Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 257 (2009) 3–18
www.elsevier.com/locate/entcs



# Closures in Binary Partial Algebras

Guo-Qiang Zhang[1]

*Department of Electrical Engineering and Computer Science*
*Case Western Reserve University*
*Cleveland, Ohio 44022, USA*

**Abstract**

Two procedures for computing closures in binary partial algebras (BPA) are introduced: a Fibonacci-style procedure for closures in associative BPAs, and a multistage procedure for closures in associative, commutative and idempotent BPAs. Ramifications in areas such as resolution theorem proving, graph-theoretic algorithms, formal languages and formal concept analysis are discussed. In particular, the multistage procedure, when applied to formal concept analysis, results in a new algorithm outperforming leading algorithms for computing concept sets.

*Keywords:* Partial algebra, algorithm, closure, formal concept analysis, resolution principle

## 1 Introduction

In computer science, algebraic structures play an essential role in the foundation and subsequent development of several areas, most notably formal languages (e.g. Kleene Algebra [2,7,15]), domain theory (see, e.g. [1,9,18,23]) and programming languages (see, e.g. [11,12]). In this paper, we study closures for a specific class of algebraic structures called binary partial algebras (BPA) and demonstrate that constructs from a number of areas in computer science, such as resolution theorem proving [19,20], graph-theoretic algorithms [6], formal languages [2,7,15,16] and formal concept analysis [8,24] can be formulated as closures in BPA, with interesting algorithmic consequences.

There is an extensive literature on *partial algebras* (see [3,4,5,10] and the references included therein). This paper differs from the existing body of work

[1] Email: gq@case.edu


doi:10.1016/j.entcs.2009.11.023

in that we focus on the the interactions of *a set of binary partial operators* in the construction of the closures of a subset of a BPA, and on general *algorithmic properties of closure and their applications* in several areas of computer science. More specifically, topics studied in this paper include:

- rank and closure in binary partial algebras;
- BPA-based formulation of resolution principle in logic programming and theorem proving, shortest path in graph theory, Kleene closure and Post Correspondence Problem in formal languages and closure systems in formal concept analysis;
- two procedures for computing closures in BPA, one of the Fibonacci style and the other called *multistage*.

We point out that the notion of *rank* is distinct from the standard notion of index for semigroups [5]. In semigroup theory, the index of an element refers to the least integer that does not produce a power of past values. In BPA, on the other hand, the rank of an element has nothing to do with power: it is the minimal number of elements from a given set that can produce the element using the given set of partial operators. Therefore, the rank of an element in BPA is relative to a chosen subset of a BPA and it represents the notion of "minimal decomposition."

We also note that multistage is a surprisingly intuitive, but non-trivial procedure. As the name suggests, the procedure computes closure in non-cumulative steps, resulting in a partition of the closure in the end. Each stage generates elements whose ranks lie within a specific range. This reduces a potentially large number of redundant operations using a trivial brute-force approach suggested by the definition. When applied to formal concept analysis [8], the multistage approach resulted in a new algorithm which outperformed all leading algorithms for computing formal concept sets [17,21]. The correctness of the multistage approach rests in the assumption of three properties of the underlying BPA: associativity, commutativity and idempotency. Without commutativity and idempotentncy, a Fibonacci-style approach provides a general approach for computing closures in associative BPAs. Neither the Fibonacci-style approach nor the multistage approach is applicable to the resolution procedure, however. This is because the BPA arising from resolution is not associative (see Section 5), a property worth being highlighted as a key reason for its computational complexity.

The rest of the paper is organized as follows. In Section 2 we review the notion of binary partial algebra, and present the notions of rank, closure and the interplay between the two. Elements in the closure are precisely those with finite ranks with respect to the starting set. In finite associative BPAs, the value of finite ranks are bounded by the size of the carrier set. In Sec-

tion 3 we formulate, as samplers, problems in resolution theorem proving, graph-theoretic algorithms, formal languages and formal concept analysis in BPAs. In Section 4 we introduce two basic approaches for computing closures in BPAs. A Fibonacci-style approach is introduced for computing closures in associative BPAs. A multistage approach is introduced for computing closures of associative, commutative and idempotent BPAs. We show that not only can these approaches be translated directly to algorithms, the algorithms can be more efficient than known ones. We then instantiate the multistage algorithm for formal concept analysis. In Section 5 we briefly revisit the topic of resolution using BPA. Concluding remarks are given at the end of the paper.

## 2 Binary Partial Algebras

In universal algebra [10,13], an algebra is defined as a set with a collection of operators. The operators can be of any arity, but they must be total. In binary partial algebra, as the name suggests, the operators can be partial [3,4], in the sense that the values for some arguments can be undefined.

**Definition 2.1** [see [10]] A binary partial algebra is a set $K$ and a set $O$ of binary partial operators of the form $K \times K \to K$. A binary operator is called partial when it may have undefined values.

