Bottom-up computation of recursive programs


<http://www.numdam.org/item?id=ITA_1976__10_1_47_0>
ABSTRACT. — In this paper, we define and study a mechanism for the implementation of recursive programs: we call it production mechanism by opposition to the usual recursion mechanism. It computes bottom-up, starting from the basic values given by halting conditions, and generates intermediate values leading to the result. We use for this purpose a translation of a recursive program into a system of equations in a space of sets. We introduce determinism conditions providing the uniqueness of the set of intermediate values; we study the structure of this set. As an application, we show the optimality of an implementation of Ackermann program (Rice's algorithm).

INTRODUCTION

When dealing with computer programs, one usually makes a distinction between flow-charts and recursive programs. We restrict here our attention to the latter ones. Their generality has been showed by Luckham, Park and Paterson [4].

Properties of programs have to be expressed in some mathematical framework. Among the possible approaches, one of the most widely used is Scott’s fixpoint semantics: a recursive program is viewed as a set of equations in some function space. The function it computes is defined as the least fixpoint of the (monotonic) functional determined by the equations.

This approach has been showed suitable for studying various properties of programs, such as correctness, termination, equivalence (see Manna-Vuillemin [6], Milner [7], etc.). It has also been used by Vuillemin [14, 15] to study the time complexity of the computations, as defined by the number of substitutions.

On the other hand, in order to develop the “fixpoint induction” technique, Park [10] presents a translation of a recursive program into a system of equations on a space of subsets of a domain. These subsets represent in fact graphs of functions, and the formalism is essentially equivalent to Scott’s one.

In this paper, we want to show that working with graphs is suited to formalise a production process, which is a dual of the usual recursion process. The
recursion mechanism can be considered as defining top-down computations, which start from the given argument to reach the halting conditions. On the other hand, certain "good" iterative programs perform the same set of computations just in the opposite way: they start from the basic values given by the halting conditions, and then produce intermediate values of the function towards the desired result. These programs generally avoid computing several times intermediate values, and use simple storage management. Well-known examples are iterative programs for computing Fibonacci numbers, integer partitions, or Ackermann function (see Rice [11]).

More generally we consider that a given recursion program determines a recursion structure rather than a computational sequence. We study the possible bottom-up computations of the function values according to this recursion structure.

The formalism presented here allows us to define in a precise way the relation \( x < y \) if "\( x \) is an intermediate value in the computation of \( y \)"; and hence the recursion structure. The basic objects we manipulate are nodes in the graphs, which identify to intermediate values and therefore to auxiliary storage cells.

In the first section, we present a way of translating a recursive program into a predicate defining a monotonic production function on graphs; we prove the correctness of the mechanism.

A class a production functions of special interest is the class of those functions for which the set of intermediate values is unambiguously determined for any argument: this can hold either at any computation step (deterministic production functions), or for this set as a whole (quasi-deterministic production functions). The relation \(<\) then becomes a well-founded partial ordering.

Second section is devoted to the study of stable discrete interpretations for programs schemes: we show that the associated production function is then deterministic.

In the third section, we study properties of production functions, independently of the programming aspects. We introduce the notion of self-reproducing set which will allow us to describe in a precise way bottom-up computations and the quasi-determinism condition. We study the structure of these sets.

In the last section, we define the notions of time and space optimality of bottom-up computations. As an example, we show that Rice's algorithm [11] realises an optimal computation of the recursion structure associated to Ackermann program.

Moreover, although this aspect is not developed in the paper, this example should convince the reader that our approach is adequate to a study of space complexity. This last point strengthens the parallel that one guesses between our formalism and that of Jean Vuillemin.
Some easy proofs are omitted, other are only sketched. Complete proofs may be found in [1].

NOTATIONS

Given a set $D$, we denote by $\mathcal{P}(D)$ the powerset of $D$.

The symbol $\subseteq$ will denote:

- when used with lower case letters, Scott's relation “less defined than”;
- when used with upper case letters, the usual set inclusion.

A $n$-tuple $(x_1, x_2, \ldots, x_n)$ is denoted by $x$; its $i$-th component is denoted by $x_i$ or $(x)_i$.

The cardinality of a finite set $X$ is denoted by $\text{card}(X)$.

I. TRANSLATION OF A RECURSIVE PROGRAM INTO A MONOTONIC PRODUCTION FUNCTION

The production mechanism can be illustrated by the Fibonacci program:

$$\text{Fib}(n) = \begin{cases} \text{if } n = 0 \vee n = 1 & \text{then } 1 \\ \text{else } \text{Fib}(n-1) + \text{Fib}(n-2). \end{cases}$$

Given a subset $X$ of the graph of Fib, we can use the program to generate new points in the graph. For instance, if $X$ contains the two points $\langle 4, \text{fib}(4) = 5 \rangle$ and $\langle 5, \text{fib}(5) = 8 \rangle$, we can generate the new point $\langle 6, \text{fib}(6) = 5 + 8 = 13 \rangle$. More precisely, the set produced by $X$ in this example can be defined by

$$\Phi(X) = \{ (n, z) \in \mathbb{N}^2 | \text{if } n = 0 \vee n = 1 \text{ then } z = 1 \\ \text{else } \exists y_1, y_2, z = y_1 + y_2 \wedge (n-1, y_1) \in X \wedge (n-2, y_2) \in X \}.$$  

The purpose of this section is to formalise a translation of a recursive program into a predicate defining the production function $\Phi$. This translation is such that the least fixpoint of $\Phi$ represents the graph of the function computed by the program. It extends Park's translation [10], which corresponds to the call-by-value rule.

I.1. Discrete domains and recursive programs

DEFINITIONS: A discrete domain is a set $D$ containing a distinguished element $\bot$, the undefined element, and ordered by the relation $x \leq y$ iff $x = \bot$ or $x = y$.

The relation $\leq$ extends to $D^p$:

$$\forall x, x' \in D^p, \ x \leq x' \iff \forall i \in \{1, 2, \ldots, p\}, \ x_i \leq x'_i.$$
In $D^p$, two elements $x$ and $x'$ always have a greatest lower bound (g. l. b.) $x \land x'$ defined by $(x \land x')_i = \bot$ if $x_i \neq x'_i$, $(x \land x')_i = x_i$ if $x_i = x'_i$.

If two elements $x$ and $x'$ have a least upper bound (l. u. b.) $x \lor x'$, they are said to be joinable (abbreviated $x \uparrow x'$).

A function $f : D^p \rightarrow D$ is monotonic (increasing), iff for all $x$, $x' \in D^p$, $x \subseteq x' \Rightarrow f(x) \subseteq f(x')$.

The set $\Delta^p$ of the monotonic functions from $D^p$ to $D$ is ordered by the relation $\subseteq$ defined by

$$\forall f, g \in \Delta^p, \quad f \subseteq g \Leftrightarrow \forall x \in D^p, \quad f(x) \subseteq g(x).$$

For each $x$, either $f(x)$ is undefined, or $f(x) = g(x)$.

The structure $(\Delta^p, \subseteq)$ is a complete partial order, whose minimal element is the constant mapping to $\bot$, also denoted by $\bot$.

I.1.1. Proposition: Let $x, x' \in D^p$. Then $x$ and $x'$ are joinable iff for all $i \in \{1, 2, \ldots, p\}$, $x_i \neq \bot$ and $x'_i \neq \bot$ imply $x_i = x'_i$. For all $f \in \Delta^p$, if $x \uparrow x'$, $f(x) \neq \bot$ and $f(x') \neq \bot$, then $f(x) = f(x')$.

Let us now define the recursive program schemes.

The basic alphabet is divided into a set of connectors, a set of variables $V = \{x_1, x_2, \ldots, x_p\}$, a set of basic function symbols $B = \{b_1, b_2, \ldots, b_n\}$, a set of unknown function symbols $F = \{F_1, F_2, \ldots, F_m\}$. To each function symbol $s$ is associated an integer $s(\rho) \geq 0$, the arity of $s$.

Definitions: The set of terms is the language generated by the context-free grammar:

$$\xi = \sum_{v \in V} v + \sum_{b \in B} b(\xi), \ldots, \xi) + \sum_{f \in F} f(\xi), \ldots, \xi).$$

A recursive program scheme $\Sigma$ is a set of equations

$$\sum_{i=0}^N f_i(x_1, x_2, \ldots, x_{\rho(f_i)}) = \tau_i,$$

where the $\tau_i$ are terms only containing variables in $x_1, x_2, \ldots, x_{\rho(f_i)}$, basic function symbols and unknown function symbols in $f_1, f_2, \ldots, f_N$.

A discrete interpretation $I$ is given by:

1) a discrete domain $D_I$;
2) for each basic function symbol $b$, a monotonic function $I(b) \in \Delta^p(b)$.

We call recursive program a pair $(\Sigma, I)$.

Notation: Let $(\Sigma, I)$ be a recursive program with $N$ equations and $k$ variables, let $g_1 \in \Delta^p(f_1), g_2 \in \Delta^p(f_2), \ldots, g_N \in \Delta^p(f_N)$, let $\tau$ be any term.
We denote by $\tau^I$ the functional from $\Delta^p(f_1) \times \Delta^p(f_2) \times \ldots \times \Delta^p(f_n)$ to $\Delta^k_I$ defined by the $\lambda$-expression $\lambda f_1, f_2, \ldots, f_N. \tau$.

Given $\bar{a} \in D^k_I$, $\tau^I(\bar{a})$ denotes the value computed by $\tau$ on $\bar{a}$ when $f_1, f_2, \ldots, f_N$ are interpreted by $g_1, g_2, \ldots, g_N$.

When no confusion arises, $\tau^I$ will be abbreviated in $\tau$.

**Theorem:** A recursive program $(\Sigma, I)$ determines a functional $\varphi$ from $\Delta^p(f_1) \times \Delta^p(f_2) \times \ldots \times \Delta^p(f_n)$ to itself: $\varphi(\bar{g}) = (\tau^I_1(\bar{g}), \tau^I_2(\bar{g}), \ldots, \tau^I_n(\bar{g}))$.

This functional is monotonic and continuous with respect to the ordering $\subseteq$, and has a least fixpoint $7(2, I)$, called the solution of the equations.

This fixpoint is limit of Kleene’s sequence $K_p = < k^1_p, k^2_p, \ldots, k^n_p >$ such that $K_0 = < 1, 1, \ldots, 1 >$ and $K_{p+1} = \varphi(K_p)$ for all $p \geq 0$.

Kleene’s sequence satisfies $K_0 \subseteq K_1 \subseteq \ldots \subseteq K_p \subseteq \ldots \subseteq Y(\Sigma, I)$.

A proof may be found in Nivat [9] or Vuillemin [15].

**I.2. Reduction to one single equation**

In the translation process, we shall prefer to manipulate programs containing a single equation. Given a recursive program, one can easily construct another single equation program performing the same computations, by giving to a generic function the name of the function to be computed as an argument (see [1]).

In this case, functions of Kleene’s sequence will be denoted by $\tau^p(\bot)$.

**I.3. Monotonic functions over subsets**

**Definitions:** $\Phi : \mathcal{P}(A) \to \mathcal{P}(A)$ is monotonic (increasing) iff:

\[ \forall X, Y \subseteq A, X \subseteq Y \Rightarrow \Phi(X) \subseteq \Phi(Y). \]

- $X \subseteq A \Phi$-produces $x$ iff $x \in \Phi(X)$.
- $X$ is a minimal $\Phi$-producer of $x$ iff $X$ produces $x$ and no proper subset of $X$ produces $x$.
- $X$ is a fixpoint of $\Phi$ iff $X = \Phi(X)$.
- $\Phi$ is continuous iff for every increasing sequence

\[ X_0 \subseteq X_1 \subseteq \ldots \subseteq X_n \subseteq \ldots, \Phi(\bigcup_{i \in N} X_i) = \bigcup_{i \in N} \Phi(X_i). \]

**I.3.1. Theorem (Knaster-Tarski):** Every monotonic function $\Phi : \mathcal{P}(A) \to \mathcal{P}(A)$ has a least fixpoint $Y(\Phi)$. If $\Phi$ is continuous, then $Y(\Phi)$ is the limit of the Kleene sequence $\Phi(\emptyset), \Phi^2(\emptyset), \ldots, \Phi^n(\emptyset), \ldots$.

Proof given in Park [10].

n° mars 1976
1.3.2. PROPOSITION : Φ is continuous iff for every point x of Φ(A), all minimal producers of x are finite.

1.4. Translation of a recursive program into a monotonic predicate

According to 1.2, we only translate single equation programs. Let

\[ \Sigma : f(x_1, x_2, \ldots, x_p) = \tau(f)(x_1, x_2, \ldots, x_p) \]

be a recursive scheme with n occurrences of f in \( \tau \). We first label these occurrences by the integers 1, 2, \ldots, n, denoting them by \( f^1, f^2, \ldots, f^n \).

DEFINITIONS : We consider a set \( Y = \{ y_1, y_2, \ldots, y_n \} \) of bound variable symbols, a result variable symbol z, and a set variable symbol X.

We denote by \( T^i = f^i(t^i_1, t^i_2, \ldots, t^i_n) \) the subterm of \( \tau \) corresponding to the occurrence \( i \) of \( f \), and by \( T^0 \) the term \( \tau \) itself.

We define a transformation \( \mathcal{T} \) on labelled terms which consists in replacing a subterm \( f^i(t^i_1, t^i_2, \ldots, t^i_n) \) by the bound variable \( y_i \):

a) \( \mathcal{T}(v) = v \) if \( v \) is a variable symbol.

b) \( \mathcal{T}(b(t_1, t_2, \ldots, t_k)) = b(\mathcal{T}(t_1), \ldots, \mathcal{T}(t_k)) \) if \( b \) is a basic function symbol.

c) \( \mathcal{T}(f^i(t_1, t^i_2, \ldots, t^i_n)) = y_i \).

