i \neq 1]$.

Claim 5 ρ_1 is ultimately obstinate.

Proof Since f_1 recursively calls f_1 either τ_1 is infinite and then z^2, ..., z^k are bounded in τ_1 or τ_1 is finite and does not finish with z^2, ..., z^k. In both cases the result follows from proposition 43. (*End of proof of claim 5)*

Since $t_1 = \rho_1[z^1 := t_1\langle x+1\rangle]$ the result follows from proposition 45.

(2.2) Assume f_1 recursively calls, say f_2. Let $\rho_1 = \tau_1[z^i := [[f_i]](r_1, \overrightarrow{\sigma}) \ / \ i \neq 2]$. By the same argument as in claim 5, ρ_1 is ultimately obstinate. Since $t_1 = \rho_1[z^2 := t_2\langle x+1\rangle]$ it is enough (by theorem 34 and proposition 44) to show that t_2 is ultimately obstinate. When f_2 recursively calls f_2, the same argument as in (2.1) gives the result. Otherwise, by repeating the argument, we get a cycle, say of length n : f_1 recursively calls f_2, ..., that recursively calls f_n, that recursively calls f_1. The following claim finishes the proof.

Claim 6 For each $j = 1, \ ..., \ n$ there is a trace s_j using only the letters x, z^j and the letters in $\overrightarrow{\sigma}$ such that the hypotheses of proposition 45 are satisfied with $t = [[f_j]](r, \overrightarrow{\sigma})$, $s = s_j$ and $y = z^j$.

Proof For the simplicity of notations, I assume that $n = 2$.

Since τ_1 is ultimately obstinate and f_1 recursively calls f_2, z^1 is bounded in τ_1 and τ_1 does not finish with z^1. Similarly for τ_2. Let $\rho_1 = \tau_1[z_1 := t_1\langle x+1\rangle]$ and $\rho_2 = \tau_2[z^2 := t_2\langle x+1\rangle]$. Then, z^i does not occur in ρ_i and $t_1 = \rho_1[z^2 := t_2\langle x+1\rangle]$, $t_2 = \rho_2[z^1 := t_1\langle x+1\rangle]$. By proposition 43, ρ_1 and ρ_2 are ultimately obstinate. Let $s_1 = \rho_1[z^2 := \rho_2\langle x+1\rangle]$ and $s_2 = \rho_2[z^1 := \rho_1\langle x+1\rangle]$. It is clear that : s_1 and s_2 are ultimately obstinate (by proposition 44), the first token of s_i is x_0, z^i does not occur in t_i , z^1 does not occur in s_2, z^2 does not occur in s_1, z^i is compatible with t_i in s_i (by proposition 33). Thus, it remains to show that $t_1 = s_1[z^1 := t_1\langle x+2\rangle]$ (the proof is similar for t_2).

$$\begin{aligned} s_1[z^1 := t_1\langle x+2\rangle] &= \rho_1[z^2 := \rho_2\langle x+1\rangle][z^1 := t_1\langle x+2\rangle] \\ &= \rho_1[z^2 := \rho_2\langle x+1\rangle[z^1 := t_1\langle x+2\rangle]] \ (*) \\ &= \rho_1[z^2 := \rho_2[z^1 := t_1\langle x+1\rangle]\langle x+1\rangle] \\ &= \rho_1[z^2 := t_2\langle x+1\rangle] \text{ because } t_2 = \rho_2[z^1 := t_1\langle x+1\rangle] \\ &= t_1 \end{aligned}$$

(*) because z^1 does not occur in ρ_1. ■

4.3 Complexity results

Definition 46 *Let f be a prc. The computation time of f is the function defined by : $time(n_1, ..., n_k) = lg(Br([[f]](S^{n_1}(0)[x^1], ..., S^{n_k}(0)[x^k])))$ where x^1, ..., x^k are distinct letters.*

In [3] the computation time of f is defined as the number of reductions in call by name strategy. It is not difficult to check that the time defined here is smaller than the one in [3]. This is due to the fact that I only count the reductions corresponding to redexes where the symbols S and 0 "come from" the named arguments and not those where these symbols are created by previous reductions. For example, assume *add* is defined by : $add(0, y) = y, add(Sx, y) = S\ add(x, y)$ and *double* is defined by : $double(0) = 0, double(Sx) = SSdouble(x)$. Let $f(x, y) = add(double(x), y)$. It is easy to check that the time function for f, as defined in [3], is (approximately) $time(n, p) = 2n + p$ whereas $lg(Br([[f]](S^n(0)[x], S^p(0)[y]))) = n + p$.

However, to prove the complexity result for the *inf* function, [3] shows that the time complexity is at least ... the time I defined here and thus, even though my result seems to be stronger than Colson's result, it is actually the same.

In order to prove theorems 48 and 49, I first need the following proposition. It essentially says that if a cell is not used in the computation of $f(e)$ and e and e' coincide on the path up to this address then $[[f]](e[x]) = [[f]](e'[x])$.

