REVUE FRANÇAISE D'AUTOMATIQUE INFORMATIQUE
RECHERCHE OPÉRATIONNELLE. INFORMATIQUE THÉORIQUE

G. BERRY

Bottom-up computation of recursive programs

Revue française d'automatique informatique recherche opérationnelle. Informatique théorique, tome 10, n° R1 (1976), p. 47-82

<http://www.numdam.org/item?id=ITA_1976__10_1_47_0>

R.A.I.R.O. Informatique Théorique
(vol. 10, n° 3, mars 1976, p. 47 à 82)

BOTTOM-UP COMPUTATION OF RECURSIVE PROGRAMS (*)

by G. BERRY (1)

Communicated by M. NIVAT

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

(*) Reçu décembre 1975.

(1) École nationale supérieure des Mines, Paris, IRIA/LABORIA.

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 $x \prec y$ if "x is an intermadiete 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 $\prec$ then besomes 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 $\mathscr{P}(D)$ 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 card (X).

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

The production mechanism can be illustrated by the Fibonacci program:

$$\mathrm{Fib}(n) = \mathit{if}\ n = 0 \vee n = 1\ \mathit{then}\ 1\ \mathit{else}\ \mathrm{Fib}(n-1) + \mathrm{Fib}(n-2).$$

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 $\langle 4, \mathrm{fib}(4) = 5 \rangle$ and $\langle 5, \mathrm{fib}(5) = 8 \rangle$, we can generate the new point $\langle 6, \mathrm{fib}(6) = 5+8 = 13 \rangle$. More precisely, the set produced by X in this example can be defined by

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

The purpose of this section is to formalise a translation of a recursive program into a predicate defining the production function Φ. This translation is such that the least fixpoint of Φ 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 $\perp$, the *undefined element*, and ordered by the relation $x \subseteq y$ iff $x = \perp$ or $x = y$.

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

$$\forall x, x' \in D^p, \quad x \subseteq x' \Leftrightarrow \forall i \in \{1, 2, \ldots, p\}, \qquad x_i \subseteq x'_i.$$

In D^p, two elements $\bar{x}$ and $\bar{x}'$ always have a greatest lower bound (g. l. b.) $\bar{x} \wedge \bar{x}'$ defined by $(\bar{x} \wedge \bar{x}')_i = \perp$ if $x_i \neq x'_{i'}$, $(\bar{x} \wedge \bar{x}')_i = x_i$ if $x_i = x'_i$.

If two elements $\bar{x}$ and $\bar{x}'$ have a least upperbound (l. u. b.) $\bar{x} \vee \bar{x}'$, they are said to be *joinable* (abbreviated $\bar{x} \uparrow \bar{x}'$).

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

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

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

For each $\bar{x}$, either $f(x)$ is undefined, or $f(\bar{x}) = g(\bar{x})$.

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

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

Let us now define the recursive program schemes.

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

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

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

A recursive program scheme Σ is a set of equations

$$\Sigma \begin{cases} f_i(x_1, x_2, \ldots, x_{\rho(f_i)}) = \tau_i, \\ \qquad i = 0, 1, \ldots, N \end{cases}$$

where the τ_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 $I(b) \in \Delta^{\rho(b)}$.

We call *recursive program* a pair (Σ, I).

NOTATION : Let (Σ, I) be a recursive program with N equations and k variables, let $g_1 \in \Delta^{\rho(f_1)}$, $g_2 \in \Delta^{\rho(f_2)}$, $\ldots$, $g_N \in \Delta^{\rho(f_N)}$, let τ be any term.

We denote by τ^I the functional from $\Delta_I^{\rho(f_1)} \times \Delta_I^{\rho(f_2)} \times \ldots \times \Delta_I^{\rho(f_N)}$ to Δ_I^k defined by the λ-expression $\lambda f_1, f_2, \ldots, f_N.\tau$.

Given $\bar{\alpha} \in D_I^k$, $\tau^I(\bar{g})(\bar{\alpha})$ denotes the value computed by τ on $\bar{\alpha}$ when f_1, $f_2, \ldots, f_N$ are interpreted by $g_1, g_2, \ldots, g_N$:

When no confusion arises, τ^I will be abbreviated in τ.

THEOREM: *A recursive program* (Σ, I) *determines a functional* φ *from* $\Delta^{\rho(f_1)} \times \Delta^{\rho(f_2)} \times \ldots \times \Delta^{\rho(f_N)}$ *to itself:* $\varphi(\bar{g}) = (\tau_1^I(\bar{g}), \tau_2^I(\bar{g}), \ldots, \tau_N^I(\bar{g}))$.

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

This fixpoint is limit of Kleene's sequence $K_p = \langle k_p^1, k_p^2, \ldots, k_p^N \rangle$ *such that* $K_0 = \langle \perp, \perp, \ldots, \perp \rangle$ and $K_{p+1} = \varphi(K_p)$ *for all* $p \geqq 0$.

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 $\tau^p(\perp)$.

I.3. Monotonic functions over subsets

DEFINITIONS : $\Phi : \mathscr{P}(A) \rightarrow \mathscr{P}(A)$ is *monotonic* (increasing) iff:

$$\forall X, Y \subseteq A, \quad X \subseteq Y \Rightarrow \Phi(X) \subseteq \Phi(Y).$$

– $X \subseteq A$ Φ-*produces* x iff $x \in \Phi(X)$.

– X *is a minimal* Φ-*producer of* x iff X produces x and no proper subset of X produces x.

– X is a *fixpoint* of Φ iff $X = \Phi(X)$.

– Φ is *continuous* iff for every increasing sequence

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

I.3.1. THEOREM (Knaster-Tarski): *Every monotonic function* $\Phi : \mathscr{P}(A) \rightarrow \mathscr{P}(A)$ *has a least fixpoint* $Y(\Phi)$. *If* Φ *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].

I.3.2. PROPOSITION : Φ *is continuous iff for every point* x *of* $\Phi(A)$, *all minimal producers of* x *are finite.*

I.4. Translation of a recursive program into a monotonic predicate

According to I.2, we only translate single equation programs. Let

$$\Sigma : \quad f(x_1, x_2, \ldots, x_\rho) = \tau(f)(x_1, x_2, \ldots, x_\rho)$$

be a recursive scheme with n occurences of f in τ. We first label these occurrences by the integers 1, 2, ..., 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 $T^i = f^i(t_1^i, t_2^i, \ldots, t_\rho^i)$ the subterm of τ corresponding to the occurrence i of f, and by T^0 the term τ itself.

We define a *transformation* $\mathscr{T}$ on labelled terms which consists in replacing a subterm $f^i(t_1^i, t_2^i, \ldots, t_n^i)$ by the bound variable y_i:

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

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

c) $\mathscr{T}(f^i(t_1, t_2^i, \ldots, t_\rho^i)) = y_i$.

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

For every integer i, $1 \leqq i \leqq n$, let I_i denote labels of bound variables that occur in subterms $\mathscr{T}(t_j^i)$ for $1 \leqq j \leqq \rho$. Let I_0 denote labels of bound variables that occur in $\mathscr{T}(\tau)$. The I_i determine a partition of $\{ 1, 2, \ldots, n \}$; I_i is empty for the innermost occurences 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 occurences of f in T^i.

