Duality for Three: Ternary Symmetry in Process Spaces

Janusz Brzozowski[1] and Radu Negulescu[2]

[1] School of Computer Science, University of Waterloo,
Waterloo, ON, Canada N2L 3G1
brzozo@uwaterloo.ca, http://maveric.uwaterloo.ca
[2] Department of Electrical and Computer Engineering
McGill University, Montreal, Québec, Canada H3A 2A7
radu@macs.ece.mcgill.ca

Abstract. Ternary algebra has been used for detection of hazards in logic circuits since 1948. Process spaces have been introduced in 1995 as abstract models of concurrent processes. Surprisingly, process spaces turned out to be special ternary algebras. We study symmetry in process spaces; this symmetry is analoguous to duality, but holds among three algebras. An important role is played here by the uncertainty partial order, which has been used since 1972 in algebras dealing with ambiguity. We prove that each process space consists of three isomorphic Boolean algebras and elements related to partitions of a set into three blocks.

1 Introduction

The concept of duality is well known in mathematics. In this paper we study a similar concept, but one that applies to three objects instead of two. The road that led to the discovery of these properties deserves to be briefly mentioned, because several diverse topics come together in this work.

The usual tool for the analysis and design of digital circuits is Boolean algebra, based on two values. As early as 1948, however, it was recognized that three values are useful for describing certain phenomena in logic circuits [10]. We provide more information about the use of ternary algebra for hazard detection in Section 2.

Ternary algebra is closely related to ternary logic [11]. This type of logic, allowing a third, ambiguous value in addition to **true** and **false**, was studied by Mukaidono in 1972 [12], who introduced the *uncertainty* partial order, in addition to the usual lattice partial order. This partial order turned out to be very useful; see, for example, [3, 6]. It also plays an important role in the ternary symmetry we are about to describe.

In 1995 Negulescu [13] introduced process spaces as abstract models of concurrent processes. Surprisingly, process spaces turned out to be special types of ternary algebras. It is in process spaces that "ternary duality" exists. Similar properties also hold in so-called *linear logic*, which has been used as another

J. Karhumäki et al. (Eds.): Theory Is Forever (Salomaa Festschrift), LNCS 3113, pp. 1–14, 2004.

framework for representing concurrent processes, and has connections to Petri nets [17]. This topic is outside the scope of the present paper.

The remainder of the paper is structured as follows. Section 2 illustrates hazard detection using ternary algebra. We also recall some basic concepts from lattice theory and summarize the properties of ternary algebras. Process spaces are defined in Section 3. Ternary symmetry is next discussed in Section 4. In Section 5 we show that each process space contains three isomorphic Boolean algebras. Section 6 characterizes elements of a process space that are outside the Boolean algebras, and Section 7 summarizes our results.

We assume that unary operations have precedence over binary operations. For example, $-x + -y$ denotes $(-x) + (-y)$. Sequences of unary operations are written without parentheses; for example, $-/-x$ denotes $-(/(-x))$. Set inclusion is denoted by $\subseteq$ and proper inclusion, by $\subset$. Proofs that are straightforward and involve only elementary set theory are omitted.

2 Ternary Algebras

The logic values are 0 and 1, and a third value, denoted here by Φ, is used to represent an intermediate or uncertain signal. This idea was used by many authors, but we mention here only Eichelberger's 1965 ternary simulation algorithm [8] and its later characterizations [6]. More information about hazard detection can be found in a recent survey [4]. The following example illustrates the use of ternary simulation to detect hazards in logic circuits.

Example 1. Consider the behavior of the circuit of Fig. 1(a) when its input x changes from 0 to 1. Initially, $x = 0$, $y = 1$, and $z = 0$. After the transition, $x = 1$, $y = 0$, and $z = 0$. Thus, z is not supposed to change during this transition. If the inverter has a sufficiently large delay, however, for a short time both inputs to the AND gate may be 1, and there may be a short 1-pulse in z. Such a pulse is undesirable, because it may cause an error in the computation.

In the first part of the ternary simulation, Algorithm A, we change the input to Φ, which indicates that the input is first going through an intermediate, uncertain value. See Fig. 1(b); the first two entries on each line illustrate Algorithm A. The circuit is then analyzed in ternary algebra to determine which gates will undergo changes; the outputs of the changing gates become Φ. In our example, the inverter output becomes uncertain because its input is uncertain. Also, since one input of the AND gate is 1 and the other uncertain, z becomes Φ.

In the second part, Algorithm B, the input is changed to its final binary value, and the circuit is again simulated in ternary algebra. Some gate outputs that became Φ in Algorithm A will become binary, while others remain Φ. In our example, both y and z become 0; see the last two entries in Fig. 1(b). If a gate output has the same (binary) value in the initial state and also at the end of Algorithm B, then that output is not supposed to change during the transition in question. If, however, that output is Φ after Algorithm A is applied, then we have detected a *hazard*, meaning that an undesired pulse may occur. This happens to the output z. □