Proposition 47 *Let f be a prc, $r = e[x], s = e'[x]$ and $\overrightarrow{t}$ be a sequence of elements of N with names distinct from x. Assume j is accessible both in e and e'. Then*

1. $[[f]](r, \overrightarrow{t}) \downarrow x_j = [[f]](S^j(\bot)[x], \overrightarrow{t})$
2. *Assume x_j does not occur in $[[f]](r, \overrightarrow{t})$. Then $[[f]](r, \overrightarrow{t}) = [[f]](s, \overrightarrow{t})$*
3. *Assume x_j does not occur in $Br([[f]](r, \overrightarrow{t})) \uparrow p$. Then $Br([[f]](r, \overrightarrow{t})) \uparrow p = Br([[f]](s, \overrightarrow{t})) \uparrow p$.*

Proof.

1. This follows immediately from proposition 21 and the fact that $r \downarrow x_j = S^j(\bot)[x]$.
2. $[[f](r, \overrightarrow{t}) = [[f]](r, \overrightarrow{t}) \underset{\downarrow}{} x_j = [[f]](S^j(\bot)[x], \overrightarrow{t}) = [[f]](s \downarrow x_j, \overrightarrow{t}) = [[f]](s, \overrightarrow{t}) \downarrow x_j = [[f]](s, \overrightarrow{t})$.
3. $Br([[f]](r, \overrightarrow{t})) \uparrow p = \{Br([[f]](r, \overrightarrow{t})) \downarrow x_j\} \uparrow p = \{Br([[f]](s, \overrightarrow{t})) \downarrow x_j\} \uparrow p = Br([[f]](s, \overrightarrow{t})) \uparrow p$. ■

Theorem 48 *There is no prc (even using mutual recursion) that computes the inf of two integers in time a function of this inf. In particular there is no prc computing the inf function in time $O(inf)$.*

Proof. Otherwise, assume f is a *prc* computing the *inf* function in time $\phi(inf)$. Let $t = [[f]](S^\omega[x], S^\omega[y])$ and w be the branch of t. The letters x and y cannot both be bounded in w : Otherwise, by proposition 47, for m, n large enough

$[[f]](S^m(0)[x], S^n(0)[y]) = t$ and thus, $f(S^m(0), S^n(0)) = Val(t)$. Hence f does not compute the *inf* function.

Thus, by theorem 13, there is exactly one unbounded letter, say x, in w. Let n be a bound for the indexes of y in w. Then (by proposition 47) $[[f]](S^\omega[x], S^n(0)[y]) = t$. Let $m = max\{\phi(n),\ n\}$. Assume that the first occurrence of x_{m+1} is the p-th token of w (since x is unbounded and regular in w, x_{m+1} does occur in w). By proposition 47, $w \uparrow p-1 = Br([[f]](S^m(0)[x], S^n(0)[y])) \uparrow p-1$. But the number of tokens in $w \uparrow p-1$ is at least $m+1$ because x is regular in t and thus x_0, ..., x_m occur in $w \uparrow p-1$. Thus the number of tokens in $Br([[f]](S^m(0)[x], S^n(0)[y]))$ is at least $m+1$ and the time to compute $f(S^m(0), S^n(0))$ is larger than m, a contradiction. ■

Theorem 48 corresponds to a computation where the rewriting strategy is call by name. The result remains true for any strategy, as the next theorem states (this result also is in [3]).

Theorem 49 *Let f be a prc computing the function inf. Let ϕ be any function. Then, there are integers n and m such that the number of reductions made to get the* normal form *of $f(S^m(0), S^n(0))$ (no matter which strategy is used) is larger than $\phi(inf(n, m))$.*

I only give a sketch of the proof. A complete proof would need a formalization of the rewriting rules on *terms*, i.e. *prc* applied to arguments of the form $S^n(0)$ or $S^n(\perp)$ and the fact that this rewriting satisfies the Church-Rosser property. The idea of the proof is the following : The definition of $[[f]]$ in proposition 9 has been made in correspondence with call by name strategy. It is possible to do the same thing for any other strategy (call $\{f\}$ the corresponding function) and to show that the properties (in particular the preservation of regularity) of $\{f\}$ are the same as those of $[[f]]$. By using the same argument as in the proof of theorem 48, the only additional point is the following : If x_j occurs in $[[f]](S^n(0)[x], S^m(0)[y])$ then x_j also occurs in $\{f\}(S^n(0)[x], S^m(0)[y])$. This is proved as follows. Assume x_j occurs in $[[f]](S^n(0)[x], S^m(0)[y])$. By proposition 47 (since $S^n(0)[x] \downarrow x_j = S^j(\perp)[x]$), the final token in $[[f]](S^j(\perp)[x],\ S^m(0)[y])$ is x_j and thus the normal form of $f(S^j(\perp), S^n(0))$ is $S^k(\perp)$ for some k. Assume x_j does not occur in $\{f\}(S^n(0)[x], S^m(0)[y])$. Then $\{f\}(S^n(0)[x], S^m(0)[y]) = \{f\}(S^j(\perp)[x], S^m(0)[y])$ and the normal form of $f(S^j(\perp), S^n(0))$ should be $S^p(0)$ for some p. This contradicts the Church-Rosser property. ■

5 The backtracking property

This section is devoted to the proof of theorem 16.

5.1 The idea of the proof

The intuition is the following. It is basically, at least at the beginning, the same as the proof of theorem 13. Let $r = S^\omega[x]$ and $s = S^\omega[z]$. I want to prove that $\rho = [[f]](r, s)$ has the BTP. Let r_i be the subtree of r at the address i, i.e. $r_i = \langle (S, x_i)\ r_{i+1} \rangle = (S, x_i)(S, x_{i+1}) \ldots$.

