

FIXPOINTS AND SEMANTICS OF CONCURRENCY

Irène GUESSARIAN *

L.I.T.P. Université Paris 6

4, Place Jussieu, 75252 Paris Cédex 05, France

e-mail: ig@litp.ibp.fr

Abstract : This paper recalls a fixpoint theorem in ordered algebraic structures and shows one way in which this theorem can be applied in computer science. It also points out the shortcomings of the classical least fixpoint theory for domains such as nondeterministic or concurrent programs, and shows how to overcome these liabilities by introducing better and more refined fixpoints.

I INTRODUCTION

The trend in Computer Science is to integrate more and more sophisticated tools in the programming language or even the underlying system, in order to give maximal flexibility to the user. One such tool is the use of recursion, which was initiated with LISP, and is now growing wider and wider. This comes from the fact that recursion allows one to express in a straightforward way the essential ideas of an algorithm, or a definition, without useless implementation details. However, if this straightforward recursive definition must then be converted into a complex imperative program, with lots of stack manipulations and book keeping, most of the gain is lost. So, once a recursive definition corresponding to a program, or describing a relation in a deductive data base, is given, usually in the form of a set of mutually recursive equations, we are interested in

- 1) finding the final state (e.g. result of a computation of the program, or elements belonging to the relation in the data base), which corresponds to a "stable state", namely a fixpoint of the set of recursive equations,

- 2) finding this stable state in the most effective and efficient way, i.e. computing the fixpoint as fast as possible and/or using as little space as possible.

The above mentioned two points form the core of the fixpoint theory, and in the past few years, researchers in domains going from semantics to data bases, and from automated proof theory to logic programming have been using various fixpoint theorems and implementations thereof. As we will see, nearly all the fixpoint techniques which are usable rely on an induction principle: they all compute the fixpoint by induction, using successive substitutions. Surprisingly enough, this very simple method leads to pretty efficient results and there is now a well established theory of least fixpoints of continuous functions which can be applied for stating and proving properties of programs, answering to queries, etc... [Scott, Van Emden-Kowalski, Gallaire-Minker-Nicolas, Guessarian89, Tiuryn]. However, least fixpoints do not always suffice, or exist. In the case of logic programs, for instance, negations might be necessary, introducing non monotonicity, and the least fixpoint does no longer exist; however, other fixpoints can be used [Apt-Blair-Walker, Makinson]. In the case of parallel or nondeterministic programs, least fixpoints are usually uninteresting or trivial, and one is then interested in various fixpoints, the greatest ones [Arnold-Nivat, Bergstra-Klop], other suitably chosen fixpoints [Darondeau-Gamatié, Rounds]. The problem then becomes: how to choose the right fixpoint, and once this choice is established, how to compute the chosen one. In the present paper, we try to improve the classical fixpoint tool and thus broaden the threshold of its applicability for parallel programs. To this end, we will study greatest or unique fixpoints, and their morphic images, which will apply in the case of parallel and non deterministic programs.

The present paper improves the results of [Guessarian89], where only non-determinism was treated; we consider also here true concurrency.

* Support from the PRC Mathématiques-Informatique and ESPRIT BRA 3230 is gratefully acknowledged.

The paper is organised as follows: section 2 contains the preliminaries and notations for algebraic semantics; section 3 describes how to obtain the model of true concurrency we are interested in, starting from the Herbrand interpretation H , and using fixpoints and morphisms.

II ALGEBRAIC SEMANTICS

II.1 Algebras

Let Σ (resp. Φ) be a finite ranked alphabet of base function symbols (resp. of variable function symbols). The rank of a symbol s in $\Sigma \cup \Phi$ is denoted by $r(s)$. Symbols in Σ are denoted f, g, h, \dots if they have rank ≥ 1 , and a, b, \dots if they have rank 0; Σ_i denotes the symbols of rank i in Σ . Symbols in Φ are denoted ϕ, ψ, \dots . Let X be a set of variables: the variables have rank 0 and are denoted by u, v, w, \dots possibly with indices.

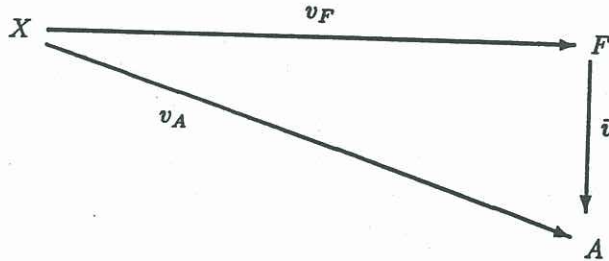
Let T_Σ denote the set of trees, or terms, well-formed over Σ (see [Guessarian] for basic notions about trees and operations on trees). A tree with variables in X is a tree over $\Sigma \cup X$, i.e. an element of $T_{\Sigma \cup X}$: intuitively, the variables are intended to range over a set of trees, e.g. T_Σ or $T_{\Sigma \cup X}$. T_Σ (resp. $T_{\Sigma \cup X}$) is endowed with a Σ -algebra structure: for f in Σ , f of rank p , and t_1, \dots, t_p in T_Σ (resp. $T_{\Sigma \cup X}$), the operation f_{T_Σ} (resp. $f_{T_{\Sigma \cup X}}$) is defined by: $(t_1, \dots, t_p) \mapsto f(t_1, \dots, t_p)$. T_Σ (resp. $T_{\Sigma \cup X}$) is the domain, or carrier set, of the free initial Σ -algebra, or Σ -magma in the terminology of [Nivat], (resp. the free Σ -algebra over generators X , $T_{\Sigma \cup X}$, which is also denoted by $T_\Sigma(X)$). This will be justified by the fact that T_Σ and $T_{\Sigma \cup X}$ have the free property, which we will recall below in full generality. We will in the sequel identify T_Σ (resp. $T_{\Sigma \cup X}$) with the corresponding free algebra.

Let us first recall the notions of ordered and complete Σ -algebras, which we will use later. A Σ -algebra A , or algebraic structure of similarity type Σ , consists of a carrier set A and for each function symbol σ in Σ , a function $\sigma_A : A^w \rightarrow A$ where $A^w = A \times \dots \times A$ with $w = \text{rank}(\sigma)$, and σ_A is a constant in A if $w = 0$.

An ordered Σ -algebra is a Σ -algebra such that the carrier set A is endowed with an ordering \leq_A and a least element \perp_A , and the operations σ_A are order preserving.

A Σ -algebra A is said to be *complete* iff, all directed subsets of A have a lub (least upper bound) in A , and the σ_A 's are continuous, i.e. preserve lub's of directed sets. The category of Σ -algebras (resp. ordered Σ -algebras, or complete Σ -algebras) is defined as follows: objects are Σ -algebras (resp. ordered Σ -algebras, or complete Σ -algebras); morphisms are Σ -homomorphisms (resp. order-preserving Σ -homomorphisms, i.e. Σ' -homomorphisms, or continuous Σ -homomorphisms that is: homomorphisms which, in addition to preserving the Σ -structure, also preserve lub's of directed sets, which implies that they preserve \perp and the order also). Recall that a Σ -homomorphism from a Σ -algebra A to a Σ -algebra B is a function $f : A \rightarrow B$ that satisfies: for σ in Σ with $\text{rank}(\sigma) = n$, and $a_i \in A$ for $i = 1, \dots, n$: $f(\sigma_A(a_1, \dots, a_n)) = \sigma_B(f(a_1), \dots, f(a_n))$; and for σ in Σ_0 : $f(\sigma_A) = \sigma_B$. Note finally that a Σ -homomorphism $h : A \rightarrow B$ is said to be *strict* if and only if: $h(\perp) = \perp$.

A (complete, ordered) $\Sigma \cup X$ -algebra F is said to be *free over generators X* iff for any (complete, ordered) $\Sigma \cup X$ -algebra A there exists a unique Σ -homomorphism (in the category of complete or ordered Σ -algebras) $\bar{v} : F \rightarrow A$ making the following diagram commute:



where, for any $\Sigma \cup X$ -algebra J , $v_J(x) = x_J$, for all x in X . F is also called the *free (complete, ordered) Σ -algebra generated by X* .

The free Σ -algebra generated by X , also called the algebra of Σ -terms with variables in X , and denoted by $T_\Sigma(X)$, exists and can be constructed as follows: its carrier set is the set of well-formed terms (or trees) on the alphabet $\Sigma \cup X$, and is defined inductively by:

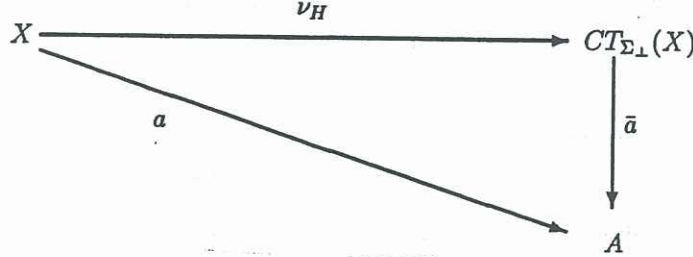
- (i) $\Sigma_0 \cup X \subseteq T_\Sigma(X)$,
- (ii) $\sigma(t_1, \dots, t_n) \in T_\Sigma(X)$ for each σ in Σ and t_i in $T_\Sigma(X)$ for $i = 1, \dots, n$, and $\text{rank}(\sigma) = n$.