We recall some basic terminologies for binary partial algebras: associativity, commutativity, idempotency, closure. We also treat the syntactic part of an algebra by employing the associated formal language. These will be used for the development in the rest of the paper.

**Definition 2.2** A binary partial algebra $(K, O)$ is called associative if for any elements $a, b, c$ in $K$,

$$\circ_2(\circ_1(a, b), c) = \circ_1(a, \circ_2(b, c))$$

for any $\circ_1, \circ_2 \in O$, whenever one side of the equality is defined. $(K, O)$ is called idempotent if for each $a \in K$, for each $\circ \in O$, $a \circ a = a$. It is called commutative if $\circ(a, b) = \circ(b, a)$ for all $\circ \in O$ and $a, b \in K$, whenever one of the values is defined.

**Observation**. Clearly, if $O$ consists of a single binary operator, then the properties of associativity, commutativity and idempotency transform from the operator level to the algebra level. Also, weaker notions of associativity and communtativity are possible, by requiring the equality to hold only when both sides are defined.

Subsequently, for clarity and conciseness, we sometimes use finite partial

binary algebras with a single associative operator $\circ$ for illustrative purposes. We will be using the infix notation and assume left-association by default. A BPA is called *finite* if the underlying sets $K$ and $O$ are finite.

**Definition 2.3** [Language and Equivalence] Let $(K, \{\circ\})$ be a finite, associative binary partial algebra with a single operator (written $(K, \circ)$ from now on). The set of strings over $K$ is called the language of the partial algebra and is denoted as $\mathcal{L}(K)$. For $s, t \in \mathcal{L}(K)$, we call $s, t$ equivalent and write $s \equiv t$ if

- $s = x_1 x_2 \cdots x_m$,
- $t = y_1 y_2 \cdots y_n$, and
- $x_1 \circ x_2 \circ \cdots \circ x_m = y_1 \circ y_2 \circ \cdots \circ y_n$, where $x_i$, $y_j$ are elements of $K$ for $i = 1, \ldots m$, $j = 1, \ldots n$. In this case, both sides are defined and equal.

For lack of a better name, for a string $s = x_1 x_2 \cdots x_m$ such as above, we call the result $x_1 \circ x_2 \circ \cdots \circ x_m$, *if defined*, an *internalization* of $s$ in the given BPA. Thus, two strings are equivalent if they have the same internalizations with respect to a BPA. Of course, some strings may not internalize – such is the case when the indicated values are undefined in the corresponding BPA.

In order to avoid confusion in some contexts, we call strings in $\mathcal{L}(K)$ formal strings. Clearly, Definition 2.3 has a straightforward generalization in the multi-operator setting.

**Definition 2.4** [Closure] Let $(K, O)$ be a binary partial algebra. For a subset $X \subseteq K$, the closure of $X$ with respect to $O$ is a set $X^* \subseteq K$ with the following properties:

- $X \subseteq X^*$,
- $\forall a, b \in X^*$, $\forall \circ \in O$, $\circ(a, b) \in X^*$ *whenever the value* $\circ(a, b)$ *is defined*,
- $X^*$ is the smallest set with the above two properties.

**Proposition 2.5** *For any binary partial algebra* $(K, O)$ *and any subset* $X \subseteq K$*, the closure* $X^*$ *always exists and is unique.*

This can be seen from the fact that the intersection of any collection of candidate closures (i.e. those that satisfy the first two items in Def. 2.4) is again a candidate closure.

We next introduce the notion of *rank*, which will be an important device for subsequent developments.

**Definition 2.6** [Rank] Let $(K, O)$ be a finite binary partial algebra. The rank of an element $x \in K$ with respect to a given set $T \subseteq K$, denoted as $\#_T x$, is defined as

$$\#_T x := \min\{\ i \mid x = x_0 \circ_1 x_1 \circ_2 \cdots x_{i-1} \circ_i x_i\ \}$$

where $x_j$ are from $T$ $(0 \leq j \leq i)$ and $\circ_j$ are from $O$ $(1 \leq j \leq i)$.

Without assuming associativity, the meaning of an expression such as $x_0 \circ_1 x_1 \circ_2 x_2 \circ_3 x_3$ is ambiguous. To avoid notational burden, we use such an abstract syntax to denote any one of a number of possibilities, such $(((x_0 \circ_1 x_1) \circ_2 x_2) \circ_3 x_3)$ or $(x_0 \circ_1 x_1) \circ_2 (x_2 \circ_3 x_3)$. But the definition of rank is precise, because the choices of elements and operations are selected among all possibilities.