$\rho = \rho_0[y^1 := [[f]](r_1, s)]$ where $\rho_0 = x_0 + [[h]](r_1, t_1, s)$ and t_1 is the named element with fresh name y^1 and value $[[f]](r_1, s)$. Repeat the same thing with $[[f]](r_1, s) = \rho_1[y^2 := [[f]](r_2, s)]$, ... We get $\rho = \rho_{n-1}[y^n := [[f]](r_n, s)]$ where $\rho_{n-1} = \rho_0[y^1 := \rho_1[y^2 := \ldots]\ldots]$. By the induction hypothesis ρ_n has the BTP and thus it remains to analyze the behaviour of the BTP with respect to the fact that $\rho = Sup\ \rho_n$.

When the only data type was N, the situation was very simple for two reasons. (1) In N, a tree has only one branch and thus there is exactly one recursive call (in the example above $[[f]](r_1, s)$). In the general case the number of recursive calls is variable and depends on the node of the tree. (2) In N, the recursive calls are similar : $[[f]](r_1, s) = [[f]](r, s)\langle x + 1\rangle$. In the general case, there is, a priori, no relations between successive recursive calls.

However, when all the data types are allowed, we can do basically the same things and get $\rho = \rho_n[\overrightarrow{Y} := \textit{the recursive calls at depth } n \textit{ in } r]$. By the induction hypothesis ρ_n has the BTP. It remains then to analyze how the BTP is propagated or created in $\rho = Sup\ \rho_n$. This is the role of next subsection. The main point is the following : If the letter z is unbounded in ρ and bounded in each ρ_n, then it must be backtracking in ρ. This is basically because the unboundedness comes from always new copies of s and, since z is regular in ρ, if z_k occurs and comes from a new copy of s then all the $z_{k'}$ for $k' \leq k$ also occur and are new.

5.2 Some preliminary results

In this subsection I examine the behaviour of the backtracking with respect to substitution. Proposition 53 gives the main cases where backtracking is *propagated* by substitution. Proposition 55 shows how a backtracking is *created* by a substitution and 56 shows that the *backtracking property is preserved* by substitution.

We will have to use traces which are not regular. This problem, which already arises in section 3 (see the proof of proposition 24), requires a more complete treatment and a slight extension of regularity is needed.

Definition 50 *Let t be a trace.*

1. *Let b be an address. A letter x is b-regular in a branch w of t if for all addresses $b \leq a \leq a'$, if $x_{a'}$ occurs in w, then x_a also occurs in w and the first occurrence of x_a is earlier than the first occurrence of $x_{a'}$.*
2. *x is b-regular in t if it is b-regular in each branch of t.*
3. *t is quasi-regular if, for each letter x, each branch w of t and each function a from N to N^* there is an n such that x is $a \uparrow n$-regular in w.*

Comment and examples

1. x is regular in t iff it is ε-regular in t. Note that, if x is b-regular in t and $b' \geq b$, then x also is b'-regular in t.
2. Let e be an element of a data type and $a \in Acc(e)$. Let $e_a[x]$ be the subtree of $e[x]$ whose root is at the address a in $e[x]$. Then, if $a \neq \varepsilon$, x is not regular in $e_a[x]$ but it is a-regular.
3. We will have to do (see definition 59) simultaneous substitutions of a-regular traces (for non-fixed a). This is the reason of the use of quasi-regularity.
4. If, for $n \geq n_0$, $x_{a\uparrow n}$ does not occur in a branch w, then x is clearly $a \uparrow n_0$-regular in w. Thus, being quasi-regular is a condition only for the functions a such that x is a-unbounded in w.

Proposition 51 *1. If t_1, ..., t_n are b-regular, then $[[f]](t_1, ..., t_n)$ is b-regular.*

2. Assume x is compatible with s in t and y is b-regular both in s and t. Then y is b-regular in $t[x := s]$.

Proof. As in section 3. ■

Proposition 52 *Let w be a word.*

1. *If x is a-BT in a subword w' of w (i.e. w' is obtained from w by deleting some, possibly infinitely many, tokens), then x is a-BT in w.*
2. *w has the BTP iff there is a final segment of w that has the BTP.*

Proof. Immediate. ■

Proposition 53 *Let r, t be traces such that x is compatible with r in t. Let w be a branch in t, a be an addressing branch for r and c be a function from N to N^*. Assume that :*

1. *Either x is a-unbounded in w and y is c-BT in $Br(r,a)$*
2. *Or x is a-BT in w and y is c-unbounded in $Br(r,a)$.*

Then y is c-BT in $w[x := r]$.

Proof.

1. Since x is regular and a-unbounded in w, $Br(r,a)$ is a subword of $w[x := r]$ and the result follows.
2. Since x is a-BT in w, for each n large enough, the word $\lambda_r(a \uparrow n)$ occurs infinitely many times in $w[x := r]$ and the result follows from the fact that y is c-unbounded in $Br(r,a)$. ■

Definition 54 *Let w be a word and d be a finite or infinite sequence of positive integers. I say that w calls (x,d) if either x is d-unbounded in w or the last token of w is some $x_{d\uparrow n}$.*

Remark Note that, if x is d-unbounded in w, then w is infinite. Also note that, if w calls (x,d) and w is infinite, then x is d-unbounded in w. This will be used without mention in the rest of the paper.