The Σ -algebra structure of $F = T_\Sigma(X)$ is defined by: $\sigma_F = \sigma$ if $\sigma \in \Sigma_0$, and otherwise: $\sigma_F(t_1, \dots, t_n) = \sigma(t_1, \dots, t_n)$, for t_i in $T_\Sigma(X)$ for $i = 1, \dots, n$.

The free ordered Σ -algebra $F' = T_{\Sigma, \perp}(X)$ generated by X has carrier sets the sets of well-formed terms on the alphabet $\Sigma \cup X \cup \{\perp\}$; its Σ -algebra structure is defined as the one of F ; its carrier set F' is endowed with the least ordering such that: (i) \perp is the least element, and (ii) the Σ -algebra operations are order-preserving.

Finally, the free complete Σ -algebra $H = CT_{\Sigma, \perp}(X)$ generated by X is the ideal completion of F' (see [Birkhoff]). $H = CT_{\Sigma, \perp}(X)$ is the set of ideals of F' , ordered by inclusion, and the Σ -algebra operations are extended by continuity; namely for σ in Σ and I_i ideal of F' for $i = 1, \dots, n$, $\sigma_H(I_1, \dots, I_n)$ is the ideal generated by $\{\sigma_{F'}(i_1, \dots, i_n)/i_j \in I_j \text{ for } j = 1, \dots, n\}$. H can be viewed as the set of well-formed finite and infinite trees generated by X . Its ordering \leq extends the ordering on F' and can be intuitively described by: $T \leq T'$ iff T' can be deduced from T by substituting some occurrences of \perp by terms different from \perp .

By the freeness of $H = CT_{\Sigma, \perp}(X)$, we know that for each valuation $a : X \rightarrow A$ in an arbitrary complete Σ -algebra A , there exists a unique morphism \bar{a} making the following diagram commute:



For T in $CT_{\Sigma, \perp}(X)$, having variables $\{x_1, \dots, x_n\}$ we will denote by T_A the application $A^n \rightarrow A$ defined by $a = (a_1, \dots, a_n) \mapsto \bar{a}(T)$, i.e. every n -tuple $a \in A^n$ determines a valuation $X \rightarrow A$, and we define $T_A(a) = \bar{a}(T)$.

Let CT_{Σ} (resp. $CT_{\Sigma}(X)$) be the subset of $CT_{\Sigma, \perp}(X)$ consisting of terms without occurrences of \perp . T_{Σ} (resp. CT_{Σ}) is the set of finite (resp. finite and infinite) terms on the alphabet Σ ; similarly, $CT_{\Sigma}(X)$ is the set of finite and infinite terms (or trees) on $F \cup X$.

II.2 Recursive schemes, rational and algebraic trees

DEFINITION II.1 A recursive scheme on Σ is a pair (S, t) , where S is a system of n equations:

$$S: \quad \phi_i(x_1^i, \dots, x_{n_i}^i) = t_i \quad , \quad i = 1, \dots, n \quad (1)$$

where, for $i = 1, \dots, n$, $\phi_i \in \Phi$, $x_j^i \in X$ for each j , $x_j^i \neq x_k^i$ for $j \neq k$, $t_i \in T_{\Sigma \cup \Phi}(X)$, and $t \in T_{\Sigma \cup \Phi}(X)$.

A recursive scheme is said to be *iterative* iff all function variables in Φ are of rank 0, or, from a programming viewpoint, iff all the t_i 's are left-linear: intuitively, an iterative scheme corresponds to left-linear or terminal recursion, which is well-known to be equivalent to iteration, since a program containing terminal recursions only can be translated into a program containing WHILE loops and no recursion by using classical program transformations. The solution of a recursive (resp. iterative) scheme is called an *algebraic* (resp. *rational*, or *regular*) tree. The terminology comes from language theory, because algebraic trees are obtained as languages generated by context-free tree grammars, and rational trees are generated by regular tree grammars.

Rational (resp. algebraic) trees are trees which are solutions of iterative (resp. recursive) schemes, and correspond intuitively to unfoldings of iterative, i.e. WHILE-loop programs (resp. LISP-like recursive programs).

EXAMPLE II.2 A recursive scheme and its algebraic tree

Having defined a binary operation *mult* on a data type, we want to extend that operation to an operation *lmult* on list of elements of that data type by means of the recursive LISP like program P:

$$\begin{aligned}
 \text{lmult}(L, L') &= \text{if } L = \text{NIL} \text{ then NIL else} \\
 &\quad \text{if } L' = \text{NIL} \text{ then NIL else} \\
 &\quad \text{cons}(\text{mult}(\text{car } L, \text{car } L'), \text{lmult}(\text{cdr } L, \text{cdr } L'))
 \end{aligned}$$

where *car* L (resp. *cdr* L) is the first element (resp. the rest) of the list L .

To this recursive program, corresponds a program scheme, naturally obtained by abstracting the meanings of the functions if-then-else, car, etc..., and replacing as much as possible of the right hand side of the above equation by a single function name. We obtain here the scheme S , where $\Phi = \{\text{lmult}\}$, $X = \{L, L'\}$, $\Sigma = \{\perp, \text{cdr}, g\}$.

$$S: \text{lmult}(L, L') = g(L, L', \text{lmult}(\text{cdr}L, \text{cdr}L'))$$

The solution of scheme S (and the corresponding program) is then computed by successive approximations. The n th approximation σ_n to that solution is defined by:

1) unwinding first the scheme S n times, i.e. replacing n times all occurrences of lmult by the right-hand side of S , starting with the term $\text{lmult}(L, L')$. This gives here, after n unwindings $s_n(L, L') = g(L, L', g(\text{cdr}L, \text{cdr}L', g(\dots, g(\text{cdr}^{n-1}L, \text{cdr}^{n-1}L', \text{lmult}(\text{cdr}^nL, \text{cdr}^nL')) \dots)))$.

2) replacing then all occurrences of lmult in s_n by \perp (the totally undefined function) yielding here: $\sigma_n(L, L') = g(L, L', g(\text{cdr}L, \text{cdr}L', g(\dots, g(\text{cdr}^{n-1}L, \text{cdr}^{n-1}L', \perp) \dots)))$.

σ_n belongs to $L(S, \text{lmult}(L, L'))$, but not s_n . In the present case, evaluating σ_n yields:

$$\begin{aligned} \sigma_n(a_1 a_2 \dots a_p, a'_1 a'_2 \dots a'_{p'}) &= \text{mult}(a_1, a'_1) \text{mult}(a_2, a'_2) \dots \text{mult}(a_q, a'_q) \\ &\quad \text{if } q = \inf(p, p') < n \\ &= \perp \quad (\text{the undefined element}) \text{ otherwise.} \end{aligned}$$

We choose here the simpler but ambiguous notation of christening of the same name \perp the undefined elements, or equivalently least elements, of the various domains, syntactic or semantic (data, lists of data), that we consider.

Thus the σ_n make their domain of definition grow larger and larger. That results in a chain $\sigma_0 < \sigma_1 < \dots < \sigma_n < \sigma_{n+1} < \dots$ where the ordering $<$ represents the relation: "to be less defined than".

A tree like representation of S , the s_n 's and σ_n 's which might help the intuition is given in Figure 2.