Intuitively, with respect to a given BPA, the rank of an element $x$ with respect to a set $T$ is the length of the shortest string over $T$ with $x$ a possible "internalization" (note that internalization has only been defined on single-operator BPA, but can easily be generalized to multiple operators). When $x \in T$, we have $\#_T x = 0$; when $x \notin T^*$, we let $\#_T x = \infty$. Even though the shortest strings with $x$ as their internalization may not be unique, the rank $\#_T x$ is always defined and unique.

**Remark**. The notion of *rank* is distinct from the standard notion of index for semigroups [5]. In semigroups, the index of an element $a$ refers to the least integer $q$ that does not produce a power $a^q$ of past values $\{a^i \mid 0 \leq i < q\}$. In BPA, on the other hand, the rank of an element has nothing to do with power; it is the minimal number of elements from a given set that can produce the element using the given set of partial operators.

The following result relates the finiteness of rank to membership in a closure. An element has finite rank with respect to a given set exactly when it belongs to the closure of the set. The proof is straightforward.

**Proposition 2.7** *With respect to a binary partial algebra $(K, O)$, for any $x \in K$ and $T \subseteq K$, we have*

$$\#_T x < \infty \text{ iff } x \in T^*.$$

The next theorem gives an upper bound on finite rank with respect to the size of an associative algebra.

**Theorem 2.8** *With respect to a finite, associative binary partial algebra $(K, O)$, for any $x \in K$ and $T \subseteq K$, if $\#_T x < \infty$ then we have*

$$\#_T x \leq |K| - |T|.$$

**Proof.** Let $\#_T x = n < \infty$. Assume $\#_T x > 0$, i.e. $x \notin T$. Then

$$x = x_0 \circ_1 x_1 \circ_2 x_2 \cdots x_{n-1} \circ_n x_n$$

for some $x_j \in T$ $(0 \leq j \leq n)$ and $\circ_j \in O$ $(1 \leq j \leq n)$. Let

$$R := \{x_0 \circ_1 x_1 \circ_2 x_2 \cdots x_{i-1} \circ_i x_i \mid 0 \leq i \leq n\}.$$

We have $|R| = n + 1$ (note that $x \neq x_0$), since otherwise for some $0 \leq \alpha < \beta \leq n$, we have

$$x_0 \circ_1 x_1 \cdots \circ_\alpha x_\alpha = x_0 \circ_1 x_1 \cdots \circ_\beta x_\beta.$$

Therefore,

$$x = x_0 \circ_1 x_1 \cdots \circ_\alpha x_\alpha \circ_{\beta+1} x_{\beta+1} \cdots x_{n-1} \circ_n x_n,$$

and this implies, By Def. 2.6, $\#_T x < n$ – a contradiction.

By similar reasoning, $T \cap R = \{x_0\}$. Since $T \cup R \subseteq K$, we have $|T| + |R| - 1 \leq |K|$. It follows that $n \leq |K| - |T|$. □

## 3 Examples

In this section we provide several examples in computer science to show that a variety of concepts can be rephrased in the framework of BPA. We demonstrate that the common notions of equivalence and rank have specific conceptual and algorithmic relevance across the subject areas. Note that the purpose here is to demonstrate the modeling capability of BPA and hence our treatments are necessarily brief. It is beyond the scope of the current paper to dive much further into the respective subject areas.

### *3.1 Graph Theory*

We first discuss the most familiar example of graphs (see [6]). Let $K = V \times V$, where $V$ is a finite set of vertices. Define $\circ((a, b), (x, y)) = (a, y)$ if $b = x$, and otherwise undefined. Let $O = \{\circ\}$. Then $(K, O)$ is a BPA and the closure $R^*$ of a relation $R \subseteq V \times V$ is the standard *transitive closure* of $R$.

Two strings $s$ and $t$ are equivalent $(s \equiv t)$ if and only if they represent paths with the same source and same sink with respect to the directed graph determined by $R$.

**Proposition 3.1** *For $(a, b) \in K$ and $R \subseteq K$, the rank $\#_R(a, b)$ is the length of* the shortest path from $a$ to $b$ *with edges from $R$.*

Note that the operator $\circ$ here is associative but not idempotent. Note also that this BPA formulation lifts the edges of a graph to first-class citizens, as they ought to be.

### 3.2 Formal Languages

There are different possibilities in formulating concepts in formal languages under BPA. The simplest one is to take $K$ to be the set of all strings over an alphabet $\Sigma$ and $O$ to consist of the single operator of string concatenation. Then for any $L \subseteq K$, $L^*$ is the *Kleene closure* in formal language theory [16].

Two formal strings $s, t \in \mathcal{L}(K)$ (see Def. 2.3) are equivalent if the concatenation of the respective string sequences give the same result. The rank of a string $x$ with respect to a set $T$ of strings is the shortest number of strings from $T$ whose concatenation gives the result $x$. This is a form of string decomposition. The operator $\circ$ is associative but not idempotent.