Proposition 55 *Let r, t be traces such that x is compatible with r in t. Let w be a branch in t, c be a function from N to N^* such that y is regular in w and $c \uparrow m$-regular in r. Assume that :*

1. *y is c-bounded in w.*
2. *(y,c) is a BT-counterexample in $w[x := r]$.*

Then there is an addressing branch d for r such that w calls (x, d) and y is c-unbounded in $Br(r,d)$.

Proof. This is proved in the following way : I assume, toward a contradiction, that for each addressing branch d for r such that w calls (x,d), y is c-bounded in $Br(r,d)$ and I show that y is c-BT in $w[x := r]$. Note that this result thus gives a condition to create a backtracking with a substitution.

Let $\lambda = lab(r)$. Denote by $w(E)$, for a set E of addresses, the result of the substitution in w of x_b by $\lambda(b)$ for each $b \in E$. Denote by E_a, for an address a, the set of addresses b such that $b \leq a$ or $b \geq a$. Let $w\{a\} = w(E_a)$. In particular, $w\{\varepsilon\} = w[x := r]$. It is easy to check that, for each address a, y is $c \uparrow m$-regular in $w\{a\}$.

Claim 1 There is an infinite addressing branch d for r such that, for each n, y is c-unbounded in $w\{d \uparrow n\}$ and $x_{d\uparrow n}$ occurs in w.

Proof $d(n)$ is defined by recursion on n, preserving the desired conditions. Note that y is c-unbounded in $w\{\varepsilon\}$ and x_ε occurs in w (otherwise, by the regularity of x in w, $w = w[x := r]$ and this contradicts the hypothesis). Assume $b = d \uparrow n$ is defined.

- b is filled in r : otherwise (because x is compatible with r) x_b would occur only as the final token of w and thus w calls (x, b). By the hypothesis, y is c-bounded in $\lambda(b)$ and, since x_b is the final token of w, also in $w\{b\}$. A contradiction.

- b is not filled in r with a terminal constructor : otherwise, for each $a \leq b$, $\lambda(a)$ is finite and again y would be c-bounded in $w\{b\}$.

Thus b is filled in r with a non-terminal constructor cf of arity p. $E_b = \bigcup_{1 \leq i \leq p} E_{b+i}$ and y is c-unbounded in $w\{b\}$. Thus, for some $1 \leq i \leq p$, x_{b+i} occurs in w and y is c-unbounded in $w\{b+i\}$. $d(n) = i$ satisfies the desired conditions. *(End of proof of claim 1)*

Since w calls (x, d), let $n_0 \geq m$ be such that, if $y_{c\uparrow n}$ occurs in w or in $Br(r, d)$, then $n < n_0$. The next claim finishes the proof.

Claim 2 For each $n \geq n_0$, $y_{c\uparrow n}$ occurs infinitely many times in $w[x := r]$.

Proof Let p be an integer and $n \geq n_0$. We must check that there is an occurrence of $y_{c\uparrow n}$ in $w[x := r]$ after the p-th token. Let n_1 be such that each token in $w[x := r] \uparrow p$ comes either from w or from the substitution of some x_a by $\lambda(a)$ and $lg(a) < n_1$. Since y is c-unbounded and $c \uparrow n$-regular in $w\{d \uparrow n_1\}$, $y_{c\uparrow n}$ occurs in $w\{d \uparrow n_1\}$. Since $n \geq n_0$, an occurrence of $y_{c\uparrow n}$ in $w\{d \uparrow n_1\}$ does not come neither from w nor from d. Then, by the defintion of $w\{d \uparrow n_1\}$, it must come from the substitution of some x_a for $a \geq d \uparrow n_1$. Thus, by the definition of n_1, this occurrence of $y_{c\uparrow n}$ appears in $w[x := r]$ after the p-th token. ■

Proposition 56 *Let r, t be traces such that x is compatible with r in t. Let $w = Br(t,b)$ be a branch in t. Assume that :*

1. *t is regular and r is quasi-regular.*
2. *r and w have the BTP.*

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

Proof. Let $\lambda = lab(r)$. By case analysis.

1. w is finite and its last token is not some x_a : $w[x := r]$ also is finite and thus has the BTP.
2. The last token of w is x_a and $w[x := r]$ is infinite : Then, $\lambda(a)$ is a final segment of $w[x := r]$. The cell a is unfilled in r (since otherwise $\lambda(a)$ is finite and thus $w[x := r]$ is finite). Thus, a is an addressing branch for r. Since the branch $Br(r, a)$ has the BTP, by using proposition 52 (2) twice, $\lambda(a)$ has the BTP and $w[x := r]$ also has the BTP.
3. w is infinite : Assume $(y, a) \neq (z, c)$ are BT-counterexamples for $w[x := r]$. By proposition 55, either y is a-unbounded in w or, for some addressing branch d in r, y is a-unbounded in $Br(r, d)$ and x is d-unbounded in w. Similarly for (z, c). There are thus 4 cases to look at.

   - y is a-unbounded in w and z is c-unbounded in w : This is impossible because, since w has the BTP, y would be a-BT (or z would be c-BT) in w and thus, by proposition 52 (1), in $w[x := r]$.