Let us now define the translated predicate P_Σ:

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

$$p_i = [(\mathscr{T}(t_1^i)(\bar{x}, \bar{y}), \mathscr{T}(t_2^i)(\bar{x}, \bar{y}), \ldots, \mathscr{T}(t_\rho^i)(\bar{x}, \bar{y}), y_i) \in X \vee y_i = \perp]$$

The predicate P_Σ associated to Σ is defined by

$$P_\Sigma(x_1, x_2, \ldots, x_\rho, z, X) = \exists y_1, y_2, \ldots, y_n, (z \neq \perp) \wedge (z = \mathscr{T}(T^0)) \wedge p_1 \wedge p_2 \wedge \ldots \wedge p_n.$$

Example: $\Sigma : f(x_1, x_2) = a(f^1(b(x_1, x_2), c(f^2(x_2, x_1))))$.

$$P_\Sigma(x_1, x_2, z, X) = \exists y_1, y_2, z \neq \perp \wedge z = a(y_1) \wedge [(b(x_1, x_2), c(y_2), y_1) \in X \vee y_1 = \perp] \wedge [(x_2, x_1, y_2) \in X \vee y = \perp].$$

We now use P_Σ to define the production function Φ_Σ:

DEFINITIONS : Let Σ be a single-equation program scheme, let I be a discrete interpretation. *The production function* $\Phi_{(\Sigma, I)} : D_I^{p+1} \to D_I^{p+1}$ associated to the program (Σ, 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_Σ and $\Phi_{(\Sigma, I)}$ in P and Φ when no confusion arises.

Remark: If we take out the $y_i = \perp$, the translation is essentialy equivalent to the one given in Park [10].

I.4.1. PROPOSITION $\Phi_{(\Sigma, I)}$ *is monotonic and continuous.*

Proof: The elementary predicates are monotonic, and the definition of Φ involves no negation: Φ is monotonic.

Since the number of elementary predicates is finite, Φ is also continuous (I.3.2). □

I.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 $z \neq \perp$ and $z = b(x_1, x_2, \ldots, x_p)$.

DEFINITION : Let (Σ, I) be a recursive program. *A n-tuple*

$$\bar{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in D_I^n$$

is admissible for (x, z, X) iff it satisfies $z = \mathscr{T}(T^0) \wedge p_1 \wedge p_2 \wedge \ldots \wedge p_n$ (i. e. P_Σ without the quantifiers) as values of $y_1, y_2, \ldots, y_n$.

Then $(\bar{x}, z) \in \Phi(X)$ iff there exists an admissible n-tuple for (x, z, X).

We can now prove the correctness of the translation.

I.5.1. THEOREM: *Let* (Σ, I) *be a recursive program. Let* h *be a monotonic function from* D^p *to* D, *let* H *denote* $dpg(h)$. *Then:*

$$(\bar{x}, z) \in \Phi_{(\Sigma, I)}(H) \Leftrightarrow z \neq \perp \wedge z = \tau(h)(\bar{x}),$$

or equivalent

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

If a n-tuple $\bar{\alpha}$ *is admissible for* $(\bar{x}, z, H)$, *then*

$$\forall i \in \{1, 2, \ldots, n\}, \quad \alpha_i \subseteq T^i(h)(\bar{x}).$$

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

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

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

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

2 *a*) Case $nd(i) = 1$. The terms $\mathscr{T}(t_j^i)(\bar{x}, \bar{\alpha})$ in p_i only contain basic functions and variables of $\bar{x}$.

Hence $\mathscr{T}(t_j^i)(\bar{x}, \bar{\alpha}) = t_j^i(\bar{x})$. By construction of p_i, either $\alpha_i = \perp$, or

$$(t_1^i(\bar{x}), t_2^i(\bar{x}), \ldots, t_\rho^i(\bar{x}), \alpha_i) \in H,$$

which means

$$\alpha_i = h(t_1^i(\bar{x}), t_2^i(\bar{x}), \ldots, t_\rho^i(\bar{x})) = T^i(h)(\bar{x}).$$

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 $\mathscr{T}(t_j^i)(\bar{x}, \bar{\alpha})$ only contain basic functions, variables in $\bar{x}$ and bound variables of nesting depth less than k.

Induction hypothesis and monotonicity of the basic functions imply:

$$\forall j, \quad 1 \leqq j \leqq \rho, \qquad \mathscr{T}(t_j^i)(\bar{x}, \bar{\alpha}) \subseteq t_j^i(\bar{x}).$$

If $\alpha_i \neq \perp$, then

$$(\mathscr{T}(t_1^i)(\bar{x}, \bar{\alpha}), \mathscr{T}(t_2^i)(\bar{x}, \bar{\alpha}), \ldots, \mathscr{T}(t_\rho^i)(\bar{x}, \bar{\alpha}), \alpha_i) \in H;$$

by monotonicity of h : $(t_1^i(\bar{x}), t_2^i(\bar{x}), \ldots, t_\rho^i(\bar{x}), \alpha_i) \in H$, which means $\alpha_i = T^i(h)(\bar{x})$.

Same argument applies to z : $z = \perp$ and $z = \mathscr{T}(\tau)(\bar{x}, \bar{\alpha})$ imply $z = \tau(h)(\bar{x}, \bar{\alpha})$. □

I.5.2. COROLLARY: *Let* (Σ, I) *be a recursive program. Then for every* $m \geqq 0$ $\Phi^m(\emptyset) = dpg(\tau^m(\perp))$; $Y(\Phi) = dpg(Y(\Sigma, I))$.

If $(\bar{x}, z) \in \Phi^m(\emptyset)$, *and if* $\bar{\alpha}$ *is admissible for* $(x, z, \Phi^{m-1}(\emptyset))$, *then for all* $i \in \{1, 2, \ldots, n\}$:

$$\alpha_i \subseteq T^i(\tau^{m-1}(\perp))(\bar{x}) \subseteq T^i(Y(\Sigma, I))(\bar{x}).$$

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 $\tau^m(\perp)$, we know that for all $\bar{x}' \supseteq \bar{x}$, $(\bar{x}', z) \in \Phi^m(\emptyset)$. It may however happen that for some X, $(\bar{x}, z) \in \Phi(X)$

and $(\bar{x}', z) \notin \Phi(X)$. For instance:

$$\Sigma : \quad f(x_1, x_2) = \textit{if } x_1 = 0 \textit{ then } 0 \textit{ else } f(x_1 - 1, x_2)$$

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

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

I.6.1. PROPOSITION: 1) Ψ *is monotonic and continuous*

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

Proof: 1) Monotonicity follows from $X \subseteq Y \Rightarrow \hat{X} \subseteq \hat{Y}$, continuity from I.3.2.