A more interesting instantiation is possible. Consider $K = \mathcal{L}(\Sigma) \times \mathcal{L}(\Sigma)$, and for any $(x, y), (u, v) \in K$, define $\circ((x, y), (u, v)) = (xu, yv)$, where $xu$, $yv$ denote the underlying string concatenation. In this case, two elements $(x, y), (u, v)$ are equivalent if they are actually equal as string pairs.

**Proposition 3.2** *The* Post Correspondence Problem *(see [14]) is equivalent to this problem: given $P \subseteq K$, is there a diagonal element $(x, x) \in K$ such that*

$$\#_P(x, x) < \infty?$$

### 3.3 Resolution Theorem-Proving

The resolution principle (see [19,20]) can be formulated as a BPA with multiple operators. Let $K$ be the set of clauses over a finite set $V$ of boolean variables. For each $x \in V$, define $\circ_x(c_1, c_2) = (c_1 \cup c_2) \setminus \{x, \neg x\}$ if $x \in c_1$ and $\neg x \in c_2$ or else $x \in c_2$ and $\neg x \in c_1$. Otherwise $o_x(c_1, c_2)$ is undefined. Note that $\circ_x(c_1, c_2)$ is simply the *resolvant* [19] of $c_1$ and $c_2$ with respect to $x$. The closure of a set of clauses $C \subseteq K$ is the standard *Robinson resolution closure* [20] of $C$.

**Proposition 3.3 (Completeness of Resolution [19])** *For any set of clauses $C \subseteq K$,*

$$\#_C\emptyset < \infty$$

*iff $C$ is unsatisfiable, where $\emptyset$ represents the empty clause.*

Note that in this case $O = \{\circ_x \mid x \in V\}$, and the BPA is commutative but neither associative nor idempotent (see Section 5).

### 3.4 Formal Concept Analysis

We follow the notation of [8] in this subsection. Readers are referred to [8] and [24] for further details. For any set $A$, let $\mathcal{P}(A)$ denote the powerset of $A$. A subset $\mathcal{C}$ of the powerset $\mathcal{P}(A)$ is called a *closure system* if $\mathcal{C}$ is closed under

arbitrary intersections, i.e., for every $X \subseteq \mathcal{C}$, $\bigcap X \in \mathcal{C}$. By convention, the whole space $A$ is always a member of a closure system $\mathcal{C}$. A *closure operator* on $A$ is a self-map $\varphi : \mathcal{P}(A) \to \mathcal{P}(A)$ which is inflationary ($X \subseteq \varphi(X)$), monotonic ($X \subseteq Y \Rightarrow \varphi(X) \subseteq \varphi(Y)$), and idempotent ($\varphi(\varphi(X)) = \varphi(X)$).

**Definition 3.4** Let $\mathbf{K} = (G, M, I)$ be a formal context where $I \subseteq G \times M$. Then its concept lattice $\mathcal{B}\mathbf{K}$ is defined to be the closure system generated by the set $\{\{g\}' \mid g \in G\}$, where $\{g\}' := \{m \mid (g, m) \in I\}$. Dually, $\mathcal{B}\mathbf{K}$ is inverse-isomorphic to the closure system generated by the set $\{\{m\}' \mid m \in M\}$.

We can formulate the notion of closure system under BPA. Let $K$ be the powerset of a finite set $G$ and let $O = \{\cap\}$. Then for each $C \subseteq K$, $C^*$ is the standard *closure system* generated by $C$.

Two formal strings $s, t \in \mathcal{L}(K)$ are equivalent iff $\bigcap s = \bigcap t$, i.e., $s$ and $t$ determines the same (formal) concept. For $x \in K$, $\#_C x$ is the minimal number of elements from $C$ whose intersection is $x$.

## 4 Procedures for Closure

As illustrated in the precious section, closures in BPAs are of particular interest since they are intimately related to algorithmic and decision-theoreic topics. In this section we provide two basic algorithms for computing closures in BPAs that are improvements from the brute-force one. The first algorithm is of Fibonacci-style (to be explained later), and the second, more efficient one is called a multistage algorithm which improves upon the Fibonacci-style algorithm but is applicable only to commutative, associative and idempotent BPAs.

Since the BPA arising from formal concept analysis (see Section 3.4) is commutative, associative and idempotent, the multistage algorithm is applicable for computing concept sets. This results in a new algorithm whose efficiency has outperformed all other leading algorithms for computing concept sets, based on an experimental study [21].

In the rest of the section we fix a finite background BPA $(K, O)$. For notational preparation, define

$$S \odot T := \{s \circ t \mid s \in S, t \in T, \circ \in O\},$$

where $S$ and $T$ are subsets of $K$.