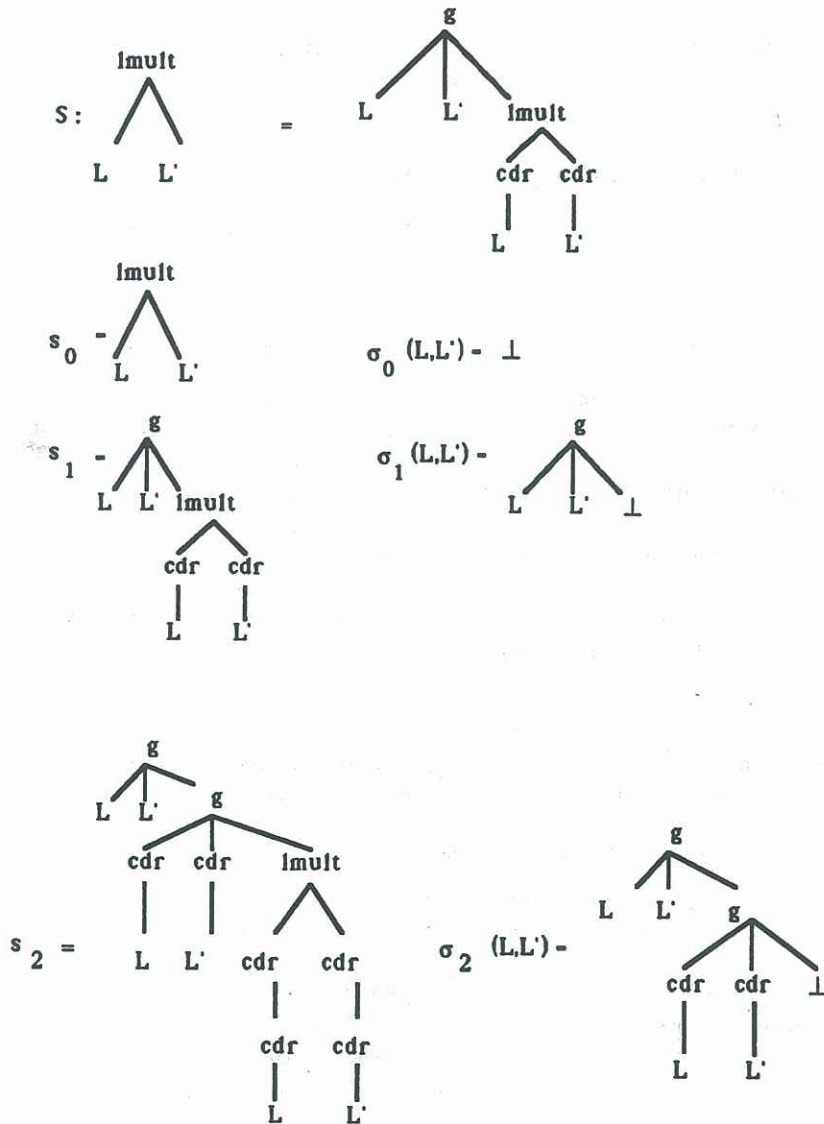


Figure 2.

Finally, the algebraic tree $T(S, \text{lmult}(L, L')) = \text{lub}\{\sigma_n / n \in \mathbb{N}\}$ can be depicted as:

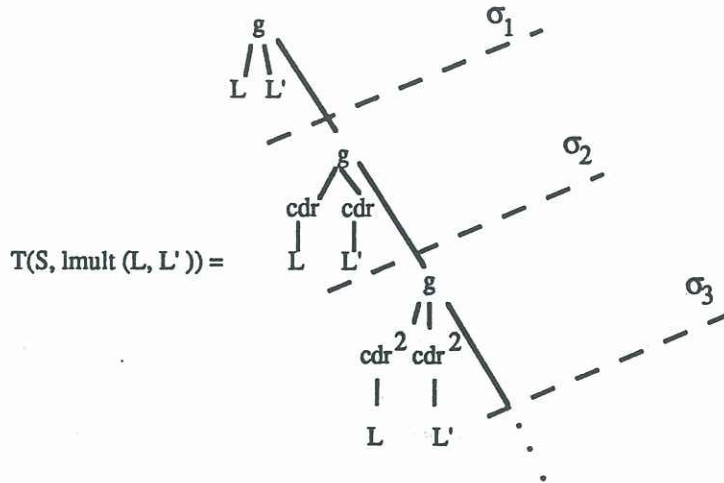


Figure 3.

II.3 Semantics

The basic idea of algebraic semantics is to characterize the meaning of a scheme in the free algebra $CT_{\Sigma}(X)$ via a tree, and to deduce from that tree the meaning of the scheme, and the corresponding programs, in all other possible models, or interpretations.

An interpretation A of Σ is a complete Σ -algebra; a valuated interpretation is an interpretation together with a valuation $a : X \rightarrow A$. An n -tuple $(\gamma_1, \dots, \gamma_n)$ of operations $\gamma_i : A^{n_i} \rightarrow A$ is said to be a solution of S iff the equations of S are satisfied in A endowed with the $\Sigma \cup \{\phi_1, \dots, \phi_n\}$ -algebra structure defined by $\phi_i|_A = \gamma_i$ for $i = 1, \dots, n$; equivalently, $(\gamma_1, \dots, \gamma_n)$ is a fixpoint of the system of equations S in A . Note that $H = CT_{\Sigma_{\perp}}(X)$ is a particular interpretation of Σ , called the Herbrand model. We then have:

THEOREM II.3 The n -tuple $(T_1, \dots, T_n) = (T(S, \phi_1(\vec{x})), \dots, T(S, \phi_n(\vec{x})))$ is the least solution, i.e. the least fixpoint, of the system of equations S in the free algebra $H = CT_{\Sigma_{\perp}}(X)$.

Note that: (i) we identify $T \in H = CT_{\Sigma_{\perp}}(X)$ with T_H , and (ii) for any t , $L(S, t)$ can be obtained by substituting $L(S, t_i)$ to all occurrences of ϕ_i in t , for $i = 1, \dots, n$. By definition, $T(S, t)$ is the function computed by the scheme (S, t) in the free model, or interpretation, H . If A is now an arbitrary interpretation, the function defined by scheme S in A will be defined as $T(S, t)_A$ (cf. Section II.1). The adequacy of this definition follows from the:

THEOREM II.4 Let A be a complete Σ -algebra, and (S, t) and (S', t') be two schemes:

- (i) $T(S, t)_A \leq T(S', t')_A$ for all A iff $T(S, t) \leq T(S', t')$
- (ii) for all A , $T(S, t)_A = \text{lub}\{t_{kA} / t_k \leq T(S, t) \text{ and } t_k \in T_{\Sigma_{\perp}}(X)\}$
- (iii) (T_{1A}, \dots, T_{nA}) is the least solution of S in A .

Theorem II.4 is an immediate consequence of Theorem II.3, together with the freeness of H . In Theorem II.4:

- (i) shows that $T(S, t)$ characterizes the behavior of (S, t) with respect to all models, namely that what happens in the model H suffices to describe what will happen in all other models; the name of Herbrand model comes from that property, together with the fact that H is a term model.
- (ii) says that the function computed by (S, t) in A can be defined as a lub of finite computations, by successive approximations.
- (iii) finally expresses the link between the algebraic semantics above described and the denotational semantics of [Scott].

Theorem II.4 (i) is fundamental in the study of abstract data types [Goguen-Meseguer]: it expresses the fact, that, due to its initiality, the free complete algebra provides an abstract (in the sense that it is

representation independent and unique up to isomorphism) way of studying data types. The initial model $CT_{\Sigma}(X)$ is the abstract data type, wherefrom all other data types can be deduced by homomorphism.

II.4 Nondeterminism

DEFINITION II.5 A non deterministic (as opposed to the deterministic ones considered up to now) recursive scheme on Σ is a pair (S, t) , where S is a system of n equations:

$$S: \quad \phi_i(x_1^i, \dots, x_{n_i}^i) = T_i \quad , \quad i = 1, \dots, n \quad (2)$$

where, for $i = 1, \dots, n$, $\phi_i \in \Phi$, $x_j^i \in X$ for each j , $x_j^i \neq x_k^i$ for $j \neq k$, $T_i \subseteq T_{\Sigma \cup \Phi}(X)$, and $t \in T_{\Sigma \cup \Phi}(X)$. (S, t) is said to be iterative if all the trees in the T_i 's and t are left-linear.

Each scheme S (recursive or iterative) is associated with a tree grammar defined by:

$$S_{\perp}: \quad \phi_i(x_1^i, \dots, x_{n_i}^i) \rightarrow T_i + \perp \quad , \quad i = 1, \dots, n$$

\Rightarrow_S and $\xRightarrow{*}_S$ denote the immediate rewriting according to S_{\perp} and its reflexive and transitive closure: i.e. $t' \xRightarrow{*}_S t''$ if and only if t'' is deduced from t' by substituting some $t_i \in T_i$, or \perp , for one occurrence of some ϕ_i in t' .

The solution of the scheme (S, t) in $CT_{\Sigma}(X)$, where X is the set of variables occurring in t , is the forest:

$$F(S, t) = Fin(S, t) \cup Inf(S, t)$$

where:

$$Fin(S, t) = \{t'/t' \in T_{\Sigma}(X) \text{ and } t \xRightarrow{*}_S t'\} = L(S_{\perp}, t)$$

and

$$Inf(S, t) = \{T'/T' \in CT_{\Sigma}(X) \text{ and } T' = lub\{t_n/n \in \mathbb{N}\} \text{ for some sequence } t_n = t'_n(\bar{1}/\bar{\phi}) \\ \text{with } t'_0 = t \text{ and } \forall n \ t'_n \xRightarrow{*}_S t'_{n+1}\}$$

$Fin(S, t)$ (resp. $Inf(S, t)$) correspond to the finite (resp. infinite) trees over Σ generated by the scheme (S, t) , i.e. the finite (resp. infinite) unfoldings of t according to S . Note that no occurrences of \perp appear in the trees of $F(S, t)$.

