Models and logics for process algebra
van der Zwaag, M.B.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
The Logic of ACP

With Alban Ponse

We distinguish two interpretations for the truth value ‘undefined’ in Kleene’s three-valued logic. Combining these two interpretations leads to a four-valued propositional logic that characterizes two particular ingredients of process algebra: “choice” and “inaction”. We study two different bases for this logic, and prove some elementary results (on expressiveness and completeness). One has the classical symmetric connective conjunction and negation, while the other one only has a ternary if-then-else connective with a sequential, operational flavor. Combining this four-valued logic with process algebra yields a direct generalization of ACP with conditional composition that establishes the characterization of choice and inaction. For this generalization we present an operational semantics in SOS-style and some completeness results.

1. Introduction

Process algebra is a generic term that refers to the study of ‘concurrency theory’ (or ‘process theory’) in an algebraic fashion. In this article we attempt to approach process algebra from a logical perspective. This is, of course, not the intended approach; process algebra is algebraically based, and focuses attention on applications (the specification and verification of distributed systems) and on algebraic (mathematical) results. Nevertheless, we think it is worth the effort to consider the primitives of process algebra from a different angle, and to weigh their merits from a logical perspective because this may further illuminate some particular design choices for the primitives and laws of process algebra.

We shall identify ‘process algebra’ with ACP (Algebra of Communicating Processes), the modular process algebra framework designed by Bergstra and Klop from 1982 onwards [15] (for an overview of the current state of the art in process algebra we refer to [23]). The most basic part of ACP is called BPA (Basic Process Algebra) and comprises two binary operations: first, sequential composition, as known from any imperative programming language
(usually written "\)); and second, \textit{alternative composition, or choice}—in principle a descriptive feature that is absent in sequential, imperative programming languages. The motivation for the alternative composition operation arises if concurrency is approached in an analytical, discrete fashion: if $a \parallel b$ expresses the concurrent execution of atomic instantaneous behaviors $a$ and $b$, then an observer experiences either $a$ followed by $b$, or $b$ followed by $a$, or $a$ and $b$ simultaneously. The last case can be thought of as a synchronization or communication between $a$ and $b$.\textsuperscript{1} Such atomic, instantaneous behaviors will henceforth be called actions. This assumption that concurrency can be analyzed or specified in terms of interleaving and synchronization of actions by means of alternative and sequential composition, is sometimes referred to as the \textit{interleaving hypothesis}. A well-known ACP axiom characterizing the interleaving hypothesis is

$$x \parallel y = (x \parallel y + y \parallel x) + x \mid y,$$

where $+$ stands for alternative composition. It states that in the parallel composition $x \parallel y$ of $x$ and $y$, either $x \parallel y$ is executed, or $y \parallel x$, or $x \mid y$. Here $x \parallel y$ is the same as parallel composition with the restriction that the first action stems from $x$, and $x \mid y$ is the same as parallel composition but with the restriction that the first action is a synchronization between a first action of $x$ and one of $y$. We note that these operations together have a simple, algebraic axiomatization in ACP (a historical reference is [15]).

Once sequential and alternative composition are accepted as primitives, it makes sense to analyze these operations in detail. The first one does not raise particular questions, but $+$ does. (choice being further away from the human condition than ordinary sequential composition). Alternative composition becomes even more involved if a notion of deadlock or inaction is included as a primitive behavior, that is, once we admit two types of behavioral stability: (1) termination (short for \textit{successful} termination)—all that should have happened, has happened—and (2) inaction (or \textit{deadlock})—a state where nothing can happen anymore because execution is stuck. Of course, at least one of these kinds of behavioral stability requires explicit notation, and in ACP this is ‘inaction’, written as $\delta$.\textsuperscript{2}

We first explain the difference between inaction and termination in terms of sequential composition, notation $\cdot$, i.e., the multiplication symbol (with the convention to omit this symbol in terms): let $a$, $b$ be actions, then

$$a \delta = (a \delta)b$$

\textsuperscript{1}If $a$ and $b$ are thought of as colored light-flashes, say yellow and blue, this makes sense: either yellow/blue, blue/yellow or a green flash may be observed.

