Dynamic Programming as Graph Searching: An Algebraic Approach

STEFANIA GNESI AND UGO MONTANARI

University of Pisa, Pisa, Italy

AND

ALBERTO MARTELLI

Istituto di Elaborazione della Informazione del C N R, Pisa, Italy

ABSTRACT. Finding the solution of a dynamic programming problem in the form of polyadic functional equations is shown to be equivalent to searching a minimal cost path in an AND/OR graph with monotone cost functions The proof is given in an algebraic framework and is based on a commutativity result between solution and interpretation of a symbolic system This approach is similar to the one used by some authors to prove the equivalence between the operational and denotational semantics of programming languages

KEY WORDS AND PHRASES: dynamic programming, functional equations, AND/OR graph, graph search, continuous algebras, algebraic semantics

CR CATEGORIES 3 64, 5 24, 5 42

1. *Introduction*

Dynamic programming [2, 22] is a well-known methodology for representing optimization problems in terms of equations on reals whose left members are unknowns and right members are expressions containing the minimum operation and monotone functions only. These equations are called *functional equations* and are usually solved with an iterative method.

Let us consider, for instance, a typical dynamic programming problem, the problem of finding a shortest path in a graph. Let G be a graph with nodes $N = \{1, \ldots, n\}$ and a positive real number c_{ij} associated with every arc (i, j). We want to determine a path of minimum length from node 1 to node n, where the length of a path is the sum of the costs of its arcs. The dynamic programming formulation of this problem is the following. Let x_i be the length of the shortest path from node 1 to node i. Then we can write the following system of *functional equations*:

$$x_1 = 0,$$
$$x_i = \min_{j=1,\ \ldots,n} (f_{ji}(x_j)), \qquad i = 2, \ldots, n,$$

Authors' addresses S Gnesi and U. Montanari, Istituto di Scienze dell'Informazione, University of Pisa, C/so Italia, 40, I-56100 Pisa, Italy, A Martelli, Istituto di Elaborazione della Informazione del C N R, Via S Maria, 46, I-56100 Pisa, Italy

where *cost functions* f_{ji} have the additive form $f_{ji}(y) = y + c_{ji}$. Such a system can be solved in many different ways [9].

In general, cost functions are neither restricted to be additive nor monadic. In fact, interesting problems such as those of finding an optimal binary search tree [11, 13] or optimal decision table conversion [17, 19] can be represented by systems of functional equations with polyadic cost functions. The property of cost functions which actually characterizes dynamic programming is *monotonicity.*

The above example shows the equivalence between the problem of determining a shortest path in a graph and that of solving a system of functional equations with additive monadic cost functions. The same equivalence result was shown to hold in the more general case of monotone monadic cost functions by Karp and Held [10], who introduced a formal model of dynamic programming, based on automata theory, called *sequential decision process.* This model was further developed by Ibaraki [7, 8]. A similar result was also proved by Martelli and Montanari [15].

The relationships between dynamic programming and graph search were further extended by Martelli and Montanari [14], who showed that in the case of polyadic cost functions, the solution of a dynamic programming problem can be obtained by searching for a minimal cost solution tree in an AND/OR graph.

In this paper the equivalence between solving a system of monotone polyadic functional equations and searching an AND/OR graph with monotone cost functions is proved in an algebraic framework by using an approach similar to the one recently used in proving the equivalence between the operational and denotational semantics of programming languages [1, 4]. An advantage of such an approach is that we can deal also with the case where the optimal cost is achieved as the limit of a sequence of finite solution tree costs.

Our equivalence proof is based on a commutativity theorem for continuous algebras. Such a theorem is given in Section 2.3, together with the algebraic background; the definition of an AND/OR graph is given in Section 2.2. In Section 3 we give an algebraic definition of interpreted functional equations. In Section 4 we introduce symbolic functional equations, that is, equations on sets of trees built with uninterpreted function symbols, and we prove a theorem stating that the solution of a system of symbolic functional equations gives all finite solution trees of a corresponding AND/OR graph. Finally, in Section 5 we introduce a system of functional equations S as a system of symbolic functional equations SS together with an interpretation I. In Section 6 we prove that to solve S we can either give the interpretation I to the symbolic system SS, obtaining an interpreted system IS, and solve it; or we can solve SS, obtaining a set of trees, and give to the resulting set of trees the interpretation I. Here to give an interpretation to a set of trees means to evaluate the costs of all trees and find the greatest lower bound of such costs. The two theorems of Section 4 and 6 imply our main result: To solve a system of functional equations $S = (SS, I)$ by deriving the interpreted system IS and solving it is equivalent to searching the AND/OR graph corresponding to SS for the cheapest (according to I) solution tree. If the minimum does not exist, we can take the greatest lower bound of all solution tree costs.

AND/OR graphs and solution trees (i.e., generalized paths) are equivalent to various well-known formalisms, as shown by Martelli and Montanari in [14]. For instance, it is easy to interpret an AND/OR graph as the transition function of a *tree automaton* [21]. The set of trees accepted by a top-down (or a bottom-up) nondeterministic tree automaton is exactly the set of all finite solution trees of the corresponding AND/OR graph. Another equivalent formalism is that of *regular tree grammars.*

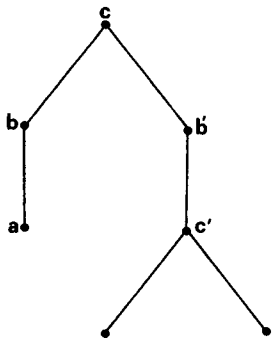


FIGURE 1

For instance, the systems of symbolic functional equations which are introduced in Section 4 might be interpreted as regular tree grammars, and their solutions as the equational languages of trees denoted by such grammars [3]. Theorem 4.1 proves that these languages are the set of all finite solution trees of the corresponding AND/OR graphs. In this paper we choose AND/OR graphs and solution trees, since they are the most immediate generalization of (transition) graphs and paths.