EXAMPLE II.6 Consider the recursive scheme:

$$S: \quad \phi(x) = x + a(\phi(b(x)))$$

$(S, \phi(x))$ generates the algebraic forest, which happens here to be identifiable with a word language: $F(S, \phi(x)) = \{a^n b^n(x)/n \geq 0\} \cup \{a^\omega\}$, where a^ω denotes the infinite tree $a^\omega = lub\{a^n(\perp)/n \geq 0\}$. Here $Inf(S, \phi(x)) = a^\omega$.

We can thus see that the case of the deterministic program schemes generating a single infinite tree corresponds to schemes whose finitary part is empty and whose infinitary part is reduced to a single element.

REMARK II.7 Theorem II.3 is no longer true in the case of non deterministic schemes. For instance, in Example II.6, $\{a^n b^n(x)/n \geq 0\} \cup \{a^\omega\}$ is the greatest fixpoint of the system of equations S in $CT_{\Sigma}(X)$; note that $CT_{\Sigma}(X) = \{a, b\}^\omega(x)$. Examples can be found where $F(S, \phi(\bar{x}))$ is neither the greatest nor the least fixpoint of S in the corresponding free algebras [Guessarian89]. This point shows the advantage of the approach of Section II.2, where all schemes were "determinized" by considering "+" and "or" as just formal uninterpreted symbols of rank 2 in Σ , and where we solved the schemes in the free algebra $CT_{\Sigma \cup \Phi}(X)$, by comparison to the "more natural" approach in the non-deterministic case, where the non-deterministic "or" is explicitly interpreted as set union. In the latter case, the natural free domains are $\mathcal{P}(CT_{\Sigma}(X))$, equipped with set inclusion as ordering, or the closed subsets thereof [Arnold-Nivat], and then, $F(S, \phi(\bar{x}))$ is no longer necessarily a component of the least fixpoint of S in the free Σ -algebra, as shown by the above examples. We will use this remark in the next section to help us when giving the semantics of concurrent processes.

III APPLICATION TO SEMANTICS OF CONCURRENCY

III.1 Fundamentals of fixpoint semantics

We first briefly recall the constructive variant of the Knaster-Tarski fixpoint theorem, most widely used in computer science.

THEOREM III.1 (*Fixpoint theorem*) *Let E be a complete Σ -algebra, and $f : E \rightarrow E$ a continuous function; then f has a least fixpoint μf such that $f(\mu f) = \mu f$, and for all $e \in E$, $f(e) = e \Rightarrow e \geq \mu f$. Moreover, μf is computed by $\mu f = \sup\{f^n(\perp) / n \in \mathbb{N}\}$.*

This theorem effectively characterizes a fixpoint computable by induction; the properties of the fixpoint can then also be proved by induction. This theorem also provides the basis of most of the work done in semantics, algebraic semantics [Nivat, ADJ], denotational semantics [Scott, Rounds], logic programming [Van Emden-Kowalski].

The basic idea then is the following: given a program scheme S consisting of a set of recursive equations on a base signature Σ , we first solve S syntactically using Theorem III.1 in the Herbrand model, namely the free continuous Σ -algebra CT_{Σ} consisting of finite and infinite well formed Σ -trees. We thus obtain a generally infinite tree $T(S)$ which is the least fixpoint of S . We then interpret $T(S)$ in an arbitrary model A , by taking its image under a strict continuous morphism ϕ , which gives us the semantics of S in the model A ; this image is the least fixpoint of the program $\phi(S)$ in the model A . The situation is illustrated by the following example.

EXAMPLE III.2 Let $S : F(n) = g(n, F(p(n)))$, where $\Sigma = \{\perp, p, g\}$. Then $T(S)(n) = g(n, g(p(n), g(p^2(n), \dots))) = \sup\{g(n, \perp), g(n, g(p(n), \perp)), \dots\}$. Consider now as model $A = \mathbb{N} \cup \{\perp_A\}$, with the discrete ordering having least element \perp_A . Define $p_A(n) = n - 1$ and $g_A(n, m) = \text{if } n = 0 \text{ then } 0 \text{ else } n + m$, where $+$ and $-$ are the usual operations on \mathbb{N} , and all functions are suitably extended to deal with \perp_A . Then $\phi(S)$ is the recursive definition $f(n) = \text{if } n = 0 \text{ then } 0 \text{ else } n + f(n - 1)$. Let $\phi : CT_{\Sigma} \rightarrow A$ be the morphism defined by $\phi(\sigma) = \sigma_A$ for σ in Σ ; we have $\phi(T(S))(n) = n(n + 1)/2$, and $n \mapsto n(n + 1)/2$ is indeed the least fixpoint of $\phi(S)$ in A , and provides us with the intended meaning of $\phi(S)$ in A .

More generally we have the following, stated for simplicity in the case of a single equation, but which holds for an arbitrary system of recursive equations.

PROPOSITION III.3 *Let Σ be a signature, $S : F = t(F)$ a recursive equation where $t(F)$ is a well-formed term on the signature $\Sigma \cup \{F\}$. Let A be a complete Σ -algebra and ϕ the morphism $\phi : CT_{\Sigma} \rightarrow A$ defined by: $\phi(\sigma) = \sigma_A$ for σ in Σ . Let $\phi(S)$ be the recursive equation $F = t_A(F)$, where t_A is deduced from t by substituting the σ_A 's for the σ 's. Let $\mu S'$ denote the least fixpoint of S' for S' in $\{S, \phi(S)\}$, then $\phi(\mu S) = \mu(\phi(S))$.*

The proof is an immediate consequence of the freeness of CT_{Σ} . We identify the least solution of S with the corresponding least fixpoint. μS gives a syntactic description of the computations of S in the free model CT_{Σ} ; to have an effective description of the computations in an actual model A , it is enough to use Proposition III.3, and take the image of μS under the morphism $\phi : CT_{\Sigma} \rightarrow A$.

III.2 The parallel case

Unfortunately, for parallel or nondeterministic programs, the image of μS under ϕ does not necessarily yield the computations we are looking for, as shown by the:

EXAMPLE III.4 Let $S : F = a \cdot F + \tau \cdot F$ be a recursive definition on the signature $\Sigma = \{\perp, a, \tau, +\}$ and let $B = a^* \cup \{a^\omega\}$. Consider the algebra $A = \mathcal{P}(B)$, ordered by inclusion, with $\perp_A = \emptyset$, $\tau_A = id$, $+_A = \cup$, and $a_A(C) = \{a \cdot c / c \in C\}$. Then μS is the rational tree T defined by $T = a \cdot T + \tau \cdot T$ in CT_{Σ} ; in the algebra A , however, we obtain $\phi(T) = \emptyset$, which is indeed the least fixpoint of $\phi(S) : F = a \cdot F \cup F$. This may be justified when dealing with deterministic programs, because $\phi(S)$ represents a loop; but it is no longer acceptable if we wish to model parallel and nondeterministic programs; in this latter case, τ would represent an invisible move, but we would nevertheless be interested in keeping some track of the infinite sequence of actions a . So, following [Arnold-Nivat] we would in the present case define the intended meaning of $x = ax$ in A as being its greatest fixpoint $\{a^\omega\}$ instead of its least fixpoint \emptyset . Similarly, the semantics of $x = ax \cup x$ in A will be its greatest fixpoint $a^* \cup \{a^\omega\}$ instead of its least fixpoint \emptyset . The same semantics is obtained in [Rounds] with slightly different tools.

Not even greatest fixpoints solve all problems, however:

EXAMPLE III.5 Consider the same signature as in Example III.4, and let $S : F(x) = a \cdot x + F(a \cdot x)$. Let A be defined as in Example III.4, then the greatest fixpoint of $\phi(S) : F(x) = a \cdot x \cup F(a \cdot x)$ is $a^+ \cup \{a^\omega\}$, and there has been some argumentation in the literature as to whether this might be attributed as semantics to $\phi(S)$ (see [Broy, Arnold-Nivat] for conflicting opinions). We consider that a^ω , which is not a limit of partial computations of $\phi(S)$, should therefore not be incorporated to its semantics, for the solace of preserving greatest fixpoints.

EXAMPLE III.6 Combining Example III.4 and Example III.5 into:

$$S : \begin{cases} F(x) = a \cdot x + F(a \cdot x) \\ G = a \cdot G + \tau \cdot G \end{cases}$$

shows that arbitrary fixpoints might be needed.