\textsuperscript{2}In CCS [68], only one kind of termination occurs (written 0, or \textit{nil}). This difference is intertwined with the fact that CCS does not have sequential composition, but a less general action prefixing mechanism for sequentiality. See [2] for a discussion.
while, of course, \( a \neq ab \). The idea that after inaction nothing can happen is axiomatized by \( \delta x = \delta \) and by the assumption that sequential composition is associative, an assumption that can hardly be rejected. (Quite naturally, \( x\delta \) cannot be further reduced.) So, \( a \) represents the execution of the action \( a \) after which termination occurs, and \( a\delta \) represents the behavior of \( a \) followed by inaction.

Having accepted the termination convention described above (explicit notation for inaction), one is faced with the question whether

\[ x + \delta \]

can be reduced, and if so, to what. In principle, two reductions seem likely: either \( x \) or \( \delta \). The axiom for the interleaving hypothesis given above yields

\[ a || \delta = (a || \delta + \delta || a) + a | \delta, \]

where the right-hand side equals

\[ (a\delta + \delta) + \delta, \]

since \( a || \delta \) equals \( a\delta \) by definition of the left merge, \( \delta || a \) equals \( \delta \) because the left argument cannot perform an action, and \( a | \delta \) equals \( \delta \) because \( \delta \) cannot participate in a synchronization. Hence, the choice \( x + \delta = x \) leads to \( a || \delta = a\delta \), while the alternative \( x + \delta = \delta \) leads to \( a || \delta = \delta \). Clearly, the latter does not match the interleaving hypothesis, hence the law \( x + \delta = x \) is an axiom of ACP. So, choice is subsidiary to the ability to perform activity in ACP. One may call this, and thus the axiom \( x + \delta = x \), optimistic choice, pessimistic choice being axiomatized by \( x + \delta = \delta \). (The latter option is characterized by the chaos constant \( \chi \) in Hoare's [30], and can be combined with \( \delta \) in a single framework, for instance as the meaningless constant in [18, 17].)

We mentioned earlier that alternative composition is primarily a descriptive feature: it is used to put together possible behaviors, while the nature of the choice between alternatives cannot be accessed. However, this reading does not combine well with the law \( x + \delta = x \), which implies that \( \delta \) is not a fair choice. On the other hand, we have a clear understanding of sequential composition, whether it is read prescriptive or descriptive.

We propose to generalize alternative composition in such a way that it becomes a prescriptive construct: we add information about the choice between alternatives as a side-condition of the composition. Thus, we obtain conditional composition:

\[ x + \phi y \]

stands for the choice between \( x \) and \( y \), under the condition \( \phi \). This construction is well-known from imperative programming languages, where it is usually written in the form \( if \ \phi \ then \ x \ else \ y \).
At this point we may adapt a logical perspective: if

\[ C \]

stands for the logical truth value that represents ‘either true or false’ or ‘overdefined’, and if

\[ D \]

stands for the logical truth value ‘neither true nor false’ or ‘undefined’, then alternative composition and inaction can be viewed as the instances \( +_C \) and \( +_D \) of conditional composition respectively. We find, with \( T \) representing ‘true’ and \( F \) representing ‘false’:

\[
\begin{align*}
  x +_C y &= x + y, \\
  x +_T y &= x, \\
  x +_F y &= y, \\
  x +_D y &= \delta.
\end{align*}
\]

In this article, we introduce a four-valued propositional logic over the truth values \( C, T, F, \) and \( D \), that takes conditional composition as a primitive in the logic, and in which the interplay between conditions can be studied. It turns out that this logic is both straightforward and elegant, and also has a classical basis. Finally, there is a straightforward correspondence with the process algebraic conditional composition, allowing one to explain the nature of choice in process algebra, and its interplay with \( \delta \), from a logical perspective.

This article follows a line of articles on the combination of process algebra and non-standard propositional logics, among which [17, 18, 20, 21]. In [20], the truth value \( C \) was introduced as a second intuition (next to \( D \)) for the third truth value in Kleene’s partial logic. Also, the correspondence between the value \( C \) and process algebraic alternative composition was first recognized in [20]. The generalization of the operations of ACP by parametrization with five-valued conditions was studied in [20, 21]. We discuss this work, and its relation with this article, in Section 6.