If \( t \) is a term, \( \mathcal{T}(t) \) only contains basic function symbols and variable symbols in \( x \) or \( y \).

For every integer \( i, 1 \leq i \leq n \), let \( I_i \) denote labels of bound variables that occur in subterms \( \mathcal{T}(t^j) \) for \( 1 \leq j \leq p \). Let \( I_0 \) denote labels of bound variables that occur in \( \mathcal{T}(\tau) \). The \( I_i \) determine a partition of \( \{ 1, 2, \ldots, n \} \); \( I_0 \) is empty for the innermost occurrences of \( f \).

The nesting depth of a label \( i \) is defined by \( nd(i) = 1 \) iff \( I_i = \emptyset \), \( nd(i) = 1 + \max \{ nd(j) \mid j \in I_i \} \) otherwise: \( nd(i) \) is the maximal number of nested occurrences of \( f \) in \( T^i \).

Let us now define the translated predicate \( P_\Sigma \):

DEFINITIONS : For every \( i \in \{ 1, 2, \ldots, n \} \), the elementary predicate \( p_i \) is defined by:

\[ p_i = [(\mathcal{T}(t^i_1)(\overline{x}, \overline{y}), \mathcal{T}(t^i_2)(\overline{x}, \overline{y}), \ldots, \mathcal{T}(t^i_n)(\overline{x}, \overline{y}), y_i) \in X \land y_i = \bot] \]

The predicate \( P_\Sigma \) associated to \( \Sigma \) is defined by

\[ P_\Sigma(x_1, x_2, \ldots, x_p, z, X) = \exists y_1, y_2, \ldots, y_n, (z \neq \bot) \land (z = \mathcal{T}(T^0)) \land p_1 \land p_2 \land \ldots \land p_n. \]
Example: \( \Sigma : f(x_1, x_2) = a(f^1(b(x_1, x_2), c(f^2(x_2, x_1)))) \).

\[
P_{\Sigma}(x_1, x_2, z, X) = \exists y_1, y_2, z \neq \bot \land z = a(y_1) \\
\land [(b(x_1, x_2), c(y_2), y_1) \in X \lor y_1 = \bot] \\
\land [(x_2, x_1, y_2) \in X \lor y = \bot].
\]

We now use \( P_{\Sigma} \) to define the production function \( \Phi_{\Sigma} \):

**Definitions:** Let \( \Sigma \) be a single-equation program scheme, let \( I \) be a discrete interpretation. The production function \( \Phi_{(\Sigma, I)} : D_p^{I+1} \rightarrow D_p^{I+1} \) associated to the program \((\Sigma, I)\) is defined by

\[
\Phi_{(\Sigma, I)}(X) = \{(x_1, x_2, \ldots, x_p, z) \mid P_{\Sigma}(x_1, x_2, \ldots, x_p, z, X) \}.
\]

We abbreviate \( P_{\Sigma} \) and \( \Phi_{(\Sigma, I)} \) in \( P \) and \( \Phi \) when no confusion arises.

**Remark:** If we take out the \( y = \bot \), the translation is essentially equivalent to the one given in Park [10].

1.4.1. Proposition \( \Phi_{(\Sigma, I)} \) is monotonic and continuous.

*Proof:* The elementary predicates are monotonic, and the definition of \( \Phi \) involves no negation: \( \Phi \) is monotonic.

Since the number of elementary predicates is finite, \( \Phi \) is also continuous (1.3.2). \( \square \)

1.5. Correctness of the translation

We first need two definitions:

**Definition:** Let \( b \) be a function from \( D_p \) to \( D \). The defined part of the graph of \( b \), denoted \( dpg(b) \), is the set of \( p + 1 \)-tuples \((x_1, x_2, \ldots, x_p, z)\) such that \( z \neq \bot \) and \( z = b(x_1, x_2, \ldots, x_p) \).

**Definition:** Let \((\Sigma, I)\) be a recursive program. An \( n \)-tuple

\[
\bar{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in D_p^I
\]

is admissible for \((x, z, X)\) iff it satisfies \( z = \mathcal{T}(T^0) \land p_1 \land p_2 \land \ldots \land p_n \) (i.e. \( P_{\Sigma} \) without the quantifiers) as values of \( y_1, y_2, \ldots, y_n \). Then \((x, z) \in \Phi(X)\) iff there exists an admissible \( n \)-tuple for \((x, z, X)\).

We can now prove the correctness of the translation.

1.5.1. Theorem: Let \((\Sigma, I)\) be a recursive program. Let \( h \) be a monotonic function from \( D_p \) to \( D \), let \( H \) denote \( dpg(h) \). Then:

\[
(\bar{\alpha}, z) \in \Phi_{(\Sigma, I)}(H) \iff z \neq \bot \land z = \tau(h)(\bar{\alpha}),
\]

or equivalent

\[
\Phi_{(\Sigma, I)}(H) = dpg(\tau(h)).
\]

n° mars 1976
If a n-tuple $\overline{\alpha}$ is admissible for $(x, z, H)$, then
$$\forall i \in \{1, 2, \ldots, n\}, \quad \alpha_i \subseteq T^i(h)(\overline{x}).$$

Proof: 1) $\text{dpg}(\tau(h)) \subseteq \Phi(H)$.

Assume $z \neq \bot \wedge z = \tau(h)(\overline{x})$. If $\overline{\alpha}$ is defined by $\alpha_i = T^i(h)(\overline{x})$ for all $i$, then by construction of $P_\Sigma$, $\overline{\alpha}$ is admissible for $(x, z, H)$.

2) $\Phi(H) \subseteq \text{dpg}(\tau(h))$.

Let $(\overline{x}, z) \in \Phi(H)$; then $z \neq \bot$. Let $\overline{\alpha}$ be admissible for $(x, z, H)$. We show by induction on the nesting depths that for all $i$, $\alpha_i \subseteq T^i(h)(\overline{x})$.

2 a) Case $\text{nd}(i) = 1$. The terms $\mathcal{T}(t^i_j)(\overline{x}, \overline{\alpha})$ in $p_i$ only contain basic functions and variables of $\overline{x}$.

Hence $\mathcal{T}(t^i_j)(\overline{x}, \overline{\alpha}) = t^i_j(\overline{x})$. By construction of $p_i$, either $\alpha_i = \bot$, or

$$(t^i_1(\overline{x}), t^i_2(\overline{x}), \ldots, t^i_p(\overline{x}), \alpha_i) \in H,$$

which means
$$\alpha_i = h(t^i_1(\overline{x}), t^i_2(\overline{x}), \ldots, t^i_p(\overline{x})) = T^i(h)(\overline{x}).$$

2 b) Assume the property true for all labels of nesting depth less than $k$, and let $i$ be of nesting depth $k$. The terms $\mathcal{T}(t^i_j)(\overline{x}, \overline{\alpha})$ only contain basic functions, variables in $\overline{x}$ and bound variables of nesting depth less than $k$.

Induction hypothesis and monotonicity of the basic functions imply:
$$\forall j, \quad 1 \leq j \leq p, \quad \mathcal{T}(t^i_j)(\overline{x}, \overline{\alpha}) \subseteq t^i_j(\overline{x}).$$

If $\alpha_i \neq \bot$, then
$$(\mathcal{T}(t^i_1)(\overline{x}, \overline{\alpha}), \mathcal{T}(t^i_2)(\overline{x}, \overline{\alpha}), \ldots, \mathcal{T}(t^i_p)(\overline{x}, \overline{\alpha}), \alpha_i) \in H;$$

by monotonicity of $h : (t^i_1(\overline{x}), t^i_2(\overline{x}), \ldots, t^i_p(\overline{x}), \alpha_i) \in H$, which means $\alpha_i = T^i(h)(\overline{x})$.

Same argument applies to $z : z = \bot$ and $z = \mathcal{T}(\tau)(\overline{x}, \overline{\alpha})$ imply $z = \tau(h)(\overline{x}, \overline{\alpha})$. □

I.5.2. Corollary: Let $(\Sigma, I)$ be a recursive program. Then for every $m \geq 0$
$$\Phi^m(\emptyset) = \text{dpg}(\tau^m(\bot)); \quad Y(\Phi) = \text{dpg}(Y(\Sigma, I)).$$

If $(\overline{x}, z) \in \Phi^m(\emptyset)$, and if $\overline{\alpha}$ is admissible for $(x, z, \Phi^{m-1}(\emptyset))$, then for all $i \in \{1, 2, \ldots, n\}$:
$$\alpha_i \subseteq T^i(\tau^{m-1}(\bot))(\overline{x}) \subseteq T^i(Y(\Sigma, I))(\overline{x}).$$

Proof: Follows From I.5.1 by induction on $m$. □

I.6. Extension of the production function

Let $(x, z) \in \Phi^m(\emptyset)$. By monotonicity of $\tau^m(\bot)$, we know that for all $\overline{x}' \supseteq \overline{x}, (\overline{x}', z) \in \Phi^m(\emptyset)$. It may however happen that for some $X, (\overline{x}, z) \in \Phi(X)$

Revue Française d'Automatique, Informatique et Recherche Opérationnelle
and \((\tilde{x}', z) \notin \Phi (X)\). For instance:

\[
\Sigma : \quad f(x_1, x_2) = \begin{cases} 
0 & \text{if } x_1 = 0 \\
x_2 & \text{else}
\end{cases}
\]

Let \(X = \{(0, 1, 0)\}\). Then \((1, 0, 0) \in \Phi (X)\) and \((1, n, 0) \notin \Phi (X)\) for \(n \in N\).
We now extend \(\Phi\) to a more powerful production function \(\Psi\).

**DEFINITIONS:** Let \(D\) be a discrete domain, let \(X \subseteq D^n\). The set \(\hat{X}\) is defined by \(\hat{X} = \{\tilde{x}' \in D^n \mid \exists \tilde{x} \in X, \tilde{x} \subseteq \tilde{x}'\}\). The extended production function \(\Psi\) is defined by: \(\forall X \subseteq D^{n+1}, \Psi (X) = \Phi (\hat{X})\).

**I.6.1. PROPOSITION:** 1) \(\Psi\) is monotonic and continuous

2) \(\forall m \in N, \Psi^m (\emptyset) = \Phi^m (\emptyset); \quad Y (\Psi) = Y (\Phi)\).

**Proof:** 1) Monotonicity follows from \(X \subseteq Y \Rightarrow \hat{X} \subseteq \hat{Y}\), continuity from 1.3.2.

2) By induction on \(n\), using \(\Phi^m (\emptyset) = \Phi^m (\emptyset)\). \(\square\)

**I.6.2. PROPOSITION:** If \((\tilde{x}, z) \in \Psi (X)\) and if \(\tilde{x}' \supseteq \tilde{x}\), then \((\tilde{x}', z) \in \Psi (X)\).
If \(\tilde{a}\) is admissible for \((\tilde{x}, z, \hat{X})\), then \(\tilde{a}\) is also admissible for \((\tilde{x}', z, \hat{X})\).

**Proof:** Let \(\tilde{a}\) be admissible for \((\tilde{x}, z, \hat{X})\). Then for every \(i\) such that \(\alpha_i \neq \bot:\nabla (t^0_i) (\tilde{x}, \tilde{a}), \nabla (t^1_i) (\tilde{x}, \tilde{a}), \ldots, \nu (t^n_i) (\tilde{x}, \tilde{a}), \alpha_i) \in \hat{X}.
\(\nabla (t^0_i) (\tilde{x}, \tilde{a}), \nabla (t^1_i) (\tilde{x}, \tilde{a}), \ldots, \nu (t^n_i) (\tilde{x}, \tilde{a}), \alpha_i) \in \hat{X}.
\)

Now \(z = \nabla (T^0) (\tilde{x}, \tilde{a})\) implies \(z = \nabla (T^0) (\tilde{x}', \tilde{a})\), and \(\tilde{a}\) is admissible for \((\tilde{x}', z, \hat{X})\). \(\square\)

**II. DETERMINISTIC PRODUCTION FUNCTIONS AND STABLE INTERPRETATIONS**

Let \(\text{or}\) denote the "parallel or" function (\(\bot\ or \text{true} = \text{true or } \bot = \text{true}\)), and consider the following program, which tests if \(z\) is the sum of \(x\) and \(y\):

\[
\text{sum}(x, y, z) = \begin{cases} 
0 & \text{if } x = 0 \land y = 0 \\
x_2 & \text{else}
\end{cases}
\]

Starting from \(x, y, z > 0\), there are several ways to perform the computation; we have no reason to choose one rather than another.

Moreover, let \(f_1(x)\) and \(f_2(x)\) be two recursively defined boolean functions, and consider the program \(f(x) = f_1(x) \text{ or } f_2(x)\). There is only one way of computing \(f(a)\): to perform the computations of \(f_1(a)\) and \(f_2(a)\) in parallel; it is even not decidable which of the two computations is the most efficient.

n° mars 1976
In his study of time complexity of computation rules, Vuillemin [14, 15] introduces a determinism condition for top-down computations: the sequentiality condition. With respect to production functions, we can consider the two following conditions:

- **Stepwise determinism**: the set of intermediate values is unambiguously determined at each production step; formally, every point has a unique minimal producer \( m_p(x) \). The whole set of intermediate values for \( x \) is then determined by transitively applying \( m_p \). This will be satisfied by stable interpretations.

- **Global determinism**: a “best” whole set of intermediate values is unambiguously determined for every argument. This weaker condition will be formalised as the quasi-determinism condition on production functions (section III); it seems difficult to characterise corresponding interpretations.

The following program satisfies the second condition but not the first (it is possible but useless to produce \((x, 0)\) from \((x+1, 0)\)):

\[
    f(x) = \begin{cases} 
        \text{true} & \text{if } x = 0 \\
        f(x-1) \text{ or } f(x+1). 
    \end{cases}
\]

**II.1. Deterministic production functions**

**Definition**: A monotonic function \( \Phi: \mathcal{P}(D) \to \mathcal{P}(D) \) is deterministic if and only if every point \( x \) in \( \Phi(D) \) has a unique minimal producer \( m_p(x) \).