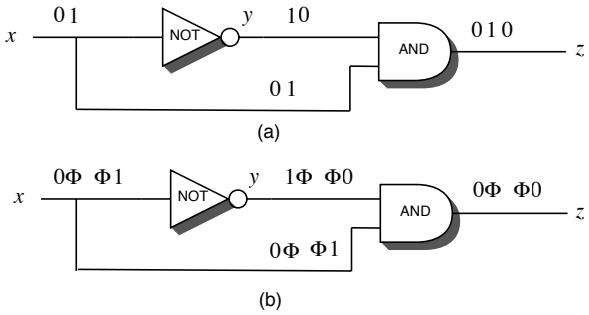

Fig. 1. Circuit with hazard: (a) binary analysis (b) ternary analysis

We now recall some concepts from algebra. For more information about lattices see [1, 7]. We use the following terminology. A *semilattice* [2] is an algebra $(S, \sqcup)$, where S is a set and $\sqcup$ is an idempotent, commutative and associative binary operation on S. We define the partial order $\sqsubseteq_\sqcup$ on S by

$$x \sqsubseteq_\sqcup y \Leftrightarrow x \sqcup y = y.$$

A *bisemilattice* is an algebra $(S, \sqcup, \sqcap)$ in which $(S, \sqcup)$ and $(S, \sqcap)$ are semilattices,

Table 1. Laws of de Morgan Algebras

M1	$x \sqcup x = x$	M1′	$x \sqcap x = x$
M2	$x \sqcup y = y \sqcup x$	M2′	$x \sqcap y = y \sqcap x$
M3	$x \sqcup (y \sqcup z) = (x \sqcup y) \sqcup z$	M3′	$x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z$
M4	$x \sqcup (x \sqcap y) = x$	M4′	$x \sqcap (x \sqcup y) = x$
M5	$x \sqcup \bot = x$	M5′	$x \sqcap \top = x$
M6	$x \sqcup \top = \top$	M6′	$x \sqcap \bot = \bot$
M7	$x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$	M7′	$x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$
M8	$--x = x$		
M9	$-(x \sqcup y) = -x \sqcap -y$	M9′	$-(x \sqcap y) = -x \sqcup -y$

i.e., laws M1–M3, M1′–M3′ of Table 1 hold. A bisemilattice has two partial orders $\sqsubseteq_\sqcup$ and $\sqsubseteq_\sqcap$, the latter defined by

$$x \sqsubseteq_\sqcap y \Leftrightarrow x \sqcap y = x.$$

If a bisemilattice satisfies the absorption laws M4 and M4′, then it is a *lattice*. The two partial orders $\sqsubseteq_\sqcup$ and $\sqsubseteq_\sqcap$ then coincide, and are denoted by $\sqsubseteq$. The converse of $\sqsubseteq$ is denoted by $\sqsupseteq$. The operations $\sqcup$ and $\sqcap$ are the *join* and *meet* of the lattice, respectively. A lattice is *bounded* if it has greatest and least elements

$\top$ (*top*) and $\bot$ (*bottom*) satisfying M5, M6, M5′, M6′. A bounded lattice is represented by $(S, \sqcup, \sqcap, \bot, \top)$. A lattice satisfying the distributive laws M7 and M7′ is *distributive*.

A *de Morgan algebra* is an algebra $(S, \sqcup, \sqcap, -, \bot, \top)$, where $(S, \sqcup, \sqcap, \bot, \top)$ is a bounded distributive lattice, and $-$ is a unary operation, called *quasi-complement*, that satisfies M8 and de Morgan's laws M9 and M9′.

A *Boolean algebra* is a de Morgan algebra $(S, \sqcup, \sqcap, -, \bot, \top)$, which also satisfies the complement laws:

$$x \sqcup -x = \top \qquad\qquad x \sqcap -x = \bot$$

A *ternary algebra* $(S, \sqcup, \sqcap, -, \bot, \Phi, \top)$ is a de Morgan algebra $(S, \sqcup, \sqcap, -, \bot, \top)$ with an additional constant Φ satisfying

T1 $-\Phi = \Phi$
T2 $(x \sqcup -x) \sqcup \Phi = x \sqcup -x$ T2′ $(x \sqcap -x) \sqcap \Phi = x \sqcap -x$

For more information about ternary algebras the reader is referred to [5, 6, 9, 12]. Here, we mention only the uncertainty partial order and the subset-pair representation of ternary algebras.

Figure 2(a) shows the lattice order $\sqsubseteq$ of the 3-element ternary algebra $\mathbf{T_3} = (\{\bot, \Phi, \top\}, \sqcup, \sqcap, -, \bot, \Phi, \top)$, and Fig. 2(b), its *uncertainty* partial order $\preceq$ [6, 12], where Φ represents the unknown or uncertain value, and $\bot$ and $\top$ are the known or certain values.

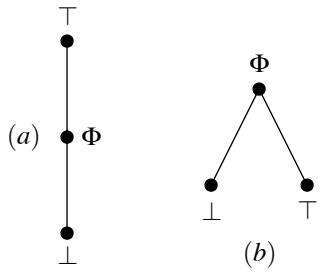

Fig. 2. Partial orders in $\mathbf{T_3}$: (a) $\sqsubseteq$ (b) $\preceq$

For any $x, y \in \mathbf{T_3}$, the least upper bound of $\{x, y\}$ in the partial order $\preceq$ can be expressed as $(x \sqcap y) \sqcup ((x \sqcup y) \sqcap \Phi)$ [6]. We extend this to any ternary algebra $(S, \sqcup, \sqcap, -, \bot, \Phi, \top)$ by defining the binary operation $\vee$ [3] as