The remainder of this article is organized as follows. In Section 2, we introduce the four-valued logic \( \mathbb{L}_4 \) that has a conditional composition connective as primitive operation. We show that this logic is equivalent with the logic that arises naturally when one distinguishes two readings of the truth value for ‘undefined’ in Kleene’s three-valued logic. We present results on expressiveness, and complete axiomatizations. In Section 3, we generalize process algebra in the manner suggested above, starting with the generalization of alternative composition in BPA, a subsystem of ACP. We present an axiom system and prove that it is complete. Furthermore, we establish a correspondence between a class of \( \mathbb{L}_4 \) identities and process algebra identities. Then, in Section 4, we also introduce a generalization of the parallel composition operation of ACP.
We give, as an example of the use of the generalized operations, a specification of a scheduling mechanism for parallel processes. Section 5 is devoted to a full and detailed proof of the completeness of our $L_4$ axioms. This (non-trivial) proof essentially uses a normal form representation for open terms.

2. Four-Valued Propositional Logic

In this section we introduce two propositional logics over the truth values discussed above. First a logic that takes conditional composition as the only operation, and second, one that is based on the classical connectives and can be seen as a natural generalization of Kleene's partial logic. We show that these logics are equal in terms of expressiveness, and provide complete axiomatizations for both.

2.1. A Logic for Conditional Composition. We introduce a four-valued logic with set $T_4 = \{C, T, F, D\}$ of truth values. These truth values can be partially ordered according to the lattice below, which we call the information ordering (see Section 6 for some more comments):

```
C
/\ /
T F  (1)
\ / \\
D
```

The value D can be read as undefined (giving less information than T or F) and C as overdefined or being either T or F. Let $x \sqcup y$ represent the least upper bound of $x$ and $y$ in the information ordering.

The primary operation that we consider is the ternary operation $\langle \rangle$ called conditional composition; it is defined by

\[
\begin{align*}
x &\langle C \rangle y = x \sqcup y, \\
x &\langle T \rangle y = x, \\
x &\langle F \rangle y = y, \\
x &\langle D \rangle y = D.
\end{align*}
\]

So, the auxiliary operation $\sqcup$ stands for $\langle C \rangle$. We prefer to view conditional composition as a primary operation because it corresponds with the process algebraic conditional composition $+_\phi$ (see Section 3) and because it has an operational, sequential flavor, i.e., it can be associated with an order of evaluation: in the evaluation of the term $x \langle y \rangle z$, first $y$ is evaluated, and depending on the outcome, possibly $x$ and/or $z$. Moreover, a logic with a single operation can be technically convenient (cf. the proof of Theorem 2.1).

Assume a set $V$ of variables. Terms are formed using the constants from $T_4$, variables from $V$, and the operations just introduced. A valuation is a mapping
from $V$ to $T_4$. Clearly, every valuation extends to an interpretation mapping from terms to $T_4$. Two terms are equivalent if they have the same interpretation under every valuation. We write $L_4$ for the resulting logic.

Having introduced the logic, we discuss some of its properties. First, conditional composition distributes over $\cup$:

\[
(x_1 \cup x_2) \triangleleft y \triangleright z = (x_1 \triangleleft y \triangleright z) \cup (x_2 \triangleleft y \triangleright z),
\]

\[
x \triangleleft (y_1 \cup y_2) \triangleright z = (x \triangleleft y_1 \triangleright z) \cup (x \triangleleft y_2 \triangleright z),
\]

\[
x \triangleleft y \triangleright (z_1 \cup z_2) = (x \triangleleft y \triangleright z_1) \cup (x \triangleleft y \triangleright z_2).
\]

Furthermore, we can define negation from conditional composition and the truth values $T$ and $F$:

\[
\neg x = F \triangleleft x \triangleright T.
\]

It follows that $\neg T = F, \neg F = T, \neg C = C,$ and $\neg D = D$. Note that the invariance of $C$ and $D$ under negation follows quite naturally from the reading giving above. Finally, negation distributes over $\cup$, and

\[
x \triangleleft y \triangleright z = z \triangleleft \neg y \triangleright x,
\]

\[
\neg(x \triangleleft y \triangleright z) = \neg x \triangleleft y \triangleright \neg z.
\]

We adopt the following binding convention: negation binds more strongly than conditional composition, which binds more strongly than $\cup$.