The problem, as illustrated by the previous examples, is to find a suitable way to translate the least fixpoint in the free model into the "right" fixpoint in the model A . The "right" fixpoint is supposed to take care of finite as well as infinite computations in A . So this problem amounts to studying alternate ways of transforming fixpoints by morphisms. Solutions have been proposed to this problem, but they basically amount to considering a case when the fixpoint is unique, as in the case of Greibach schemes [Arnold-Nivat], or guarded schemes [Bergstra-Klop]. These ideas will not apply because the scheme $\phi(S)$ will not be Greibach or guarded (Example III.5), or the operator corresponding to $\phi(S)$ will not be contracting in A (Example III.4). Example III.4 also shows that we will not always be able to transform a least fixpoint into a greatest one via a closure operator: in that example, the closure of the empty set would have to be $a^* \cup \{a^\omega\}$, which is too demanding.

In the rest of this section, we will address an instance of the problem of transforming least (or unique) fixpoints via morphisms, which will be tailored for describing the semantics of communicating processes. Our formalism will be inspired from CCS (or ACP) - like languages [Milner, Bergstra-Klop]. We will adopt a modular approach, and consider first the nondeterministic case, and then the concurrent case, that we will try to solve using the results obtained in the nondeterministic case.

III.3 Fixpoints and morphisms for nondeterminism

Let A be an alphabet of unary action symbols, let NIL be a constant symbol representing deadlock (or termination), let τ be a distinguished symbol representing an invisible action and $+$ be a binary symbol (representing nondeterministic choice). Let Σ be the signature $\Sigma = \Sigma_1 = \langle NIL, A, \tau, + \rangle$, and let T_Σ (resp. CT_Σ) denote the set of finite (resp. finite and infinite) Σ -trees.

Let $S : F(x) = t(F)(x)$ be a recursive equation, with $t(F)(x) \in T_{\Sigma \cup \{F, x\}}$; t is thus a finite term on the signature Σ . For simplicity of notations we will assume that we have a single recursive equation, the case of several mutually recursive equations would be similar. S is said to be *weakly guarded* (or weakly Greibach) iff it is not of the form $F(x) = F(t'(x))$.

PROPOSITION III.7 Let $S : F(x) = t(F)(x)$ be weakly guarded, then S has a unique fixpoint $T(S)$ in CT_Σ . $T(S)$ is thus also the least fixpoint of the operator $\lambda F. t(F)$, with the notations of [Rounds].

The scheme S is recursive whereas for modelling CCS (or ACP) like processes we need only rational schemes of the form $F = t(F)$. We will keep this more general framework as long as it will not cost us any extra work. Our goal will be to use $T(S)$ to give the semantics of S in the classical model $\mathcal{P}(A^\infty)$ consisting of languages of finite and infinite words on the alphabet A , in a way which would render a good account of its operational semantics. We would like to define ϕ so as to obtain a mapping:

$$\phi : CT_\Sigma \longrightarrow \mathcal{P}(A^\infty) \quad (13)$$

where $\phi(T(S))$ would represent all the computations (i.e. possible finite or infinite sequences of actions of $T(S)$). Moreover, the semantics of NIL , $+$, τ in $\mathcal{P}(A^\infty)$ should satisfy:

$$\begin{aligned} \phi(NIL) &= \epsilon, \quad \phi(T + T') = \phi(T) \cup \phi(T'), \\ \phi(aT) &= a \cdot \phi(T), \quad \phi(\tau T) = \phi(T). \end{aligned} \quad (14)$$

One may try and define ϕ

- inductively for finite trees by the equations (14)
- by "continuity" for infinite trees by:

$$\varphi_1(\sup\{t_n/n \in \mathbb{N}\}) = \sup\{\phi(t_n)/n \in \mathbb{N}\}$$

or by:

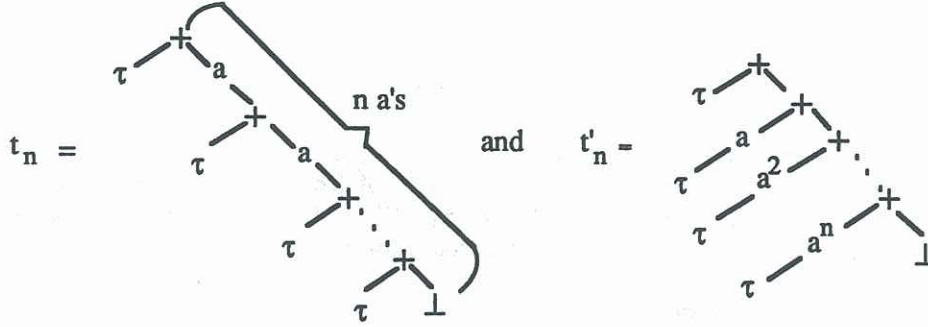
$$\varphi_2(\sup\{t_n/n \in \mathbb{N}\}) = \text{adh}(\sup\{\phi(t_n)/n \in \mathbb{N}\})$$

Recall that:

DEFINITION III.8 For L a language in $\mathcal{P}(A^\infty)$, $\text{adh}(L)$ is the set of infinite words $w \in A^\omega$ such that all left factors of w belong to the set of left factors of L .

But then neither φ_1 nor φ_2 give the desired semantics for infinite trees as shown by the following example:

EXAMPLE III.9 Consider the trees:



the \perp 's are present for technical soundness reasons which will be justified in the next section. We obtain by (14):

$$\phi(t_n) = \phi(t'_n) = \{\epsilon, a, \dots, a^n\}$$

Letting: $T = \sup\{t_n/n \in \mathbb{N}\}$ and $T' = \sup\{t'_n/n \in \mathbb{N}\}$, we will thus obtain, by continuity:

- $\varphi_1(T) = \varphi_1(T') = a^*$, which does not give the right semantics for T , since a^ω should belong to $\varphi_1(T)$
- $\varphi_2(T) = \varphi_2(T') = a^* \cup \{a^\omega\}$, which does not give the right semantics for T' , since a^ω should not belong to $\varphi_2(T')$.

If parallel composition operators are present, the problems might even get worse, because ϕ , as defined by the equations (14) might even be non monotone, rendering impossible an extension by continuity.

We will consider here the case when $\Sigma = \Sigma_1 = \langle \text{NIL}, A, \tau, + \rangle$. For T in CT_{Σ_1} , define $\text{br}(T)$ the set of maximal paths in T , i.e. the root to leaf (or infinite) sequences of pairs of node labels and node numbers along a path in T . We will thus have, for b^1, \dots, b^n , in Σ_1 , and $1, \dots, n$ in \mathbb{N}^* : $(b^1, 1) \dots (b^n, n) \dots \in \text{br}(T)$ iff $1 \dots n \dots$ is a maximal path in T and $b^n = T(n)$ is the label of node n in T . We can formally define $\text{br}(T)$ by induction on k , by letting $T = \sup\{t_k/k \in \mathbb{N}\}$, the t_k 's being finite trees ordered by the relation "to be an initial subtree of".

Let then $\pi_A : \Sigma_1 \times \mathbb{N}^* \rightarrow A$ be the projection morphism defined by:

$$\pi_A(b^n, n) = \begin{cases} b^n, & \text{if } b^n \in A; \\ \epsilon, & \text{if } b^n \in \{\tau, +\}. \end{cases}$$

Define finally: $\phi(T) = \pi_A(\text{br}(T))$. $\phi(T)$ will provide us with the required mapping for the case when $\Sigma = \Sigma_1$. ϕ clearly satisfies the morphism requirements (14), and consequently defines an adequate semantics for finite trees in T_{Σ_1} . Moreover, ϕ also defines a suitable semantics for infinite trees (the reader is invited to check ϕ against Example III.9).

THEOREM III.10 Let

$$S : T = t(T) \tag{15}$$

be a left linear "à la CCS" recursion and $T(S)$ be the least fixpoint of S in CT_{Σ_1} . Then:

(i) $\phi(T(S))$ is a fixpoint of the equation

$$X = \phi[t](X) \tag{16}$$

on $\mathcal{P}(A^\infty)$, where $\phi[t] : \mathcal{P}(A^\infty) \rightarrow \mathcal{P}(A^\infty)$ is defined by induction on the depth of t by, for $X \subseteq \mathcal{P}(A^\infty)$:

$$\phi[t](X) = \begin{cases} X & \text{if } t(X) = X, \\ \varepsilon & \text{if } t(X) = NIL, \\ \phi[t_1](X) \cup \phi[t_2](X) & \text{if } t = t_1 + t_2, \\ a \cdot \phi[t_1](X) & \text{if } t = at_1, \\ \phi[t_1](X) & \text{if } t = \tau t_1. \end{cases}$$

(ii) If moreover $\phi[t](X)$ is weakly guarded (or Greibach), then $\phi(T(S))$ is the greatest fixpoint of (16).