- y is a-unbounded in w and, for some addressing branch d for r, z is c-unbounded in $Br(r,d)$ and x is d-unbounded in w : Since w has the BTP, either y is a-BT in w and thus in $w[x := r]$ (and this is a contradiction) or x is d-BT in w and thus, by proposition 53, z is c-BT in $w[x := r]$ and this is again a contradiction.
- the symmetrical case for (y,a) and (z,c).
- For some addressing branches d and d' for r, x is d and d'-unbounded in w, y is a-unbounded in $Br(r,d)$ and z is c-unbounded in $Br(r,d')$.
  - Assume $d = d'$. Since $Br(r,d)$ has the BTP, y is a-BT (or z is c-BT) in $Br(r,d)$) and thus, by proposition 53, y would be a-BT (or z would be c-BT) in $w[x := r]$. A contradiction.
  - Assume $d \neq d'$. Since w has the BTP, x is d-BT (or d'-BT) in w and thus, by the proposition 53, y is a-BT (or z is c-BT) in $w[x := r]$. A contradiction. ■

5.3 Proof of theorem 16

By induction on f. The only non-trivial case is when f is defined by recursion. Let r be the recursive argument and $\overrightarrow{\sigma}$ be the sequence of the other arguments. By theorem 34 and proposition 56, I may assume that $\overrightarrow{\sigma}$ are named elements and $r = e[x]$. Let $\rho = [[f]](r, \overrightarrow{\sigma})$ and denote by $y \in \overrightarrow{\sigma}$ the fact that y is the name of some element in $\overrightarrow{\sigma}$ i.e. some σ_i is $e_i[y]$.

Definition 57 *For $a \in Acc(e)$, let r_a be the trace obtained by restricting r to the subtree at address a and $s_a = [[f]](r_a, \overrightarrow{\sigma})$.*

Example Let $r = S^\omega[x]$, then $r_n = (S, x_n)(S, x_{n+1})...$

Definition 58 *The sets A and A_n of addresses, the families $(\tau_a)_{a\in A}$ of traces, $(h_a)_{a\in A}$ of prc, $(t_a)_{a\in A^*}$ of named elements and $(X^a)_{a\in A^*}$ of letters are defined, by induction on $lg(a)$, in the following way :*

1. *$\varepsilon \in A$.*
2. *For $a \in A$, assume the recursive equation concerning the constructor $cf = e(a)$ is : $f(cf(z_1,...,z_{p_a}), \overrightarrow{y}) = h_a(f(z_1, \overrightarrow{y}),...,f(z_{m_a}, \overrightarrow{y}), z_1,...,z_{p_a}, \overrightarrow{y})$. Note that h_a, m_a and p_a depend on the constructor $e(a)$.*
3. *For $a \in A$:*
   - *$a + j \in A$ iff $1 \leq j \leq m_a$.*
   - *For $j = 1,...,m_a$, let t_{a+j} be the element with fresh name X^{a+j} such that $Val(t_{a+j}) = Val(s_{a+j})$.*
   - *Let τ_a be the trace : $x_a + [[h_a]](t_{a+1},...,t_{a+m_a}, r_{a+1},...,r_{a+p_a}, \overrightarrow{\sigma})$.*
4. *Let A_n denote the set $\{a \in A \ / \ lg(a) = n\}$.*

Comment and examples

1. By proposition 9, the clause 2. implies that $s_a = x_a + [[h_a]](s_{a+1},...,s_{a+m_a}, r_{a+1},...,r_{a+p_a}, \overrightarrow{\sigma})$.
2. In the previous definition A^* represents $A - \{\varepsilon\}$, i.e. t_ε and X^ε are not defined ... and not used. Note that in clause 2, m_a may be 0 and that, in this case, no extension of a is in A.

3. A represents the set of recursice calls in $f(e)$. Note that, for $a \in A$, the arguments numbered from 1 to m_a in h_a are recursive arguments of the constructor $e(a)$ and thus have the same type as e (the notion of recursive argument has been given in the notations at the beginning of section 2.1) but m_a may be less than the number of recursive arguments of $e(a)$.

4. Define Δ (the data type of sequences of elements of $\{0,1\}$) by : $\Delta = \{nil : \Delta,$ $s_0 : \Delta \to \Delta,\ s_1 : \Delta \to \Delta\}$. Let e be the infinite sequence $[0,1,0,1,\ ...]$. If f satisfies : $f(s_0(l)) = S\ f(l)$ and $f(s_1(l)) = f(l)$ (e.g. if f computes the number of 0 in a list), then $A = \{n\ /\ n \geq 0\}$ and, for a of even (respectively odd) length, h_a is the successor (respectively the identity) function.

5. Define D (the data type of binary trees whose leaves are labelled by integers) by : $D = \{L\_of\ :\ N \to D,\ T\_of\ : D \times D \to D\}$. Let e be the complete and infinite binary tree, i.e. $Acc(e)$ is the set of finite lists of elements of $\{1,2\}$ and for each a, $e(a) = T\_of$.