**II.1.1. Proposition**: A monotonic function \( \Phi \) is deterministic iff for every finite or infinite family \( X_i, i \in I \), of subsets of \( D \), \( \Phi \left( \bigcap_{i \in I} X_i \right) = \bigcap_{i \in I} \Phi(X_i) \).

**Proof**: Assume \( \Phi \left( \bigcap_{i \in I} X_i \right) = \bigcap_{i \in I} \Phi(X_i) \). Then \( m_p(x) = \bigcap \{ X \mid x \in \Phi(X) \} \).

Converse follows from the definition of \( m_p \). \( \Box \)

**II.2. Stable functions**

Let us define the restricted class of functions we shall use in programs:

**Definition**: Let \( D \) be a discrete domain, let \( b \) be a monotonic function from \( D^p \) to \( D \). Then \( b \) is stable by minimalisation (or simply stable) if and only if for every joinable \( x \) and \( x' \), \( b(x \land x') = b(x) \land b(x') \).

**II.2.1. Proposition**: A function \( b \in \Delta^p \) is stable if and only if for every \( x \in D^p \) there exists in \( D^p \) a unique minimal element \( m_b(x) \subseteq \bar{x} \) such that \( b(m_b(x)) = b(x) \).

**Proof**: If \( b \) is stable, then \( m_b(x) \) is the g. l. b. of the set \( Y = \{ \bar{y} \in D^p \mid \bar{x} \uparrow \bar{y} \land b(\bar{y}) = b(\bar{x}) \} \).

---

*Revue Française d'Automatique, Informatique et Recherche Opérationnelle*
Assume conversely the existence of $m_b(x)$ for every $x \in D^p$. Let $x$ and $x'$ be joinable. If $b(x) = \perp$ or $b(x') = \perp$, then $b(x \land x') = \perp$ by monotonicity. If $b(x) = b(x') \neq \perp$, then

$$b(x \lor x') = b(x) = b(x').$$

This implies $m_b(x \lor x') \subseteq x \land x'$ and by monotonicity

$$b(x \land x') = b(m_b(x \lor x')) = b(x) = b(x').$$

Let us study the connection with Vuillemin's sequential functions [14]:

**Definition:** A function $b \in \Delta^p$ is sequential iff for every $x \in D^p$ there exist an integer $i$, $1 \leq i \leq p$, such that $y \supseteq x$ and $y_i = x_i$ imply $b(y) = b(x)$. This integer is called the critical index of $x$ for $b$.

**Example:** Consider the usual if-then-else function. For $x = (\perp, x_2, x_3)$ the critical index is 1. For $x = (true, x_2, x_3)$ (resp. $(false, x_2, x_3)$), the critical index is 2 (resp. 3).

**II.2.2. Proposition:** Every sequential function is stable. There exist stable functions which are not sequential.

**Proof:** a) Assume $b \in \Delta^p$ is sequential but not stable. Let $x$ and $y$ satisfy $x \uparrow y$, $b(x) = b(y) \neq \perp$, $b(x \land y) = \perp$. For every $i \in \{1, 2, \ldots, p\}$, either $(x \land y)_i = x_i$ or $(x \land y)_i = y_i$. In the first case, $y \supseteq x \land y$ and $b(y) \supseteq b(x \land y)$, in the second case $y \supseteq x \land y$ and $b(y) \supseteq b(x \land y)$. Therefore $x \land y$ has no critical index, which is impossible.

b) Let $D = \{ \perp, 0, 1 \}$, let $b$ be the least function such that $b(0, 1, \perp) = 0$, $b(1, \perp, 0) = 0$, $b(\perp, 0, 1) = 0$. Then $b$ is stable, but $(\perp, \perp, \perp)$ has no critical index.

**II.3. Stable interpretations**

**Definition:** Let $\Sigma$ be a program scheme, let $I$ be a discrete interpretation. Then $I$ is stable if and only if all basic functions $I(b)$ are stable.

**II.3.1. Lemma:** Let $I$ be a stable interpretation, let $T$ be a term only containing basic function symbols. Then $T$ defines a stable function.

**Proof:** By structural induction on $T$. Case $T = x_i$ is obvious.

Let $T(x) = b(T_1(x), T_2(x), \ldots, T_m(x))$ assuming that the $T_i$ define stable functions. Assume $x \uparrow y$ and $T(x) = T(y) \neq \perp$. Then the $m$-tuples

$$(T_1(x), T_2(x), \ldots, T_m(x)) \quad \text{and} \quad (T_1(y), T_2(y), \ldots, T_m(y))$$

are joinable; by induction hypothesis, their g.l.b. is the $m$-tuple

$$(T_1(x \land y), T_2(x \land y), \ldots, T_m(x \land y)).$$

n° mars 1976
The result follows from the stability of $b$. □

II. 3.2. Theorem: Let $\Sigma$ be a recursive scheme, let $I$ be a stable interpretation. Then the functions $\tau^n(\bot)$ and the fixpoint $Y(\Sigma, I)$ are stable.

Proof: For every $m$, $\tau^n(\bot)$ is defined by a term containing only basic functions and the undefined function: Lemma applies. Let $x, y \in D^p$, $x \uparrow y$; there exist a least $m \geq 0$ such that $Y(\Sigma, I)(x) = \tau^m(\bot)(x)$ and $Y(\Sigma, I)(y) = \tau^m(\bot)(y)$. Then $Y(\Sigma, I)(x \land y) = \tau^m(\bot)(x \land y)$ imply

$$Y(\Sigma, I)(x \land y) = Y(\Sigma, I)(x) \land Y(\Sigma, I)(y).$$

We now prove a crucial property of stable interpretations:

II. 3.3. Theorem: Let $\Sigma$ be a program scheme, let $I$ be a stable interpretation. If $x$ is such that $\tau^m(\bot)(x) = \bot$, if $x'$ is joinable to $x$ and such that $Y(\Sigma, I)(x') \neq \bot$, then $\tau^m(\bot)(x') = \bot$.

Proof: By induction on $m$. Let $g$ denote $Y(\Sigma, I)$.

1) $m = 0$. Then $z = \tau(\bot)(x) = T^0(x, \bot)$, where $\bot$ denotes the $n$-tuple $(\bot, \bot, \ldots, \bot)$. Let $x'$ be such that $x \uparrow x'$ and $g(x') = z$. Since $g$ is fixpoint, $z = \tau(g)(x') = T^0(x', \alpha)$ with $\alpha = T^i(g)(x')$. The $p+n$-tuples $(x, \bot)$ and $(x', \alpha)$ are joinable, and their g. l. b. is $(x \land x', \bot)$. Therefore $T^0(x \land x', \bot) = z$ and by monotonicity $T^0(x', \bot) = z$, which is equivalent to $z = \tau(\bot)(x')$.

2) Assume the property true for $m$:

$$\forall y, y' \in D^p, \quad (\tau^m(\bot)(y) = z) \land (y \uparrow y') \land (g(y') = z) \Rightarrow \tau^m(\bot)(y') = z.$$ 

We show by structural induction that for every term $T$:

$$\forall y, y' \in D^p,$$

$$(T(\tau^m(\bot))(y) = z) \land (y \uparrow y') \land (T(g)(y') = z) \Rightarrow T(\tau^m(\bot))(y') = z.$$ 

Case $T = x_i$ is obvious. Assume $T = b(t_1, t_2, \ldots, t_k)$ where the property holds for every $t_k$. Then:

$$T(g)(x') = b(t_1(g)(x'), t_2(g)(x'), \ldots, t_k(g)(x')) = b(\alpha),$$

$$T(\tau^m(\bot))(x) = b(t_1(\tau^m(\bot))(x), t_2(\tau^m(\bot))(x), \ldots, t_k(\tau^m(\bot))(x)) = b(\beta).$$

But $\tau^m(\bot) \subseteq g$ implies $t_i(\tau^m(\bot))(x) \subseteq t_i(g)(x)$. By I.1.1, $\alpha$ and $\beta$ are joinable, and $b(\alpha) = b(\beta) = b(\alpha \land \beta)$. For every $i$ such that $(\alpha \land \beta)_i \neq \bot$, we have

$$(\alpha \land \beta)_i = t_i(g)(x') = t_i(\tau^m(\bot))(x) \neq \bot.$$ 

By induction hypothesis on $t_1, t_i(g)(x') = t_i(\tau^m(\bot))(x')$. 

Revue Française d'Automatique, Informatique et Recherche Opérationnelle
Therefore \((\alpha \land \beta) \subseteq (t_1 (\tau^m (\bot)) (\bar{x}'), t_2 (\tau^m (\bot)) (\bar{x}'), \ldots, t_k (\tau^m (\bot)) (\bar{x}'))\).

The result follows by monotonicity of \(b\).

Assume now \(T = f (t_1, t_2, \ldots, t_p)\). Since \(g\) is stable, the previous argument holds, and we can replace \(g\) by \(\tau^m (\bot)\) in the subterms \(t_i:\)

\[
T(g)(\bar{x}') = g (t_1 (\tau^m (\bot))(\bar{x}'), t_2 (\tau^m (\bot))(\bar{x}'), \ldots, t_p (\tau^m (\bot))(\bar{x}')) = g(\alpha),
\]

\[
T(\tau^m (\bot))(\bar{x}) = \tau^m (\bot)(t_1 (\tau^m (\bot))(\bar{x}'), t_2 (\tau^m (\bot))(\bar{x}'), \ldots, t_p (\tau^m (\bot))(\bar{x})) = \tau^m (\bot)(\beta).
\]

But \(\bar{x} \uparrow \bar{x}'\) implies \(\bar{\alpha} \uparrow \bar{\beta}\); the global induction hypothesis applies and yields

\[
T(g)(\bar{x}') = \tau^m (\bot)(t_1 (\tau^m (\bot))(\bar{x}'), t_2 (\tau^m (\bot))(\bar{x}'), \ldots, t_p (\tau^m (\bot))(\bar{x}'))
\]

which is the desired result.

Let now \(\bar{x}, \bar{x}' \in D^p\) be such that \(\tau^{m+1} (\bot) (\bar{x}) = z \neq \bot, g (\bar{x}') = z\) and \(\bar{x} \uparrow \bar{x}'\). Then \(z\) is the value of the term \(\tau (\tau^m (\bot)) (\bar{x})\), and also the value of the term \(\tau (g) (\bar{x}')\) since \(g\) is the fixpoint.

Applying to \(\tau\) the preceding result for terms, we get:

\[
\tau (g)(\bar{x}') = \tau (\tau^m (\bot))(\bar{x}') = \tau^{m+1} (\bot)(\bar{x}'). \quad \square
\]

This property is not true under general interpretations. Consider for instance the program

\[
f(x, y) = (x = 0) \text{ or } (x = 1 \land y = 0), \text{ or } f(x - 1, 0).
\]

Here \(\tau (\bot) (1, 0) = \text{true}, \tau (\bot) (1, \bot) = \bot, \tau^2 (\bot) (1, \bot) = \text{true}\).

Let us finally study the admissible \(n\)-tuples.

\(\Pi \ 3.4. \ \text{Theorem: Let } \Sigma \text{ be a recursive scheme, let } I \text{ be a stable interpretation. Let } (\bar{x}, z) \text{ belong to } \Phi^m (\emptyset), m > 0.\)

1) The set of admissible \(n\)-tuples for \((\bar{x}, z, Y(\Phi))\) has a minimal element \(\bar{\gamma}\) under the ordering \(\subseteq\).

2) \(\bar{\gamma}\) is also the minimal admissible \(n\)-tuple for \((m_\# (\bar{x}), z, Y(\Phi))\).

3) \(\bar{\gamma}\) is admissible for \((\bar{x}, z, \Phi^{m-1} (\emptyset))\).

\(\text{Proof: Let } g \text{ denote } Y(\Sigma, I).\)

1) Let \(I_i\) be the set of labels of the bound variables occurring in \(\mathcal{F}(T^i)\) (cf. I. 4). Let \(l_i\) denote the cardinality of \(I_i\), let \(\beta_i = T^i (g) (\bar{x})\).

We construct for every \(i\) a subset \(J_i\) of \(I_i\) satisfying the following two conditions:

\(a)\) there exists an \(n\)-tuple \(\bar{\alpha}\) admissible for \((\bar{x}, z, Y(\Phi))\) such that \(\alpha_j = \beta_j\) if \(j \in J_i, \alpha_i = \bot\) if \(j \notin J_i\) and \(j \in I_i\).

n° mars 1976
b) for every \( n \)-tuple \( \bar{\alpha} \) admissible for \((\bar{x}, z, Y(\Phi))\), for all \( j \in J_1 \), \( \alpha_j = \beta_j \neq \bot \).

If \( I_1 = \emptyset \), or if there exists \( \bar{\alpha} \) admissible for \((\bar{x}, z, Y(\Phi))\) such that \( \alpha_i = \bot \), then \( J_1 = \emptyset \) satisfies \( a \) and \( b \).

If every \( \bar{\alpha} \) admissible for \((\bar{x}, z, Y(\Phi))\) satisfies \( \alpha_i \neq \bot \), then

\[
\alpha_i = \mathcal{T}(T_i)(\bar{x}, \alpha_{i_1}, \alpha_{i_2}, \ldots, \alpha_{i_k})
\]

with \( i_j \in I_j \); there exists a minimal \( I_j \)-tuple \( \bar{\delta} = (\delta_{i_1}, \delta_{i_2}, \ldots, \delta_{i_k}) \) such that \( \alpha_i = \mathcal{T}(T_i)(\bar{x}, \bar{\delta}) \). Therefore \( J_i = \{ i_k | \delta_{i_k} \neq \bot \} \) satisfies \( a \) and \( b \).