$$x \vee y = (x \sqcap y) \sqcup ((x \sqcup y) \sqcap \Phi).$$

It is easily verified that $(S, \vee)$ is a semilattice. The semilattice partial order is

$$x \preceq y \Leftrightarrow x \vee y = y.$$

Let $\mathcal{E}$ be a nonempty set, and $\mathcal{P}$, a collection of ordered pairs (X, X') of subsets of $\mathcal{E}$ such that $X \cup X' = \mathcal{E}$. For $(X, X'), (Y, Y') \in \mathcal{P}$, let

$$(X, X') \sqcup (Y, Y') = (X \cap Y, X' \cup Y'), \tag{1}$$

$$(X, X') \sqcap (Y, Y') = (X \cup Y, X' \cap Y'), \tag{2}$$

$$-(X, X') = (X', X). \tag{3}$$

Let $\bot = (\mathcal{E}, \emptyset)$, $\Phi = (\mathcal{E}, \mathcal{E})$, and $\top = (\emptyset, \mathcal{E})$. Then $(\mathcal{P}, \sqcup, \sqcap, -, \bot, \Phi, \top)$ is a *subset-pair algebra* [5] if $\mathcal{P}$ is closed under $\sqcup$, $\sqcap$, and $-$, and contains the constants $\bot$, Φ, and $\top$. The following result was shown in [5, 9]:

Theorem 1. *Every subset-pair algebra is a ternary algebra, and every ternary algebra is isomorphic to a subset-pair algebra.*

It is easy to verify that

$$(X, X') \sqsubseteq (Y, Y') \Leftrightarrow X \supseteq Y, \text{ and } X' \subseteq Y', \tag{4}$$

$$(X, X') \vee (Y, Y') = (X \cup Y, X' \cup Y'), \tag{5}$$

$$(X, X') \preceq (Y, Y') \Leftrightarrow X \subseteq Y \text{ and } X' \subseteq Y'. \tag{6}$$

3 Process Spaces

The material in this section is based on [14, 15]. The discussion of applications of process spaces is beyond the scope of this paper, and we treat process spaces only as mathematical objects. However, we do give a simple example to motivate the reader.

Let $\mathcal{E}$ be any nonempty set; a *process* x over $\mathcal{E}$ is an ordered pair $x = (X, X')$ of subsets of $\mathcal{E}$ such that $X \cup X' = \mathcal{E}$.

We refer to $\mathcal{E}$ as a set of *executions*. Several different examples of execution sets have been used [14, 15]. For the purposes of this paper, however, we may think of $\mathcal{E}$ as the set of all sequences of *actions* from some action *universe* $\mathcal{U}$; thus $\mathcal{E} = \mathcal{U}^*$. A process $x = (X, X')$ represents a contract between a device and its environment: the device guarantees that only executions from X occur, and the environment guarantees that only executions from X' occur. Thus, $\overline{X} = \mathcal{E} \setminus X$ is the set of executions in which the device violates the contract. Similarly, for executions in $\overline{X'}$, the environment violates the contract. The condition $X \cup X' = \mathcal{E}$, or equivalently $\overline{X} \cap \overline{Y} = \emptyset$, means that the blame for violating the contract can be assigned to either the device or the environment, but not both. The set X is called the set of *accessible* executions of x, and X' is the set of *acceptable* executions.

Example 2. Figure 3 (a) shows a symbol for a buffer, and Fig. 3 (b) shows a sequential machine describing its behavior. The buffer starts in the state marked by an incoming arrow. If it receives a signal on its input a, it moves to a new state. It is expected to respond by producing a signal on its output b and returning to the original state. Thus, the normal operation of the buffer consists of an alternating sequence of a's and b's starting with a. The two states involved in this normal operation are marked g, representing the fact that they are the *goal* states of the process.

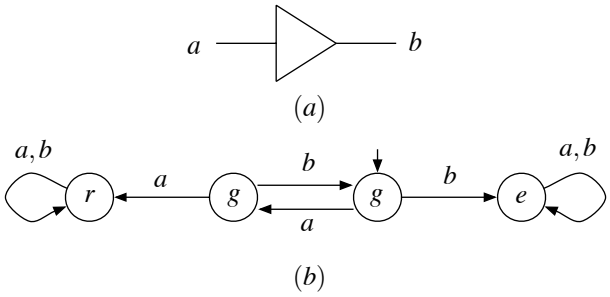

Fig. 3. Buffer process: (a) block diagram (b) behavior

It is possible that the environment of the buffer does not behave according to the specified goal, and produces two consecutive a's in the initial state. From the point of view of the buffer, this environment behavior can be rejected as illegal; hence the state diagram moves to a *reject* state marked r, and remains in that state thereafter. It is also possible that the buffer malfunctions by producing b in the initial state. This is a violation of the contract by the buffer, and the process moves to the state labelled e; such executions have been called the *escapes* of the process.

Let L_g be the set of all words taking the machine of Fig. 3 (b) to a state marked g, and let L_e and L_r be defined similarly. One verifies that $L_g = (ab)^*(\epsilon \cup a)$, where ϵ is the empty word, $L_e = (ab)^*b(a \cup b)^*$, and $L_r = (ab)^*aa(a \cup b)^*$. The buffer process is $(X, X') = (L_g \cup L_e, L_g \cup L_r)$. □

The *process space over* $\mathcal{E}$ is denoted by $\mathcal{P}_\mathcal{E}$, and it is the set of all processes over $\mathcal{E}$. Note that each set $\mathcal{E}$ defines a unique process space.