A term $\theta = \bigcup_{i=1}^n w_i X \cup U$ is said to be weakly guarded if at least one of the w_i 's is different from ε , i.e. if θ is not of the form $X \cup U$. t is applied to (possibly infinite) terms in CT_{Σ_1} , whereas $\phi[t]$ is applied to (possibly infinite) sets of words in $\mathcal{P}(A^\infty)$. With the notations of [Guessarian], t would be denoted by $t_{CT_{\Sigma_1}}$, whereas $\phi[t]$ would be denoted by $t_{\mathcal{P}(A^\infty)}$.

Our Theorem III .10 also holds if S is a system of mutually recursive equations of the form:

$$S: \begin{cases} T_1 = t_1(T_1, \dots, T_k) \\ \vdots \\ T_k = t_k(T_1, \dots, T_k) \end{cases} \quad (17)$$

Thus, for systems S of the form (15) or (17), on the signature $\Sigma_1 = \langle NIL, A, \tau, + \rangle$, Theorem III .10 enables us to deduce the computations of S in an arbitrary model A from its syntactic computation in the free model CT_{Σ_1} , via the morphism ϕ defined by $\phi(T) = \pi_A(br(T))$. This yields, in that case, a clean semantics for parallel programs, based on initial algebras and morphisms.

III.4 Extension to true concurrency

The case when $\Sigma = \Sigma_2 = \langle NIL, A, \tau, +, \parallel \rangle$, where \parallel is a binary symbol representing a parallel composition, e.g. the Δ -synchronized shuffle of [Rounds], where actions in Δ are synchronized, and actions out of Δ are interleaved, is more complex. We will adopt a modular approach to the problem, and try to reduce it to the nondeterministic case.

For technical reasons, we will introduce in our signatures a new symbol \perp representing undefined, or "pending" computations, as in the theory of algebraic semantics [Guessarian], or, more recently, [Aceto-Hennessy].

We will in fact consider the more general case of a signature $\Sigma_2 = \langle NIL, A, \tau, +, \perp \rangle \cup \{o_1, \dots, o_n\}$, where the o_i 's are symbols which can be interpreted as \parallel_Δ [Rounds], \parallel [Milner], \parallel [Bergstra-Klop], the restriction or hiding operators, substitutions or renamings, the sequential composition ";", of [Rounds], etc. ...

The only constraint will be that each o_i be interpreted in $\mathcal{P}(A^\infty)$ as an operator \bar{o}_i which can be defined inductively by Definition III .11. For simplicity we state this definition in the case of a single binary operator o , the general case of an operator of arbitrary rank being similar but more tedious.

DEFINITION III.11 Let \bar{o} be defined on $\mathcal{P}(A^\infty)$ as follows: (i) for $w = a_1 \dots a_n = a_1 u$ and $w' = a'_1 \dots a'_p = a'_1 u'$ two finite words in A^* :

$$\begin{aligned} \bar{o}(\varepsilon, \varepsilon) &= \varepsilon \\ \bar{o}(w, w') &= \bigcup_k \chi_k(a_1, a'_1) \cdot \bar{o}(u, u') + \bigcup_k \theta_k(a_1) \cdot \bar{o}(u, w') + \bigcup_k \theta'_k(a'_1) \cdot \bar{o}(w, u') \end{aligned}$$

This definition implies that: $\bar{o}(w, \varepsilon) = \theta(w)$ where θ is a morphism for concatenation. We will in the sequel write the above definition in the shorthand form:

$$\bar{o}(w, w') = \bigcup_k \chi_k(a_1, a'_1) \cdot \bar{o}(\nu_k(w), \nu'_k(w'))$$

(ii) For w, w' two infinite words in A^∞ ,

$$\bar{o}(w, w') = adh\{\cup\{\bar{o}(u, u')/u \text{ left factor of } w, u' \text{ left factor of } w'\}\}.$$

(iii) For L, L' two languages in $\mathcal{P}(A^\infty)$,

$$\bar{o}(L, L') = \cup\{\bar{o}(w, w')/w \in L, \text{ and } w' \in L'\}.$$

In the sequel and in order to simplify notations, we will assume that there is a single binary operator o satisfying Definition III .11. The case of several o_i 's of various ranks is similar.

EXAMPLE III.12 An example of operator o is the Δ -synchronized parallel composition \parallel_{Δ} which is a combination of Milner's parallel composition and Rounds' [Rounds] Δ -synchronized shuffle: it interleaves actions outside of Δ , and can either interleave or synchronize events in Δ . Formally, it satisfies the following expansion laws:

- if $p = \sum_{i \in I} a_i p_i$ and $q = \sum_{j \in J} b_j q_j$ then:

$$p \parallel_{\Delta} q = \sum_{i \in I} a_i \cdot (p_i \parallel_{\Delta} q) + \sum_{j \in J} b_j \cdot (p \parallel_{\Delta} q_j) + \sum_{a_i = \overline{b_j} \in \Delta} \tau \cdot (p_i \parallel_{\Delta} q_j) \quad (18)$$

- if $p = \sum_{i \in I} a_i p_i + \perp$ and $q = \sum_{j \in J} b_j q_j$ or $q = \sum_{j \in J} b_j q_j + \perp$, or if $p = \sum_{i \in I} a_i p_i$ and $q = \sum_{j \in J} b_j q_j + \perp$, then:

$$p \parallel_{\Delta} q = \sum_{i \in I} a_i \cdot (p_i \parallel_{\Delta} q) + \sum_{j \in J} b_j \cdot (p \parallel_{\Delta} q_j) + \sum_{a_i = \overline{b_j} \in \Delta} \tau \cdot (p_i \parallel_{\Delta} q_j) + \perp \quad (18')$$

\bar{o} is then the operator $shuffle_{\Delta}$ which is defined along the lines of the Definition III.11 by: (i) For $w = a_1 \dots a_n, w' = a'_1 \dots a'_p$, two finite words in A^* ,

$$\begin{aligned} shuffle_{\Delta}(\varepsilon, w) &= shuffle_{\Delta}(w, \varepsilon) = w \\ shuffle_{\Delta}(w, w') &= a_1 \cdot shuffle_{\Delta}(a_2 \dots a_n, a'_1 \dots a'_p) \cup a'_1 \cdot shuffle_{\Delta}(a_1 \dots a_n, a'_2 \dots a'_p) \cup S \end{aligned}$$

where

$$S = \begin{cases} shuffle_{\Delta}(a_2 \dots a_n, a'_2 \dots a'_p) & \text{if } a_1 = \overline{a'_1} \in \Delta \\ \emptyset & \text{otherwise.} \end{cases}$$

(ii) For w, w' two infinite words in A^{∞} ,

$$shuffle_{\Delta}(w, w') = adh\{\cup\{shuffle_{\Delta}(u, u')/u \text{ left factor of } w, u' \text{ left factor of } w'\}\}.$$

(iii) For L, L' two languages in $\mathcal{P}(A^{\infty})$,

$$shuffle_{\Delta}(L, L') = \cup\{shuffle_{\Delta}(w, w')/w \in L, \text{ and } w' \in L'\}.$$

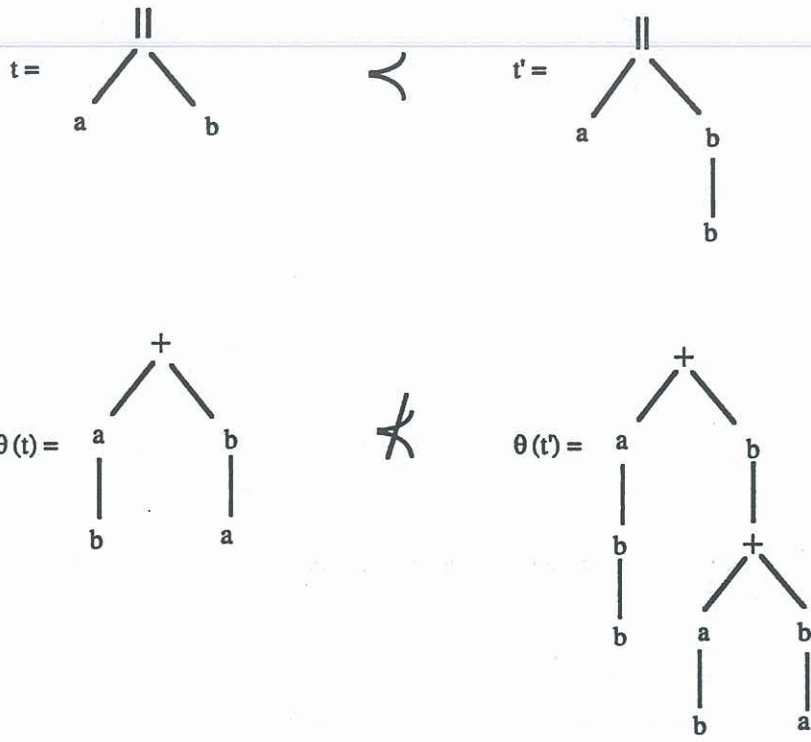
REMARK III.13 Milner's parallel composition \parallel corresponds to \parallel_A .