The equivalence result proved in this paper has practical applications, since it allows various graph searching techniques to be translated into techniques for solving functional equations of dynamic programming, and vice versa.

2. *Background*

2.1 Σ-TREES. In this paper we use a tree domain which is based on CT_Σ as defined in [4]. Let Σ be a *ranked alphabet*, that is, a family of disjoint alphabets Σ_i ($i \in \omega$, $\omega = \{0, 1, 2, \ldots\}$) indexed by the natural numbers. The symbols of alphabet Σ_k may be interpreted as k-adic function symbols.

A Σ-*tree* is a partial tree labeled with symbols of Σ, where partial means that some of its leaves may be unlabeled. For instance, in Figure 1 we have a Σ-tree, with Σ defined as follows:

$$\Sigma_0 = \{a\}, \qquad \Sigma_1 = \{b, b'\}, \qquad \Sigma_2 = \{c, c'\}, \qquad \Sigma_3 = \Sigma_4 = \cdots = \varnothing,$$

where a is a zeradic function (constant) symbol, b and b' are monadic function symbols, c and c' are dyadic function symbols.

A Σ-tree t may be defined formally as a partial function

$$t: \omega^* \multimap \Sigma,$$

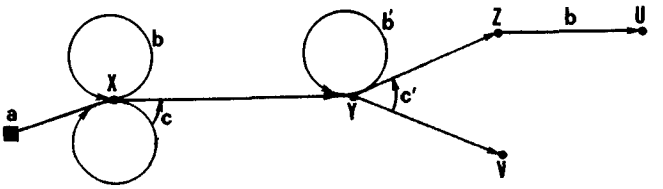
where ω^* is the set of strings of natural numbers of length 0, 1, 2, ... In order to represent a Σ-tree, this function must satisfy the following conditions for all $u \in \omega^*$ and $i \in \omega$:

(a) $ui \in \text{def}(t)$ implies $u \in \text{def}(t)$;
(b) $ui \in \text{def}(t)$ implies $t(u) \in \Sigma_n$ and $i < n$ for some $n > 0$;

where def(t) is the definition domain of partial function t.

The strings of natural numbers in the domain of t may be interpreted as encoding of paths in the corresponding tree from the root to any labeled node, and function t gives the label of this node. Thus condition (a) means that if path ui exists, then certainly path u exists as well. Condition (b) means that if path ui exists, then the last

FIGURE 2



node of path u must be labeled with a correctly ranked symbol. For instance, the tree in Figure 1 is represented by the function

$$t:\{\lambda, 0, 1, 00, 10\} \rightarrow \{a, b, b', c, c'\}$$

where $\lambda \mapsto c$, $0 \mapsto b$, $1 \mapsto b'$, $00 \mapsto a$, $10 \mapsto c'$, and λ is the empty string.

The set CT_Σ of all such trees is ordered according to the standard order of partial functions, namely, $t' \preceq t''$ iff $\text{def}(t') \subseteq \text{def}(t'')$, and $\forall u \in \text{def}(t')$ we have $t'(u) = t''(u)$. If $t' \preceq t''$, we say that t' *approximates* t''.

According to the above definition, Σ-trees can be finite or infinite. A Σ-tree t is *finite* if $\text{def}(t)$ is finite and infinite otherwise. In this paper we use only finite Σ-trees.

2.2 AND/OR GRAPHS. AND/OR graphs are well known in artificial intelligence as a model of problem reduction [18]. An *AND/OR graph* can be defined as a hypergraph, that is, a graph where instead of arcs connecting pairs of nodes there are hyperarcs connecting n-tuples of nodes ($n = 1, 2, 3, \ldots$). Hyperarcs are called *connectors*, and they must be considered as being directed from their first node to all the others. Formally, an AND/OR graph G [14, 17] is a pair (N, C), where N is a finite set of nodes and C is a set of connectors

$$C \subseteq N \times \bigcup_{k=0}^{m} N^k.$$

Each k-connector $(A, A_1, \ldots, A_k)$ is an ordered $(k + 1)$-tuple, where A is the *input node* and $A_1, \ldots, A_k$ are the *output nodes*. When $C \subseteq N^2$, we have a usual graph whose arcs are the 1-connectors. 0-connectors are connectors with one input and no output node. Given a ranked alphabet Σ, a *Σ-labeled AND/OR graph* is an AND/OR graph whose k-connectors are labeled with symbols in the alphabet Σ_k.

As an example let us consider the Σ-labeled AND/OR graph in Figure 2. The nodes are denoted by capital letters, 0-connectors are represented as a line ending with a square, and k-connectors ($k > 0$) as k directed lines connected together. Alphabet Σ was given in the example of Section 2.1. For instance, node X is the input node of a 0-connector (X) labeled a, a 1-connector (X, X) labeled b, and a 2-connector (X, X, Y) labeled c.

A *Σ-labeled AND tree* is a Σ-labeled AND/OR graph where every node is input node of at most one connector and every node is output node of exactly one connector, except for a node, the *root*, which is the output node of no connector. An example is given in Figure 3, where X_1 is the root and nodes V_1 and Z_1 are input nodes of no connector.

It is easy to see that a Σ-labeled AND tree can properly represent a Σ-tree, and vice versa. For instance, the AND tree in Figure 3 corresponds to the tree given in Section 2.1 and represented in Figure 1. Owing to this isomorphism, from now on we will speak interchangeably of AND trees or Σ-trees.

Let H be a Σ-labeled AND/OR graph, X a node of H, and T a finite Σ-labeled AND tree. Then we say that T is a *solution tree of H rooted at node X* iff there exists

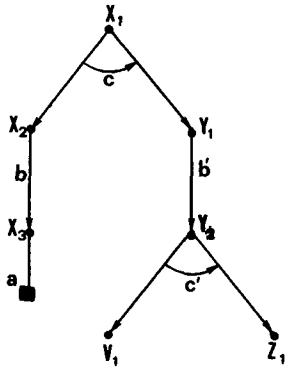


FIGURE 3

a homomorphism mapping T into H. More precisely, there must exist a function h mapping nodes of T into nodes of H such that if a connector $(A, A_1, \ldots, A_k)$ exists in T and is labeled with σ, then a connector $(h(A), h(A_1), \ldots, h(A_k))$ labeled with σ must also exist in H. Furthermore, h must map the root of T into node X.