In constructing $\mathcal{P}_\mathcal{E}$ we must put each element of $\mathcal{E}$ in X or X' or both. Hence, if $\mathcal{E}$ has cardinality n, then $\mathcal{P}_\mathcal{E}$ has cardinality 3^n. The smallest process space has three elements. If $\mathcal{E} = \{1\}$, say, then the three processes are: $(\{1\}, \emptyset)$, $(\{1\}, \{1\})$, and $(\emptyset, \{1\})$.

In every process space we identify three special elements: *bottom*, $\bot = (\mathcal{E}, \emptyset)$, *void*, $\Phi = (\mathcal{E}, \mathcal{E})$, and *top*, $\top = (\emptyset, \mathcal{E})$.

Next, we define the main operations on processes. These operations are motivated by applications to concurrent systems, and are related to operations in several theories of concurrency. For more details see [14, 15].

- *Reflection,* defined as in (3). Reflection permutes the roles of the device and the environment. If a process $x = (X, X')$ is the contract seen by the device, then $-x = (X', X)$ represents the same contract as seen by the environment.
- *Refinement,* defined as in (4). If $x = (X, X')$, $y = (Y, Y')$, and $x \sqsubseteq y$, then y is an acceptable substitute for x, because y accesses fewer executions than x, *i.e.*, its device obeys tighter constraints ($Y \subseteq X$), and accepts more executions than x, *i.e.*, its environment has weaker constraints ($Y' \supseteq X'$).
- *Product,* written $\times$, is a binary operation such that

$$(X, X') \times (Y, Y') = (X \cap Y, (X' \cap Y') \cup \overline{(X \cap Y)}). \tag{7}$$

  Product models a system formed by two devices operating jointly. The system's accessible executions are those that are accessible to both components. Its set of acceptable executions consists of the executions that are acceptable to both components and those that must be avoided by one of the components.

Refinement is a partial order which induces a lattice over a process space, whose join is given by (1) and meet, by (2). Furthermore, this lattice has $\bot$ and $\top$ as bounds. This, together with reflection, which is defined as in (3), makes an arbitrary process space $(\mathcal{P}_\mathcal{E}, \sqcup, \sqcap, -, \bot, \Phi, \top)$ a subset-pair algebra, and, by Theorem 1, a ternary algebra. Reflection is an involution and it reverses refinement. Thus

$$- - x = x, \tag{8}$$

$$x \sqsubseteq y \Leftrightarrow -x \sqsupseteq -y. \tag{9}$$

One verifies that $(\mathcal{P}_\mathcal{E}, \times, \Phi, \top)$ is a semilattice with identity Φ and greatest element $\top$.

An example of a process space is shown in Fig. 4. Here $\mathcal{E} = \{1, 2\}$, and $\mathcal{P}_\mathcal{E} = \mathbf{P_9}$ has nine elements. Its partial orders $\sqsubseteq$ and $\preceq$ are shown in the figure. To simplify the notation, we denote $\{1, 2\}$ simply by 12, *etc.*

4 Ternary Symmetry

Process spaces admit a ternary symmetry [14, 16] based on a unary operation, called *rotation* ($/$), and defined by:

$$/x = (\overline{X} \cup \overline{X'}, X), \tag{10}$$

for all $x = (X, X')$.

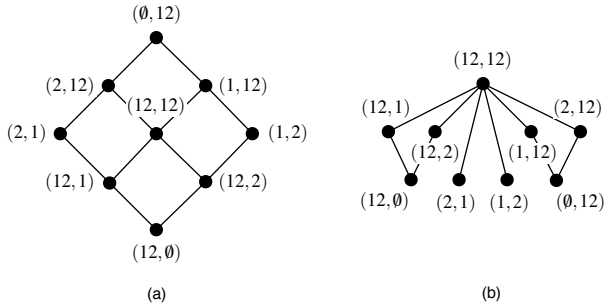

Fig. 4. Partial orders in $\mathbf{P_9}$: (a) $\sqsubseteq$ (b) $\preceq$

Proposition 1. *For any processes x and y we have:*

$$///x = x, \tag{11}$$

$$x \times y = //(/x \sqcap /y), \tag{12}$$

$$-/x = //-x, \quad /-x = -//x, \tag{13}$$

$$/\bot = \Phi,\ /\Phi = \top,\ /\top = \bot. \tag{14}$$

Proposition 1 shows that / is bijective, since it is a root of identity, and therefore admits an inverse, namely //. Furthermore, Prop. 1 reveals that map / is an isomorphism of the semilattices $(\mathcal{P}_\mathcal{E}, \times)$ and $(\mathcal{P}_\mathcal{E}, \sqcap)$.

The ternary symmetry brought out by Prop. 1 justifies an alternate representation of processes in a process space as *(set) triplets* [13, 14], defined below. For process $x = (X, X')$, we also write $x = (X_1, X_2, X_3)$ where $X_1 = X \setminus X'$, $X_2 = X \cap X'$, and $X_3 = X' \setminus X$.