Next, we look at the expressivity of the logic. We show that, with respect to the information ordering (1), the logic $L_4$ is truth-functionally complete for monotone functions. Recall that an $n$-ary function $f$ over $T_4$ is monotone with respect to a partial ordering $\leq$ on $T_4$, if whenever $a_i \leq b_i$ for $1 \leq i \leq n$, then

\[
f(a_1, \ldots, a_n) \leq f(b_1, \ldots, b_n).
\]

Note that, according to the information ordering lattice, the operation for conditional composition is monotone. This follows from the fact that $x \leq y$ if and only if $x \cup y = y$ and that it distributes over $\cup$. Furthermore, an $n$-ary function $f$ over $T_4$ can be expressed in $L_4$ if there is a term $t$ with variables $x_1, \ldots, x_n$, and no others, such that

\[
f(a_1, \ldots, a_n) = t[a_1/x_1, \ldots, a_n/x_n]
\]

for all $a_1, \ldots, a_n \in T_4$. If every monotone function over the truth values can be expressed in a logic, then that logic is called expressively adequate (this terminology is taken from [27]).

**Theorem 2.1.** The logic $L_4$ is expressively adequate.

**Proof.** Let $f$ be a $(k + 1)$-ary monotone function on $T_4$, and write $\bar{x}, y$ for $(k + 1)$-tuples ($\bar{x}$ may be empty). Then

\[
f(\bar{x}, y) = f(\bar{x}, D) \cup f(\bar{x}, T) \triangleleft y \triangleright f(\bar{x}, F) \cup (D \triangleleft y \triangleright f(\bar{x}, C)) \triangleleft y \triangleright D
\]
by monotonicity of \( f \). By induction on \( k \), the function \( f \) is expressible (because by induction hypothesis \( f(\bar{x}, a) \) is expressible, for all \( a \in T_4 \)).

Non-monotone functions cannot be expressed in \( L_4 \). However, we shall see that the inclusion of a single non-monotone operation results in a logic that is truth-functionally complete (Theorem 2.2).

### 2.2. An Extension of Kleene's Logic

In the previous section, we introduced the four-valued propositional logic \( L_4 \), that has a single operation that may be considered not so standard. In this section we show that this logic can be obtained also by extending Kleene's three-valued logic \([60]\), which we call \( K_3 \), in the following way: we distinguish two interpretations of Kleene's third truth value 'undefined' and show that the resulting logic has exactly the same expressivity as \( L_4 \) (where of course Kleene's logic has the familiar primitive operations negation and conjunction).

First we present the three-valued logic \( \mathbb{K}_3 \) that is also known as partial logic. This logic has, besides the classical truth values true (T) and false (F), a third truth value \( * \), that may be read as either undefined or overdefined (being either true or false, but one cannot predict which of the two). Its basic operations are negation and conjunction defined by the truth tables below.

| \( \neg \) \ |  | \( \wedge \) \ |  | T \ |  | F \ |  | * \ |
|---|---|---|---|---|---|---|---|
| T \ | F \ | T \ | T \ | F \ | * \ |
| F \ | T \ | F \ | F \ | F \ | F \ |
| * \ | * \ | * \ | F \ | * \ | * \ |

Other operations, like disjunction and implication, are defined in terms of these in the familiar way; in particular, disjunction is defined by

\[ x \lor y = \neg (\neg x \land \neg y). \]

Kleene's three-valued logic was designed in order to deal with partial recursive functions: if a partial function \( f \) is not defined for argument \( a \), and the truth value of the term \( t \) depends on \( f(a) \), then \( t \) may be classified as \( * \). However, a term may still make sense, that is, have a definite truth value, even if it has indefinite subterms; for example, \( F \land t \) equals \( F \), even if \( t \) is classified as \( * \).