   - If f satisfies : $f(T\_of\ (e_1, e_2)) = add(f(e_1), f(e_2))$ (e.g. if f computes the *sum* of the leaves of the tree), then $A = Acc(e)$ and for each $a \in A, h_a$ $= add$ and $m_a = 2$.
   - If f satisfies : $f(T\_of\ (e_1, e_2)) = f(e_1)$ (e.g. if f computes the value of the leftmost *leaf* in the tree), then $A = \{1^n\ /\ n \geq 0\}$ and, for each $a \in A, h_a$ is the identity function and $m_a = 1$ (where as T_of has two recursive arguments).

6. Note that if A is finite the fact that ρ has the BTP follows immediately from the induction hypothesis (by a trivial induction on $Card(A)$). Also note that A_n is finite for each n.

7. If A is infinite, A has, by König's lemma, an infinite branch d and d is an addressing branch for r. The reader might think that (x, d) is the only possible BT-counterexample for ρ. Even if this intuition is mainly correct, the situation is much more complicated ... simply because d may be not used in ρ, i.e. x may be d-bounded in a branch of ρ.

Definition 59 *The sequence (ρ_n) of traces is defined by : Let $\rho_0 = \tau_\varepsilon$. $\rho_{n+1} = \rho_n[X^a := \tau_a\ /\ a \in A_{n+1}]$.*

Remark The fact that the sequence (ρ_n) is well defined follows easily from proposition 33 and the lemma 60 below.

Lemma 60 *Let $a \in A_{n+1}$ and $y \in \overrightarrow{\sigma}$. Then :*

1. *The letter y is regular in τ_a. The letter x is a-regular in τ_a. The traces τ_a and s_a are quasi-regular.*
2. *X^a is compatible with τ_a and s_a in ρ_n.*

Proof. Immediate. ■

Lemma 61 *For each n, ρ_n has the BTP and $\rho = \rho_n[X^a := s_a\ /\ a \in A_{n+1}]$.*

Proof. By the induction hypothesis, τ_a has the BTP. The first point is proved by induction on n (use proposition 56). The second is immediate (use theorem 34). ■

Lemma 62 $\rho = Lim\ (\rho_n)$.

Proof. Let b be an addressing branch in ρ. By lemma 61, it is enough to show that, for each p, $Br(\rho_n, b) \uparrow p$ has, for n large enough, no occurrences of some X^a. This is done by an immediate induction on p, using the fact that the first symbol of s_a is x_a. This point has been more detailed in the proof of proposition 45. ■

Lemma 63 *Let α be an addressing branch in ρ. Let $w = Br(\rho, \alpha)$ and $w_n = Br(\rho_n, \alpha)$.*

1. *Assume $a \in A_{n+1}$ and $a' \geq a$. Each occurrence of $x_{a'}$ in w comes from $w_n[X^a := s_a]$.*
2. *Assume $a \notin A$, $lg(a) = n+1$ and $a' \geq a$. Each occurrence of $x_{a'}$ in w comes from w_n.*

Proof. By lemma 61, $w = w_n[X^c := s_c \ / \ c \in A_{n+1}]$.

1. Assume $x_{a'}$ comes from the substitution of X^c by s_c for $c \neq a$. Then $a' \geq c$, and this is a contradiction since $a' \geq a$, $c \neq a$ and $lg(a) = lg(c)$.

2. Assume $x_{a'}$ comes from the substitution of X^c by s_c for some $c \in A_{n+1}$. Then $a' \geq c$, and this is a contradiction since $a' \geq a$, $c \neq a$ and $lg(a) = lg(c)$. ■

Lemma 64 *Let α be an addressing branch in ρ and c be a function from N to N^*. Assume $y \in \overrightarrow{\sigma}$ is c-bounded in each $w_n = Br(\rho_n, \alpha)$ and c-unbounded in $w = Br(\rho, \alpha)$. Then y is c-BT in w.*

Proof. Let k be an integer. I show that $y_{c\uparrow k}$ occurs infinitely many times in w. Let p be an integer. Since, by lemma 62, $w = Sup\ w_n$ let n be such that $w \uparrow p = w_n \uparrow p$. Let $k_0 > k$ be such that if $y_{c\uparrow k'}$ occurs in w_n, then $k' < k_0$. Since y is c-unbounded in w, there is a $k' \geq k_0$ such that $y_{c\ \uparrow\ k'}$ occurs in w. Let m be the least such that $y_{c\ \uparrow\ k'}$ occurs in w_m. Since $w_m = w_{m-1}[X^a := \tau_a \ / \ a \in A_m]$, $y_{c\ \uparrow\ k'}$ comes from the substitution of some $(X^a)_d$ by $\lambda_{\tau_a}(d)$. By the definition of k_0, $m > n$. By the regularity of X^a in w_{m-1} and the regularity of y in τ_a, $y_{c\uparrow k}$ also has an occurrence in w_m (and thus in w) coming from the substitution in w_{m-1} of some $(X^a)_d$. This occurrence of $y_{c\uparrow k}$ cannot be in $w \uparrow p$. ■

End of the proof the theorem

Let α be an addressing branch for ρ. Let $w = Br(\rho, \alpha)$ and, for each n, $w_n = Br(\rho_n, \alpha)$. Assume w has not the BTP and $(y, c) \neq (z, b)$ be BT-counterexamples for w. I show, by examining the different cases, that this is impossible.

1. $y, z \in \overrightarrow{\sigma}$: By lemma 64, for some n, y must be c-unbounded and z be b-unbounded in w_n. Since w_n has the BTP, y is, for example, c-BT in w_n and thus in w. A contradiction.