Note that the entries in a set triplet "split" $\mathcal{E}$ in the following sense. If $x = (X_1, X_2, X_3)$, then $X_1 \cap X_2 = X_1 \cap X_3 = X_2 \cap X_3 = \emptyset$ and $X_1 \cup X_2 \cup X_3 = \mathcal{E}$. (This "split" is not a "partition" because some of the blocks might be empty.)

We redefine below the main operations on processes in the set triplet representation. These definitions are equivalent to the definition on set pairs given previously.

$$(X_1, X_2, X_3) \times (Y_1, Y_2, Y_3) = ((X_1 \setminus Y_3) \cup (Y_1 \setminus X_3), X_2 \cap Y_2, X_3 \cup Y_3). \tag{15}$$

$$(X_1, X_2, X_3) \sqsubseteq (Y_1, Y_2, Y_3) \Leftrightarrow X_1 \supseteq Y_1 \wedge X_3 \subseteq Y_3. \tag{16}$$

Note that there is no condition on X_2 and Y_2.

$$-(X_1, X_2, X_3) = (X_3, X_2, X_1). \tag{17}$$

$$\bot = (\mathcal{E}, \emptyset, \emptyset), \quad \Phi = (\emptyset, \mathcal{E}, \emptyset), \quad \top = (\emptyset, \emptyset, \mathcal{E}). \tag{18}$$

Rotation has a simpler form in the set triplet representation:

$$/(X_1, X_2, X_3) = (X_3, X_1, X_2). \tag{19}$$

One also verifies that the operation // of *double rotation*, being the composition of two single rotations, satisfies

$$//x = (X', \overline{X} \cup \overline{X'}) \tag{20}$$

and, equivalently,

$$//(X_1, X_2, X_3) = (X_2, X_3, X_1). \tag{21}$$

Several new operations are defined using ternary symmetry. Each definition relates two operations in a way similar to that between $\times$ and $\sqcap$ in Prop. 1. For completeness, we repeat (12).

Definition 1. *For arbitrary processes x and y from process space $\mathcal{P}_\mathcal{E}$*

$$x \times y = //(/x \sqcap /y),$$
$$x \otimes y = //(/x \times /y),$$
$$x + y = //(/x \sqcup /y),$$
$$x \oplus y = //(/x + /y).$$

The refinement partial order $\sqsubseteq$ is related to the uncertainty partial order, as shown below. This is remarkable because the notions of refinement and uncertainty as formally defined here were motivated by totally different applications.

Proposition 2. *For arbitrary processes x and y from process space $\mathcal{P}_\mathcal{E}$*

$$x \preceq y \Rightarrow /x \sqsubseteq /y.$$

5 Boolean Trios

Let $\mathcal{E}$ be any nonempty set and let $\mathcal{P} = \mathcal{P}_\mathcal{E}$ be the process space on $\mathcal{E}$. Let $(\mathcal{P}, \sqcup, \sqcap, -, \bot, \Phi, \top)$ be the process space viewed as a ternary algebra. Let Q_T be the set of all elements comparable to Φ in $\mathcal{P}$.

Theorem 2. *The structure $(Q_T, \sqcup, \sqcap, -, \bot, \Phi, \top)$ is a sub-ternary-algebra of $(\mathcal{P}, \sqcup, \sqcap, -, \bot, \Phi, \top)$.*

Proof. It is easy to verify that Q_T is closed under the two binary operations, and contains the three constants. Since de Morgan's laws hold, we have $x \sqsubseteq y \Leftrightarrow -x \sqsupseteq -y$. In particular, $x \sqsubseteq \Phi \Leftrightarrow -x \sqsupseteq \Phi$, in view of T1. Hence Q_T is also closed under $-$. □

Let Q_M be the set of all the processes of $\mathcal{P}$ that are minimal in the uncertainty partial order $\preceq$. Since $(X, X') \preceq (Y, Y')$ if and only if $X \subseteq Y$ and $X' \subseteq Y'$, a process $x = (X, X')$ is minimal if and only if $X \cap X' = \emptyset$. Otherwise, if there is a common element in X and X', we can remove it from X (or X'), and obtain a smaller process. Thus, if $x \in Q_M$, then x has the form $x = (X, \overline{X})$, for some $X \subseteq \mathcal{E}$. Note that Q_M includes $\top$ and $\bot$.

Let $(\mathcal{P}, \sqcup, \sqcap, -, \bot, \top)$ be the process space $(\mathcal{P}, \sqcup, \sqcap, -, \bot, \Phi, \top)$ viewed as a de Morgan algebra.

Theorem 3. *The structure* $(Q_M, \sqcup, \sqcap, -, \bot, \top)$ *is a sub-de-Morgan-algebra of* $(\mathcal{P}, \sqcup, \sqcap, -, \bot, \top)$. *Furthermore,* $(Q_M, \sqcup, \sqcap, -, \bot, \top)$ *is a Boolean algebra.*

Proof. Let x, y be in Q_M. Then $x = (X, \overline{X})$ and $y = (Y, \overline{Y})$ for some X and Y. Then $x \sqcap y = (X \cup Y, \overline{X} \cap \overline{Y}) = (X \cup Y, \overline{X \cup Y})$, $x \sqcup y = (X \cap Y, \overline{X} \cup \overline{Y}) = (X \cap Y, \overline{X \cap Y})$, *i.e.*, Q_M is closed under both binary operations. Furthermore, $-x = (\overline{X}, X)$, and Q_M is also closed under $-$. We have already noted that Q_M contains $\bot$ and $\top$.