For instance, the AND tree in Figure 3 is a solution tree of the graph in Figure 2 rooted at node X. In fact, the following suitable function h exists:

$$h(X_1) = h(X_2) = h(X_3) = X, \qquad h(V_1) = V,$$
$$h(Y_1) = h(Y_2) = Y, \qquad h(Z_1) = Z.$$

The concept of a solution tree of an AND/OR graph, together with the function mapping it into the graph, can be seen as a generalization of the concept of path in ordinary graphs (in fact it reduces to it if the graph contains only 1-connectors).

Finally we define the *depth* of an AND tree t as follows:

(a) if the root of t has no outgoing connector, then the depth of t is 0;
(b) if the root of t is input node of a 0-connector, then the depth of t is 1;
(c) if the root of t is input node of a k-connector ($k > 0$), then the depth of t is obtained by adding 1 to the maximum depth of all subtrees rooted at the output nodes of the connector.

For instance, the depth of the AND tree in Figure 3 is 3.

2.3 CONTINUOUS ALGEBRAS. Let A be an algebra (a set with operations whose names are taken from a ranked alphabet) and I be the set of its elements. Set I is called the *carrier* of A [4]. Let us then assume a *partial order* $\sqsubseteq$ over I. Algebra A is called *continuous* if the following properties are satisfied:

(i) The carrier I of A is a *complete* partial order (cpo), namely, every chain $x_0 \sqsubseteq x_1 \sqsubseteq x_2 \sqsubseteq \cdots$, $x_i \in I$, $i = 0, 1, 2, \ldots$, has a least upper bound $\bigsqcup_{i=0}^{\infty} x_i = \bar{x} \in I$; and a bottom element $\perp$ exists, namely $\forall x \in I$, $\perp \sqsubseteq x$.
(ii) Every (elementary) operation of A is a monotone, continuous function, namely, if σ_A is an n-adic operation of A, then

$$x'_k \sqsubseteq x''_k \quad \text{implies} \quad \sigma_A(x_1, \ldots, x'_k, \ldots, x_n) \sqsubseteq \sigma_A(x_1, \ldots, x''_k, \ldots, x_n) \qquad \text{(monotonicity)},$$

and

$$\bigsqcup_{i=0}^{\infty} \sigma_A(x_1, \ldots, x_k^i, \ldots, x_n) = \sigma_A(x_1, \ldots, \bigsqcup_{i=0}^{\infty} x_k^i, \ldots, x_n), \qquad x_k^0 \sqsubseteq x_k^1 \sqsubseteq \cdots \qquad \text{(continuity)}.$$

Given a continuous algebra A, it is possible to obtain its *derived* operations by composing its elementary operations. It is easy to see also that derived operations are monotone, continuous functions. Let us now write a *system of equations* in A

$$\begin{aligned} x_1 &= \tau_1(x_1, x_2, \ldots, x_n), \\ x_2 &= \tau_2(x_1, x_2, \ldots, x_n), \\ &\vdots \\ x_n &= \tau_n(x_1, x_2, \ldots, x_n), \end{aligned} \tag{2.1}$$

where x_i, $i = 1, \ldots, n$, are variables over I and τ_i are (derived) operations. If we define a partial order over tuples of I,

$$(x'_1, \ldots, x'_k, \ldots x'_n) \sqsubseteq (x''_1, \ldots, x''_k, \ldots, x''_n) \quad \text{iff} \quad x'_k \sqsubseteq x''_k, \quad k = 1, \ldots, n,$$

we can apply the *fixpoint theorem* [20], namely, that system (2.1) has a least solution $(\bar{x}_1, \ldots, \bar{x}_n)$ which is the least upper bound of the chain of n-tuples $(x_1^i, \ldots, x_n^i)$, $i = 0, 1, \ldots$, obtained by starting with $x_k^0 = \perp$, $k = 1, \ldots, n$, and iteratively applying the operators in the right member of the equations:

$$x_k^{i+1} = \tau_k(x_1^i, \ldots, x_n^i), \qquad k = 1, \ldots, n.$$

We now need to introduce some concepts concerning *homomorphism* between continuous algebras.

Let A and B be two continuous algebras, I_A and I_B be their carriers, and A and B have the same *signature* (i.e., the same operation names). A function $h: I_A \to I_B$ defines a *homomorphism* from A to B iff for every zeradic function (i.e., for every constant) name a, we have

$$h(a_A) = a_B, \tag{2.2}$$

and for every n-adic function name σ, $n = 1, 2, \ldots, m$, we have

$$h(\sigma_A(x_1, x_2, \ldots, x_n)) = \sigma_B(h(x_1), h(x_2), \ldots, h(x_n)). \tag{2.3}$$

Given a derived operation τ_A in algebra A, it is possible to define a corresponding derived operation τ_B in algebra B. It is sufficient to consider any expression in terms of primitive operations of A denoting τ_A and evaluate it in terms of the homonymous operations of B. It is easy to see that τ_B does not depend on the particular chosen expression and that the commutative property holds for derived operations as well:

$$h(\tau_A(x_1, x_2, \ldots, x_n)) = \tau_B(h(x_1), h(x_2), \ldots, h(x_n)).$$

A homomorphism is called *strict* if it maps bottom into bottom,

$$h(\perp_A) = \perp_B, \tag{2.4}$$

and it is called *monotone* if it preserves the partial orders of A and B,

$$x \sqsubseteq_A y \qquad \text{implies} \qquad h(x) \sqsubseteq_B h(y). \tag{2.5}$$

A monotone homomorphism is called *continuous* iff it preserves the limits of chains,

$$h\left(\bigsqcup_{i=0}^{\infty}{}_A x_i\right) = \bigsqcup_{i=0}^{\infty}{}_B h(x_i), \qquad x_0 \sqsubseteq x_1 \sqsubseteq \cdots. \tag{2.6}$$

We can now prove the following theorem, which is a generalization of [4, Prop. 5.1].