To understand the proposed new procedures, it would be helpful to briefly discuss the brute-force procedure first. To compute the closure $G^*$ for $G \subseteq K$, the brute-force procedure amounts to cumulatively collecting elements obtained from applying an operator to pairs of elements and performing such

an operation repeatedly. Formally, the brute-force procedure can be defined as:

$$\begin{aligned} B_0 &:= G \\ B_i &:= B_{i-1} \cup (B_{i-1} \odot B_{i-1}) \text{ for } i \geq 1. \end{aligned}$$

**Proposition 4.1** *With respect to a finite binary partial algebra $(K, O)$ and $G \subseteq K$, we have*

$$G^* = \bigcup_{i \geq 0} B_i.$$

This can be shown using induction on the syntactic size of expressions for elements in $G^*$, with the stronger conclusion that for all $n \geq 0$, all expressions of the form $x_0 \circ_1 x_1 \circ_2 x_2 \cdots x_{j-1} \circ_j x_j$ belongs to $B_n$, with $0 \leq j \leq n$.

### *4.1 Fibonacci-Style Procedure*

The Fibonacci-style approach computes the closure by an iterative procedure that uses the results obtained from the two immediately previous rounds. This is captured more formally as follows.

With respect to a given subset $G \subseteq K$, define

$$\begin{aligned} L_0 &:= G \\ L_1 &:= G \\ L_i &:= (L_{i-1} \odot L_{i-1}) \cup (L_{i-1} \odot L_{i-2}) \text{ for } i > 1. \end{aligned}$$

Let

$$L := \bigcup_{|K| > i \geq 0} L_i.$$

Figure 1 illustrates the idea. For example, when computing $L_2$, $L_0$ and $L_1$ are used and operations are performed both within two elements of $L_1$ and between elements of $L_1$ and $L_0$. Similarly, $L_3$ involves intra-operations within $L_2$ and inter-operations between $L_2$ and $L_1$.

**Theorem 4.2 (Correctness of Fibonacci Procedure)** *With respect to an associative, finite, binary partial algebra $(K, O)$ and any $G \subseteq K$, we have $L = G^*$, i.e.,*

$$G^* = \bigcup_{|K| > i \geq 0} L_i.$$

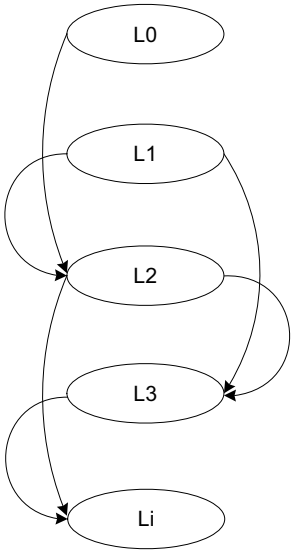


Fig. 1. Fibonacci-style procedure for computing $G^*$.

**Proof.** From the definition of closure (Def. 2.4) it is clear that $G^*$ contains $L$.

We show the containment $G^* \subseteq L$ by induction on the ranks of elements in $G^*$, since all elements in $G^*$ have finite rank (see Prop. 2.7). Clearly, every element in $G^*$ with rank 0 is in $L$ because such an element belongs to $G = L_1$. Every element in $G^*$ with rank 1 is in $L$ because such an element belongs to $L_2$.

Now suppose all elements in $G^*$ with rank less than $k$ are in $L$. Let $x$ be an element such that $\#_G x = k$. Then $x = x_0 \circ_1 x_1 \circ_2 \cdots x_{k-1} \circ_k x_k$ for some $x_0, x_1, \cdots x_k \in G$ and $\circ_1, \circ_2, \cdots \circ_k \in O$. If $k$ is even, then by associativity we have

$$x = (x_0 \circ_1 x_1 \circ_2 \cdots \circ_{k/2} x_{k/2}) \circ_{(k/2)+1} (x_{(k/2)+1} \circ_{(k/2)+2} \cdots \circ x_k),$$

with

$$x_0 \circ_1 x_1 \circ_2 \cdots \circ_{k/2} x_{k/2} \in L_{(k/2)+1}$$

and

$$x_{(k/2)+1} \circ_{(k/2)+2} \cdots \circ x_k \in L_{k/2}.$$

Therefore $x \in L_{(k/2)+1} \odot L_{k/2} \subseteq L$. Similarly, if $k$ is odd then $x \in L_{(k+1)/2)} \odot L_{(k+1)/2} \subseteq L$ and this completes the inductive step. Note that by Theorem 2.8, we have $\#_G x = k < |K|$.

Therefore by induction we have $G^* \subseteq L$. □

Two remarks are in order.

*Remark 1.* The above proof using induction goes through without using rank; it can be based on syntactic form, i.e., the length of an expression. Also, induction based on rank will be needed for the next corollary, the idea of which is explained in the second remark.