2) By induction on n, using $\widehat{\Phi^m(\emptyset)} = \Phi^m(\emptyset)$. □

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

Proof: Let $\bar{\alpha}$ be admissible for $(\bar{x}, z, \hat{X})$. Then for every i such that $\alpha_i \neq \perp$:

$$(\mathscr{T}(t_1^i)(\bar{x}, \bar{\alpha}), \mathscr{T}(t_2^i)(\bar{x}, \bar{\alpha}), \ldots, \mathscr{T}(t_\rho^i)(\bar{x}, \bar{\alpha}), \alpha_i) \in \hat{X}.$$

$\mathscr{T}(t_j^i)(\bar{x}', \bar{\alpha}) \supseteq \mathscr{T}(t_j^i)(\bar{x}, \bar{\alpha})$ implies

$$(\mathscr{T}(t_1^i)(\bar{x}', \bar{\alpha}), \mathscr{T}(t_2^i)(\bar{x}', \bar{\alpha}), \ldots, \mathscr{T}(t_\rho^i)(\bar{x}', \bar{\alpha}), \alpha_i) \in \hat{X}.$$

Now $z = \mathscr{T}(\mathrm{T}^0)(\bar{x}, \bar{\alpha})$ implies $z = \mathscr{T}(\mathrm{T}^0)(\bar{x}', \bar{\alpha})$, and $\bar{\alpha}$ is admissible for $(x', z, \hat{X})$. □

II. DETERMINISTIC PRODUCTION FUNCTIONS AND STABLE INTERPRETATIONS

Let *or* denote the "parallel or" function ($\perp$ *or true* = *true or* $\perp$ = *true*), and consider the following program, which tests if z is the sum of x and y: sum $(x, y, z) =$ *if* $x = 0 \wedge y = 0$ *then* $z = 0$ *else* sum $(x-1, y\, z-1)$ *or* sum $(x, y-1, z-1)$.

Starting from $x, y, z > 0$, there are several to ways 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 $f(x) = f_1(x)$ *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.

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 $mp(x)$. The whole set of intermediate values for x is then determined by transitively applying mp. 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 $(x+1,0)$):

$$f(x) = \textit{if } x = 0 \textit{ then true else } f(x-1) \textit{ or } f(x+1).$$

II.1. Deterministic production functions

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

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

Proof: Assume $\Phi(\bigcap_{i\in I} X_i) = \bigcap_{i\in I} \Phi(X_i)$. Then $mp(x) = \cap \{X \mid x \in \Phi(X)\}$.

Converse follows from the definition of mp. □

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 $\bar{x}$ and x', $b(\bar{x} \wedge \bar{x}') = b(\bar{x}) \wedge b(\bar{x}')$.

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 $m_b(\bar{x}) \subseteq \bar{x}$ such that $b(m_b(\bar{x})) = b(x)$.*

Proof: If b is stable, then $m_b(x)$ is the g. l. b. of the set

$$Y = \{\bar{y} \in D^p \mid \bar{x} \uparrow \bar{y} \wedge b(\bar{y}) = b(\bar{x})\}.$$

Assume conversely the existence of $m_b(\bar{x})$ for every $\bar{x} \in D^p$. Let $\bar{x}$ and $\bar{x}'$ be joinable. If $b(\bar{x}) = \bot$ or $b(\bar{x}') = \bot$, then $b(\bar{x} \wedge \bar{x}') = \bot$ by monotonicity. If $b(\bar{x}) = b(\bar{x}') \neq \bot$, then

$$b(\bar{x} \vee \bar{x}') = b(\bar{x}) = b(\bar{x}'), \qquad \text{and} \qquad m_b(\bar{x} \vee \bar{x}') \subseteq \bar{x},\ m_b(\bar{x} \vee \bar{x}') \subseteq \bar{x}'.$$

This implies $m_b(\bar{x} \vee \bar{x}') \subseteq \bar{x} \wedge \bar{x}'$ and by monotonicity

$$b(\bar{x} \wedge \bar{x}') = b(m_b(\bar{x} \vee \bar{x}')) = b(\bar{x}) = b(\bar{x}'). \quad \square$$

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

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

Example: Consider the usual *if-then-else* function. For $\bar{x} = (\bot, x_2, x_3)$ the critical index is 1. For $\bar{x} = (\mathit{true}, x_2, x_3)$ (resp. $(\mathit{false}, x_2, x_3)$), 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 $\bar{x}$ and $\bar{y}$ satisfy $\bar{x} \uparrow \bar{y}$, $b(\bar{x}) = b(\bar{y}) \neq \bot$, $b(\bar{x} \wedge \bar{y}) = \bot$. For every $i \in \{1, 2, \ldots, p\}$, either $(\bar{x} \wedge \bar{y})_i = x_i$, or $(\bar{x} \wedge \bar{y})_i = y_i$. In the first case, $\bar{x} \supseteq \bar{x} \wedge \bar{y}$ and $b(\bar{x}) \supset b(\bar{x} \wedge \bar{y})$, in the second case $\bar{y} \supseteq \bar{x} \wedge \bar{y}$ and $b(\bar{y}) \supset b(\bar{x} \wedge \bar{y})$. Therefore $\bar{x} \wedge \bar{y}$ has no critical index, which is impossible.

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

II.3. Stable interpretations

DEFINITION: Let Σ 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 $T(x) = b(T_1(\bar{x}), T_2(\bar{x}), \ldots, T_m(x))$ assuming that the T_i define stable functions. Assume $\bar{x} \uparrow \bar{y}$ and $T(\bar{x}) = T(\bar{y}) \neq \bot$. Then the m-tuples

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

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

$$(T_1(\bar{x} \wedge \bar{y}), T_2(\bar{x} \wedge \bar{y}), \ldots, T_m(\bar{x} \wedge \bar{y})).$$

The result follows from the stability of b. □

II.3.2. THEOREM: *Let Σ 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, $\tau^m(\bot)$ is defined by a term containing only basic functions and the undefined function: Lemma applies. Let $\bar{x}, \bar{y} \in D^p$, $\bar{x} \uparrow \bar{y}$; there exist a least $m \geqq 0$ such that $Y(\Sigma, I)(\bar{x}) = \tau^m(\bot)(\bar{x})$ and $Y(\Sigma, I)(\bar{y}) = \tau^m(\bot)(\bar{y})$. Then $Y(\Sigma, I) \supseteq \tau^m(\bot)$ and $\tau^m(\bot)(\bar{x} \wedge \bar{y}) = \tau^m(\bot)(\bar{x}) \wedge \tau^m(\bot)(\bar{y})$ imply

$$Y(\Sigma, I)(\bar{x} \wedge \bar{y}) = Y(\Sigma, I)(\bar{x}) \wedge Y(\Sigma, I)(\bar{y}). \quad \square$$

We now prove a crucial property of stable interpretations:

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

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

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

2) Assume the property true for m:

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

We show by structural induction that for every term T:

$$\forall \bar{y}, \bar{y}' \in D^p,$$
$$(T(\tau^m(\bot))(\bar{y}) = z) \wedge (y \uparrow \bar{y}') \wedge (T(g)(\bar{y}') = z) \quad \Rightarrow \quad T(\tau^m(\bot))(\bar{y}') = z.$$

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:

$$T(g)(\bar{x}') = b(t_1(g)(\bar{x}'), t_2(g)(\bar{x}'), \ldots, t_k(g)(\bar{x}')) = b(\bar{\alpha}),$$
$$T(\tau^m(\bot))(\bar{x}) = b(t_1(\tau^m(\bot))(\bar{x}), t_2(\tau^m(\bot))(\bar{x}), \ldots, t_k(\tau^m(\bot))(\bar{x})) = b(\bar{\beta}).$$

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

$$(\bar{\alpha} \wedge \bar{\beta})_i = t_i(g)(\bar{x}') = t_i(\tau^m(\bot))(\bar{x}) \neq \bot.$$

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

Therefore $(\bar{\alpha} \wedge \bar{\beta}) \subseteq (t_1(\tau^m(\bot))(\bar{x}'), t_2(\tau^m(\bot))(\bar{x}'), \ldots, t_k(\tau^m(\bot))(\bar{x}'))$.