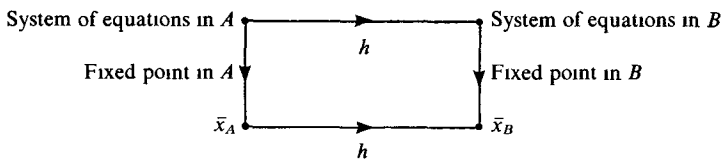


FIGURE 4

THEOREM 2.1. *Let A and B be continuous algebras and h be a strict, continuous homomorphism from A to B. Let*

$$x_k = \tau_k^A(x_1, x_2, \ldots, x_n), \qquad k = 1, \ldots, n,$$

be a system of equations in algebra A and $\bar{x}_k^A$ *be its least solution. Let*

$$x_k = \tau_k^B(x_1, x_2, \ldots, x_n), \qquad k = 1, \ldots, n,$$

be the corresponding system of equations in algebra B, obtained by replacing in the right members of equations the derived operations of A with the corresponding derived operations of B, and let $\bar{x}_k^B$ *be its least solution. We have*

$$h(\bar{x}_k^A) = \bar{x}_k^B, \qquad k = 1, \ldots, n.$$

PROOF. Let $((x_1^{A,0}, \ldots, x_n^{A,0}), (x_1^{A,1}, \ldots, x_n^{A,1}), \ldots)$ and $((x_1^{B,0}, \ldots, x_n^{B,0}), (x_1^{B,1}, \ldots, x_n^{B,1}), \ldots)$ be the approximating chains for $\bar{x}_n^A$ and $\bar{x}_k^B$ obtained in A and B by applying the fixpoint theorem. We will prove inductively that

$$h(x_k^{A,i}) = x_k^{B,i}, \qquad k = 1, \ldots, n, \quad i = 0, 1, \ldots.$$

In fact, we have

$$h(x_k^{A,0}) = h(\bot_A) = \bot_B = x_k^{B,0}, \qquad k = 1, \ldots, n,$$

since h is strict. Let us assume

$$h(x_k^{A,i}) = x_k^{B,i}, \qquad k = 1, \ldots, n.$$

We have also

$$\begin{aligned} h(x_k^{A,i+1}) &= h(\tau_k^A(x_1^{A,i}, \ldots, x_n^{A,i})) = \tau_k^B(h(x_1^{A,i}), \ldots, h(x_n^{A,i})) \\ &= \tau_k^B(x_1^{B,i}, \ldots, x_n^{B,i}) = x_k^{B,i+1}, \qquad k = 1, \ldots, n. \end{aligned}$$

Finally, since h is continuous, it also preserves the limits of the approximating chains, namely,

$$\begin{aligned} h(\bar{x}_k^A) &= h\left(\bigsqcup_{i=0}^{\infty}{}_A\, x_k^{A,i}\right) = \bigsqcup_{i=0}^{\infty}{}_B\, h(x_k^{A,i}) \\ &= \bigsqcup_{i=0}^{\infty}{}_B\, x_k^{B,i} = \bar{x}_k^B, \qquad k = 1, \ldots, n. \end{aligned}$$

□

We can represent the result of this theorem with the commutativity of the diagram in Figure 4.

3. *Interpreted Functional Equations*

In this section we introduce the concept of interpreted functional equations by defining a suitable algebra A_R. The relation with functional equations of dynamic programming is explained in Section 5.

Algebra A_R has one carrier, I_R, defined as the real numbers with an ordering $\sqsubseteq$ which is the reciprocal of the usual ordering $\leq$ plus a "bottom" element $+\infty$ and a "top" element $-\infty$, to obtain a complete partial order.

The elementary operations on A_R (*the cost functions*) are denoted by the symbols of a given ranked alphabet Σ. As we mentioned before, the property of cost functions which characterizes dynamic programming is *monotonicity*. Thus, given a function $\sigma_R: I_R^n \to I_R$, where $\sigma \in \Sigma_n$, we must have

$$x_k' \geq x_k'' \qquad \text{implies} \qquad \sigma_R(x_1, \ldots, x_k', \ldots, x_n) \geq \sigma_R(x_1, \ldots, x_k'', \ldots, x_n).$$

Furthermore, we require these functions to be continuous from above; that is,

$$\lim_{\substack{x_k \to \bar{x}_k \\ x_k > \bar{x}_k}} \sigma_R(x_1, \ldots, x_k, \ldots, x_n) = \sigma_R(x_1, \ldots, \bar{x}_k, \ldots, x_n).$$

This property is required to guarantee the convergence of the iterative algorithm for solving a system of equations.

Besides these functions, algebra A_R possesses a minimum operation. Since we have

$$x' \geq x'' \qquad \text{implies} \qquad \min(x', y) \geq \min(x'', y)$$

and

$$\lim_{x \to \bar{x}} \min(x, y) = \min(\bar{x}, y),$$

min is a monotone continuous function as well. Therefore we can conclude that A_R is a continuous algebra.

A *system of interpreted functional equations* is a finite system of equations in the algebra A_R.

4. *Symbolic Functional Equations and AND/OR Graphs*

We introduce here *symbolic functional equations*, that is, equations in an algebra A_L defined as follows.

Let Σ be the same ranked alphabet of the previous section. The carrier I_L of A_L consists of all sets of finite Σ-trees, with the following restrictions:

(i) Prefix property: If a set contains a Σ-tree t, then it must also contain all Σ-trees t' such that $t' \preceq t$.