We now define \( \bar{\gamma} \) by \( \gamma_j = \beta_j \) if \( j \in j_i \) for some \( i \), \( \gamma_j = \bot \) otherwise. Then \( \bar{\gamma} \) is by construction the unique minimal acceptable \( n \)-tuple for \((\bar{x}, z, Y(\Phi))\).

2) Let \( \bar{x}' \) denote \( m_g(\bar{x}) \). Since \( g(\bar{x}') = \bar{z} \neq \bot \), there exists a \( n \)-tuple \( \bar{\gamma}' \) admissible for \((\bar{x}', z, Y(\Phi))\). By I.6.2, \( \bar{\gamma}' \) is also admissible for \((\bar{x}, z, Y(\Phi))\), and hence \( \bar{\gamma}' \supseteq \bar{\gamma} \). Therefore \((\bar{x}, \bar{\gamma})\) and \((\bar{x}', \bar{\gamma}')\) are joinable, with

\[
(\bar{x}, \bar{\gamma}) \vee (\bar{x}', \bar{\gamma}') = (\bar{x}, \bar{\gamma}') \quad \text{and} \quad (\bar{x}, \bar{\gamma}) \wedge (\bar{x}', \bar{\gamma}') = (\bar{x}', \bar{\gamma}).
\]

Using the stability of \( \mathcal{T}(\tau) \), \( \mathcal{T}(t_j) \), \( g \), one easily shows that \( \bar{\gamma} \) is admissible for \((\bar{x}', z, Y(\Phi))\) (cf. [1]).

3) Assume \( (\bar{x}, z) \in \Phi^m(\emptyset) \). There exists \( \bar{\gamma}' \) admissible for \((x, z, \Phi^{m-1}(\emptyset))\).

Since \( \Phi^{m-1}(\emptyset) \subseteq Y(\Phi) \), \( \bar{\gamma}' \) is also admissible for \((\bar{x}, z, Y(\Phi))\) and \( \gamma' \supseteq \bar{\gamma} \).

For each \( i \) such that \( \gamma_i \neq \bot \), we have :

\[
\xi_i = (\mathcal{T}(t_i^1)(\bar{x}, \bar{\gamma}), \mathcal{T}(t_i^2)(\bar{x}, \bar{\gamma}), \ldots, \mathcal{T}(t_i^k)(\bar{x}, \bar{\gamma}), \gamma_i) \in Y(\Phi),
\]

\[
\xi'_i = (\mathcal{T}(t_i^1)(\bar{x}, \bar{\gamma}'), \mathcal{T}(t_i^2)(\bar{x}, \bar{\gamma}'), \ldots, \mathcal{T}(t_i^k)(\bar{x}, \bar{\gamma}'), \gamma'_i) \in \Phi^{m-1}(\emptyset).
\]

\((\bar{x}, \bar{\gamma})\) and \((\bar{x}, \bar{\gamma}')\) being joinable, \( \xi_i \) and \( \xi'_i \) are joinable, an by II.3.3:

\[
(\mathcal{T}(t_i^1)(\bar{x}, \bar{\gamma}), \mathcal{T}(t_i^2)(\bar{x}, \bar{\gamma}), \ldots, \mathcal{T}(t_i^k)(\bar{x}, \bar{\gamma}), \gamma_i) = \xi_i \in \Phi^{m-1}(\emptyset),
\]

which means that \( \bar{\gamma} \) is admissible for \((\bar{x}, z, \Phi^{m-1}(\emptyset))\). \( \square \)

II.4. Stable interpretations and \( \Psi \)-producers

Even under stable interpretations, the extended production function \( \Psi \) is in general not deterministic. Consider the example:

\[
f(x_1, x_2) = \text{if } x_1 = 0 \text{ then } 0 \text{ else } f(x_1 - 1, f(x_1 - 2, x_2))
\]

Here \((2, 1, 0)\) has two distinct minimal \( \Psi \)-producers \( P_1 = \{(1, \bot, 0)\} \) and \( P_2 = \{(1, 0, 0), (0, 1, 0)\} \).

But \( P_1 \) is of smaller cardinality than \( P_2 \), and contains more compact information: the existence of \((1, \bot, 0)\) in \( \Psi(\emptyset) \) implies the existence of \((1, 0, 0)\) by monotonicity, and the point \((0, 1, 0)\) is not really useful for the computation.
To formalise this intuitive comparison between $\Psi$-producers, we define the following partial preoder:

**Definition:** Let $D$ be a discrete domain, let $X, Y \subseteq D^p$.

The relation $\sqsubseteq$ is defined by

$$X \sqsubseteq Y \iff \forall \bar{x} \in X, \exists y \in Y, \bar{x} = \bar{y}.$$

**II.4.1. Theorem:** Let $\Sigma$ be a recursive program scheme, let $I$ be a stable interpretation.

Let $(\bar{x}, z) \in Y (\Psi)$, let $MP (\bar{x}, z)$ denote the family of the minimal $\Psi$-producers of $(\bar{x}, z)$.

1) The relation $\sqsubseteq$ induces a partial ordering on $MP (\bar{x}, z)$.

2) $MP (\bar{x}, z)$ has a unique minimal element under $\sqsubseteq$, which will be called the optimal producer of $(\bar{x}, z)$, and denoted $\text{opt} (\bar{x}, z)$.

3) If $(\bar{x}, z) \in \Psi^m (\emptyset)$, then $\text{opt} (\bar{x}, z) \sqsubseteq \Psi^{m-1} (\emptyset)$.

4) If $\bar{y}$ is the minimal admissible $n$-tuple for $(\bar{x}, z, Y (\Psi))$, then $\bar{y}$ is admissible for $(\bar{x}, z, \text{opt} (\bar{x}, z))$.

5) If $x'$ is joinable to $\bar{x}$ and if $(\bar{x}', z) \in Y (\Psi)$ then $\text{opt} (\bar{x}', z) = \text{opt} (\bar{x}, z)$.

6) For every $\Phi$-producer or $\Psi$-producer $P$ of $(\bar{x}, z)$, the cardinality of $\text{opt} (\bar{x}, z)$ is less than or equal to the cardinality of $P$.

**Proof:**

1) Let $P \in MP (\bar{x}, z)$. Then for every $p, p' \in P$, $p \sqsubseteq p'$ is false: otherwise $P - \{p'\} \Psi$-produces $(\bar{x}, z)$. Proposition follows easily.

2) Let $(\bar{x}, z) \in Y (\Psi)$, let $\bar{y}$ be the minimal admissible $n$-tuple for $(\bar{x}, z, Y (\Psi))$. Let $P_0$ be the set of the points:

$$\xi_i = (\mathcal{F} (t_1^i) (\bar{x}, \bar{y}), \mathcal{F} (t_2^i) (\bar{x}, \bar{y}), \ldots, \mathcal{F} (t_p^i) (\bar{x}, \bar{y}), \gamma_i) \quad \text{with} \quad \gamma_i \neq \bot.$$

Then $P_0 \Phi$-produces $(\bar{x}, z)$. But $Y (\Sigma, I)$ is a stable function: for every $\xi_i$ there exists a least $\eta_i \subseteq \xi_i$ such that $\eta_i \in Y (\Phi)$.

Let $\text{opt} (\bar{x}, z)$ be the set of the $\eta_i$. By construction $\text{opt} (\bar{x}, z) \sqsubseteq P_0$, and therefore $\text{opt} (\bar{x}, z) \sqsubseteq P_0$: $\text{opt} (\bar{x}, z) \Psi$-produces $(\bar{x}, z)$.

Assume $R \subseteq Y (\Psi)$ $\Psi$-produces $(\bar{x}, z)$, and let $\bar{\alpha}$ be admissible for $(\bar{x}, z, \hat{R})$.

By II.3.4, $\bar{\alpha} \sqsubseteq \bar{y}$, and $\hat{R}$ contains at least the points:

$$\xi'_i = (\mathcal{F} (t_1^i) (\bar{x}, \bar{\alpha}), \mathcal{F} (t_2^i) (\bar{x}, \bar{\alpha}), \ldots, \mathcal{F} (t_p^i) (\bar{x}, \bar{\alpha}), \gamma_i) \quad \text{with} \quad \gamma_i \neq \bot.$$

For each $i$ such that $\gamma_i \neq \bot$ we have $\xi_i \subseteq \xi'_i$ and hence $\eta_i \subseteq \xi'_i$. But $R$ contains at least one $\eta'_i$ such that $\eta'_i \subseteq \xi'_i$. Necessarily, $\eta_i \subseteq \eta'_i$, and $\text{opt} (\bar{x}, z) \sqsubseteq R$.

n° mars 1976
Since opt \((x, z)\) is finite, we prove that it is a minimal \(\Psi\)-producer of \((x, z)\) by showing that it is a \(\Psi\)-producer of minimal cardinality. Assume that \(U\) \(\Psi\)-produces \((x, z)\) and assume \(\text{card} (U) < \text{card} (\text{opt} (x, z))\). Since \(\text{opt} (x, z) \subseteq U\), there exist \(n_1, n_2 \in \text{opt} (x, z)\) and \(u \in U\) such that \(n_1 \leq u\) and \(n_2 \leq u\). This is impossible, since by construction \(n_1\) and \(n_2\) cannot be joinable.

3) Since \(\tilde{\gamma}\) is also admissible for \((\bar{x}, z, \Phi^n (\emptyset))\), we have \(P_0 \subseteq \Phi^{n-1} (\emptyset)\). Now II.3.3 implies \(\text{opt} (\bar{x}, z) \subseteq \Phi^{n-1} (\emptyset)\).

4) The \(n\)-tuple \(\tilde{\gamma}\) is admissible for \(P_0\), and \(\text{opt} (\bar{x}, z) \supseteq P_0\).

5) If \(\bar{x} \uparrow \bar{x}'\) and if \((\bar{x}', z) \in Y(\Psi)\), then by II.3.4 \(\tilde{\gamma}\) is also the minimal admissible \(n\)-tuple for \((\bar{x}', z, Y (\Phi))\).

6) We showed in 2 that \(\text{opt} (\bar{x}, z)\) is a \(\Psi\)-producer of minimal cardinality. But every \(\Phi\)-producer is also a \(\Psi\)-producer. \(\Box\)

II.5. Construction of the deterministic production function \(\theta_{(\bar{x}, \bar{y})}\)

Our last step is to construct a new production function \(\theta\) by selecting among the \(\Psi\)-producers the optimal one. In fact \(\theta\) itself will not be derivistic: we only defined the optimal producers for the points in \(Y (\Psi)\). However is "restriction" to its fixpoint will be determinist: every point of \(Y (\theta)\) will have a unique minimal producer included in \(Y (\theta)\), and all the results of the next section will hold. We call minimal a point \((x, z)\) such that \((x, z) \in Y (\Psi)\), and \(x' \leq x\) implies \((x', z) \notin Y (\Psi)\). By construction every point of \(\text{opt} (\bar{x}, z)\) is minimal. Given, for some \(m\), \(\Psi^m (\emptyset)\) and \((\bar{x}, z)\) in \(\Psi^m (\emptyset)\), we can effectively test if \((\bar{x}, z)\) is minimal: we just test if \(\Psi^m (\emptyset)\) contains no point \((x, z)\) with \(x' \leq x\); the number of possible points is finite. However this method fails if we only know a proper subset \(X\) of \(\Psi^m (\emptyset)\), as will be the case in most of the actual bottom-up computations (see Chapter III). We shall modify \(\Psi\) by adding to every point \((x, z)\) a flag \(f\) set to 1 iff \((x, z)\) is minimal.

**Definition:** If \(X \subseteq D^{p+1} \times \{0, 1\}\), \(p (X)\) denotes the projection of \(X\) ob \(D^{p+1}\).

**Definition:** Let \((\Sigma, I)\) be a recursive program.

The production function \(\theta_{(\bar{x}, \bar{y})} : \mathcal{P} (D^{p+1}_I \times \{0, 1\}) \rightarrow \mathcal{P} (D^{p+1}_I \times \{0, 1\})\) is defined by

\[
\theta (X) = \{(\tilde{x}, z, fI) \in D^{p+1}_I \times \{0, 1\} \mid S (\tilde{x}, z, X) \\
\land [fI = 1 \iff \forall x' \subseteq x \sim S (\tilde{x}', z, X)]\}
\]

with \(S (x, z, X) = \exists Y \subseteq X \forall (\bar{y}, t, fly) \in Y, fly = 1) \land P (\bar{x}, z, p (Y)).\)

**II.5.1. Proposition:** 1) \(\theta\) is monotonic and continuous.

2) \(\forall m \in N\), \(p (\theta^m (\emptyset)) = \Psi^m (\emptyset); p (Y (\emptyset)) = Y (\Psi).\)
3) If \((\bar{x}, z, fI) \in Y(\theta)\), \(fI = 1\) is equivalent to \((\bar{x}, z)\) minimal.

4) If the interpretation \(I\) is stable, then every point \((\bar{x}, z, fI)\) in \(Y(\theta)\) has a unique minimal producer included in \(Y(\theta)\); this minimal producer is \(\text{opt}(\bar{x}, z) \times \{1\}\).

\textit{Proof:} 1) 2) 3) directly follow from the construction (see [1]).

4) Every point of \(\text{opt}(\bar{x}, z)\) is minimal by construction; therefore \(\text{opt}(\bar{x}, z) \times \{1\}\) produces \((\bar{x}, z, 0)\) or \((\bar{x}, z, 1)\).