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:

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

$$T(\tau^m(\bot))(\bar{x}) = \tau^m(\bot)(t_1(\tau^m(\bot))(\bar{x}), t_2(\tau^m(\bot))(\bar{x}), \ldots, t_p(\tau^m(\bot))(\bar{x})) = \tau^m(\bot)(\bar{\beta}).$$

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 the term $\tau(g)(\bar{x}')$ since g is the fixpoint.

Applying to τ the preceeding 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

$$f(x, y) = (x = 0) \text{ or } (x = 1 \wedge y = 0) \text{ or } f(x-1, 0).$$

Here $\tau(\bot)(1, 0) = true$, $\tau(\bot)(1, \bot) = \bot$, $\tau^2(\bot)(1, \bot) = true$.

Let us finally study the admissible n-tuples.

II.3.4. THEOREM: *Let Σ be a recursive scheme, let I be a stable interpretation. Let $(\bar{x}, z)$ 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 $(m_g(\bar{x}), z, Y(\Phi))$.*

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

Proof: Let g denote $Y(\Sigma, I)$.

1) Let I_i be the set of labels of the bound variables occuring in $\mathscr{T}(T^i)$ (*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 $j \in I_i$.

b) for every n-tuple $\bar{\alpha}$ admissible for $(\bar{x}, z, Y(\Phi))$, for all $j \in J_i$, $\alpha_j = \beta_j \neq \perp$.

If $I_i = \emptyset$, or if there exists $\bar{\alpha}$ admissible for $(\bar{x}, z, Y(\Phi))$ such that $\alpha_l = \perp$, then $J_i = \emptyset$ satisfies *a* and *b*.

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

$$\alpha_i = \mathscr{T}(T^i)(\bar{x}, \alpha_{i_1}, \alpha_{i_2}, \ldots, \alpha_{i_{l_i}})$$

with $i_j \in I_i$; there exists a minimal l_i-tuple $\bar{\delta} = (\delta_{i_1}, \delta_{i_2}, \ldots, \delta_{i_{l_i}})$ such that $\alpha_i = \mathscr{T}(T^i)(\bar{x}, \bar{\delta})$. Therefore $J_i = \{i_k | \delta_{i_k} \neq \perp\}$ satisfies *a* and *b*.

We now define $\bar{\gamma}$ by $\gamma_j = \beta_j$ if $j \in j_i$ for some i, $\gamma_j = \perp$ 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 $g(\bar{x}') = z \neq \perp$, 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

$$(\bar{x}, \bar{\gamma}) \vee (\bar{x}', \bar{\gamma}') = (\bar{x}, \bar{\gamma}') \qquad \text{and} \qquad (\bar{x}, \bar{\gamma}) \wedge (\bar{x}', \bar{\gamma}') = (\bar{x}', \bar{\gamma}).$$

Using the stability of $\mathscr{T}(\tau)$, $\mathscr{T}(t_j^i)$, 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 $\gamma_i \neq \perp$, we have :

$$\xi_i = (\mathscr{T}(t_1^i)(\bar{x}, \bar{\gamma}), \mathscr{T}(t_2^i)(\bar{x}, \bar{\gamma}), \ldots, \mathscr{T}(t_p^i)(\bar{x}, \bar{\gamma}), \gamma_i) \in Y(\Phi),$$

$$\xi_i' = (\mathscr{T}(t_1^i)(\bar{x}, \bar{\gamma}'), \mathscr{T}(t_2^i)(\bar{x}, \bar{\gamma}'), \ldots, \mathscr{T}(t_p^i)(\bar{x}, \bar{\gamma}'), \gamma_i') \in \Phi^{m-1}(\emptyset).$$

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

$$(\mathscr{T}(t_1^i)(\bar{x}, \bar{\gamma}), \mathscr{T}(t_2^i)(\bar{x}, \bar{\gamma}), \ldots, \mathscr{T}(t_p^i)(\bar{x}, \bar{\gamma}), \gamma_i) = \xi_i \in \Phi^{m-1}(\emptyset),$$

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

II.4. Stable interpretations and Ψ-producers

Even under stable interpretations, the extended production function Ψ is in general not deterministic. Consider the example:

$$f(x_1, x_2) = \textit{if } x_1 = 0 \textit{ then } 0 \textit{ else } f(x_1 - 1, f(x_1 - 2, x_2))$$

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

But P_1 is of smaller cardinality than P_2, and contains more compact infor mation: the existence of (1, ⊥, 0) in Ψ(∅) 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 Ψ-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

$$X \sqsubseteq Y \Leftrightarrow \forall \bar{x} \in X,\ \exists y \in Y,\ \bar{x} \subseteq \bar{y}.$$

II.4.1. THEOREM: *Let* Σ *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* Ψ*-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 opt* $(\bar{x}, z)$.

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

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

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

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

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

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

$$\xi_i = (\mathscr{T}(t_1^i)(\bar{x}, \bar{\gamma}), \mathscr{T}(t_2^i)(\bar{x}, \bar{\gamma}), \ldots, \mathscr{T}(t_\rho^i)(\bar{x}, \bar{\gamma}), \gamma_i) \qquad \text{with} \quad \gamma_i \neq \perp.$$

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

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

Assume $R \subseteq Y(\Psi)$ Ψ-produces $(\bar{x}, z)$, and let $\bar{\alpha}$ be admissible for $(\bar{x}, z, \hat{R})$. By II.3.4, $\bar{\alpha} \supseteq \bar{\gamma}$, and $\hat{R}$ contains at least the points:

$$\xi_i' = (\mathscr{T}(t_1^i)(\bar{x}, \bar{\alpha}), \mathscr{T}(t_2^i)(\bar{x}, \bar{\alpha}), \ldots, \mathscr{T}(t_\rho^i)(\bar{x}, \bar{\alpha}), \gamma_i) \qquad \text{with} \quad \gamma_i \neq \perp.$$

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

Since opt $(\bar{x}, z)$ is finite, we prove that it is a minimal Ψ-producer of $(\bar{x}, z)$ by showing that it is a Ψ-producer of minimal cardinality. Assume that U Ψ-produces $(\bar{x}, z)$ and assume card (U) < card (opt $(\bar{x}, z)$). Since opt $(\bar{x}, z) \sqsubseteq U$, there exist $\eta_1, n_2 \in$ opt $(\bar{x}, z)$ and $u \in U$ such that $\eta_1 \subseteq u$ and $\eta_2 \subseteq u$. This is impossible, since by construction η_1 and η_2 cannot be joinable.