*Remark 2.* There may be redundancies in the unions involved in constructing $L$. Typically, the sets $L_i$ and $L_{i+1}$ may have overlapping elements, and this is one source of redundancy for constructing $L_{i+2}$. Another redundancy is that the end result $L$ is the union of $L_i$ for $|K| > i \geq 0$, and as long as an element is found in $L_{i+1}$, we do not need it in $L_i$. This results in the following slightly modified, but potentially more efficient way of computing the closure:

$$\begin{aligned} S_0 &:= G, \\ S_1 &:= G, \text{ and} \\ S_{i+1} &:= ((S_i \odot S_i) \cup (S_i \odot S_{i-1})) - \bigcup_{1 \leq k \leq i} S_k \end{aligned}$$

for $i \geq 1$. The key distinction lies in the removal of all existing elements $\bigcup_{1 \leq k \leq i} S_k$ when forming $S_{i+1}$. This way, only newly generated (and necessary) elements are kept for the next iteration.

**Corollary 4.3** *With respect to an associative, finite, binary partial algebra $(K, O)$ and any $G \subseteq K$, we have*

$$G^* = \bigcup_{0 \leq i \leq 1 + \lceil \log_2 |K| \rceil} S_i,$$

*where $\lceil\ \rceil : \mathcal{R} \to \mathcal{R}$ denotes the ceiling function from reals to reals, that is, for any $x \in \mathcal{R}$, $\lceil x \rceil$ is the least integer not less than $x$.*

We omit the proof for Corollary 4.3 since it amounts to combining those for Theorem 4.2 and Theorem 4.4 in the next section.

### *4.2 Non-overlapping Multistage Procedure*

An even more concise and effective algorithm is possible for computing closures in an associative, commutative and idempotent BPA $(K, O)$. We call this

procedure a multistage one:

$$\begin{aligned} M_0 &:= G, \text{ and} \\ M_{i+1} &:= (M_i \odot M_i) - \bigcup_{1 \le k \le i} M_k \text{ for } i \ge 0. \end{aligned}$$

The key distinction lies in the replacement of the previous union $(S_i \odot S_i) \cup (S_i \odot S_{i-1})$ by a single term $M_i \odot M_i$.

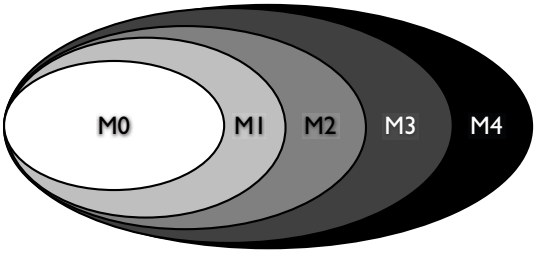


Fig. 2. Multistage procedure; areas of different grades of shade represent non-overlapping elements.

The multistage approach (see Fig. 2) computes the closure $G^*$ in a sequence of stages. Each stage involves the collection of all elements satisfying two criteria: (1) it must be obtained by a single operation of two elements from the immediate previous stage; (2) it must be an element not found in any previous stages. Because of these, distinct stages contain no common elements. The non-cumulative character is a key distinction from the brute-force approach, and the dependance on the *immediate* previous stage, instead of previous two consecutive stages, is a key distinction from the "Fibonacci" idea.

**Theorem 4.4 (Correctness of Multistage Procedure)** *With respect to an idempotent, associative, commutative, finite, binary partial algebra* $(K, O)$ *and any* $G \subseteq K$*, we have*

$$G^* = \bigcup_{0 \le i \le 1 + \lceil \log_2 |K| \rceil} M_i.$$

**Proof.** By Theorem 2.8, the rank of any element in $G^*$ cannot be greater than $|K|$. So it suffices to show by induction that every element in $G^*$ with rank $k$ belongs to $M_t$, where $t = 1 + \lceil \log_2 k \rceil$ (we fix $t = 0$ for $k = 0$ and $t = 1$ for $k = 1$). For conciseness, define $\tau(k) := 1 + \lceil \log_2 k \rceil$. Note that $\tau(k) = 1 + \lceil \log_2(k) \rceil = 2 + \lceil \log_2(k/2) \rceil = 1 + \tau(k/2)$.

To further alleviate notational burden, we denote the rank of $x$ with respect to $G$ simply as $\#x$ without the subscript $G$ in the rest of the proof.

The base cases ($\#x = 0, 1$) follow directly from the definition of $M_0$ and

$M_1$.

For the induction step, assume that any element $x \in G^*$ with rank $\#x \leq k$ belongs to $M_{\tau(\#x)}$. Now let $x \in G^*$ be an element with $\#x = k+1$. Then by the definition of rank, there exist $x_0, x_1, \cdots, x_k, x_{k+1} \in G$ and $\circ_1, \circ_2 \cdots, \circ_{k+1} \in O$ such that

$$x = x_0 \circ x_1 \circ x_2 \cdots x_k \circ x_{k+1}.$$

To improve readability, here we dropped subscripts for $\circ$ with the understanding that they are not necessarily the same operators from $O$. Such an abbreviation will not lead to technical oversimplification. Therefore, the ordering (or positions) of the operators remain the same despite the fact that associativity is freely applied.