2. $z = x$ and $y \in \overrightarrow{\sigma}$: By lemma 64, y is c-unbounded in some w_n. Since w_n has the BTP, x must be b-bounded in w_n. Let $\beta = b \uparrow (n+1)$.

   - $\beta \notin A$: By lemma 63, for $p \geq n+1$, if $x_{b\uparrow p}$ occurs in w, it comes from w_n. A contradiction.
   - $\beta \in A$: By lemma 63, for $p \geq n+1$, if $x_{b\uparrow p}$ occurs in w, it comes from $w_n[X^\beta := s_\beta]$ and thus, x is b-unbounded in s_β.

- $\{a \ / \ (X^\beta)_a$ occurs in $w_n\}$ is finite : The word w_n is infinite (because y is c-unbounded in it) and thus (because X^β is compatible with s_β) each $\lambda_\beta(a)$ substituting $(X^\beta)_a$ is finite. Since x is b-bounded in w_n and only finitely many distinct finite words are substituted, x is b-bounded in $w_n[X^\beta := s_\beta]$. A contradiction.
- $\{a \ / \ (X^\beta)_a$ occurs in $w_n\}$ is infinite : Since X^β is regular in w_n, this set is a finitely branching tree. Thus, by König's lemma, it has an infinite branch d. Since w_n has the BTP, since (y, c) is a BT-counterexample in w_n and X^β is d-unbounded in w_n, X^β is d-BT in w_n and, by proposition 53, x is b-BT in $w_n[X^\beta := s_\beta]$ and thus in w. A contradiction.

3. $y = z = x$: Let n_0 be the least such that $b(n_0) \neq c(n_0)$.

   - For some m, $b \uparrow m \notin A$ and $c \uparrow m \notin A$: By lemma 63, for each $p \geq m$, an occurrence of $x_{b\uparrow p}$ (respectively $x_{c\uparrow p}$) in w, comes from w_m. Thus, x is both b and c-unbounded in w_m. This is a contradiction since w_m has the BTP.
   - For each n, $b \uparrow n \in A$ and $c \uparrow n \in A$: Let $v = w_{n_0}$, $n_1 = n_0 + 1$, $b' = b \uparrow n_1$ and $c' = c \uparrow n_1$. Note that x cannot be both b and c-unbounded in v : Otherwise, since v has the BTP, x would be either b or c-BT in v and thus in w. A contradiction. Thus x is, say, b-bounded in v.

     By lemma 63, for $p \geq n_1$, each occurrence of $x_{b\uparrow p}$ (respectively $x_{c\uparrow p}$) in w comes from $v[X^{b'} := s_{b'}]$ (respectively in $v[X^{c'} := s_{c'}]$. Thus (x, b) (respectively (x, c)) is a BT-counterexample in $v[X^{b'} := s_{b'}]$ (respectively in $v[X^{c'} := s_{c'}]$.
     - x is c-unbounded in v : Then v is infinite and, since x is not c-BT in w, it is not c-BT in v. By proposition 55, there is an addressing branch d for $s_{b'}$ such that $X^{b'}$ is d-unbounded in v and x is b-unbounded in $Br(s_{b'}, d)$. Since v has the BTP and x is not c-BT in v, $X^{b'}$ is d-BT in v and thus, by proposition 53, x is b-BT in $v[X^{b'} := s_{b'}]$ and thus in w. A contradiction.
     - x is c-bounded in v : By proposition 55, there is an addressing branch d for $s_{b'}$ (respectively d' for $s_{c'}$) such that v calls $(X^{b'}, d)$ (respectively v calls $(X^{c'}, d')$). Since a word cannot finish by two distinct tokens and, if it is finite, a letter cannot be unbounded, the only possible case is : $X^{b'}$ is d-unbounded in v, x is b-unbounded in $Br(s_{b'}, d)$, $X^{c'}$ is d'-unbounded in v, x is c-unbounded in $Br(s_{c'}, d')$. This is impossible : Since $(X^{b'}, d) \neq (X^{c'}, d')$ and v has the BTP, $X^{b'}$ for example would be d-BT in v and thus, by proposition 53, x would be b-BT in $v[X^{b'} := s_{b'}]$ and thus in w. A contradiction.
   - For each n, $c \uparrow n \in A$ and for some $n_1 \geq n_0$, $b \uparrow n_1 \notin A$: Let $v = w_{n_1}, n_2 = n_1 + 1$ and $c' = c \uparrow n_2$. By lemma 63, for each $p \geq n_2$, each occurrence of $x_{b\uparrow p}$ in w, comes from v, and thus x is b-unbounded in v.
     - x is c-unbounded in v : Since v has the BTP x would be b or c-BT in v and thus in w. A contradiction.
     - x is c-bounded in v : Since x is c-unbounded in $v[X^{c'} := s_{c'}]$, by proposition 55, there is an addressing branch d for $s_{c'}$ such that $X^{c'}$ is d-unbounded in v and x is c-unbounded in $Br(s_{c'}, d)$. Since v has the BTP and x is not b-BT in v, $X^{c'}$ is d-BT in v and thus, by proposition 53, x is c-BT in $v[X^{c'} := s_{c'}]$ and thus in w. A contradiction. ■

6 Conclusion

The trace is a mathematical representation of the intuitive notion of "the way an algorithm uses its arguments". The intuitive meaning of the main results of this paper is the following :

- The ultimate obstination : A primitive recursive algorithm (even using mutual recursion) cannot use alternatively its arguments.