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

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

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

6) We showed in 2 that opt $(\bar{x}, z)$ is a Ψ-producer of minimal cardinality. But every Φ-producer is also a Ψ-producer. □

II.5. Construction of the deterministic production function $\theta_{(\Sigma, I)}$

Our last step is to construct a new production function θ be selecting among the Ψ-producers the optimal one. In fact θ itself will not be deterministic: we only defined the optimal producers for the points in $Y(\Psi)$. However is "restriction" to its fixpoint will be deterministic: 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 $(\bar{x}, z)$ such that $(\bar{x}, z) \in Y(\Psi)$, and $\bar{x}' \subset \bar{x}$ implies $(\bar{x}', z) \notin Y(\Psi)$. By construction every point of 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 $\bar{x}' \subset \bar{x}$; 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 Ψ by adding to every point $(\bar{x}, z)$ a flag fl set to 1 iff $(\bar{x}, z)$ 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 (Σ, I) be a recursive program.

The *production function* $\theta_{(\Sigma, I)} : \mathcal{P}(D_I^{p+1} \times \{0, 1\}) \to \mathcal{P}(D_I^{p+1} \times \{0, 1\})$ is defined by

$$\theta(X) = \{(\bar{x}, z, fl) \in D_I^{p+1} \times \{0, 1\} \mid S(\bar{x}, z, X) \wedge [fl = 1 \Leftrightarrow \forall \bar{x}' \subset \bar{x} \quad \sim S(\bar{x}', z, X)]\}$$

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

II.5.1. PROPOSITION: 1) θ *is monotonic and continuous.*

2) $\forall\, m \in N$, $p(\theta^m(\emptyset)) = \Psi^m(\emptyset)$; $p(Y(\theta)) = Y(\Psi)$.

3) *If* $(\bar{x}, z, fl) \in Y(\theta)$, $fl = 1$ *is equivalent to* $(\bar{x}, z)$ *minimal.*

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

Proof: 1) 2) 3) directly follow from the construction (*see* [1]).

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

If P θ-produces $(\bar{x}, z, fl)$, there exists $P' \subseteq P$ such that every point of P' is minimal and such that $p(P')$ Ψ-produces $(\bar{x}, z)$. Then $P \sqsupseteq$ opt $(\bar{x}, z)$ and $P \supseteq$ opt $(\bar{x}, z)$. □

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

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

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

DEFINITION: A *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

$$X_0 = \emptyset \quad \text{and} \quad \forall n \geqq 0, \; X_n \subseteq X_{n+1} \subseteq X_n \cup \Phi(X_n).$$

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

$$X_0 = \emptyset \quad \text{and} \quad \forall n \geqq 0, \; X_{n+1} \subseteq X_n \cup \Phi(X_n).$$

A s. c. s. $\{ X_n, n \in N \}$ is *simple* iff it satisfies

$$\forall x \in \bigcup_n X_n \quad \sim \exists p, q, r \in N, p < q < r, x \in X_p \wedge x \notin X_q \wedge x \in X_r.$$

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 $X_n = \bigcup_{0 \leqq p \leqq n} Y_p$ is a b. u. c. s. An example of simple s. c. s. is Kleene's sequence $\emptyset, \Phi(\emptyset), \ldots, \Phi^n(\emptyset), \ldots$

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.

DEFINITION: A subset S of D is *self-reproducing* (abbreviated s. r.) iff it satisfies $S \subseteq \Phi(S)$.

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.*

III.2.2. PROPOSITION: *Every member or limit of a b. u. c. s. is s. r.*

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:

DEFINITION: Φ is *quasi-deterministic* iff every point $x \in \Phi(D)$ has a unique minimal self-reproducing producer *prod* (x).

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

Proof: See II.1.1. □

Clearly, every deterministic function is quasi-deterministic.

III.2.4. PROPOSITION: *Let Φ 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* $\Phi^m(\emptyset)$ *for some* $m \geqq 0$.

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 $P_2 \nsubseteq$ 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 $x : X_n \supseteq$ 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 $\prec$

DEFINITION: Let Φ be quasi-deterministic. *The domain of* x, *dom* (x), is the smallest s. r. set containing x. The relation $\prec$ is defined by $x \prec y$ iff $x \in$ dom (y).

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

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

2) $x \notin$ *prod* (x);

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

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

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

Then 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 prod $(x) \subseteq \Phi^{n-1}(\emptyset)$ and $x \notin$ prod (x).

3) The relation $\prec$ is obviously reflexive and transitive. It is also antisymmetric: assume $x \prec y$, $y \prec x$, $x \neq y$. Then $x \in$ prod (y) implies prod $(x) \subseteq$ prod (y), and $y \in$ prod (x) implies $y \in$ prod (y), contradicting 2.

If $x \in \Phi^n(\emptyset)$ and $y \prec x$, then $\text{prod}(x) \subseteq \Phi^{n-1}(\emptyset)$ and $y \in \Phi^{n-1}(\emptyset)$. The relation $\prec$ is well-founded. □

We extend the definitions of prod (x) and dom (x) to subsets of D:

DEFINITION: Let Φ be quasi-deterministic.

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

III.3.2. PROPOSITION: 1) $dom\ (X) = \bigcup_{x \in X} dom\ (x)$.

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

3) $dom\ (X) = prod\ (X) \cup X$.

4) $prod\ (X) = prod\ (dom\ (X))$.

5) S *s. r.* $\Leftrightarrow S = dom\ (S) \Leftrightarrow S \supseteq prod\ (S)$.

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

A set S is s. r. iff $S = \bigcup_{x \in S} \text{dom}\ (x)$. 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 Φ 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 $\forall\ e, e' \in E \quad \sim e \prec e'$. 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* $S = dom\ (E)$ *and* $E \cap prod\ (S) = \emptyset$.

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 $\exists\ s \in S \quad e \in \text{prod}\ (s)$. Since $S = \text{dom}\ (E)$, $\exists\ e' \in E \quad s \in \text{dom}\ (e')$. Finally $e \in \text{prod}\ (e')$, which is impossible. □

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

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

Proof: By induction on n. □

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

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

Assume Φ deterministic, assume $y \in$ ext (prod (x)) satisfies $y \notin mp(x)$.

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

III.4.6. COROLLARY 2: *Let Φ 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 prod $(X_n) \subseteq X_{n-1}$. Therefore 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 Φ be quasi-deterministic. A set $E \subseteq D$ is *extremal* iff E is extremal for dom (E).

III.5.1. PROPOSITION: *Let Φ be quasi-deterministic*

$$E \text{ extremal} \Leftrightarrow E \cap \text{prod}(E) = \emptyset \Leftrightarrow \text{prod}(E) = \text{dom}(E) - E.$$

The relation $\prec$ extends to extremal sets:

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

III.5.2. PROPOSITION: *The relation* $\prec$ *is a partial ordering on EXT.*

Proof: Reflexivity and transitivity are obvious.

Assume $E_1 \prec E_2$, $E_2 \prec E_1$. 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. □

III.5.3. LEMMA: *Let* $E_1, E_2 \in EXT$. *Then*

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

Proof: We have $\text{prod}(E_1) \subseteq \text{dom}(E_1) \subseteq \text{dom}(E_2) = \text{prod}(E_2) \cup E_2$. Assume $\exists\, e_2 \in E_2\ e_2 \in \text{prod}(E_1)$. Then $\exists\, e_1 \in E_1\ e_2 \in \text{prod}(e_1)$. Since $e_1 \in \text{dom}(E_2)$, $\exists\, e_2' \in E_2,\ e_1 \in \text{dom}(e_2')$.