We shall now extend this three-valued logic by making an explicit distinction between the two possible readings of the third truth value: we replace the value \( * \) by the two distinct truth values \( C \) and \( D \). The resulting logic should preserve the equational theory of \( \mathbb{K}_3 \). Furthermore, it should contain \( \mathbb{K}_3 \) (with \( * \) read as either \( C \) or \( D \)) as a subalgebra. This last assumption leads immediately
to the following (incomplete) truth tables:

<table>
<thead>
<tr>
<th>¬</th>
<th></th>
<th>C</th>
<th>T</th>
<th>F</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>C</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>F</td>
<td>D</td>
</tr>
</tbody>
</table>

In the following we argue that \( C ∧ D = D ∧ C = F \) (and hence that \( C ∨ D = D ∨ C = T \)), and that there are no more than two possible readings of the third truth value \( * \). Observe that absorption \((x = x ∧ (x ∨ y))\) is valid in \( K_3 \), and so are commutativity, associativity and idempotence of conjunction. Now \( C ∧ D ∉ \{C, D\} \) by absorption and the identity \( C ∨ D = \neg(C ∧ D) \). For suppose \( C ∧ D = D \), then

\[
C = C ∧ (C ∨ D) = C ∧ \neg(C ∧ D) = C ∧ \neg D = D.
\]

(In the same way, \( C ∧ D = C \) can be refuted.) By associativity and idempotence of conjunction, \( C ∧ D ≠ T \) (consider \( C ∧ C ∧ D \)). Now assume that \( * \) admits a third interpretation, say \( E \), and \( C ∧ D = E \) (and thus \( C ∨ D = E \)). Then we derive \( E = C \) as follows. First, we have that

\[
C = C ∧ (C ∨ D) = C ∧ E = E ∧ C,
\]
and hence

\[
C = \neg C = \neg(C ∧ E) = \neg C ∨ \neg E = C ∨ E = E ∨ C.
\]

It follows that

\[
E = E ∧ (E ∨ C) = E ∧ C = C.
\]

This shows that \( C ∧ D = F \), and it remains to be shown that with this identity the assumption above, i.e., the existence of a third reading \( E \), is not compatible with \( C \) and \( D \). Suppose the contrary. Then, as above, it follows that \( C ∧ E = D ∧ E = F \). Because distributivity is valid in \( K_3 \), we can derive

\[
C = C ∧ T = C ∧ (D ∨ E) = (C ∧ D) ∨ (C ∧ E) = F ∨ F = F,
\]

which concludes our argument.

Thus, we have extended \( K_3 \) in a natural way to a four-valued logic that we shall refer to as \( K_4 \) (this logic was introduced in [20]). We mention some properties of the operations of \( K_4 \). First, conjunction and disjunction are the greatest lower bound and the least upper bound according to the following ordering:

\[
T

\begin{array}{c}
\text{C} \\
\text{D} \\
\text{F}
\end{array}
\]
Moreover, this lattice, with \( \land \) and \( \lor \), is distributive, and negation is a so-called involution with respect to it (cf. [59]), that is, we have \( \neg
eg\neg x = x \). Below we shall see that this characterization of the logic as a distributive lattice with involution leads directly to a complete axiomatization.

### 2.3. Expressiveness

We show that the logics \( \mathbb{K}_4 \) and \( \mathbb{L}_4 \) have exactly the same expressivity, that is, their operations can be defined in terms of the operations of the other logic. Hence, the two logics can be considered “the same”, but with a different functional basis. So, we can freely use those operations that seem most appropriate. We adopt the following binding convention: negation binds more strongly than conjunction and disjunction, which bind more strongly than conditional composition, which binds more strongly than \( \sqcup \). The operations negation, conjunction and disjunction can all be defined in terms of conditional composition and the truth values \( C, T, \) and \( F \) (recall that \( \sqcup \) abbreviates \( \lhd C \rhd \)):

\[
\neg x = F \lhd x \rhd T, \quad (3)
\]

\[
x \land y = y \lhd x \rhd F \sqcup x \lhd y \rhd F, \quad (4)
\]

\[
x \lor y = T \lhd x \rhd y \sqcup T \lhd y \rhd x. \quad (5)
\]