(ii) All sets must contain the empty Σ-tree $\perp = \{\ \}$. In particular, the empty set does not belong to I_L.

Algebra A_L has as many elementary operations as there are symbols in the ranked alphabet Σ, plus the union operation, which corresponds to the min operation of algebra A_R.

The result of applying the operation $\sigma_L(\sigma \in \Sigma_m)$, $m = 0, 1, \ldots$ to the m elements $L_1, L_2, \ldots, L_m \in I_L$ is defined as follows:

$$\sigma_L(L_1, L_2, \ldots, L_m) = \{\sigma(x_1, x_2, \ldots, x_m) \mid x_i \in L_i,\ i = 1, \ldots, m\} \cup \{\perp\}, \quad (4.1)$$

where $\sigma(x_1, x_2, \ldots, x_m)$ is a Σ-tree whose root is labeled with σ and has the Σ-trees $x_1, x_2, \ldots, x_m$ as sons. Notice that the resulting set satisfies restrictions (i) and (ii). Notice also that the union operation commutes with any operation σ_L, namely,

$$\sigma_L(L_1, \ldots, L_k' \cup L_k'', \ldots, L_m) = \sigma_L(L_1, \ldots, L_k', \ldots L_m) \cup \sigma_L(L_1, \ldots, L_k'', \ldots L_m).$$

As a consequence we can denote any derived operation with an expression having the union operations (if any) at the outermost level. Furthermore, it is easy to see that any derived operation can be represented as the union of a unique standard set

of terms. A set of terms is *standard* if no term contains the union operation and if, whenever a term of type $t_1(t_2)$ appears in it, the term $t_1(\perp)$ does not. In fact, the sets $\{t_1(t_2)\}$ and $\{t_1(t_2), t_1(\perp)\}$ would represent the same derived operation in algebra A_L, owing to the presence of $\{\perp\}$ in (4.1). In the following, we will often represent a derived operation in A_L by its standard set of terms.

Algebra A_L can now be given an ordering relation, that is, simply set inclusion among sets of Σ-trees. Thus the set $\{\perp\}$ is the bottom element of the partial order, and since every chain has a least upper bound (just the set union of all sets of Σ-trees in the chain), then the carrier I_L is a complete partial order. Furthermore, it is easy to see that all elementary operations defined above are monotone, continuous functions. Therefore algebra A_L is a continuous algebra.

As an example, let us consider the following system of symbolic functional equations:

$$\begin{aligned} x &= a_L \cup b_L(x) \cup c_L(x, y), \\ y &= b'_L(y) \cup c'_L(v, z), \\ z &= b_L(u), \\ u &= \{\perp\}, \\ v &= \{\perp\}. \end{aligned} \tag{4.2}$$

As a consequence of the fixpoint theorem, such a system can be solved iteratively. The solutions for z, u, and v are clearly $\{b(\perp); \perp\}$, $\{\perp\}$, and $\{\perp\}$, respectively. Let us apply the iterative algorithm to the two first equations. We start with a pair of bottom elements,

$$x_0 = \{\perp\}, \qquad y_0 = \{\perp\},$$

and at the first and second iteration we get

$$\begin{aligned} x_1 &= \{a;\ b(\perp);\ c(\perp, \perp);\ \perp\}, \\ y_1 &= \{b'(\perp);\ c'(\perp, \perp);\ \perp\}, \\ x_2 &= \{a;\ b(a);\ b(\perp);\ b(c(\perp, \perp));\ b(b(\perp));\ c(a, b'(\perp));\ c(a, c'(\perp, \perp)); \\ &\quad c(a, \perp);\ c(b(\perp), b'(\perp));\ c(b(\perp), c'(\perp, \perp));\ c(b(\perp), \perp); \\ &\quad c(c(\perp, \perp), b'(\perp));\ c(c(\perp, \perp), c'(\perp, \perp));\ c(c(\perp, \perp), \perp); \\ &\quad c(\perp, b'(\perp));\ c(\perp, c'(\perp, \perp));\ c(\perp, \perp);\ \perp\}, \\ y_2 &= \{b'(b'(\perp));\ b'(c'(\perp, \perp));\ b'(\perp);\ c'(\perp, b(\perp));\ c'(\perp, \perp);\ \perp\}. \end{aligned}$$

Given a system of symbolic functional equations, it is very easy to put it in correspondence with a labeled AND/OR graph. Before doing this, we give the following definition.

Definition. A derived operation of algebra A_L is called *normal* iff its standard set of terms is either $\{\perp\}$ or a set of terms of the form $f_L(x_1, \ldots, x_n)$, where $f \in \Sigma_n$ and $x_1, \ldots, x_n$ are variables. A system of symbolic functional equations is in *normal form* if all its derived operations are normal.

For instance, system (4.2) is in normal form.

Any system can be easily transformed into an equivalent system in normal form by applying well-known techniques for obtaining regular grammars from equivalent formalisms [6]. For instance, the system

$$x = y \cup f(g(x, y)), \qquad y = c \cup h(x),$$

can be transformed into the system

$$x = c \cup h(x) \cup f(z), \qquad y = c \cup h(x), \qquad z = g(x, y),$$

which is in normal form.

Given a system of symbolic functional equations in normal form, the *corresponding AND/OR graph* has as many nodes as there are variables of the system. Furthermore, every term $t_j = f_L(x_1, \ldots, x_n)$ contained in an equation $x = t_1 \cup \cdots \cup t_j \cup \cdots$ corresponds to an n-connector $(X, X_1, \ldots, X_n)$ labeled with f, where X is the node corresponding to variable x and X_i, $1 \leq i \leq n$, is the node corresponding to x_i. The bottom element $\{\perp\}$ does not correspond to any connector. For instance, the labeled AND/OR graph corresponding to system (4.2) is given in Figure 2.