We get $e_2 \in \text{prod}(e_2')$, which is impossible since E_2 is extremal. □

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

1) $max(E_1, E_2) = ext(dom(E_1) \cup dom(E_2))$;
2) $max(E_1, E_2) = \{ e_1 \in E_1 \mid e_1 \notin prod(E_2) \} \cup \{ e_2 \in E_2 \mid e_2 \notin prod(E_1) \}$;
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 $x \in \text{dom}(E_1) \cup \text{dom}(E_2)$; we can assume $x \in \text{dom}(E_1)$. Then $\exists\, e_1 \in E_1\ x \in \text{dom}(e_1)$. If $e_1 \notin \text{prod}(E_2)$, then $e_1 \in E$ and $x \in \text{dom}(E)$. If $e_1 \in \text{prod}(E_2)$, there exists $e_2 \in E_2$ such that $e_1 \in \text{prod}(e_2)$; $e_1 \in \text{prod}(E_1)$ being impossible, we have $e_2 \notin \text{prod}(E_1)$, 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

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

The reverse inclusion follows from III.5.3, and property 3 is proved. □

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

Proof: If $\mathrm{dom}(E_1) \cap \mathrm{dom}(E_2)$ 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 $\mathrm{dom}(E) \not\subseteq \mathrm{dom}(E_1) \cap \mathrm{dom}(E_2)$. Let

$$a \notin \mathrm{dom}(E),\ a \in \mathrm{dom}(E_1) \cap \mathrm{dom}(E_2),$$

let $E' = \max(\{ a \}, E)$. Then $E' \neq E$, $E' \succ E$, $E' \prec E_1$, $E' \prec E_2$, which is impossible. □

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 Φ 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.

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* $\forall x \in S\text{-}dom(C)$ $\quad dom(x) \cap C \neq \emptyset$.

Proof: 1) Assume C S-complete, let $x \in S$-dom (C) satisfy $\mathrm{dom}(x) \cap C = \emptyset$. Then for all c in C, $c \notin \mathrm{dom}(x)$ and $x \notin \mathrm{dom}(c)$; hence $C \cup \{ x \}$ is extremal, is included in S and strictly contains C: contradiction with C S-complete. Conversely, assume $\forall x \in S\text{-dom}(C)$ $\mathrm{dom}(x) \cap C \neq \emptyset$. Assume there exists $a \in S$ such that $C \cup \{ a \}$ is extremal. Then $\mathrm{dom}(a) \cap C = \emptyset$ and $a \notin \mathrm{dom}(C)$, 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) $\forall x \in S$ $\quad prod(x) \subseteq prod(C) \Rightarrow x \in dom(C)$.

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

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

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

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

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

Let $e \in E$, Lemma III.5.3 implies prod $(e) \subseteq$ prod (C), and first condition implies $e \in$ dom (C). Therefore $e \in$ dom $(x) \cap C$, and C is S-complete by III.6.1. □

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 $\forall x \in S$ prod $(x) \subseteq$ prod $(C_1) \Rightarrow x \in$ 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:

$$Y_S(S') = (\varphi(S') - S') \cap S = \{ s \in S \mid s \notin S' \wedge \text{prod}(s) \subset S' \}.$$

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

1) $cmax_S(C_1, C_2) = Y_S(prod(C_1) \cup prod(C_2))$.
2) $prod(cmax_S(C_1, C_2)) = prod(C_1) \cup prod(C_2)$.
3) $cmin_S(C_1, C_2) = Y_S(prod(C_1) \cap prod(C_2))$.
4) $cmin_S(C_1, C_2) = min(C_1, C_2) = ext(dom(C_1) \cap dom(C_2))$.
5) $cmin_S(C_1, C_2) = (C_1 \cap dom(C_2)) \cup (C_2 \cap dom(C_1))$.
6) $prod(cmin_S(C_1, C_2)) = prod(C_1) \cap prod(C_2)$.

Proof: a) Let $C = Y_S(prod(C_1) \cup prod(C_2))$.

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, prod $(C) \supseteq$ prod $(C_1) \cup$ prod (C_2). By construction, prod $(C) \subseteq$ prod $(C_1) \cup$ prod (C_2), and 2 holds.

Assume $x \in S$ and prod $(x) \subseteq$ prod (C).

If $x \notin$ prod $(C_1) \cup$ prod (C_2), then $x \in C$ by definition of C. Otherwise $x \in$ dom $(C_1) \cup$ dom $(C_2) \subseteq$ 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, prod $(C_1) \cup$ prod $(C_2) \subseteq$ prod (C'). Hence $c \in$ dom (C') by III.6.2. Therefore $C \prec C'$, and C is the l. u. b. of C_1 and C_2 in SCOMP.

b) Let $C = Y_S(prod(C_1) \cap prod(C_2))$.

b 1) Let $c \in C$. Then prod $(c) \subseteq$ prod $(C_1) \cap$ prod (C_2) implies by III.6.2 $c \in$ dom $(C_1) \cap$ dom (C_2). Since $c \notin$ prod $(C_1) \cap$ prod (C_2), we have either $c \notin$ prod (C_1) and $c \in C_1 \cap$ dom (C_2), or $c \notin$ prod (C_2) and $c \in C_2 \cap$ 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.

b 2) We now have

$$\mathrm{dom}(C) = \mathrm{dom}(C_1 \cap \mathrm{dom}(C_2)) \cup \mathrm{dom}(C_2 \cap \mathrm{dom}(C_1)),$$

and therefore $\mathrm{dom}(C) = \mathrm{dom}(C_1) \cap \mathrm{dom}(C_2)$; (*cf.* III.3.2). Since C is extremal by construction, property 4 holds.

b 3) To show that C is S-complete, we apply III.6.1.

Assume $x \in S\text{-dom}(C)$. Three cases are possible:

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

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

Case 3: $x \notin \mathrm{dom}(C_1)$ and $x \notin \mathrm{dom}(C_2)$. Then $\mathrm{dom}(x) \cap C_1 \neq \emptyset$.

Let $c_1 \in \mathrm{dom}(x) \cap C_1$. If $c_1 \in \mathrm{dom}(C_2)$, then $c_1 \in C$. Otherwise

$\mathrm{dom}(c_1) \cap C_2 \neq \emptyset$, $\mathrm{dom}(x) \cap (C_2 \cap \mathrm{dom}(C_1)) \neq \emptyset$ and $\mathrm{dom}(x) \cap C \neq \emptyset$.

Eventually, C being the g. l. b. of C_1 and C_2 in (EXT, $\prec$) is also their g. l. b. in (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* $\Phi : \mathscr{P}(D) \to \mathscr{P}(D)$ *be monotonic and continuous. Let* X_n *be the sequence of subsets of* D *defined by:*

$X_0 = \emptyset$;

$X_n = X_{n-1} \cup \{ d_i \}$ *if there exist a least integer* i *such that* $d_i \notin X_{n-1}$ *and* $d_i \in \Phi(X_{n-1})$;

$X_n = X_{n-1}$ *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 $\bigcup_n X_n \subseteq Y(\Phi)$ holds by induction on n.