- The backtracking property : A primitive recursive algorithm (even using any kind of first order data types) cannot alternate without backtracking.

The first property has a consequence in terms of complexity and, though I have no such consequences for the second, the notion of trace and the backtracking property are useful tools to study the behaviour of primitive recursive algorithms (see the forthcoming papers [8] and [11]) because the trace contains a very rich information on the computation. However (at least until now) this information is somehow under-used : In the ultimate obstination we essentially only look at the least occurrence of a token. In the backtracking property we consider a bit more : how many times a token appears.

1) Are there other *intensional* properties of algorithms that can be captured by the notion of trace, i.e. are there other intensional properties for which we can prove the analog of theorems 16 and 13 ?

2) Valarcher conjectures that there is no *prc* computing the *inf* function both in the good time (i.e. $O(inf)$) and in the good way (i.e. $f(S^n(\bot), S^m(\bot) = S^{inf(n,m)}(\bot)$). A much finer analysis (i.e. defining a stronger notion of trace with more information) will probably be necessary.

I give below some questions (in terms of complexity) that could be solved by using this kind of technique.

3) The term given in [9] computes the *inf* in time $O(inf)$ but it is not really a good algorithm because it does not use its arguments in *real time*. A definition of real time could be the following. A *prc* f computes a function (e.g. from N^2 to N) in real time if there is a constant c such that : for each integer i, the length of the subword of $Br([[f]](S^\omega[x], S^\omega[y]))$ between the i-th and the $(i+1)$-th least occurrence of x_i and x_{i+1} is less than c and similarly for y. It is easy to see that the term given in [9] has not a real time computation. I conjecture that there is no *prc* computing the *inf* function in real time.

4) Let f be a *prc* and $t_1, ..., t_n$ be named elements. The ultimate obstination theorem says that, if the only data type to be used is N and $[[f]](t_1, ..., t_n)$ is infinite, there is a *leading* argument : the (unique) unbounded one. The experience seems to show that, even if f uses other data types there is such a leading argument, i.e. an argument that can be somehow distinguished. Is it possible to define a property of $[[f]](t_1, ..., t_n)$ distinguishing a unique argument ? This property is certainly not : being unbounded and not backtracking. Theorem 16 says that such an element, if it exists, is necessarily unique but it is easy to find examples where there is no such element.

References

[1] R. Amadio and PL. Curien. *Domains and Lambda Calculi*. Cambridge University Press, 1998.

[2] G. Berry and P.-L. Curien. Sequential algorithms on concrete data structures. *Theoretical Computer Science*, 20 : 265-321, 1982.

[3] L. Colson. About primitive recursive algorithms. *Theoretical Computer Science*, 83 : 57-69, 1991.

[4] L. Colson. Représentation intentionnelle d'algorithmes dans les systèmes fonctionnels : Une étude de cas. *Thèse de doctorat, Université Paris 7*, 1991.

[5] L. Colson. A unary representation result for system T. *Annals of Mathematics and Artificial Intelligence*, 16 : 385-403, 1996.

[6] L. Colson. A unary representation result for system T. *Mejanne le Clap Act special issue of Theoretical Computer Science*, 1996.

[7] T. Coquand. Une preuve directe du théorème d'ultime obstination. *Comptes Rendus de l'Académie des Sciences*, 314, Srie I, 489-492, 1992.

[8] R. David. Decidability results for primitive recursive algorithms. *In preparation.*

[9] R. David. Un algorithme primitif récursif pour la fonction inf. *Comptes Rendus de l'Académie des Sciences*, 317 (Série I) 899-902, 1993.

[10] R. David. The inf function in the system F. *Theoretical Computer Science*, 135 : 423-431, 1994.

[11] R. David and P. Valarcher. Traces of some primitive recursive schemata. *In preparation.*

[12] M. Hotzel Escardo. On lazy natural numbers with applications. *SIGACT News*, 24(1), 1993.

[13] D. Fredholm. Intensional aspects of function definitions. *PhD Thesis and TCS 152 1-66*, 1995.

[14] D. Fredholm. Computing minimum with primitive recursion over lists. *Theoretical Computer Science*, 163 269-276, 1996.

[15] J.-L. Krivine. Un algorithme non typable dans le systeme F. *Comptes Rendus de l'Académie des Sciences*, 304(5), 1987.

[16] R. Peter. *Recursive Functions.* Academic Press, 1968.

[17] H. Rogers. *Theory of recursive functions and effective computability.* MIT Press, 1988.

[18] P. Valarcher. A complete characterization of intensional behaviours of primitive recursive algorithms. *Rapport de Recherche du LIR 96.11 (To appear in TCS)*, 1996.

[19] P. Valarcher. Contribution à l'étude du comportement intentionnel des algorithmes : le cas de la récursion primitive. *Thèse de doctorat, Université Paris 7*, 1996.

[20] P. Valarcher. Intensionality vs extensionality and primitive recursion. *ASIAN Computing Science Conference - LNCS*, 1179, 1996.

[21] J.E. Vuillemin. *Proof techniques for recursive programs.* PhD thesis, Standford, 1973.