We can now prove the important result that, given a system of symbolic functional equations in normal form and its corresponding AND/OR graph, the minimal solution of the system coincides with the set of all finite solution trees of the AND/OR graph. (Recall that solution trees are Σ-trees.)

Let us first prove the following lemma.

LEMMA 4.1. *Let S be a system of symbolic functional equations in normal form with a set of variables V and G be the labeled AND/OR graph corresponding to S. Furthermore, let x_i be the set of Σ-trees which is the value of variable x ($x \in V$) at the ith step of the iterative algorithm applied to system S and $T(x)_i$ be the set of all solution trees of depth at most i rooted at the node corresponding to x in the graph G. Then*

$$x_i = T(x)_i, \qquad i = 0, 1, \ldots, \quad x \in V.$$

PROOF. By induction. We have $x_0 = T(x)_0 = \{\perp\}$. Let us now assume that $x_i = T(x)_i$ for every $x \in V$.

Let $t_j = f_L(y_1, \ldots, y_m)$ be the jth term of the equation whose left member is x in the system S. By applying such an operation in step i of the iterative algorithm we get the set

$$s_j = f_L((y_1)_i, \ldots, (y_m)_i) = \{f(z_1, \ldots, z_m) \mid z_k \in (y_k)_i, k = 1, \ldots, m\} \cup \{\perp\}.$$

Let us now consider the node corresponding to x in G, and let s_j' be the set of solution trees of depth $\leq i + 1$ obtained through the connector corresponding to the above term t_j. We have

$$s_j' = \{f(z_1, \ldots, z_m) \mid z_k \in T(y_k)_i, k = 1, \ldots, m\} \cup \{\perp\}.$$

Thus, since $(y_k)_i = T(y_k)_i$, $k = 1, \ldots, m$, by our induction hypothesis, we have $s_j = s_j'$. But

$$x_{i+1} = \bigcup_j s_j \qquad \text{and} \qquad T(x)_{i+1} = \bigcup_j s_j',$$

and thus

$$x_{i+1} = T(x)_{i+1}. \qquad \square$$

For instance, the value x_2 in the iterative solution of system (4.2) given above coincides with the set of all solution trees of depth ≤ 2 rooted at node X in the AND/OR graph of Figure 2.

Now we can prove

THEOREM 4.1. *Let S be a system of symbolic functional equations in normal form with a set of variables V and G be the labeled AND/OR graph corresponding to S. Furthermore, let $\bar{x}$ be the minimal solution of system S for variable x ($x \in V$) and $T(x)$ be the set of all finite solution trees rooted at the node corresponding to x in G. Then we have*

$$\bar{x} = T(x), \qquad x \in V.$$

PROOF. This result follows directly from Lemma 4.1, if we notice that the solution $\bar{x}$ is given by

$$\bar{x} = \bigcup_{i=0}^{\infty} x_i,$$

and that the set of all solution trees is given by

$$T(x) = \bigcup_{i=0}^{\infty} T(x)_i. \qquad \square$$

5. *Functional Equations of Dynamic Programming*

A *system of functional equations of dynamic programming S* is a symbolic system SS, as defined in the previous section, together with an interpretation I, which associates with each elementary operation of A_L a monotone function on $R \cup \{+\infty\} \cup \{-\infty\}$. In particular it maps the union operation in the min operation. Given an interpretation I, we thus obtain an algebra A_R having such functions as elementary operations.

We can now define a homomorphism between A_L and A_R through the function h computed as follows. Let T be a set of Σ-trees belonging to I_L.

(i) For every tree in T, interpret every symbol σ on the nodes as the operation σ_R and evaluate the tree. The empty tree evaluates to $+\infty$. Let us call g this function which given a T returns a set of values belonging to I_R, namely, the set of "costs" of every tree.

(ii) Take the greatest lower bound of all values obtained in (i). Thus $h(T) = \text{glb}(g(T))$.

Notice that for the operation symbols belonging to Σ, A_L acts as the initial algebra and h as the evaluation homomorphism [4]. Thus for such operations conditions (2.2) and (2.3) are clearly satisfied. For the union–min operation (2.3) is also valid:

$$\begin{aligned} h(T_1 \cup T_2) &= \text{glb}(g(T_1 \cup T_2)) = \text{glb}(g(t_1) \cup g(T_2)) \\ &= \min(\text{glb}(g(T_1)), \text{glb}(g(T_2))) \\ &= \min(h(T_1), h(T_2)). \end{aligned}$$

Thus h is a homomorphism.

Given a system $S = (\text{SS}, I)$, its solution is obtained, by definition, as follows. One derives from the symbolic system SS an interpreted system IS using homomorphism h and solves it.

For instance, let us consider again system (4.2):

$$\begin{aligned} x &= a_L \cup b_L(x) \cup c_L(x, y), \\ y &= b'_L(y) \cup c'_L(v, z), \\ z &= b_L(u), \\ u &= \{\perp\}, \\ v &= \{\perp\}, \end{aligned}$$

together with the interpretation

$$\begin{aligned} a_R &= 5, \\ b_R(x) &= \frac{x}{2} + 2, \\ b'_R(x) &= \textbf{if } x \leq 1 \textbf{ then } 1 \textbf{ else } 2 - \frac{1}{x}, \\ c_R(x, y) &= \frac{x}{2} + y, \\ c'_R(x, y) &= x + y. \end{aligned}$$

The interpreted system IS is

$$x = \min\left(5, \frac{x}{2} + 2, \frac{x}{2} + y\right)$$

$$y = \textbf{if } y \leq 1 \textbf{ then } 1 \textbf{ else } 2 - \frac{1}{y},$$

$$z = u = v = +\infty.$$

It is well known in the context of dynamic programming that functional equations are a means for describing the class of optimization problems characterized by the so-called "Bellman's principle of optimality" [2, 22]. According to this principle, the solution of an optimization problem can be reduced to the solution of several subproblems, whose solution, in turn, can be reduced to the solution of other subproblems. In the example of Section 1 the problem of finding a minimum length path from node 1 to node n is reduced to the subproblems of finding the minimum length path from node 1 to every node adjacent to node n, and similarly for all other nodes. Functional equations express the optimal solution of a problem in terms of the optimal solutions of its subproblems.