Vice versa, conditional composition can be defined in terms of negation, conjunction, disjunction and the truth value \( D \):

\[
x \lhd y \rhd z = (((x \land y) \lor (z \land \neg y)) \lor (((x \land z) \land D) \lor ((y \land \neg y) \land D)). \quad (6)
\]

We conclude that the two logics are equally expressive, and in particular that \( \mathbb{K}_4 \) is expressively adequate (see Theorem 2.1). Because all operations of \( \mathbb{L}_4 \) (and thus \( \mathbb{K}_4 \)) are monotone, we cannot express non-monotone functions on the truth values. We show that with the addition of one non-monotone operation, we can express every truth-functional operation. The unary definedness operation \( \downarrow \) (see [13]) is defined by

\[
\downarrow C = F, \quad \downarrow T = T, \quad \downarrow F = T, \quad \downarrow D = F.
\]

This operation is not monotone; for example, we have \( T \leq C \) while \( \downarrow T \nleq \downarrow C \).

**Theorem 2.2.** *With the addition of the definedness operation \( \downarrow \) to \( \mathbb{K}_4 \) or \( \mathbb{L}_4 \), we obtain a logic that is truth-functionally complete.*

**Proof.** It is sufficient to prove this for \( \mathbb{K}_4 \). We introduce auxiliary operations \( \kappa_a(\_): \) that satisfy

\[
\kappa_a(b) = \begin{cases} 
T & \text{if } a = b, \\
F & \text{otherwise}, 
\end{cases}
\]

for \( a, b \in T_4 \):

\[
\kappa_C(x) = \downarrow ((x \land \neg x) \lor D),
\]

Finally, the following axiomatization of \( \mathbb{K}_4 \) is complete.

\[
\begin{align*}
\kappa_C(x) = \kappa_C(y) \rightarrow (x \land \neg x), \\
\kappa_C(x) = \kappa_C(y) \rightarrow (x \lor \neg x), \\
\kappa_C(x) = \kappa_C(y) \rightarrow (x \rightarrow y). \\
\end{align*}
\]
The Logic of ACP

TABLE 1. Axioms of $\mathbb{K}_4$.

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N0)</td>
<td>$\neg(x \land y) = \neg x \lor \neg y$</td>
</tr>
<tr>
<td>(N1)</td>
<td>$\neg\neg x = x$</td>
</tr>
<tr>
<td>(N2)</td>
<td>$\neg T = F$</td>
</tr>
<tr>
<td>(N3)</td>
<td>$\neg C = C$</td>
</tr>
<tr>
<td>(N4)</td>
<td>$\neg D = D$</td>
</tr>
<tr>
<td>(K1)</td>
<td>$x \land y = y \land x$</td>
</tr>
<tr>
<td>(K2)</td>
<td>$x \land (y \land z) = (x \land y) \land z$</td>
</tr>
<tr>
<td>(K3)</td>
<td>$x \land (y \lor z) = (x \land y) \lor (x \land z)$</td>
</tr>
<tr>
<td>(K4)</td>
<td>$x \lor (x \land y) = x$</td>
</tr>
<tr>
<td>(K5)</td>
<td>$T \land x = x$</td>
</tr>
<tr>
<td>(K6)</td>
<td>$C \land D = F$</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\kappa_T(x) &= \downarrow x \land x,
\kappa_F(x) &= \kappa_T(\neg x),
\kappa_D(x) &= \downarrow ((x \land \neg x) \lor C).
\end{align*}
\]

Let $f$ be a $(k+1)$-ary function on $T_4$. Write $\bar{x}$, $y$ for $(k+1)$-tuples. We define

\[f(\bar{x}, y) = \bigvee_{a \in T_4} (\kappa_a(y) \land f(\bar{x}, a)).\]

Hence, the theorem follows by induction on $k$. \qed

2.4. Axioms for the Logics. An axiomatization of $\mathbb{K}_4$ is presented in Table 1. The axioms K1–K4 reflect that (2) is a distributive lattice, and axiom N1 reflects that negation is a so-called involution for this lattice. Axiom N0 is, in the presence of axiom N1, equivalent with the definition of disjunction in terms of negation and conjunction. The proof for the following theorem is due to Bas Luttik and Piet Rodenburg; it is based on [59].