Since the rank of each element in $M_i$ is bounded by $2^{i-1}$ for all $i \geq 1$, $x$ does not belong to $\bigcup_{1 \leq i \leq \tau((k+1)/2)} M_i$.

Now if $k$ is even, decompose $x$, by associativity, as

$$x = (x_0 \circ x_1 \circ \cdots \circ x_{k/2}) \circ (x_{1+k/2} \circ \cdots \circ x_{k+1}).$$

Note that the ranks of both $x_0 \circ x_1 \circ \cdots \circ x_{k/2}$ and $x_{1+k/2} \circ \cdots \circ x_{k+1}$ are *equal* to $k/2$ ($\leq k$). By the induction hypothesis, the decomposed elements belong to $M_{\tau(k/2)}$. Thus $x$ belongs to $M_{\tau(k/2)} \odot M_{\tau(k/2)}$. Note that with $k$ even, we have $\tau(k+1) = 1 + \tau(k/2)$. Therefore $x \in M_{\tau(k+1)}$.

If $k$ is odd, decompose $x$, by both idempotency and associativity, as

$$x = (x_0 \circ x_1 \circ \cdots \circ x_{(k+1)/2}) \circ (x_{(k+1)/2} \circ \cdots \circ x_{k+1}).$$

Here, we used idempotency to rewrite $x_{(k+1)/2}$ as $x_{(k+1)/2} \circ x_{(k+1)/2}$. Clearly, the rank of the element $y := x_0 \circ x_1 \circ \cdots \circ x_{(k+1)/2}$ is $(k+1)/2$. More important, the rank of the element $z := x_{(k+1)/2} \circ \cdots \circ x_{k+1}$ is also $(k+1)/2$. Suppose a shorter expression for $z$ exists using elements from $G$. If $x_{(k+1)/2}$ is not part of the expression, then we achieve a shorter expression for $x = y \circ z$ than permitted by the rank of $x$, which is impossible. On the other hand, if $x_{(k+1)/2}$ is part of the shorter expression, we can also achieve a shorter expression for $x$ than permitted by the rank of $x$, *by using commutativity* to move $x_{(k+1)/2}$ to the front of the expression for $z$ and then applying idempotency. Therefore, $\#z = (k+1)/2$.

By induction hypothesis, $y$ and $z$ both belong to $M_{\tau((k+1)/2)}$. Thus $x$ belongs to $M_{\tau((k+1)/2)} \odot M_{\tau((k+1)/2)}$. Therefore $x \in M_{\tau(k+1)}$. This completes the proof. □

As corollaries, the next two observations highlight the key non-redundant and non-overlapping characteristics of the multistage procedure.

**Corollary 4.5** *With respect to an idempotent, associative, commutative, finite, BPA* $(K, O)$ *and* $G \subseteq K$, $\{M_i \mid 0 \leq i \leq \tau(|K|)\}$ *is a partition of* $G^*$.

**Corollary 4.6** *For any* $x \in M_0$, $\#x = 0$ *and for any* $x \in M_1$, $\#x = 1$. *For any* $i \geq 2$ *and any* $x \in M_i$, *we have* $2^{i-1} \leq \#x < 2^i$.

## 5 Resolution Revisited

Since associativity is a property affecting the available methods for computing closures, one might be curious why the BPA induced by the resolution procedure (see Subsection 3.3) is not associative (although it is commutative).

Here is a simple example illustrating non-associativity. With three clauses $\{a, \neg b, \neg c\}$, $\{b\}$, $\{c\}$, we have

$$(\{a, \neg b, \neg c\} \circ_b \{b\}) \circ_c \{c\} = \{a\},$$

but we cannot perform $\{b\} \circ_c \{c\}$ because it is undefined.

There is a simple example to show directly why "multistage" does not work for resolution. Let $G = \{\{a, b\}, \{\neg a\}, \{\neg b\}\}$. Then $M_1 = \{\{a\}, \{b\}\}$, and $M_2$ is empty. However, clearly the empty clause $\{\ \}$ belongs to $G^*$.

In these examples, the non-associativity seems to be caused by the interaction of distinct operators such as $\circ_b$ and $\circ_c$. By removing trivial clauses in which both a literal and its negation appear, one can check that by fixing a single operator, we do get an associative operator. This begs the question of achieving an overall closure by obtaining closures with respect to one operators at a time, making room for possible use of efficient algorithmic methods for individual closures with respect to single operators. This is indeed possible, and is proved, for example, in [22] for resolution.

We state without proof a general result with a stronger condition than desired. This allows for the computation of an overall closure by computing the individual closures *in any sequence*.