In practice, the problem–subproblem relations often give rise to a partial ordering among problems. In this case functional equations can be solved efficiently using the technique, known as the "dynamic programming technique," of finding the optimal solution of all subproblems of a given problem before finding the optimal solution of the problem itself. In the general case, when cyclic relations among problems exist, a system of functional equations can be solved iteratively starting from an initial value of $+\infty$ for the variables. Of course, in our framework this is a consequence of the fixpoint theorem, since, as we showed, algebra A_R is continuous. In our example we have $z = u = v = +\infty$, and for the remaining variables we get

$$\begin{array}{ll} x_0 = +\infty, & y_0 = +\infty, \\ x_1 = 5, & y_1 = 2, \\ x_2 = 4.5, & y_2 = 1.5, \\ \vdots & \vdots \\ \bar{x} = 2, & \bar{y} = 1. \end{array}$$

6. *A Commutativity Result*

Let us now apply Theorem 2.1 to algebras A_L and A_R introduced in Sections 3 and 4. In order to do this we have to prove that the homomorphism h introduced in Section 5 is strict and continuous. In fact, (2.4) is satisfied, since

$$h(\perp_L) = \text{glb}(g(\perp_L)) = \text{glb}(\{+\infty\}) = +\infty = \perp_R.$$

Therefore h is strict. Moreover, (2.5) is satisfied, since

$$\begin{aligned} T_1 \sqsubseteq_L T_2 \quad &\text{implies} \quad T_1 \subseteq T_2 \quad \text{implies} \quad g(T_1) \subseteq g(T_2) \\ &\text{implies} \quad \text{glb}(g(T_1)) \geq \text{glb}(g(T_2)) \quad \text{implies} \quad h(T_1) \sqsubseteq_R h(T_2). \end{aligned}$$

Thus h is monotone. Finally, (2.6) is satisfied, since

$$\begin{aligned} h\left(\bigsqcup_{i=0}^{\infty}{}_L\, T_i\right) &= \text{glb}\left(g\left(\bigsqcup_{i=0}^{\infty}{}_L\, T_i\right)\right) = \text{glb}\left(\bigcup_{i=0}^{\infty} g(T_i)\right) \\ &= \underset{i=0}{\overset{\infty}{\text{glb}}}\,(\text{glb}(g(T_i))) = \bigsqcup_{i=0}^{\infty}{}_R\, h(T_i), \qquad T_0 \sqsubseteq T_1 \sqsubseteq \cdots. \end{aligned}$$

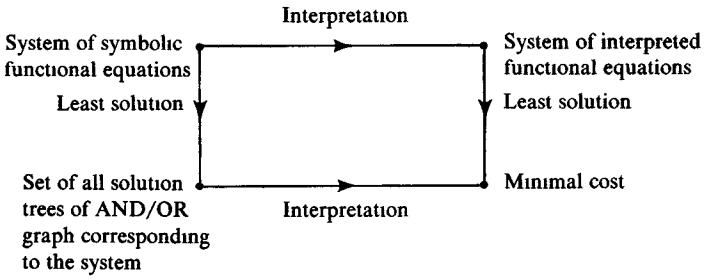


FIGURE 5

Therefore h defines a strict, continuous homomorphism, and Theorem 2.1 can be applied. Thus the solution of a system of functional equations $S = (SS, I)$ can be obtained either by solving in algebra A_R the interpreted system derived from the symbolic system SS using h or by solving in algebra A_L the symbolic system SS and then interpreting the solution.

A system of equations in algebra A_L corresponds to an AND/OR graph, and its solution for the variable x, by Theorem 4.1, is exactly the set of all solution trees rooted at the node corresponding to x. Applying function h to such a set means evaluating the "cost" of every solution tree and taking the cost of the "cheapest" solution tree (or the glb of the costs of all solution trees, if no cheapest solution tree exists). Alternatively, as defined in Section 5, the same value can be obtained by first applying function h to the symbolic system, thus getting a system of interpreted functional equations, and then solving the system for variable x. In short, we can state our main result as the commutativity of the diagram in Figure 5.

Since the diagram commutes not only for the fixpoint but also for all elements in the chains, the value obtained at the ith step of the iterative solution of the interpreted system is the minimum cost of all solution trees of depth at most i in the corresponding AND/OR graph. For instance, for the system of functional equations given in the previous section we have,

$$h(x_1^L) = \text{glb}(g(x_1^L)) = \text{glb}(5, \infty) = 5 = x_1^R,$$
$$h(y_1^L) = \text{glb}(g(y_1^L)) = \text{glb}(2, \infty) = 2 = y_1^R,$$
$$h(x_2^L) = \text{glb}(5, 4.5, \infty) = 4.5 = x_2^R,$$
$$h(y_2^L) = \text{glb}(1.5, 2, \infty) = 1.5 = y_2^R,$$

where x_i^L and y_i^L are the values of x and y at the ith step of the iterative solution of the symbolic system, that is, by Lemma 4.1, the sets of solution trees of depth at most i in the AND/OR graph of Figure 2.

The minimal cost $\bar{x} = 2$ is achieved by the *infinite* tree,

$$c(c(c(\ldots), b'(b'(\ldots))), b'(b'(\ldots))),$$

which does not belong to the solution of system (4.2), since the elements of algebra A_L are only sets of *finite* Σ-trees. In general, however, the glb may be not even achievable by an infinite tree. For instance, given the functional equation

$$x = \min(4, f(x)) \qquad \text{with} \quad f(x) = \tfrac{1}{2}x + 1,$$

the minimal solution $\bar{x} = 2$ is achieved as the limit of the costs $4, f(4), f(f(4)), \ldots,$ whereas the cost of the infinite tree $f(f(f(\ldots)))$ is ∞.