We would like $\psi = \phi \circ \phi_1 : CT_{\Sigma_2} \rightarrow \mathcal{P}(A^{\infty})$ to satisfy the equations (14) suitably extended in order to deal with the undefined element \perp and the operator o , i.e.:

$$\begin{aligned} \psi(NIL) &= \varepsilon, \psi(T + T') = \psi(T) \cup \psi(T'), \\ \psi(aT) &= a \cdot \psi(T), \psi(\tau T) = \psi(T), \\ \psi(T o T') &= \bar{o}(\psi(T), \psi(T')), \psi(\perp) = \emptyset. \end{aligned} \quad (14')$$

For the same reason as in Example III.9, a direct definition of ψ is not possible. The idea then is to decompose $\psi : CT_{\Sigma_2} \rightarrow \mathcal{P}(A^{\infty})$ into $\psi = \phi \circ \theta$, where $\theta : CT_{\Sigma_2} \rightarrow CT_{\Sigma_1}$, and $\phi : CT_{\Sigma_1} \rightarrow \mathcal{P}(A^{\infty})$. We can here remark that \perp has been added to the signatures Σ_1 and Σ_2 in order to ensure that θ is monotone on finite terms, and thus well-defined (by continuity) on infinite terms. The following counterexample shows that θ is not monotone on T_{Σ_2} if the latter is equipped with the ordering "to be an initial subtree of", without an element \perp .

EXAMPLE III.14 Let $t = a \parallel b$ and $t' = a \parallel b^2$; then, t is an initial subtree of t' , but $\theta(t) = ab + ba$ is not an initial subtree of $\theta(t') = ab^2 + b(ab + ba)$.



Moreover, in order to simplify notations, we will take into account the associativity and commutativity of $+$ and introduce $\phi_1 : CT_{\Sigma_2} \rightarrow CT_{\Sigma_1}/\sim$, where CT_{Σ_1}/\sim is the suitable factor of CT_{Σ_1} , instead of $\theta : CT_{\Sigma_2} \rightarrow CT_{\Sigma_1}$. Notice that the introduction of ϕ_1 is of a "syntactic sugary" nature, whereas the introduction of \perp is a technical necessity. We thus will try and construct a commutative diagram as shown by Figure 4, where \sim is defined by:

DEFINITION III.15 (i) Let \sim be the congruence defined on T_{Σ_2} (the finite terms in CT_{Σ_2}) by the axioms:

$$\begin{aligned} \perp \cdot p &\sim \perp \\ (p + q) + r &\sim p + (q + r) \\ p + q &\sim q + p \end{aligned}$$

(ii) \sim is extended to infinite terms $T, T' \in CT_{\Sigma_2}$ by: $T \sim T'$ iff there exist increasing chains t_n and t'_n such that $T = \sup\{t_n/n \in \mathbb{N}\}$, $T' = \sup\{t'_n/n \in \mathbb{N}\}$ and $t_n \sim t'_n$ for all n .

The restriction of the congruence \sim on T_{Σ_1} (resp. CT_{Σ_1}), is also denoted by \sim .

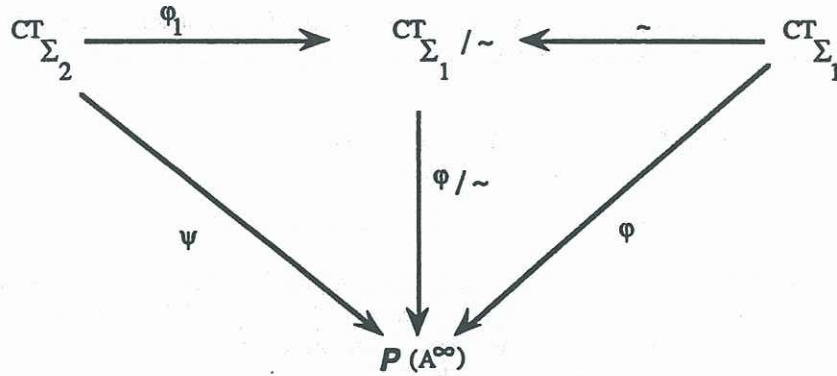


Figure 4.

Recall that CT_{Σ_2} and CT_{Σ_1} now contain an element \perp , representing the undefined process (or computations which are partial but will eventually terminate). \perp is the least element in CT_{Σ_2} , namely $\perp \prec t$, for all t , and CT_{Σ_2} is ordered by the least ordering compatible with the operations in Σ_2 , and having \perp as least

element: i.e. $t \prec t'$ iff t' is deduced from t by substituting trees for occurrences of \perp in t , and \perp 's indicate the cutpoints where branches of t can be extended into branches of t' . The relation t is an initial subtree of t' is now replaced by " $t \prec t'$ ". We then need, in order to fully define the diagram in Figure 4, to first redefine $\phi : CT_{\Sigma_1} \rightarrow \mathcal{P}(A^\infty)$ to take into account the undefined element \perp . To this end, it suffices to extend the previously defined ϕ to trees possibly containing \perp by excluding paths ending with \perp from the set of maximal paths; we thus will not account for such paths in $\phi(T)$, intuitively, "only terminated paths count". ϕ can thus be factored through \sim : for T in CT_{Σ_1}/\sim , $\beta(T) = \phi/\sim(T)$ is defined by:

- 1) taking the set of maximal paths in T , i.e. paths that either are infinite, or are root to leaf paths with a leaf different from \perp .
- 2) erasing labels $+$ and τ along such paths.

We then will associate with each T in CT_{Σ_2} a tree $\phi_1(T)$ in CT_{Σ_1}/\sim ; $\phi_1(T)$ will represent a "normal form" of T after elimination of the \bar{o} 's and taking into account the associativity and commutativity of $+$. We finally will apply a slight variation of ϕ to obtain $\psi(T) = \phi(\phi_1(T))$ in $\mathcal{P}(A^\infty)$. This will yield the commutative diagram of Figure 4. We now define formally $\phi_1(T)$ for T in CT_{Σ_2} .

DEFINITION III.16 (i) For t a finite tree in T_{Σ_2} , $\phi_1(t) \in T_{\Sigma_1}/\sim$ is defined inductively as follows:

- if $t \in CT_{\Sigma_1}$, then $\phi_1(t) = [t]_\sim$,
- if $t = NILot'$, or $t = t'ONIL$, then $\phi_1(t) = \phi_1(t')$,
- if $t = poq$, with $p = \sum_{i \in I} a_i p_i + \perp$, and $q = \sum_{j \in J} a'_j q_j + \perp$, then

$$\phi_1(t) = \sum_k \sum_{i \in I, j \in J} \chi_k(a_i, a'_j) \cdot \phi_1(\delta_k(p) o \delta'_k(q)) + \perp$$

$$\text{where: } \delta_k(p) = \begin{cases} p & \text{if } \nu_k = id \\ p_i & \text{otherwise} \end{cases} \quad \delta'_k(q) = \begin{cases} q & \text{if } \nu'_k = id \\ q_j & \text{otherwise} \end{cases}$$

(The \perp in $\phi_1(t)$ disappears if there is no \perp in neither p nor q),

- if $t = poq$, with p and q not of the above form, then $\phi_1(t) = \phi_1(\phi_1(p) o \phi_1(q))$,
- if $t = \sum_{i \in I} a_i p_i (+ \perp)$, then $\phi_1(t) = \sum_{i \in I} a_i \phi_1(p_i) (+ \perp)$.