If \(P\) \(\theta\)-produces \((\bar{x}, z, fI)\), there exists \(P' \subseteq P\) such that every point of \(P'\) is minimal and such that \(P(P')\) \(\Psi\)-produces \((\bar{x}, z)\). Then \(P \supseteq \text{opt}(\bar{x}, z)\) and \(P \supseteq \text{opt}(\bar{x}, z)\). \(\square\)

\textit{Remark:} It may obviously be the case that \(I\) is not stable and \(\theta(\varepsilon, I)\) is deterministic. However, given a non-stable function \(b \in A^\varepsilon\), one can easily construct a non-deterministic program just by using sequential \texttt{if-then-else} and constant functions (see [1]).

III. BOTTOM-UP COMPUTATIONS. QUASI-DETERMINISTIC PRODUCTION FUNCTIONS

Let \(D\) be an arbitrary set and \(\Phi\) be a monotonic function from \(\mathcal{P}(D)\) to \(\mathcal{P}(D)\). Every notion is defined with respect to \(\Phi\), and symbols may be prefixed by \(\Phi\) if necessary: \(\Phi\)-b. u. c. s., \(\Phi\)-s. r., \(\Phi\)-dom...

We first formalise the notion of bottom-up computation of a point in \(Y(\Phi)\). We next express the quasi-determinism condition (cf. section II), and then study some properties of the quasi-deterministic functions. Lastly, we present induction techniques based on the structure of the set of intermediate values.

Illustrations are given in diagram 1 (in annex).

III.1. Bottom-up computation sequences; storage configuration sequences

For defining bottom-up computations, we use as basic operation the production of a point by a subset of \(D\), starting from initially empty information.

\textbf{Definition:} A \textit{bottom-up computation sequence} (abbreviated b. u. c. s.) is a sequence \(X_0, X_1, \ldots, X_n, \ldots\) of subsets of \(D\) such that

\[X_0 = \emptyset \quad \text{and} \quad \forall n \geq 0, \ X_n \subseteq X_{n+1} \subseteq X_n \cup \Phi(X_n).\]

A \textit{storage configuration sequence} (abbreviated s. c. s.) is a sequence \(X_0, X_1, \ldots, X_n, \ldots\) of subsets of \(D\) such that

\[X_0 = \emptyset \quad \text{and} \quad \forall n \geq 0, \ X_{n+1} \subseteq X_n \cup \Phi(X_n).\]
A s. c. s. \{ X_n, n \in N \} is \textbf{simple} iff it satisfies
\[
\forall x \in \bigcup_n X_n \sim \exists p, q, r \in N, p < q < r, x \in X_p \wedge x \notin X_q \wedge x \in X_r.
\]

A point \( x \in D \) is computed by a s. c. s. \{ X_n, n \in N \} iff \( x \in \bigcup_n X_n \).

s. c. s. correspond to actual computations, b. u. c. s. correspond to histories of computations. In a s. c. s., we are allowed to discard intermediate values when they are no longer useful for the computation. In simple s. c. s. we are however not allowed to recompute intermediate values.

Clearly, every b. u. c. s. is a simple s. c. s.; on the other hand, if \{ Y_n, n \in N \} is a s. c. s., then the sequence \( X_n = \bigcup_{0 \leq p \leq n} Y_p \) is a b. u. c. s. An example of simple s. c. s. is Kleene's sequence \( \emptyset, \Phi(\emptyset), \ldots, \Phi^n(\emptyset), \ldots \)

\section{III. 2. Self-reproducing sets. Quasi-deterministic functions}

We now introduce the basic property which will allow us to characterise the members and limits of b. u. c. s., and to express the quasi-determinism condition.

\textbf{Définition:} A subset \( S \) of \( D \) is \textbf{self-reproducing} (abbreviated s. r.) iff it satisfies \( S \subseteq \Phi(S) \).

\section{III. 2.1. Proposition:} If \( S \) is s. r., so is \( \Phi(S) \); A finite or infinite union of s. r. sets is s. r.; For every \( n \in N \), \( \Phi^n(\emptyset) \) is s. r.

\section{III. 2.2. Proposition:} Every member or limit of a b. u. c. s. is s. r.

\textit{Proof:} Let \{ \( X_n, n \in N \) \} be a b. u. c. s. If \( n = 0 \), then \( X_0 = \emptyset \), and \( X_0 \subseteq \Phi(X_0) \). Assume \( X_n \subseteq \Phi(X_n) \). Then \( X_{n+1} \subseteq X_n \cup \Phi(X_n) \) implies \( X_{n+1} \subseteq \Phi(X_n) \), and \( X_n \subseteq X_{n+1} \) implies \( \Phi(X_n) \subseteq \Phi(X_{n+1}) \); finally \( X_{n+1} \subseteq \Phi(X_{n+1}) \). The limit \( \bigcup_n X_n \) is s. r. since it is a union of s. r. sets.

The intuitive definition of the quasi-determinism condition was the existence of a least set of necessary and sufficient intermediate values for any given argument.

We now give the following formal definition:

\textbf{Définition:} \( \Phi \) is \textbf{quasi-deterministic} iff every point \( x \in \Phi(D) \) has a unique minimal self-reproducing producer \textit{prod}(x).

\section{III. 2.3. Proposition:} A function \( \Phi \) is quasi-deterministic iff for every finite or infinite family \( S_i, i \in I \), of s. r. sets, \( \Phi(\bigcap_{i \in I} S_i) = \bigcap_{i \in I} \Phi(S_i) \).

\textit{Proof:} See II. 1. 1. \( \square \)

Clearly, every deterministic function is quasi-deterministic.
III.2.4. Proposition: Let $\Phi$ be quasi-deterministic and continuous.

a) $X \subseteq Y(\Phi)$ is the limit of a b. u. c. s. iff $X$ is s. r.

b) $X \subseteq Y(\Phi)$ is member of a b. u. c. s. iff $X$ is s. r. and included in $\Phi^n(\emptyset)$ for some $m \geq 0$.

Proof: Let $\{X_n, n \in N\}$ be a b. u. c. s. Then $X_n$ and $\bigcup_{n \in N} X_n$ are s. r. by III.2.2. By induction on $n$, $X_n \subseteq \Phi^n(\emptyset)$. Conversely, if $S$ is s. r., then the sequence $S \cap \Phi^n(\emptyset)$ is a b. u. c. s.

A quasi-deterministic function may not be deterministic; however nothing can be gained by using the non determinism facilities: let $P_1$ be a minimal producer of $x$ included in prod $(x)$, let $P_2$ be another minimal producer such that $P_2 \notin$ prod $(x)$. We can produce $x$ by first producing either $P_1$ or $P_2$. Assume $\{X_n, n \in N\}$ is a b. u. c. s. such that $X_n \supseteq P_2$. Then $x \in \Phi(X_n)$. But $X_n$ is s. r. and produces $x : X_n \supseteq \text{prod} (x)$, and $X_n \supseteq P_1$; computing $P_2$ is simply a waste of time. Therefore prod $(x)$ is actually the set of necessary and sufficient intermediate values we wanted to define.

III.3. Producers and domains. The ordering $<$

Definition: Let $\Phi$ be quasi-deterministic. The domain of $x$, $\text{dom} (x)$, is the smallest s. r. set containing $x$. The relation $<$ is defined by $x \prec y$ iff $x \in \text{dom} (y)$.

III.3.1. Proposition: Let $\Phi$ be quasi-deterministic and continuous, let $x \in Y(\Phi)$:

1) $\text{dom} (x) = \text{prod} (x) \cup \{x\}$;

2) $x \notin \text{prod} (x)$;

3) The relation $<$ induces a well-founded ordering on $Y(\Phi)$.

Proof: 1) Since $\text{dom} (x)$ is s. r. and contains $x$, it produces $x$ and $\text{dom} (x) \supseteq \text{prod} (x) \cup \{x\}$. Now

$$\Phi(\text{prod} (x) \cup \{x\}) \supseteq \Phi(\text{prod} (x)) \supseteq \text{prod} (x) \cup \{x\}.$$ 

Then $\text{prod} (x) \cup \{x\}$ is s. r. and contains $x$; hence

$$\text{prod} (x) \cup \{x\} \supseteq \text{dom} (x).$$

2) Let $n$ be the smallest integer such that $x \in \Phi^n(\emptyset)$.

Then $\Phi^{n-1}(\emptyset)$ is s. r. and produces $x$; hence $\text{prod} (x) \subseteq \Phi^{n-1}(\emptyset)$ and $x \notin \text{prod} (x)$.

3) The relation $<$ is obviously reflexive and transitive. It is also antisymmetric: assume $x \prec y$, $y \prec x$, $x \neq y$. Then $x \in \text{prod} (y)$ implies $\text{prod} (x) \subseteq \text{prod} (y)$, and $y \in \text{prod} (x)$ implies $y \in \text{prod} (y)$, contradicting 2.
If $x \in \Phi^n(\emptyset)$ and $y < x$, then $\text{prod}(x) \subseteq \Phi^{n-1}(\emptyset)$ and $y \in \Phi^{n-1}(\emptyset)$. The relation $<$ is well-founded. □

We extend the definitions of $\text{prod}(x)$ and $\text{dom}(x)$ to subsets of $D$:

**Definition:** Let $\Phi$ be quasi-deterministic.

Let $X \subseteq Y(\Phi)$. Then $\text{dom}(X)$ denotes the smallest s. r. set containing $X$, $\text{prod}(X)$ denotes the smallest s. r. set producing $X$.

III. 3.2. **Proposition:**

1) $\text{dom}(X) = \bigcup_{x \in X} \text{dom}(x)$.

2) $\text{prod}(X) = \bigcup_{x \in X} \text{prod}(x)$.

3) $\text{dom}(X) = \text{prod}(X) \cup X$.

4) $\text{prod}(X) = \text{prod}(\text{dom}(X))$.

5) $S$ s. r. $\Leftrightarrow S = \text{dom}(S) \Leftrightarrow S \supseteq \text{prod}(S)$.

III. 4. Extremal points and extremal set of a s. r. set

A set $S$ is s. r. iff $S = \bigcup_{x \in X} \text{dom}(x)$. We note that if $y \in S$ is such that $\exists z \in S$, $y \in \text{prod}(z)$, then $S = \bigcup_{x \in S-(y)} \text{dom}(x)$. We are interested here in specifying a least set $E$ which characterises $S$ by $S = \text{dom}(E)$.

**Definition:** Let $\Phi$ quasi-deterministic, let $S$ be s. r. A point $x \in S$ is extremal for $S$ iff $x \notin \text{prod}(S)$. A subset $E$ of $S$ is extremal for $S$ iff $S = \text{dom}(E)$ and $\forall e, e' \in E$, $e < e'$. By convention, $\emptyset$ is extremal for $\emptyset$.

III. 4.1. **Proposition:** A point $x$ is extremal for a s. r. set $S$ iff $S - \{x\}$ is s. r.

**Proof:** Let $x$ be extremal for $S$. Then $S - \{x\} \supseteq \text{prod}(S)$, and $\Phi(S - \{x\}) \supseteq S - \{x\}$. Assume conversely that $S - \{x\}$ is s. r. Then $S - \{x\} \supseteq \text{prod}(S) - \{x\}$, and $S - \{x\} \supseteq \text{prod}(x)$, since $S$ produces $x$. Finally $S - \{x\} \supseteq \text{prod}(S)$ and $x \notin \text{prod}(S)$. □

III. 4.2. **Proposition:** Let $S$ be s. r. Then $E \subseteq S$ is extremal for $S$ iff $S = \text{dom}(E)$ and $E \cap \text{prod}(S) = \emptyset$.

**Proof:** Clearly, if $E$ satisfies $S = \text{dom}(E)$ and $E \cap \text{prod}(S) = \emptyset$, then $E$ is extremal for $S$. Assume conversely $E$ extremal for $S$. Assume $\exists e \in E e \in \text{prod}(S)$. Then $\exists s \in S e \in \text{prod}(s)$. Since $S = \text{dom}(E)$, $\exists e' \in E s \in \text{dom}(e')$. Finally $e \in \text{prod}(e')$, which is impossible. □

III. 4.3. **Proposition:** A s. r. set $S$ has at most one extremal set $\text{ext}(S)$. If it exists, $\text{ext}(S)$ is the set of extremal points of $S$. If $E' \subseteq S$ satisfies $S = \text{dom}(E')$, then $E' \supseteq \text{ext}(S)$.

*Revue Française d'Automatique, Informatique et Recherche Opérationnelle*
Proof: Let $E$ be extremal for $S$. If $e \in E$, then $e \notin \text{prod} (S)$ and $e$ is extremal for $S$.

Let $e$ be extremal for $S$. Assume $e \notin E$. Then $S - \{ e \}$ is s. r., and $E \subseteq S - \{ e \}$ implies $\text{dom} (E) \subseteq S - \{ e \}$, contradicting $\text{dom} (E) = S$.

If $E'$ satisfies $\text{dom} (E') = S$, then $E'$ clearly contains the extremal points of $S$. □

A s. r. set $S$ may have no extremal set: this is the case for $Y (\Phi)$ if $Y (\Phi) \neq \Phi^n (\emptyset)$ for all $n$. This is however not the case for the bounded sets:

**Definition:** A set $X$ is $n$-bounded iff $X \subseteq \Phi^n (\emptyset)$. The $n$-th slice of $Y (\Phi)$ is defined by $\tau^n = \Phi^n (\emptyset) - \Phi^{n-1} (\emptyset)$.

**III. 4.4. Theorem:** Let $\Phi$ be quasi-deterministic, let $S$ be s. r. included in $\Phi^n (\emptyset)$. Then $S$ has an extremal set; $\text{ext} (S)$ is the union of $S \cap \tau^n$ and of the points of $\text{ext} (S \cap \Phi^{n-1} (\emptyset))$ which do not belong to $\text{dom} (S \cap \tau^n)$.

**Proof:** By induction on $n$. □

**III. 4.5. Corollary 1:** Let $x \in Y (\Phi)$. Then $\text{prod} (x)$ has an extremal set. Moreover, if $\Phi$ is deterministic, then $\text{ext} (\text{prod} (x)) \supseteq \text{mp} (x)$.