7. *Conclusions*

The commutativity result stated in Section 6 has already been mentioned in the literature, to our knowledge, only for special cases. In [10] Karp and Held proved the result for monadic cost functions under the restriction that a cheapest finite path exists. In [15] Martelli and Montanari eliminated this restriction, but required that all functions be infinite-preserving (namely, $\sigma(+\infty) = +\infty$; notice that $b'_R(x)$ in our example is not infinite preserving). In [14] they extended the result to polyadic cost functions, still in the infinite-preserving case. The general case has been achieved in this paper by a suitable definition of algebra A_L, which guarantees the prefix property and the nonemptyness of its elements.

There are interesting computational applications based on the main result of this and of our previous papers [14, 15, 16], namely, that the solution of a system of functional equations can always be reduced to the problem of searching a minimal cost solution tree in an AND/OR graph. For instance, the iterative algorithm for solving a system of functional equations can be immediately applied to an AND/OR graph for finding a minimal cost solution tree. In fact, the kth step of this algorithm corresponds to finding for every node the minimal cost of all solution trees of depth at most k, knowing the minimal cost of all solution trees of depth at most $k - 1$ of all adjacent nodes. The most interesting results from a computational viewpoint are achieved, however, by applying algorithms for searching AND/OR graphs to the solution of systems of functional equations, since efficient algorithms are known in important special cases, such as, for instance, acyclic graphs [8] or positively monotone cost functions, that is, functions such that

$$f(x_1, \ldots, x_n) \geq x_i, \qquad 1 \leq i \leq n.$$

In the latter case a solution can be found efficiently by applying the Dijkstra–Knuth algorithm [12].

The most important cases in practice are those in which an estimate of the cost of the minimal solution is provided for every node of the graph. In these cases, the so-called heuristic search algorithms can be used, such as the well-known algorithm by Hart, Nilsson, and Raphael [5], which can be applied to graphs with monadic and additive cost functions.

Martelli and Montanari developed heuristic search algorithms for the general case of monadic cost functions [16] and the case of positively monotone polyadic cost functions [13, 17]. In [17] we have an example of an important practical problem, optimal decision table conversion, which has been solved more efficiently using heuristically guided search than using standard dynamic programming techniques.

REFERENCES

1. ARNOLD, A., AND NIVAT, M. Nondeterministic recursive program schemes. In *Fundamentals of Computation Theory 1977*, Lecture Notes in Computer Science 56, M. Karpinski, Ed., Springer Verlag, Berlin, Heidelberg, 1977, pp. 12–21.
2. BELLMAN, R.E. *Dynamic Programming* Princeton University Press, Princeton, N J , 1957
3. BLIKLE, A. Equational languages. *Inf. Control 21* (1972), 134–147
4. GOGUEN, J A., THATCHER, J.W., WAGNER, E.G , AND WRIGHT, J.B. Initial algebra semantics and continuous algebras. *J. ACM 24*, 1 (Jan. 1977), 68–95.
5. HART, P., NILSSON, N., AND RAPHAEL, B A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern. SSC-4*, 2 (July 1968), 100–107.
6. HOPCROFT, J.E., AND ULLMAN, J.D *Formal Languages and Their Relation to Automata.* Addison-Wesley, Reading, Mass , 1969.
7. IBARAKI, T. Classes of discrete optimization problems and their decision problems. *J. Comput. Syst. Sci. 8* (1974), 84–116.

8 IBARAKI, T On the optimality of algorithms for finite state sequential decision processes. *J Math. Anal Appl 53*, 3 (March 1976), 618–643
9 JOHNSON, D B Algorithms for shortest paths Ph D Dissertation, Cornell Univ, Ithaca, N.Y., May 1973
10 KARP, R.M, AND HELD, M. Finite-state processes and dynamic programming *SIAM J. Appl Math. 15* (1967), 693–718
11 KNUTH, D.E Optimum binary search trees *Acta Inf 1* (1971), 14–25
12 KNUTH, D E A generalization of Dijkstra's algorithm *Inf Proc Lett 6* (Feb 1977), 1–4
13 MARTELLI, A., AND MONTANARI, U Additive AND/OR graphs Proc 3rd Int. Joint Conf on Artificial Intelligence, Stanford, Calif, 1973, pp 1–11
14 MARTELLI, A, AND MONTANARI, U Programmazione dinamica e punto fisso Proc Convegno di Informatica Teorica, Mantova, Italy, Nov 1974, pp 1–19
15 MARTELLI, A, AND MONTANARI, U On the foundations of dynamic programming In *Topics in Combinatorial Optimization*, S. Rinaldi, Ed., Springer Verlag, Vienna, New York, 1975, pp 145–163
16 MARTELLI, A, AND MONTANARI, U From dynamic programming to search algorithms with functional costs Proc 4th Int. Joint Conf on Artificial Intelligence, Tbilisi, USSR, Sept. 1975, pp 345–350
17 MARTELLI, A, AND MONTANARI, U Optimizing decision trees through heuristically guided search *Commun ACM 21*, 12 (Dec. 1978), 1025–1039
18 NILSSON, N J *Problem Solving Methods in Artificial Intelligence* McGraw-Hill, N Y., 1971
19 SCHUMACHER, H, AND SEVCIK, K C The synthetic approach to decision table conversion. *Commun ACM 19*, 6 (June 1976), 343–351
20 TARSKI, A A lattice-theoretical fixpoint theorem and its applications *Pacific J. Math 5* (1955), 285–309.
21 THATCHER, J W Tree automata An informal survey In *Currents in the Theory of Computing*, A V Aho, Ed, Prentice Hall, Englewood Cliffs, N.J, 1973, pp 143–172
22 WHITE, D J *Dynamic Programming* Oliver & Boyd, Edinburgh, 1969

RECEIVED JUNE 1979, REVISED FEBRUARY 1980, ACCEPTED JUNE 1980