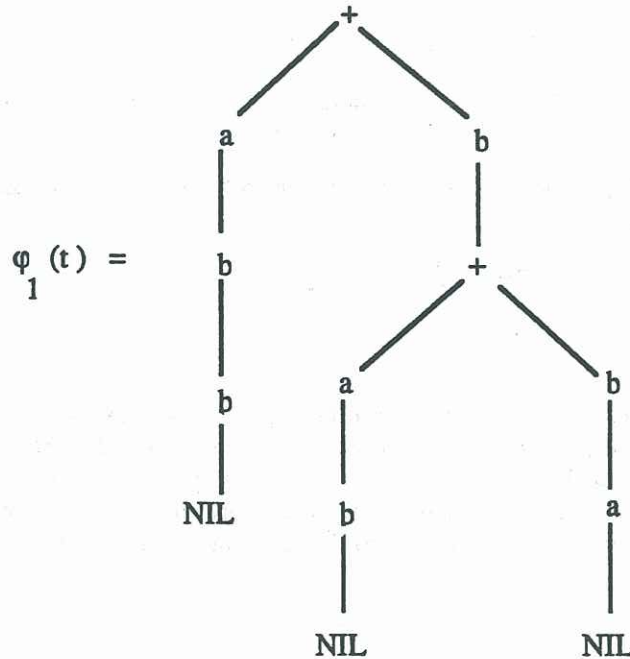
(ii) for $T = \sup\{t_n/n \in N\}$ an infinite tree in CT_{Σ_2} , we define $\phi_1(T) = \sup\{\phi_1(t_n)/n \in N\} \in CT_{\Sigma_1}/\sim$.

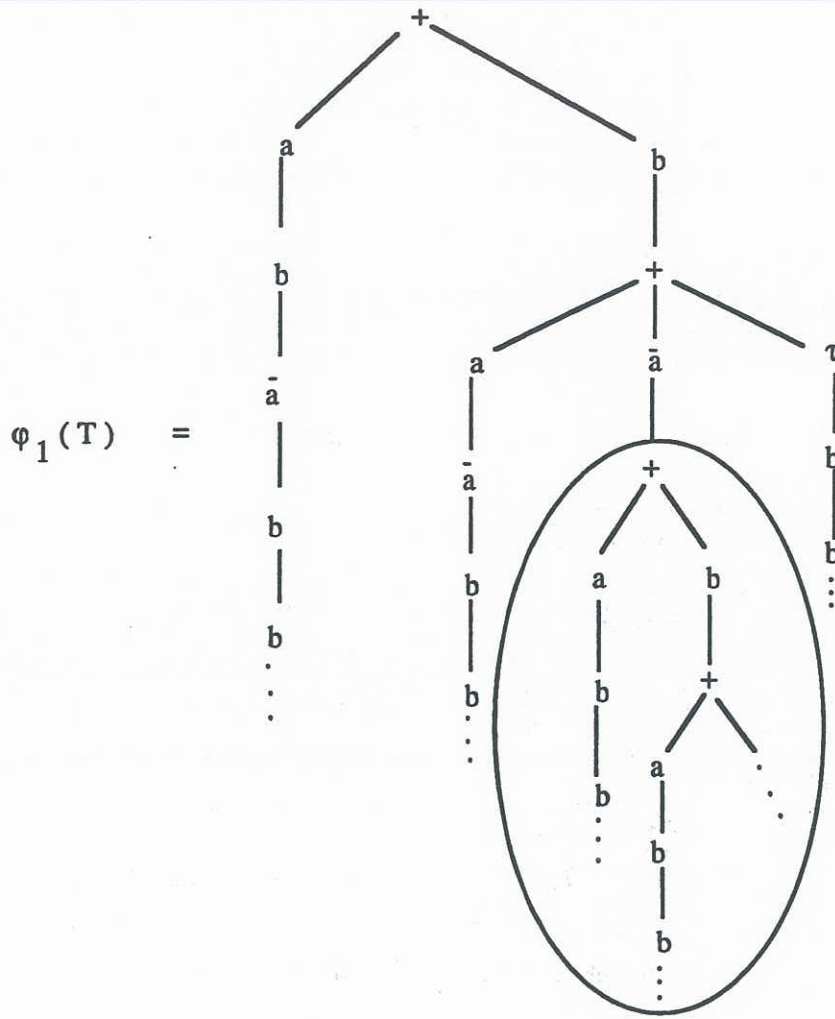
$\phi_1(t)$ can be considered to be an o -free normal form of t in CT_{Σ_1}/\sim .

EXAMPLE III.17 Let $\Delta = \{a, \bar{a}\}$, $\|\Delta$ be defined as in Example III.12, $t = aNIL \|\Delta b^2NIL$ and $T = aNIL \|\Delta b\bar{a}b^\omega$; then:

$$\phi_1(T) = ab\bar{a}b^\omega + b(a\bar{a}b^\omega + \bar{a}T_1 + \tau b^\omega),$$

where $T_1 = ab^\omega + bT_1$, i.e. $\phi_1(t)$ is a $\|\Delta$ -free normal form of t in CT_{Σ_1}/\sim .





T_1 is circled.

$\psi : CT_{\Sigma_2} \rightarrow \mathcal{P}(A^\infty)$ can now be defined by:

$$\psi = \beta \circ \phi_1 = (\phi/\sim) \circ \phi_1$$

as announced in Figure 4. We then have to check the adequacy of the above defined ψ . We will first consider the case of finite trees in CT_{Σ_2} , and check that ψ gives us the right semantics.

PROPOSITION III.18 *Let $\psi_1 : T_{\Sigma_2} \rightarrow \mathcal{P}(A^*)$ be defined inductively using the equations (14'). For finite trees in T_{Σ_2} , $\psi_1(t) = \psi(t)$.*

COROLLARY III.19 *ψ satisfies the equations (14').*

III.5 Conclusion

We have in the present paper showed how fixpoints can be applied to semantics of concurrency, through the study of transformations of solutions to program schemes, or more generally sets of equations, under factoring morphisms. We obtained, in the case of the nondeterministic programs, a morphism transforming the syntactical least fixpoint into the semantical meaning of the nondeterministic program; we then used that morphism to define another morphism giving the semantics of a program possibly involving true concurrency together with nondeterminism.

IV REFERENCES

- [Aceto-Hennessy] L. Aceto, M. Hennessy, Termination, deadlock and divergence, to appear in *J. Assoc. Comput. Mach.*
- [ADJ] J. Goguen, J. Thatcher, E. Wagner, J. Wright, Initial Algebra Semantics and Continuous Algebras, *J. Assoc. Comput. Mach.* 24 (1977), 68-95.
- [Apt-Blair-Walker] K.R. Apt, H. Blair, A. Walker, Towards a theory of declarative knowledge, in *Foundations of Deductive Databases and Logic Programming*, J. Minker ed., Morgan-Kaufmann, Los Altos (1988), 89-148.
- [Arnold-Nivat] A. Arnold, M. Nivat, The metric space of infinite trees: Algebraic and Topological properties, *Fund. Inform.* 3 (1980), 445-476.
- [Bergstra-Klop] J. Bergstra, J. Klop, Algebra of communicating processes, *Proc. CWI Symposium Mathematics and Computer Science*, J. de Bakker, M. Hazenwinkel and J. Lenstra, eds. (1986).
- [Birkhoff] G. Birkhoff, *Lattice Theory*, 3rd edition, *AMS Coll.*, New York (1979).
- [Broy] M. Broy, On the Herbrand-Kleene universe for nondeterministic computations, MFCS 84, LNCS 176, Springer-Verlag (1981), 214-222.
- [Darondeau-Gamatié] P. Darondeau, B. Gamatié, A fully observational model for infinite behaviors of communicating systems, to appear.
- [van Emden-Kowalski] M.H. Van Emden, R.A. Kowalski, The Semantics of Predicate Logic as a Programming Language, *Jour. Assoc. Comput. Mach.* 23 (1976), 733-742.
- [Gallaire-Minker-Nicolas] H. Gallaire, J. Minker, J. M. Nicolas, Logic and data bases: a deductive approach, *Assoc. Comput. Mach. Comput. Surveys*, 16 (1984), 153-185.
- [Goguen-Meseguer] J. Goguen, J. Meseguer, Initiality, Induction and Computability, in *Algebraic Methods in Semantics*, M. Nivat and J. Reynolds eds., Cambridge Univ. Press (1985), 459-540.
- [Guessarian89] I. Guessarian, Improving fixpoint tools for computer science, *IFIP'89 Proc.*, G. Ritter ed., North-Holland, Amsterdam (1989), 1109-1114.
- [Guessarian] I. Guessarian, *Algebraic Semantics*, Springer-Verlag, LNCS 99, Berlin (1981).
- [Makinson] D. Makinson, General theory of cumulative inference, unpublished manuscript.
- [Milner] R. Milner, A calculus of communication systems, LNCS 92, Springer-Verlag, Berlin (1980).
- [Nivat] M. Nivat, On the interpretation of recursive polyadic program schemes, *Symp. Mathematica* 15 (1975) 255-281.
- [Rounds] W. Rounds, On the relationships between Scott domains, synchronization trees, and metric spaces, *Inf. and Control* 66 (1985), 6-28.
- [Scott] D. Scott, Data types as lattices, *SIAM Jour. Comput.* 5 (1976), 522-587.
- [Tiuryn] J. Tiuryn, Unique fixed points vs. least fixed points, *Theor. Comput. Sci.* 12 (1980), 229-254.