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

$$d_{i_{q+l}} \in X_{q+l}, \qquad d_{i_{q+l}} \notin X_{q+l-1}, \qquad d_{i_{q+l}} \in \varphi(X_{q+l-1}), \qquad i_{q+l} < p.$$

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 Φ be continuous. Let $h : D \to \mathcal{P}(D)$ be such that $\forall x \in \Phi^n(\emptyset) \quad h(x) \subseteq \Phi^{n-1}(\emptyset)$. 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 $Y(\Phi)$.*

$$[\forall x \in Y(\Phi), [\forall y \in h(x)\, P(y)] \Rightarrow P(x)] \vdash [\forall x \in Y(\Phi) \quad P(x)].$$

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

Assume $\forall y \in \Phi^n(\emptyset) \quad P(y)$. Let $x \in \Phi^{n+1}(\emptyset)$. Then $h(x) \subseteq \Phi^n(\emptyset)$ implies $\forall y \in h(x) \quad P(y)$, 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) $h(x) = prod(x)$. We obtain the usual structural induction principle applied to the relation $\prec$.

3) $h(x) = ext(prod(x))$.

4) $h(x) = mp(x)$ if Φ is deterministic. This principle seems to be the most useful when dealing with recursive programs: the set $mp(x)$ 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.

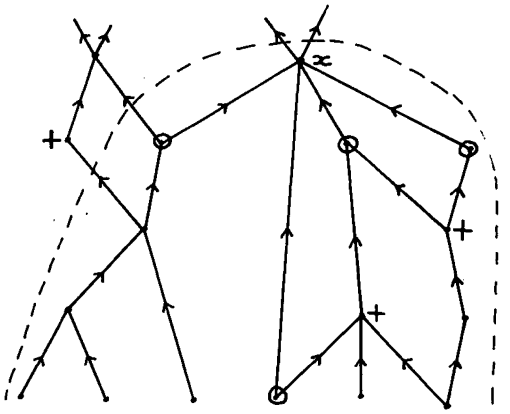

Diagram 1.1

A deterministic production function.

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

– – – – – – – – dom (x)

O $mp(x)$

+ + + an extremal set.

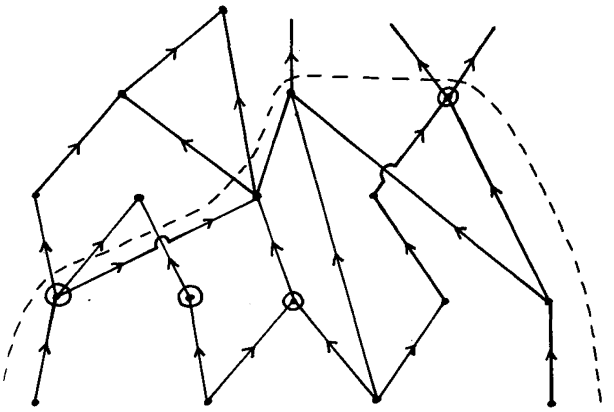

Diagram 1.2

– – – – – – – – a s. r. set S.

O O O O a S-complete 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).

– 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 Fibonnaci function can be defined by the two following programs:

P 1. fib (n) = *if* $n = 0 \vee n = 1$ *then* 1 *else* fib $(n-1)+$fib $(n-2)$.

P 2. fib (n) = *if* $n = 0 \vee n = 1$ *then* 1 *else if* $n = 2$ *then* 2.
else (fib $(n-1)+3\times$fib $(n-2)+$fib $(n-3))/2$.

An optimal implementation of P 2 is not optimal for the computation of 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 $N \cup \{ \perp \}$ by the following program scheme:

$$\mathrm{A}(m, n) = \textit{if } m = 0 \textit{ then } n+1 \textit{ else if } n = 0 \textit{ then } \mathrm{A}(m-1, 1) \textit{ else } \mathrm{A}(m-1, \mathrm{A}(m, n-1)).$$

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

$$\forall m, n \in N \quad \mathrm{A}(\perp, \perp) = \mathrm{A}(m, \perp) = \mathrm{A}(\perp, n) = \perp.$$

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

$$\Phi(X) = \{(m, n, z) \in N^3 \mid \textit{if } m = 0 \textit{ then } z = n+1 \textit{ else if } n = 0 \textit{ then } (m-1, 1, z) \in X \textit{ else } \exists y \quad (m-1, y, z) \in X \wedge (m, n-1, y) \in X\}.$$

As the function θ in II.5, the function Φ 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:* $\forall\, m, n \in N,\ A(m, n) \in N$.

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

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

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

We now characterise the domains:

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

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

2) *if* $m \neq 0$, *then*

$$dom(m, n, z) = \{(m', n', z') \in Y(\Phi) \mid m' \leqq m \wedge z' \leqq z\} - \{0, 0, 1\}.$$

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

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

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

Case $m > 1$, $n \neq 0$: let $z1 = A(m, n-1)$; assume

$$\mathrm{dom}(m, n-1, z1) = D(m, n-1, z1) \text{ and } \mathrm{dom}(m-1, z1, z) = D(m-1, z1, z).$$

Then:

$$\mathrm{dom}(m, n, z) = D(m, n-1, z1) \cup D(m-1, z1, z) \cup \{(m, n, z)\}.$$

Proposition IV.2.1 implies $z1 < z$, and therefore $\mathrm{dom}(m, n, z) \subseteq D(m, n, z)$.

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

- If $m' = m$, then IV.2.1 implies $n' \leqq n$; if $n' = n$, then $(m', n', z') = (m, n, z)$, otherwise $(m', n', z') \in D(m, n-1, z1)$.
- If $m' < m$, then $(m', n', z') \in D(m-1, z1, z)$.

In both cases, $(m', n', z') \in \mathrm{dom}(m, n, z)$. Eventually,

$$\mathrm{dom}(m, n, z) = D(m, n, z).$$

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

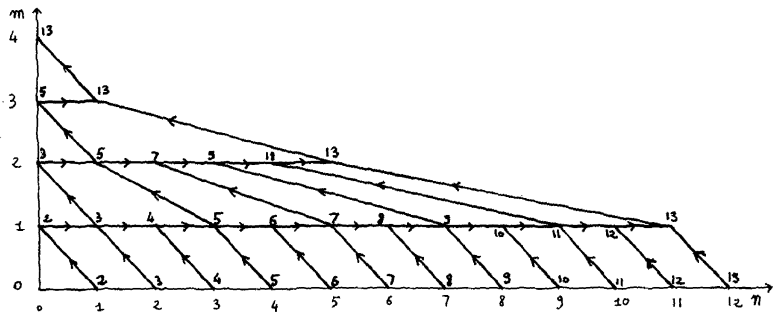

Diagram 2

Ackermann recursion structure.

The domain of (4, 0, 13); numbers are values of the function.

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 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 Φ 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, PLACE $[I]$, 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":

$$d_t = (\mathrm{I}_t, \mathrm{PLACE}[\mathrm{I}_t]_t, \mathrm{VAL}[\mathrm{I}_t]_t),$$

$$Y_t = \{(\mathrm{J}, \mathrm{PLACE}[\mathrm{J}]_t, \mathrm{VAL}[\mathrm{J}]_t) \mid 0 \leqq \mathrm{J} \leqq M \wedge \mathrm{VAL}[\mathrm{J}]_t > 0\}.$$

```
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;
```

TABLE

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

Let $T0$ be the value of T when the program is exited.