**Proof:** If $x \in Y (\Phi)$, then $x \in \Phi^n (\emptyset)$ for some $n$ and $\text{prod} (x)$ is $(n-1)$-bounded.

Assume $\Phi$ deterministic, assume $y \in \text{ext} (\text{prod} (x))$ satisfies $y \notin \text{mp} (x)$.

Then $\text{prod} (x) - \{ y \}$ is s. r., contains $\text{mp} (x)$, and produces $x$: contradiction with the minimal of $\text{prod} (x)$. □

**III. 4.6. Corollary 2:** Let $\Phi$ be quasi-deterministic, let $X_n$ be a b. u. c. s.

Then $X_n$ has an extremal set, which contains $X_n - X_{n-1}$.

**Proof:** By induction on $n$, $X_n$ is $n$-bounded, and III.4.3 applies. Now $X_n \subseteq \Phi (X_{n-1})$ implies $\text{prod} (X_n) \subseteq X_{n-1}$. Therefore $\text{ext} (X_n) \supseteq X_n - X_{n-1}$. □

**III. 5. Extremal sets**

According to the previous results, we can define the extremal sets in an intrinsic way.

**Definition:** Let $\Phi$ be quasi-deterministic. A set $E \subseteq D$ is extremal iff $E$ is extremal for $\text{dom} (E)$.

**III. 5.1. Proposition:** Let $\Phi$ be quasi-deterministic

$$E \text{ extremal} \iff E \cap \text{prod} (E) = \emptyset \iff \text{prod} (E) = \text{dom} (E) - E.$$
The relation $<$ extends to extremal sets:

**Définition:** Let $\Phi$ be quasi-deterministic, let $\text{EXT}$ denote the family of the extremal sets included in $Y(\Phi)$. The relation $<$ is defined on $\text{EXT}$ by $E_1 < E_2 \iff \text{dom}(E_1) \subseteq \text{dom}(E_2)$.

III.5.2. *Proposition:* The relation $<$ is a partial ordering on $\text{EXT}$.

*Proof:* Reflexivity and transitivity are obvious.

Assume $E_1 < E_2$, $E_2 < E_1$. Then $\text{dom}(E_1) = \text{dom}(E_2)$.

Since a s.r. set has a unique extremal set, $E_1 = E_2$. The relation is antisymmetric. $\square$

III.5.3. *Lemma:* Let $E_1, E_2 \in \text{EXT}$. Then

$$E_1 < E_2 \Rightarrow \prod(E_1) \subseteq \prod(E_2).$$

*Proof:* We have $\prod(E_1) \subseteq \text{dom}(E_1) \subseteq \text{dom}(E_2) = \prod(E_2) \cup E_2$.

Assume $\exists e_2 \in E_2$ $e_2 \in \prod(E_1)$. Then $\exists e_1 \in E_1$ $e_2 \in \prod(e_1)$. Since $e_1 \in \text{dom}(E_2)$, $\exists e_2' \in E_2$, $e_1 \in \text{dom}(e_2')$.

We get $e_2 \in \prod(e_2')$, which is impossible since $E_2$ is extremal. $\square$

III.5.4. *Theorem:* Let $E_1, E_2 \in \text{EXT}$. Then $E_1$ and $E_2$ have a least upperbound $\max(E_1, E_2)$ in $(\text{EXT}, <)$. This l.u.b. satisfies the following properties:

1) $\max(E_1, E_2) = \text{ext}(\text{dom}(E_1) \cup \text{dom}(E_2));$
2) $\max(E_1, E_2) = \{e_1 \in E_1 | e_1 \notin \prod(E_2)\} \cup \{e_2 \in E_2 | e_2 \notin \prod(E_1)\};$
3) $\prod(\max(E_1, E_2)) = \prod(E_1) \cup \prod(E_2)$.

*Proof:* Consider the set $E$ defined by property 2; $E$ is extremal by construction. Since $E \subseteq E_1 \cup E_2$, we have $\text{dom}(E) \subseteq \text{dom}(E_1) \cup \text{dom}(E_2)$. Let $x \in \text{dom}(E) \subseteq \text{dom}(E_1) \cup \text{dom}(E_2)$; we can assume $x \in \text{dom}(E_1)$. Then $\exists e_1 \in E_1$ $x \in \text{dom}(e_1)$. If $e_1 \notin \prod(E_2)$, then $e_1 \in E$ and $x \in \text{dom}(E)$. If $e_1 \in \prod(E_2)$, there exists $e_2 \in E_2$ such that $e_1 \in \prod(e_2)$; $e_1 \in \prod(E_1)$ being impossible, we have $e_2 \notin \prod(E_1)$, and hence $e_2 \in E$ and $x \in \text{dom}(E)$.

Finally $\text{dom}(E) = \text{dom}(E_1) \cup \text{dom}(E_2)$. Property 1. is proved, and $E$ clearly is the l.u.b. of $E_1$ and $E_2$.

By construction, $E$ is of the form $E_1' \cup E_2'$ with $E_1' \subseteq E_1$ and $E_2' \subseteq E_2$. Hence

$$\prod(E) = \prod(E_1') \cup \prod(E_2'),$$
and $\prod(E) \subseteq \prod(E_1) \cup \prod(E_2)$.

The reverse inclusion follows from III.5.3, and property 3 is proved. $\square$

III.5.5. *Proposition:* Two elements $E_1$ and $E_2$ of $\text{EXT}$ have a g.l.b. in $(\text{EXT}, <)$ iff $\text{dom}(E_1) \cap \text{dom}(E_2)$ has an extremal set. If it exists, this g.l.b. satisfies $\min(E_1, E_2) = \text{ext}(\text{dom}(E_1) \cap \text{dom}(E_2))$. 

*Revue Française d'Automatique, Informatique et Recherche Opérationnelle*
Proof: If \( \text{dom}(E_1) \cap \text{dom}(E_2) \) has an extremal set \( E \), then \( E \) clearly is the g.l.b. of \( E_1 \) and \( E_2 \). Assume conversely that \( E_1 \) and \( E_2 \) have a g.l.b. \( E \), assume \( \text{dom}(E) \subseteq \text{dom}(E_1) \cap \text{dom}(E_2) \). Let
\[
a \notin \text{dom}(E), \ a \in \text{dom}(E_1) \cap \text{dom}(E_2),
\]
let \( E' = \max (\{ a \}, E) \). Then \( E' \neq E, E' > E, E' < E_1, E' < E_2 \), which is impossible.

\[\Box\]

III. 6. S-complete sets

Among the extremal sets included in a s.r. set, those which are maximal with respect to inclusion present a very rich structure. They seem to be also useful for determining the storage requirements of bottom-up computations (see section IV).

**Definition:** Let \( \Phi \) be quasi-deterministic and continuous, let \( S \) be s.r included in \( Y(\Phi) \). A set \( C \subseteq S \) is S-complete iff \( C \) is extremal and no extremal set included in \( S \) properly contains \( C \).

\( \text{SCOMP} \) denotes the family of the S-complete sets.

**III. 6.1. Proposition:** Let \( C \subseteq S \) be extremal. Then \( C \) is S-complete iff \( \forall x \in S-\text{dom}(C) \quad \text{dom}(x) \cap C \neq \emptyset \).

Proof: 1) Assume \( C \) S-complete, let \( x \in S-\text{dom}(C) \) satisfy \( \text{dom}(x) \cap C = \emptyset \).

Then for all \( c \) in \( C \), \( c \notin \text{dom}(x) \) and \( x \notin \text{dom}(c) \); hence \( C \cup \{x\} \) is extremal, is included in \( S \) and strictly contains \( C \); contradiction with \( C \) S-complete. Conversely, assume \( \forall x \in S-\text{dom}(C) \quad \text{dom}(x) \cap C \neq \emptyset \). Assume there exists \( a \in S \) such that \( C \cup \{a\} \) is extremal. Then \( \text{dom}(a) \cap C = \emptyset \) and \( a \notin \text{dom}(C) \), which is impossible.

The next characterisation does not directly involve the points of \( C \), but the domain of \( C \):

**III. 6.2. Theorem:** Let \( C \subseteq S \) be extremal. Then \( C \) is S-complete iff the following two conditions hold:

1) \( \forall x \in S \quad \text{prod}(x) \subseteq \text{prod}(C) \Rightarrow x \in \text{dom}(C). \)
2) \( \forall x \in S-\text{prod}(C) \quad \text{dom}(x) \cap \text{dom}(C) \neq \emptyset. \)

Proof: a) Assume \( C \) S-complete. The second condition holds by III.6.1. Assume there exists \( x \in S \) such that \( x \notin \text{dom}(C) \) and \( \text{prod}(x) \subseteq \text{prod}(C) \).

Then \( x \notin \text{prod}(C) \), and for all \( c \in C \), \( c \notin \text{prod}(x) \); this means that \( C \cup \{x\} \) is extremal, contradicting the S-completeness of \( C \).

b) Assume both conditions hold, let \( x \in S-\text{prod}(C) \).

Since \( x \in \Phi^* (\emptyset) \) for somen, the set \( \text{dom}(x) \cap \text{dom}(C) \) is n-bounded and has an extremal set \( E \) (III.4.4). Because of the second condition, \( E \) is not empty.

\[\text{n\textsuperscript{e} mars 1976}\]
Let \( e \in E \), Lemma III.5.3 implies \( \text{prod} (e) \subseteq \text{prod} (C) \), and first condition implies \( e \in \text{dom} (C) \). Therefore \( e \in \text{dom} (x) \cap C \), and \( C \) is \( S \)-complete by III.6.1. \( \square \)

III.6.3. **COROLLARY:** Let \( C \) be \( S \)-complete, let \( C_1 \subseteq S \) be such that \( C \prec C_1 \). Then \( C_1 \) is \( S \)-complete iff \( \forall x \in S \text{ prod} (x) \subseteq \text{prod} (C_1) \Rightarrow x \in \text{dom} (C_1) \).

III.7. **Lattice of the \( S \)-complete sets**

**DEFINITION:** Let \( S \) be s. r., let \( S' \) be s. r. included in \( S \). The yield of \( S' \) in \( S \) is defined by:

\[
Y_S (S') = (\Phi(S') - S') \cap S = \{ s \in S | s \notin S' \land \text{prod} (s) \subseteq S' \}.
\]

III.7.1. **THEOREM:** Let \( \Phi \) quasi-deterministic and continuous, let \( S \) be s. r. and included in \( Y (\Phi) \). Then \( (\text{SCOMP}, \prec) \) is a lattice; the l. u. b. and g. l. b. satisfy the following properties:

1) \( \text{cmax}_S (C_1, C_2) = Y_S (\text{prod} (C_1) \cup \text{prod} (C_2)). \)
2) \( \text{prod} (\text{cmax}_S (C_1, C_2)) = \text{prod} (C_1) \cup \text{prod} (C_2). \)
3) \( \text{cmin}_S (C_1, C_2) = Y_S (\text{prod} (C_1) \cap \text{prod} (C_2)). \)
4) \( \text{cmin}_S (C_1, C_2) = \min (C_1, C_2) = \text{ext} (\text{dom} (C_1) \cap \text{dom} (C_2)). \)
5) \( \text{cmin}_S (C_1, C_2) = (C_1 \cap \text{dom} (C_2)) \cup (C_2 \cap \text{dom} (C_1)). \)
6) \( \text{prod} (\text{cmin}_S (C_1, C_2)) = \text{prod} (C_1) \cap \text{prod} (C_2). \)

**Proof:** a) Let \( C = Y_S (\text{prod} (C_1) \cup \text{prod} (C_2)). \)

Clearly, \( C \) is extremal and contains \( \max (C_1, C_2) \) (cf. III.5.4). Hence \( C \succ C_1 \) and \( C \succ C_2 \). By III.5.3, \( \text{prod} (C) \subseteq \text{prod} (C_1) \cup \text{prod} (C_2) \). By construction, \( \text{prod} (C) \subseteq \text{prod} (C_1) \cup \text{prod} (C_2) \), and 2 holds.

Assume \( x \in S \) and \( \text{prod} (x) \subseteq \text{prod} (C) \).

If \( x \notin \text{prod} (C_1) \cup \text{prod} (C_2) \), then \( x \in C \) by definition of \( C \). Otherwise \( x \in \text{dom} (C_1) \cup \text{dom} (C_2) \subseteq \text{dom} (C) \). Hence \( C \) is \( S \)-complete by III.6.3.

Assume \( C' \) \( S \)-complete, \( C' \succ C_1 \), \( C' \succ C_2 \). Let \( c \in C \). Then

\[
\text{prod} (c) \subseteq \text{prod} (C) = \text{prod} (C_1) \cup \text{prod} (C_2).
\]

But by III.5.3, \( \text{prod} (C_1) \cup \text{prod} (C_2) \subseteq \text{prod} (C') \). Hence \( c \in \text{dom} (C') \) by III.6.2. Therefore \( C \prec C' \), and \( C \) is the l. u. b. of \( C_1 \) and \( C_2 \) in \( \text{SCOMP} \).

b) Let \( C = Y_S (\text{prod} (C_1) \cap \text{prod} (C_2)). \)

\[b \] 1) Let \( c \in C \). Then \( \text{prod} (c) \subseteq \text{prod} (C_1) \cap \text{prod} (C_2) \) implies by III.6.2 \( c \in \text{dom} (C_1) \cap \text{dom} (C_2) \). Since \( c \notin \text{prod} (C_1) \cap \text{prod} (C_2) \), we have either \( c \notin \text{prod} (C_1) \) and \( c \in C_1 \cap \text{dom} (C_2) \), or \( c \notin \text{prod} (C_2) \) and \( c \in C_2 \cap \text{dom} (C_1) \). Therefore \( C \subseteq (C_1 \cap \text{dom} (C_2)) \cup (C_2 \cap \text{dom} (C_1)) \). The reverse inclusion holds by construction, and property 5 is proved.