Laws M1–M9, and M1′–M7′, M9′ hold because they hold in $\mathcal{P}$. Hence we need only to verify the complement laws. We have $x \sqcap -x = (X, \overline{X}) \sqcap (\overline{X}, X) = (X \cup \overline{X}, X \cap \overline{X}) = (\mathcal{E}, \emptyset) = \bot$. Similarly, $x \sqcup -x = \top$. □

We now consider the set of all the processes in Q_T that are below Φ. Let $(\mathcal{P}, \sqcup, \sqcap, -, \bot, \Phi, \top)$ be a process space, and let $Q_L = \{x \in \mathcal{P} \mid x \sqsubseteq \Phi\}$. If $x \in Q_L$, then x has the form $x = (\mathcal{E}, X')$, for some $X' \subseteq \mathcal{E}$. Note that Q_L contains $\bot$ and Φ, and that

$$/x = (\overline{X'}, \mathcal{E}). \tag{22}$$

Secondly, consider the set of all elements above Φ. Let $Q_U = \{x \in \mathcal{P} \mid x \sqsupseteq \Phi\}$. If $x \in Q_U$, then $x = (X, \mathcal{E})$ for some $X \subseteq \mathcal{E}$. Note that Q_L contains Φ and $\top$, and that

$$/x = (\overline{X}, X), \tag{23}$$

for all $x = (\mathcal{E}, X')$ in Q_L.

Finally, recall that elements of Q_M have the form $x = (X, \overline{X})$ and note that

$$/x = (\mathcal{E}, X). \tag{24}$$

If $Q \subseteq \mathcal{E}$, we define $/Q = \{/q \mid q \in Q\}$ and $-Q = \{-q \mid q \in Q\}$.

Proposition 3. $/Q_L = Q_U$, $/Q_U = Q_M$, *and* $/Q_M = Q_L$.

Proof. In view of (22)–(24), we have $/Q_L \subseteq Q_U$, $/Q_U \subseteq Q_M$, and $/Q_M \subseteq Q_L$. Since $///x = x$, it follows that $Q_U = ///Q_U \subseteq //Q_M \subseteq /Q_L$. Thus $/Q_L = Q_U$. The other two equalities follow similarly. □

Proposition 4. *For* $x, y \in Q_L$,

(a)	$x \sqcup -/x = \Phi$,	$x \sqcap -/x = \bot$,
(b)	$/(x \sqcup y) = /x \sqcup /y$,	$/(x \sqcap y) = /x \sqcap /y$,
(c)	$x \sqcup y = x \otimes y = x + y$,	$x \sqcap y = x \times y = x \oplus y$.

Theorem 4. $(Q_L, \sqcup, \sqcap, -/, \bot, \Phi)$ *is a Boolean algebra with join* $\sqcup$, *meet* $\sqcap$, *complement* $-/$, *least element* $\bot$ *and greatest element* Φ.

Proof. We verify that Q_L is closed under $\sqcup, \sqcap$ and $-/$, and contains $\bot$ and Φ. Next we check that the laws of Boolean algebra hold for Q_L. Laws M1–M7, M1′–M7′ hold in Q_L, since they hold in $\mathcal{P}$. The complement laws hold by Prop. 4 (a). The involution and de Morgan's laws follow easily using the fact that each element in Q_L is of the form $x = (\mathcal{E}, X')$. □

Proposition 5. *For $x, y \in Q_U$,*

$$\begin{array}{lll} \text{(a)} & x \sqcup / - x = \top, & x \sqcap / - x = \Phi, \\ \text{(b)} & /(x \sqcup y) = /x \sqcap /y, & /(x \sqcap y) = /x \sqcup /y, \\ \text{(c)} & x \sqcup y = x \times y = x \oplus y, & x \sqcap y = x + y = x \otimes y. \end{array}$$

Theorem 5. *$(Q_U, \sqcup, \sqcap, /-, \Phi, \top)$ is a Boolean algebra with join $\sqcup$, meet $\sqcap$, complement $/-$, least element Φ and greatest element $\top$. Moreover, the algebras $(Q_L, \sqcup, \sqcap, -/, \bot, \Phi)$ and $(Q_U, \sqcup, \sqcap, /-, \Phi, \top)$ are isomorphic, an isomorphism being $/ : Q_L \to Q_U$.*

Proof. By Prop. 1, / is a bijection. By Prop. 4 (b), / preserves $\sqcup$ and $\sqcap$. Also $/(-/x) = /-(/x)$, showing that / maps complements correctly. Finally, $/\bot = \Phi$, and $/\Phi = \top$. □

Proposition 6. *For $x, y \in Q_M$,*

$$\begin{array}{lll} \text{(a)} & x \sqcap -x = \bot, & x \sqcup -x = \top, \\ \text{(b)} & /(x \sqcap y) = /x \sqcup /y, & /(x \sqcup y) = /x \sqcap /y, \\ \text{(c)} & x \sqcap y = x \oplus y = x + y, & x \sqcup y = x \times y = x \otimes y. \end{array}$$