IV.3.1. PROPOSITION: *For every* $M, N > 0$, *the sequence* $\{ Y_t, 1 \leqq t \leqq T0 \}$ *is a simple s. c. s. with limit 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 $\mathrm{VAL}[0]_t = \mathrm{PLACE}[0]_t + 1$; hence $d_t \in \Phi(\emptyset)$.

– *Case* $I_t \neq 0$, PLACE $[I_t]_t = 0$: block B was exited at time t, and also at time $t-1$; the following relations hold:

$$I_{t-1} = I_t - 1; \qquad Z_{t-1} = Z_t.$$

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

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

Since the *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 $d_t \in \Phi(\{ d_{t-1} \}) \subseteq \Phi(Y_{t-1})$.

– *Case* $I_t \neq 0$, 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 $\leqq_l$ on N^2; no value is computed twice.

3) $ACK(M, N) = A(M, N)$: 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 $I_{T0} = M$, PLACE $[I_{T0}]_{T0} = N$ and

$$\text{ACK} = \text{VAL}[I_{T0}]_{T0} = A(M, N).$$

4) $\bigcup_{1 \leqq t \leqq T0} Y_t = dom(M, N, A(M, N))$: Let $S = \bigcup_{1 \leqq t \leqq T0} Y_t$; for every t, we have $I_t \leqq M$ and $Z_t \leqq Z_{T0}$. 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 $S \supseteq \text{dom}(M, N, A(M, N))$. □

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 $\{ Y_n, 0 \leqq n \leqq N \}$ such that $(x, y, A(x, y)) \in Y_N$).

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

DEFINITION: Let $x \in Y(\Phi)$. *A path from* x *to* $\Phi(\emptyset)$ is a sequence $x_0, x_1, \ldots, x_n$ of points of $Y(\Phi)$ such that $x_0 = x$, $x_n \in \Phi(\emptyset)$ $\forall p$, $1 \leqq p < n$, $x_{p+1} \in mp(x_p)$.

The proof relies on the following lemma:

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

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

Proof: Assume the property false. Then there exists a path

$$P = \{ x_0, x_1, \ldots, x_q \}$$

and an index k', $k \leqq k' \leqq N$, 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 $P \cap Y_{k'} \neq \emptyset$ (*cf.* definition of simplicity).

Since $x_{q'} \notin Y_{k'}$ and $x_{q'} \in Y_m$ for some $m < k'$, $x_{q'} \notin Y_{m'}$ holds for $m' \geqq k'$. 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 $m' \geqq k'$ $mp(x_{q'-1}) \nsubseteq Y_{m'}$, and hence $x_{q'-1} \notin Y_{m'+1}$. Finally, $x_{q'-1} \notin Y_m$ for all $m \leqq N$. 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 dom (x)-complete set C. Let us define the following transformations:

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

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

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

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

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

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

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 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 $c_x^3 < c_{x-1}^3 < \ldots < c_2^3 < c_1^3 < z = c_0^3$; 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 dom (x, y, z)-complete. □

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

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

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 Φ denote the Ackermann production function, let $(x, y, z) \in Y(\Phi)$, $x > 0$, $y > 0$. Let $\{ Y_n, 0 \leqq n \leqq N \}$ be a simple s. c. s. with limit dom (x, y, z). Then* $\max_{0 \leqq n \leqq N} card(Y_n) \geqq x+1$.

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 μ $x-m$ times and then ν, to reach c_m; we then apply μ m times to reach $\Phi(\emptyset)$, thus keeping the third coordinate equal to c_m^3. Let a denote (x, y, z):

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

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

$$\vdots$$

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

$$\vdots$$

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

Let $m < m'$; by IV.4.3, P_m and $P_{m'}$ only intersect at points a, μa, $\mu^2 a$, ..., $\mu^{x-m'}$, 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 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 reccurence relations 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.

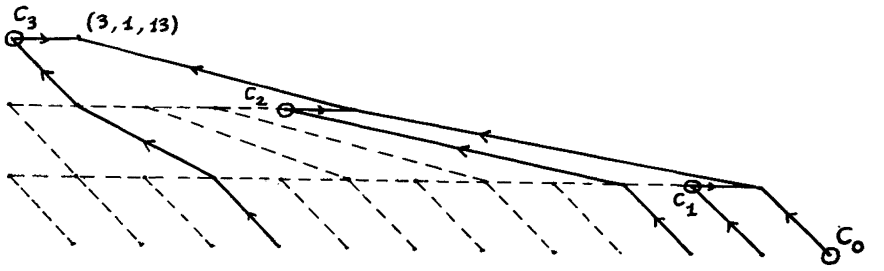

Diagram 3

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

1. G. BERRY. *Calculs ascendants des programmes récursifs.* Thèse de 3e cycle, Université Paris VII, 1976.
2. R. BURSTALL. *Proving Properties of Programs by Structural Induction.* Computer Journal, vol. 12, 1969, p. 41-48.
3. S. COOK and R. SETHI. *Storage Requirements for Deterministic Polynomial Time Recognizible Languages.* Proc. 6th annual symposium on theory of computing, Seattle, Washington, 1974, p. 33-39.
4. D. LUCKHAM, D. PARK and M. PATERSON. *On Formalised Computer Programs.* Journal of Computer and System Sciences, vol. 4, No. 3, 1970, p. 220-250.
5. Z. MANNA. *Mathematical theory of Computation.* McGraw-Hill, (Computer science series), 1975.
6. Z. MANNA and J. VUILLEMIN. *Fixpoint Approach to the Theory of Computation.* Comm. ACM, vol. 15, No. 7, 1972, p. 528-536.
7. R. MILNER. *Implementation and Applications of Scott's Logic for Computable Functions.* Proceedings ACM Conference on Proving assertions about programs, Las Cruces, New Mexico, 1972, p. 1-5.
8. J. H. MORRIS. *Another Recursion Induction Principle.* Comm. ACM, vol. 14, No. 5, 1971, p. 351-354.

9. M. NIVAT. *On the Interpretation of Recursive Program Schemes.* Symposia Mathematica, Vol. XV, Instituto Nazionale di Alta Matematica, Italy, 1975, p. 255-281.
10. D. PARK. *Fixpoint Induction and Proofs of Program Properties.* Machine Intelligence 5, Edinburgh University press, 1969, p. 59-77.
11. H. G. RICE. *Recursion and Iteration.* Comm. ACM, vol. 8, No. 2, 1965, p. 114-115.
12. D. SCOTT. *Outline of a Mathematical Theory of Computation.* Programming research group monography n° 2, Oxford University, 1970.
13. D. SCOTT and C. STRACHEY. *Towards a Mathematical Semantics for Programming Languages.* Programming research group monography No. 6, Oxford University, 1972.
14. J. VUILLEMIN. *Proof Techniques for Recursive Programs.* Ph. D. thesis, Computer Science Department, Stanford University, U.S.A., 1973.
15. J. VUILLEMIN. *Syntaxe, sémantique et axiomatique d'un language de programmation simple.* Thèse de doctorat d'état ès-sciences mathématiques, Université Paris VI, Paris, 1974.