Revue Française d'Automatique, Informatique et Recherche Opérationnelle
2) We now have 
\[ \text{dom}(C) = \text{dom}(C_1 \cap \text{dom}(C_2)) \cup \text{dom}(C_2 \cap \text{dom}(C_1)), \]
and therefore \( \text{dom}(C) = \text{dom}(C_1) \cap \text{dom}(C_2); \) (cf. III.3.2). Since \( C \) is extremal by construction, property 4 holds.

3) To show that \( C \) is \( S \)-complete, we apply III.6.1. Assume \( x \in S \cdot \text{dom}(C) \). Three cases are possible:

Case 1: \( x \notin \text{dom}(C_1) \) and \( x \in \text{dom}(C_2) \). Then \( \text{dom}(x) \cap C_1 \neq \emptyset, \text{dom}(x) \cap (C_1 \cap \text{dom}(C_2)) \neq \emptyset \) and \( \text{dom}(x) \cap C \neq \emptyset. \)

Case 2: \( x \notin \text{dom}(C_1) \) and \( x \in \text{dom}(C_2) \). Symmetric to case 1.

Case 3: \( x \notin \text{dom}(C_1) \) and \( x \notin \text{dom}(C_2) \). Let \( c_1 \in \text{dom}(x) \cap C_1 \). If \( c_1 \in \text{dom}(C_2), \) then \( c_1 \in C. \) Otherwise \( \text{dom}(c_1) \cap C_2 \neq \emptyset, \text{dom}(x) \cap (C_2 \cap \text{dom}(C_1)) \neq \emptyset \) and \( \text{dom}(x) \cap C \neq \emptyset. \)

Eventually, \( C \) being the g. l. b. of \( C_1 \) and \( C_2 \) in \( (\text{EXT}, \prec) \) is also their g. l. b. in \( (\text{SCOMP}, \prec). \)

III.8. Bottom-up computations and induction techniques

We do not define precisely "bottom-up algorithms" or "bottom-up computation rules". However we remark that there is always a way of effectively producing every point of \( Y(\Phi) \) when \( D \) is denumerable:

III.8.1. Proposition: Let \( D = \{ d_n, \ n \in N \} \), let \( \Phi : \mathcal{P}(D) \rightarrow \mathcal{P}(D) \) be monotonic and continuous. Let \( X_n \) be the sequence of subsets of \( D \) defined by:

\[ X_0 = \emptyset; \]
\[ X_n = X_{n-1} \cup \{ d_i \} \text{ if there exist a least integer } i \text{ such that } d_i \notin X_{n-1} \text{ and } d_i \in \Phi (X_{n-1}); \]
\[ X_n = X_{n-1} \text{ otherwise.} \]

Then \( \{ X_n, \ n \in N \} \) is a b. u. c. s., and has limit \( Y(\Phi). \)

Proof: The sequence is a b. u. c. s. by construction, and \( \bigcup X_n \subseteq Y(\Phi) \) holds by induction on \( n. \)

Assume that \( \bigcup X_n \) is not a fixpoint; then there would exist an integer \( p \) such that \( d_p \notin \bigcup X_n \) and \( d_p \in \Phi (\bigcup X_n). \) By continuity, \( d_p \in \bigcup \Phi (X_n). \) Let \( q \) be such that \( d_p \in \Phi (X_q). \) Then \( d_p \in \Phi (X_r) \) for \( r \geq q, \) and there exists a sequence \( d_{i_q}, d_{i_{q+1}}, \ldots, d_{i_{q+i}}, \ldots \) of distinct points of \( D \) such that for every \( l \in N: \)

\[ d_{i_{q+l}} \in X_{q+l}, \quad d_{i_{q+l}} \notin X_{q+l-1}, \quad d_{i_{q+l}} \in \Phi (X_{q+l-1}), \quad i_{q+l} < p. \]
This is impossible, since there only exist a finite number of integers less than \( p \).

Top-down computation rules do not avoid recomputation of intermediate values (except by using memo-functions, which may involve additional tests and a waste of storage). It seems that when using “uniform bottom-up computation rules”, we cannot avoid to compute useless intermediate values; there is probably no uniform way of characterising the domain of a given point.

We now develop some induction techniques which might allow to characterise the domains in particular examples. All the induction principles we present derive from the following one:

**III. 8.2. PROPOSITION:** Let \( \Phi \) be continuous. Let \( h : D \to \mathcal{P}(D) \) be such that \( \forall x \in \Phi^n(\emptyset) \ h(x) \subseteq \Phi^{n-1}(\emptyset) \). Let \( P \) be a property such that for every \( x \in Y(\Phi) \), if \( P \) is true for every member of \( h(x) \), then \( P \) is true of \( x \). Then one can conclude that \( P \) is true of every member of \( Y(\Phi) \).

\[
\forall x \in Y(\Phi), \left[ \forall y \in h(x) \ P(y) \right] \Rightarrow P(x) \quad \leftarrow \quad \forall x \in Y(\Phi) \quad P(x).
\]

**Proof:** By induction on \( n \): Assume \( \forall x \in Y(\Phi) \left[ \forall y \in h(x) \ P(y) \right] \Rightarrow P(x) \). Let \( x \in \Phi(\emptyset) \). Then \( h(x) = \emptyset \), and \( P(x) \) holds.

Assume \( \forall y \in \Phi^n(\emptyset) \ P(y) \). Let \( x \in \Phi^{n+1}(\emptyset) \). Then \( h(x) \subseteq \Phi^n(\emptyset) \) implies \( \forall y \in h(x) \ P(y) \), and therefore \( P(x) \).

Some possible choices for \( h(x) \) are:

1) \( h(x) = \Phi^{n-1}(\emptyset) \). We obtain the usual truncation induction principle (Morris [8]).

2) \( h(x) = \text{prod}(x) \). We obtain the usual structural induction principle applied to the relation \(<\).

3) \( h(x) = \text{ext}(\text{prod}(x)) \).

4) \( h(x) = \text{mp}(x) \) if \( \Phi \) is deterministic. This principle seems to be the most useful when dealing with recursive programs: the set \( \text{mp}(x) \) is directly given by the elementary predicates of the translated formula.

An example of application of this last principle is given in the next section.

**IV. TIME AND SPACE OPTIMALITY. APPLICATION TO ACKERMANN PROGRAM**

**IV. 1. Time and space optimality of storage configuration sequences**

Given a recursive program, there exist several different implementations, using either recursion mechanism or production mechanisms. It is natural to compare these implementations. We shall use as time and space complexity measures the number of recursion or production steps and the number of intermediate values respectively.
A deterministic production function.

The arrows represent the relation $x \in mp(y)$. The relation $\leq$ is obtained by transitivity.

```
--- --- --- _ dom(x)
O  mp(x)
++  an extremal set.
```

Our formalism provides naturally these evaluations for the implementations using production mechanisms: let $\{ Y_n, n \in N \}$ be a storage configuration sequence which computes a point $(x, z)$.

n° mars 1976
The amount of space used by this sequence is the maximal cardinality of $Y_n$.

The sequence is optimal with respect to time if its limit is the domain of $(x, z)$ and if it is simple (cf. III.1): that is no intermediate value is computed twice, only useful intermediate values are computed.

The sequence is optimal iff it is optimal with respect to time, and iff it is optimal with respect to space among those s. c. s. which are optimal with respect to time. (It may be the case that one can use less space by recomputing some intermediate values.)

An algorithm realising an implementation is said optimal iff for every argument, the s. c. s. it determines is optimal.

We emphasise the fact that optimality is defined with respect to the implementation of a given recursive program, and not with respect to the function it computes. For instance, the Fibonacci function can be defined by the two following programs:

$P_1. \ \text{fib} (n) = \begin{cases} \text{if } n = 0 \vee n = 1 & \text{then } 1 \\ \text{else fib} (n-1)+\text{fib} (n-2) \end{cases}$

$P_2. \ \text{fib} (n) = \begin{cases} \text{if } n = 0 \vee n = 1 & \text{then } 1 \\ \text{else if } n = 2 & \text{then } 2 \\ \text{else fib} (n-1)+3\times\text{fib} (n-2)+\text{fib} (n-3))/2. \end{cases}$

An optimal implementation of $P_2$ is not optimal for the computation of $\text{fib} (n), n > 2$: there always exist better implementations of $P_1$. In fact, no implementation of $P_1$ is an implementation of $P_2$ and conversely.

IV.2. Application: Ackermann program

The purpose of the next sections is to show that Rice's algorithm [11] realises an optimal bottom-up implementation of Ackermann program.

Ackermann program is defined on $N \cup \{ \bot \}$ by the following program scheme:

$A (\cdot, n) = \begin{cases} \text{if } m = 0 & \text{then } n+1 \\ \text{else if } n = 0 & \text{then } A (m-1, 1) \\ \text{else } A (m-1, A (m, n-1)) \end{cases}$

The tests $m = 0$ and $n = 0$ imply

$\forall m, n \in N \ A (\bot, \bot) = A (m, \bot) = A (\bot, n) = \bot.$

Hence no bound variable can be set to $\bot$, and the associated production function may be simply defined by:

$\Phi (X) = \{(m, n, z) \in N^3 \mid \begin{cases} \text{if } m = 0 & \text{then } z = n+1 \\ \text{else if } n = 0 & \text{then } (m-1, 1, z) \in X \\ \text{else } \exists y (m-1, y, z) \in X \land (m, n-1, y) \in X \end{cases} \}.$
As the function $\theta$ in II.5, the function $\Phi$ is not deterministic, but is such
that every point in $Y(\Phi)$ has a unique minimal producer in $Y(\Phi)$.

- If $m = 0$, then $mp(m, n, z) = \emptyset$.
- If $m \neq 0$ and $n = 0$, then $mp(m, n, z) = \{(m-1, 1, z)\}$.
- If $m \neq 0$ and $n \neq 0$, then
  \[ mp(m, n, z) = \{(m, n-1, A(m, n-1)), (m-1, A(m, n-1), z)\}. \]

IV. 2.1. **Proposition:** 1) The function is total: $\forall m, n \in \mathbb{N}, A(m, n) \in \mathbb{N}$.

2) $\forall m, n, n' \in \mathbb{N}, n' > n \Rightarrow A(m, n') > A(m, n)$.

3) $\forall m, m', n \in \mathbb{N}, m' > m \Rightarrow A(m', n) > A(m, n)$.

**Proof:** A proof of 1 is given in Manna [5] (p. 410); it uses the well-foundedness of the lexicographical ordering $\leq_l$ on $\mathbb{N}^2$: $(x, y) \leq_l (x', y')$ holds iff $x < x'$ or $x = x'$ and $y \leq y'$. Properties 2 and 3 are well-known, and can be easily proved by induction on $(N, \leq)$ and $(N^2, \leq_l)$.

We now characterise the domains:

IV. 2.2. **Proposition:** Let $(m, n, z) \in Y(\Phi)$. Then:

1) if $m = 0$, then $\text{dom} (m, n, z) = \{(m, n, z)\}$;

2) if $m \neq 0$, then
  \[ \text{dom} (m, n, z) = \{(m', n', z') \in Y(\Phi) \mid m' \leq m \wedge z' \leq z\} \cup \{0, 0, 1\}. \]

**Proof:** By induction on the minimal producers (III.8.2).

Let $D(m, n, z) = \{(m', n', z') \in Y(\Phi) \mid m' \leq m \wedge z' \leq z\} \cup \{0, 0, 1\}$.

**Case** $m = 0$: then $mp(m, n, z) = \emptyset$ implies property 1.

**Case** $m > 1, n \neq 0$: let $z 1 = A(m, n-1)$; assume $\text{dom}(m, n-1, z 1) = D(m, n-1, z 1)$ and $\text{dom}(m-1, z 1, z) = D(m-1, z 1, z)$.

Then:
  \[ \text{dom} (m, n, z) = D(m, n-1, z 1) \cup D(m-1, z 1, z) \cup \{(m, n, z)\}. \]

Proposition IV.2.1 implies $z 1 < z$, and therefore $\text{dom} (m, n, z) \subseteq D(m, n, z)$.

Let $(m', n', z') \in D(m, n, z)$:

- If $m' = m$, then IV.2.1 implies $n' \leq n$; if $n' = n$, then $(m', n', z') = (m, n, z)$, otherwise $(m', n', z') \in D(m, n-1, z 1)$.

- If $m' < m$, then $(m', n', z') \in D(m-1, z 1, z)$.

In both cases, $(m', n', z') \in \text{dom} (m, n, z)$. Eventually,
  \[ \text{dom} (m, n, z) = D(m, n, z). \]

**Cases** $m \neq 0, n = 0$ and $m = 1, n \geq 0$: similar proofs.

n° mars 1976
IV. 3. A bottom-up implementation of Ackermann program

The given PASCAL program implements a variant of Rice’s algorithm [11].

Remark: The arrays PLACE and VAL should have dimension 0..M and 0..M+1 (PASCAL does not allow flexible arrays).

If \( M = 0 \), the program is directly exited with \( \text{ACK} = N+1 \); if \( N = 0 \), the argument \((M, N)\) is transformed into \((M-1, 1)\); therefore we only investigate the case \( M > 0, N > 0 \).

A termination proof is given in Manna [2] (p. 200-202), and we only give an informal correctness proof; a formal one could be constructed by using the defining predicate of \( \Phi \) in assertions.

The algorithm uses the fact that for every \( z > 1 \), the arguments such that \( A(m, n) = z \) form a finite sequence of the form \((0, n 1), (1, n 2), \ldots, (k, nk)\).