Theorem 6. *$(Q_M, \sqcap, \sqcup, -, \top, \bot)$ is a Boolean algebra with join $\sqcap$, meet $\sqcup$, complement $-$, least element $\top$ and greatest element $\bot$. Moreover, $(Q_U, \sqcup, \sqcap, -, \Phi, \top)$ and $(Q_M, \sqcap, \sqcup, -/, \top, \bot)$ are isomorphic, an isomorphism being $/ : Q_U \to Q_M$.*

Proof. The first claim follows by Theorem 3 and duality in Boolean algebras. Mapping / is a bijection, which behaves like an isomorphism with respect to the binary operations because of Prop. 5 (b). For the unary operation, we have $/(/ - x) = // - x = -(/x)$, as required. Finally, $/\Phi = \top$ and $/\top = \bot$. □

In a similar fashion we verify the following:

Theorem 7. *$(Q_M, \sqcap, \sqcup, -/, \top, \bot)$ and $(Q_L, \sqcup, \sqcap, -/, \bot, \Phi)$ are isomorphic, an isomorphism being $/ : Q_M \to Q_L$.*

We refer to the three Boolean algebras of a process space as its *Boolean trio*. The Hasse diagram for the partial order $\sqsubseteq$ within the Boolean algebras of the 27-element process space $\mathbf{P_{27}}$ is shown in Fig. 5. Note that rotation of the Boolean algebras is counterclockwise, whereas rotation of the complement operations in the three algebras is clockwise, since we have $-$, $/-$, and $//- = -/$.

We close this section by showing that $-$ is an isomorphism between Q_L and the dual of Q_U.

Theorem 8. *$(Q_L, \sqcup, \sqcap, -/, \bot, \Phi)$ and $(Q_U, \sqcap, \sqcup, /-, \top, \Phi)$ are isomorphic, an isomorphism being $- : Q_L \to Q_U$.*

Proof. Since $x \sqsubseteq \Phi \Leftrightarrow -x \sqsupseteq \Phi$, we have $Q_U = -Q_L$. Next, $-(x \sqcup y) = -x \sqcap -y$, and $-(x \sqcap y) = -x \sqcup -y$. Also, $-(-/x) = - - //x = /x = / - -x = / - (-x)$, as required. Finally, $-\bot = \top$, and $-\Phi = \Phi$. □

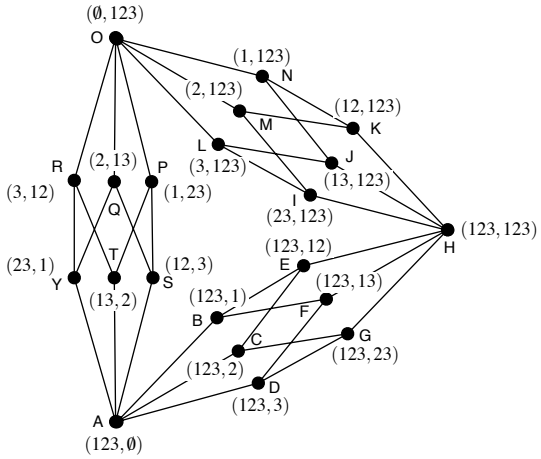

Fig. 5. Boolean trio of $\mathbf{P_{27}}$

By duality of Boolean algebras, algebra $(Q_U, \sqcap, \sqcup, /-, \top, \Phi)$ is isomorphic to $(Q_U, \sqcup, \sqcap, /-, \Phi, \top)$.

Exercise. We now offer the readers an exercise to check their understanding of the concepts presented. To simplify the notation, we introduce the following symbols: $A = (123, \emptyset), B = (123, 1)$, *etc.*, as shown in Fig. 5. The reader who evaluates the following expressions will be richly rewarded (example: $/B = I$):
$/A, \quad //H, \quad -Y, \quad P \sqcup G, \quad // - I;$
$- /G, \quad -/(Q \sqcap R), \quad -Q \sqcup -P, \quad /M, \quad /// - H, \quad -(I \sqcup J), \quad // - H, \quad -/I;$
$/O, \quad Y \sqcup /M, \quad P \oplus R, \quad -/ - A.$ □

6 Ordered Tripartitions

We now consider elements $x = (X, X') = (X_1, X_2, X_3)$ that are outside the Boolean trio. Let T be the set of all such elements; these are elements of $\mathcal{P}$ that are incomparable to Φ and are not minimal in the partial order $\preceq$.

Proposition 7. $T = \{(X, X') \mid \emptyset \subset X, X' \subset \mathcal{E},$ *and* $X \cap X' \neq \emptyset\}$.

We refer to partitions of a set into three blocks as *tripartitions* of the set. An *ordered tripartition* of $\mathcal{E}$ is an ordered triple (X_1, X_2, X_3) of subsets of $\mathcal{E}$, such that $\{X_1, X_2, X_3\}$ is a tripartition of $\mathcal{E}$.

Proposition 8. $T = \{(X_1, X_2, X_3) \mid \{X_1, X_2, X_3\}$ *is a tripartition of* $\mathcal{E}\}$.