**Proposition 5.1** *Let* $(K, O)$ *be a finite, associative binary partial algebra with* $O = \{\circ_i \mid 1 \leq i \leq n\}$. *For* $X \subseteq K$, *let* $(X)_i^*$ *be the closure of* $X$ *with respect to* $(K, \circ_i)$. *We have, for any* $G \subseteq K$,

$$G^* = (\cdots((G)_1^*)_2^* \cdots)_n^*.$$

This result does not yet fully explain why "literal resolution" [22] is complete, and we leave the issue of a generic result in the BPA framework that explicates Literal Resolution as a topic for future investigation.

## 6 Conclusion

We have made a case for using binary partial algebras as a possible unifying framework for studying a group of algorithmic problems from computer science. We have illustrated how the notions of rank and closure interact with each other, and introduced two algorithms for computing closures in binary partial algebras. The multistage procedure is a more efficient method for computing closures, with the algebraic properties of the underlying BPA supporting its algorithmic validity.

Further developments in BPA could take two broad flavors. One is the development of properties for BPA, including those for computing closures, so they can be applied to specific topics such as graph algorithms. Second and in return, algorithms underlying many existing efficient procedures known in special cases such as resolution theorem provers may be seen under BPA in a new light, making it possible to reapply the ideas to a different setting.

## References

[1] S. Abramsky and A. Jung. *Domain Theory*. In S. Abramsky, D. M. Gabbay, T. S. E. Maibaum (editors), Handbook of Logic in Computer Science Vol III. Oxford University Press, 1994.

[2] S. Bloom and Z. Ésik. The equational theory of regular words. Inf. Comput. 197:55-89, 2005.

[3] P. Burmeister. Partial algebras – survey of a unifying approach towards a two-valued model theory for partial algebras. Algebra Universalis, 15:306-358, 1982.

[4] P. Burmeister. *A Model Theoretic Oriented Approach to Partial Algebras (Introduction to Theory and Application of Partial Algebras)*. Mathematical Research Vol. 32, Akademie-Verlag, Berlin, 1986.

[5] A.H. Clifford. *The Algebraic Theory of Semigroups*, Volume II (Mathematical Survey, No 7), American Mathematical Society, 2nd edition, 1967.

[6] T. Cormen, C. Leiserson, R. Rivest, C. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2001.

[7] M. Droste and P. Gastin. The Kleene-Schützenberger theorem for formal power series in partially commuting variables. Information and Computation 153:47-80, 1999.

[8] B. Ganter and R. Wille. *Formal Concept Analysis*. Springer-Verlag, 1999.

[9] G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, D. S. Scott. *Continuous Lattices and Domains*. In Series: Encyclopedia of Mathematics and its Applications (No. 93), Cambridge University Press, 2003.

[10] G. Grätzer: *Universal Algebra*, 2nd Edition, Springer, New York, 1978.

[11] J. Goguen, J. Thatcher, E. Wagner, J. Wright. Initial algebra semantics and continuous algebras. J. ACM, 24:68-95, 1977.

[12] J. Goguen and G. Malcolm. *Algebraic Semantics of Imperative Programs*. MIT Press, 1996.

[13] M. Goldstein, V. Kudryavtsev, I. Rosenberg (eds.). Structural Theory of Automata, Semigroups, and Universal Algebra: Proc. NATO Advanced Study Institute on Structural Theory of Automata, Semigroups and Universal Algebra (NATO Science Series II: Mathematics, Physics and Chemistry, Vol. 207). Springer, 2005.

[14] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[15] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. Infor. and Comput., 110:366-390, 1994.

[16] D. Kozen. *Automata and Computability*. Springer, 1997.

[17] S. Kuznetsov and S. Obiedkov. Comparing performance of algorithms for generating concept lattices. J. Exp. Theor. Artif. Intell. 14:189-216, 2002.

[18] M. Mislove. Nondeterminism and probabilistic choice: obeying the laws. CONCUR 2000: 350-364.

[19] A. Nerode and R. Shore. *Logic for Applications*. Springer, 1997.

[20] Z. Stachniak. *Resolution Proof Systems: An Algebraic Theory*. Springer, 1996.

[21] A. Troy, G.-Q. Zhang, Y. Tian. Faster concept analysis. Proc. 15th International Conference on Conceptual Structures, Lecture Notes in Artificial Intelligence, 4604:206-219, 2007.

[22] G.-Q. Zhang. Literal resolution: A simple proof of resolution completeness. DAIMI-Report, Aarhus University. Computer Science Department, 11 pages, 1989.

[23] G.-Q. Zhang. *Logic of Domains*. Birkhäuser, Boston, 1991.

[24] G.-Q. Zhang. Chu spaces, concept lattices, and domains. 19th Conference on the Mathematical Foundations of Programming Semantics, Montreal, Canada, March 19-22, 2003, Electronic Notes in Theoretical Computer Science, Vol. 83, 17 pages, 2004.