Points in the graph are represented by triples \((I, \text{PLACE}[I], \text{VAL}[I])\). A new point is created each time one of the blocks \( A \) or \( B \) is exited. The variable \( T \) count the number of such exists, and will be called the time counter: \( T \) is useless for the algorithm itself. For every variable \( X \), let \( X_t \) denote the value of \( X \) at time \( t \); let \( d_t \) be the new triple introduced, let \( Y_t \) be the set of intermediate triples “in storage”:

\[
d_t = (I_t, \text{PLACE}[I_t], \text{VAL}[I_t]),
Y_t = \{(J, \text{PLACE}[J], \text{VAL}[J]), 0 \leq J \leq M \land \text{VAL}[J] > 0\}.
\]
function ACK(M:INTEGER;N:INTEGER):INTEGER;
var VAL,PLACE:array[0..10]of INTEGER;
I,Z,T:INTEGER;
begin
if M=0 then ACK:=N+1
else begin
  if N=0 then
    begin
      M:=M-1;
      N:=1;
      end;
  for I:=1 to M do %INITIALISATIONS%
    begin
      VAL[I]:=1;
      PLACE[I]:=-1;
    end;
  PLACE[0]:=0;
  T:=0;
  Z:=2;
  repeat %LOOP ON THE FUNCTION VALUE%
    begin
      %BLOCK A%
      PLACE[0]:=PLACE[0]+1;
      VAL[0]:=Z;
      T:=T+1;
    end %A%
  I:=1;
  while(VAL[I]=PLACE[I-1])and(I<=M)do
    begin
      %BLOCK B%
      VAL[I]:=Z;
      PLACE[I]:=PLACE[I]+1;
      I:=I+1;
      T:=T+1;
    end %B%;
  Z:=Z+1;
  until PLACE[M]=N %END TEST%;
  ACK:=VAL[M];
end;
end;

We are in fact not interested in the program itself, but in the s. c. s. \( Y_t \) it defines.

Let \( T_0 \) be the value of \( T \) when the program is exited.

IV.3.1. PROPOSITION: For every \( M, N > 0 \), the sequence \( \{ Y_t, 1 \leq t \leq T_0 \} \) is a simple s. c. s. with limit \( \text{dom} (M, N, A (M, N)); ACK (M, N) = A (M, N) \).

Proof: 1) \( Y_t \) is a s. c. s.

- Case \( I_t = 0 \): block \( A \) was exited, and \( VAL[0] = PLACE[0] + 1 \); hence \( d_t \in \Phi(\emptyset) \).

n° mars 1976
Case \( I_t \neq 0 \), \( \text{PLACE} \{ I_t \}_t = 0 \): block \( B \) was exited at time \( t \), and also at time \( t-1 \): the following relations hold:

\[
I_{t-1} = I_t - 1; \quad Z_{t-1} = Z_t.
\]

\[
\text{VAL} \{ I_t \}_{t-1} = 1; \quad \text{PLACE} \{ I_t \}_{t-1} = -1.
\]

\[
\text{VAL} \{ I_t \}_t = Z_t.
\]

Since the \texttt{do} loop was entered, we also have:

\[
\text{PLACE} \{ I_t - 1 \}_{t-1} = \text{VAL} \{ I_t \}_{t-1} = 1.
\]

Hence \( d_{t-1} = (I_{t-1}, \text{PLACE} \{ I_{t-1} \}_{t-1}, \text{VAL} \{ I_{t-1} \}_{t-1}) = (I_t - 1, 1, Z_t) \). Therefore \( d_t \in \Phi (\{ d_{t-1} \}) \subseteq \Phi (Y_{t-1}) \).

\textbf{Case} \( I_t \neq 0 \), \( \text{PLACE} \{ I_t \}_t \neq 0 \): similar proof: one easily shows that \( Y_{t-1} \) contains the two points

\[
(I_t, \text{PLACE} \{ I_t \}_t - 1, \text{VAL} \{ I_t \}_{t-1}) \text{ and } (I_t - 1, \text{VAL} \{ I_t \}_{t-1}, Z_t).
\]

This implies \( d_t \in \Phi (Y_{t-1}) \).

2) \( Y_t \) is simple: the pairs \((Z_t, I_t)\) strictly increase with respect to the lexicographic ordering \( \leq \) on \( \mathbb{N}^2 \): no value is computed twice.

3) \( \text{ACK} (M, N) = A (M, N) \): since \( Y_t \) is a s. c. s., the results of the first sections imply for every \( t \) \( \text{VAL} \{ I_t \}_t = A (I_t, \text{PLACE} \{ I_t \}_t) \). When the program is exited, we have \( I_{t_0} = M \), \( \text{PLACE} \{ I_{t_0} \}_{t_0} = N \) and

\[
\text{ACK} = \text{VAL} \{ I_{t_0} \}_{t_0} = A (M, N).
\]

4) \( \bigcup_{1 \leq t \leq T_0} Y_t = \text{dom} (M, N, A (M, N)) \): Let \( S = \bigcup_{1 \leq t \leq T_0} Y_t \); for every \( t \), we have \( I_t \leq M \) and \( Z_t \leq Z_{t_0} \). By IV.2.2, \( d_t \in \text{dom} (M, N, A (M, N)) \).

Hence \( S \subseteq \text{dom} (M, N, A (M, N)) \).

But \( S \) is s. r. (III.1.1, III.2.4), and contains \((M, N, A (M, N))\): therefore \( S \supseteq \text{dom} (M, N, A (M, N)) \). \( \square \)

\textbf{IV.4. Optimality of the s. c. s. defined by Rice’s algorithm}

Proposition IV.3.1 shows that the algorithm is optimal with respect to time. Given \( x, y > 0 \), the simple s. c. s. it defines uses \( x + 1 \) space units to compute \( A (x, y) \). Our purpose is to show that no simple s. c. s. can use less than \( x + 1 \) space units (we only consider s. c. s. of the form \( \{ Y_n, 0 \leq n \leq N \} \) such that \( (x, y, A (x, y)) \in Y_N \)).

We use a technique similar to that developed by Cook and Sethi in [3].

\textbf{Definition:} Let \( x \in Y (\Phi) \). A \textit{path from} \( x \) to \( \Phi (\emptyset) \) is a sequence \( x_0, \ x_1, \ \ldots, \ x_n \) of points of \( Y (\Phi) \) such that \( x_0 = x \), \( x_n \in \Phi (\emptyset) \) \( \forall p, \ 1 \leq p < n, \ x_{p+1} \in mp (x_p) \).
The proof relies on the following lemma:

**IV.4.1. Lemma:** Let $\Phi$ be deterministic, let $x \in Y(\Phi)$, let \(\{ Y_n, 0 \leq n \leq N \}\) be a simple s. c. s. with limit $\text{dom}(x)$, and assume $x \in Y_N$. Let \(\{ X_n, 0 \leq n \leq N \}\) be the corresponding b. u. c. s.: $X_n = \bigcup_{0 \leq p \leq n} Y_p$. Let $k$ be the smallest integer such that $X_k \supseteq \text{dom}(x) \cap \Phi(\emptyset)$. Then for every path $P = \{ x_0, x_1, \ldots, x_q \}$ from $x$ to $\Phi(\emptyset)$, the following property holds:

$$
\forall k', \quad k \leq k' \leq N, \quad P \cap Y_{k'} \neq \emptyset.
$$

**Proof:** Assume the property false. Then there exists a path $P = \{ x_0, x_1, \ldots, x_q \}$ and an index $k', k \leq k' \leq N$, such that $P \cap Y_{k'} = \emptyset$.

The set $P \cap X_k$ is not empty, since it contains at least $x_q$. Let $q'$ be the least integer such that $x_q \in X_k$.

We cannot have $q' = 0$: otherwise $x \in X_k$, and $x \in Y_N$ would imply $x \in Y_k$, and $P \cap Y_k = \emptyset$ (cf. definition of simplicity).

Since $x_q \notin X_k$ and $x_q \in Y_m$ for some $m < k'$, $x_q \notin Y_m$. holds for $m' \geq k'$.

Consider now the point $x_{q'-1}$: by hypothesis, $x_{q'} \in mp(x_{q'-1})$ and $x_{q'-1} \notin X_k$ hold. This implies for every $m' \geq k'$ mp ($x_{q'-1}$) $\notin Y_{m'}$, and hence $x_{q'-1} \notin Y_{m'+1}$. Finally, $x_{q'-1} \notin Y_m$ for all $m \leq N$. This is impossible, since $x_{q'-1} \in \text{dom}(x)$ and $\bigcup_m Y_m = \text{dom}(x)$.

Let us come back to Ackermann production function. Given $(x, y, z) \in Y(\Phi)$, our purpose is now to construct $x+1$ paths which are distinct in $X_k$ for every simple s. c. s. We use for this the construction of a $\text{dom}(x)$-complete set $C$. Let us define the following transformations:

**Definitions:** Let $(x, y, z) \in Y(\Phi)$, $x > 0$, $y > 0$.

$$
\mu(x, y, z) = (x-1, y, z) \quad \text{with} \quad z = A(x-1, y, 1).
$$

$$
\nu(x, y, z) = (x, y-1, z + 1) \quad \text{with} \quad z + 1 = A(x, y-1).
$$

$$
C(x, y, z) = \{ c_m(x, y, z) \mid 0 \leq m \leq x \}
$$

with

$$
c_m(x, y, z) = \nu \mu^{x-m}(x, y, z) \quad \text{for} \quad 0 < m \leq x,
$$

$$
c_0(x, y, z) = \mu^x(x, y, z).
$$

We denote $c_m(x, y, z) = (m, c_m^2, c_m^3)$.

**IV.4.2. Lemma:** \(\forall x, \ y > 0, \ A(x, \ y-1) < A(x-1, \ A(x, \ y-1)-1)\).

**Proof:** Left to the reader.
IV. 4.3. PROPOSITION: Let $x, y > 0$; the set $C(x, y, z)$ is $\text{dom}(x, y, z)$-complete.

Proof: by IV. 2.1, IV. 2.2, $C(x, y, z) \subseteq \text{dom}(x, y, z)$ holds.

By lemma IV. 4.2, we have $c_2 x < c_2 x^2 < \ldots < c_2 x^3 < c_3 = z = c_0^3$; hence $C(x, y, z)$ is extremal.

Let $(x', y', z') \in \text{dom}(x, y, z)$; if $x' > 0$, then either $(x', y', z') \in \text{dom}(c_x)$, or $c_x \in \text{dom}(x', y', z')$. If $x' = 0$, then either $(x', y', z') = c_0$ or $(x', y', z') \in \text{dom}(c_1)$. Therefore $C(x, y, z)$ is $\text{dom}(x, y, z)$-complete. □

IV. 4.4. PROPOSITION: Let $(x, y, z) \in Y(\Phi), x, y > 0$; let $\{Y_n, 0 \leq n \leq N\}$ be a simple s. c. s. with limit $\text{dom}(x, y, z)$, let $X_n, 0 \leq n \leq N$ be the associated b. u. c. s.; let $k$ be the least integer such that

$$X_k \supseteq \text{dom}(x, y, z) \cap \Phi(\emptyset).$$

Then $(x', y', z') \in X_k$ implies $x' = 0$.

Proof: Assume $(x', y', z') \in X_k, x' \neq 0$; then $X_{k-1} \supseteq \text{prod}(x', y', z)$. But by IV. 2.2, $\text{prod}(x', y', z) \cap \Phi(\emptyset) = \text{prod}(x, y, z) \cap \Phi(\emptyset)$; this contradicts the minimality of $k$.

We now show the final result:

IV. 4.5. THEOREM: Let $\Phi$ denote the Ackermann production function, let $(x, y, z) \in Y(\Phi), x > 0, y > 0$. Let $\{Y_n, 0 \leq n \leq N\}$ be a simple s. c. s. with limit $\text{dom}(x, y, z)$. Then $\max_{0 \leq n \leq N} \text{card}(Y_n) \geq x + 1$.

Proof: To each $c_m$ we associate a path $P_m$ from $(x, y, z)$ to $\Phi(\emptyset)$ in the following way: we first apply $\mu x - m$ times and then $\nu$, to reach $c_m$; we then apply $\mu m$ times to reach $\Phi(\emptyset)$, thus keeping the third coordinate equal to $c_3^m$.

Let a denote $(x, y, z)$:

$P_0 = (a, \mu a, \mu^2 a, \ldots, \mu^{x-1} a, \mu^x a = c_0),$

$P_1 = (a, \mu a, \mu^2 a, \ldots, \mu^{x-1} a, \nu \mu^{x-1} a = c_1, \mu c_1)$

$\vdots$

$P_m = (a, \mu a, \mu^2 a, \ldots, \mu^{x-m} a, \nu \mu^{x-m} a = c_m, \mu c_m, \mu^2 c_m, \ldots, \mu^m c_m)$

$P_x = (a, \nu a = c_x, \mu c_x, \mu^2 c_x, \ldots, \mu^x c_x)$.

Let $m < m'$; by IV. 4.3, $P_m$ and $P_{m'}$ only intersect at points $a, \mu a, \mu^2 a, \ldots, \mu^{x-m'} a$, and by IV. 4.4, these points do not belong to $X_k$. Lemma IV. 4.1 implies the result. □

Remark: The fact that $C(x, y, z)$ is $\text{dom}(a)$-complete is not essential to the proof; however looking for $S$-complete sets seems to be a good heuristic in this class of problems.
This result has been improved in [1]. The s. c. s. associated to Rice's algorithm is directly defined by recurrences instead of a program. Using a similar method, we show that it is optimal with respect to storage among all possible s. c. s., thus even allowing recomputation of intermediate values.

Diagram 3

The set $C(3, 1, 13)$, and the corresponding paths $P_0, P_1, P_2, P_3$.

ACKNOWLEDGEMENTS

I am grateful to Pr. M. Nivat for his advice and encouragement. Many thanks to Bruno Courcelle, Philippe Flajolet, Gérard Huet, Jean-Jacques Lévy, Charles Rackoff, and particularly Jean-Marc Steyaert, who helped me in this work.

REFERENCES