A *sextet* is an algebra $(S_6, /, -)$, where S_6 is a set of six elements, and / and $-$ are unary operations satisfying $///x = x$, $--x = x$, and $-/x = //-x$, for all $x \in S_6$. An example of a sextet is shown in Fig. 6(a). Here $S_6 = \{0, 1, \ldots, 5\}$, $/x =$

$x+2 \pmod 6$ and $-x = 5-x$, where the $-$ on the right-hand side is subtraction of integers. Figure 6(b) shows another example of a sextet. Here, S_6 consists of all the ordered tripartitions generated by the tripartition $\{\{1,4\},\{2\},\{3\}\}$ of $\mathcal{E} = \{1,2,3,4\}$ under the two unary operations / and $-$, the rotation and reflection of triplets.

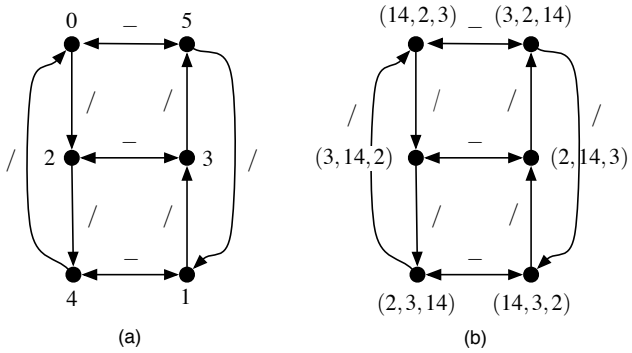

Fig. 6. Illustrating sextets

Proposition 9. *T is a disjoint union of sextets generated by all tripartitions of $\mathcal{E}$.*

If $\mathcal{E}$ has cardinality n, there are 3^n elements in $\mathcal{P}_\mathcal{E}$. In each Boolean algebra of the trio there are 2^n elements, for a total of $3 \times (2^n - 1)$ elements in the trio. Thus there are $3^n - 3 \times (2^n - 1)$ elements in T. For $n = 1$ and $n = 2$, T is empty. For $n = 3$, there are six elements in T. These belong to the sextet generated by the tripartition $\{\{1\},\{2\},\{3\}\}$. For $n = 4$, there are 36 elements belonging to the sextets generated by the six tripartitions $\{\{1,2\},\{3\},\{4\}\}, \{\{1,3\},\{2\},\{4\}\}$, $\{\{1,4\},\{2\},\{3\}\}, \{\{1\},\{2,3\},\{4\}\}, \{\{1\},\{2,4\},\{3\}\}, \{\{1\},\{2\},\{3,4\}\}$.

7 Conclusions

We have demonstrated a ternary symmetry similar to duality. We have shown that every process space consists of a trio of Boolean algebras and a disjoint union of sextets generated by all tripartitions of the underlying set.

Acknowledgment

This research was supported by Grants No. OGP0000871 and RGPIN119122 from the Natural Sciences and Engineering Research Council of Canada.

References

1. Balbes, R., Dwinger, P.: Distributive Lattices, University of Missouri Press (1974)
2. Brzozowski, J. A.: De Morgan Bisemilattices. Proc. 30th Int. Symp. on Multiple-Valued Logic, IEEE Comp. Soc. (2000) 173–178
3. Brzozowski, J. A.: Involuted Semilattices and Uncertainty in Ternary Algebras. Int. J. Algebra and Comput. (2004) to appear
4. Brzozowski, J. A., Ésik, Z., Iland, Y.: Algebras for hazard detection. Beyond Two: Theory and Applications of Multiple-Valued Logic, Fitting, M., Orłowska, E., eds., Physica-Verlag, (2003) 3–24
5. Brzozowski, J. A., Lou, J. J., Negulescu, R.: A Characterization of Finite Ternary Algebras. Int. J. Algebra and Comput. **7** (6) (1997) 713–721
6. Brzozowski, J. A., Seger, C-J. H.: Asynchronous Circuits, Springer-Verlag (1995)
7. Davey, B. A., Priestley, H. A.: Introduction to Lattices and Order, Cambridge University Press (1990)
8. Eichelberger, E. B.: Hazard detection in combinational and sequential circuits. IBM J. Res. and Dev. **9** (1965) 90–99
9. Ésik, Z.: A Cayley Theorem for Ternary Algebras. Int. J. Algebra and Comput. **8** (3) (1998) 311–316
10. Goto, M.: Application of Three-Valued Logic to Construct the Theory of Relay Networks (in Japanese). Proc. IEE, IECE, and I. of Illum. E. of Japan (1948)
11. Kleene, S. C.: Introduction to Metamathematics, North-Holland (1952)
12. Mukaidono, M.: On the B-Ternary Logical Function—A Ternary Logic Considering Ambiguity. Trans. IECE, Japan, **55–D** (6) (1972) 355–362. In English in Systems, Computers, Controls **3** (3) (1972) 27–36
13. Negulescu, R.: Process Spaces. Technical Report CS-95-48, Dept. of Comp. Science, University of Waterloo, ON, Canada (1995)
14. Negulescu, R.: Process Spaces and Formal Verification of Asynchronous Circuits. PhD Thesis, Dept. of Comp. Science, University of Waterloo, ON, Canada (1998)
15. Negulescu, R.: Process Spaces. Proc. 11th Int. Conf. on Concurrency Theory (2000) 196–210
16. Negulescu, R.: Generic Transforms on Incomplete Specifications of Asynchronous Interfaces. Proc. 19th Conf. on Math. Found. of Programming Semantics (2003)
17. Troelstra, A. S.: Lectures on Linear Logic, Center for the Study of Language and Information, Stanford University, Stanford, CA (1992)