Theorem 2.3. The axioms for $\mathbb{K}_4$ in Table 1 are complete.

Proof. Let the $\mathbb{K}_4$ axioms in Table 1 denote the variety of algebras with conjunction, disjunction, negation, and the four constants $C$, $T$, $F$, and $D$. First, it is easy to see that the initial $\mathbb{K}_4$ algebra is the four element distributive lattice (2) with involution and with the two distinct fixed points of negation $C$ and $D$.

We apply the following theorem from [59]:

Any distributive lattice with involution is isomorphic with a subdirect product of isomorphic images of the four element distributive lattice (2) with involution and with two distinct fixed points of negation.
Table 2. Axioms of $\mathbb{L}_4$.

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>(L1)</td>
<td>$x \triangleright (x' \triangleright y \triangleright z') \triangleright z = (x \triangleright x' \triangleright z) \triangleright y \triangleright (x \triangleright z' \triangleright z)$</td>
</tr>
<tr>
<td>(L2)</td>
<td>$(x \triangleright y \triangleright z) \triangleright y' \triangleright (x' \triangleright y \triangleright z') = (x \triangleright y' \triangleright x') \triangleright y \triangleright (z \triangleright y' \triangleright z')$</td>
</tr>
<tr>
<td>(L3)</td>
<td>$(x \triangleright y \triangleright x') \triangleright y \triangleright z = x \triangleright y \triangleright (x' \triangleright y \triangleright z)$</td>
</tr>
<tr>
<td>(L4)</td>
<td>$T \triangleright x \triangleright F = x$</td>
</tr>
<tr>
<td>(LT)</td>
<td>$x \triangleright T \triangleright y = x$</td>
</tr>
<tr>
<td>(LF)</td>
<td>$x \triangleright F \triangleright y = y$</td>
</tr>
<tr>
<td>(LD)</td>
<td>$x \triangleright D \triangleright y = D$</td>
</tr>
<tr>
<td>(LC1)</td>
<td>$x \triangleright C \triangleright y = y \triangleright C \triangleright x$</td>
</tr>
<tr>
<td>(LC2)</td>
<td>$x \triangleright C \triangleright D = x$</td>
</tr>
<tr>
<td>(LC3)</td>
<td>$C \triangleright C \triangleright x = C$</td>
</tr>
</tbody>
</table>

From this theorem it follows that the $\mathbb{K}_4$ axioms completely axiomatize the initial $\mathbb{K}_4$ algebra $K$. Suppose that $K \models t = u$. Then this identity holds in any subdirect power of $K$, and since any $\mathbb{K}_4$ algebra is isomorphic to such a subdirect power, we may conclude that $\mathbb{K}_4 \models t = u$. Hence $\mathbb{K}_4 \vdash t = u$ follows by Birkhoff's completeness theorem for equational logic [26].

We present an alternative axiomatization for our four-valued logic in Table 2, this time taking conditional composition as primitive operation. This axiomatization is complete as well:

**Theorem 2.4.** The axioms for $\mathbb{L}_4$ in Table 2 are complete.

Using the completeness of $\mathbb{K}_4$, we prove this theorem by exploiting translations in the following way. If the translation of each $\mathbb{K}_4$ axiom is derivable in $\mathbb{L}_4$, then each $\mathbb{K}_4$ derivation can be mimicked in $\mathbb{L}_4$. To complete the proof we argue that the translations are invariant with respect to derivability. We explain this in some more detail: for $t$ a term in the $\mathbb{L}_4$ signature, we write $t'$ for its translation to $\mathbb{K}_4$ (cf. equation (6)), and for $t$ a term in the $\mathbb{K}_4$ signature, we write $t^*$ for its translation to $\mathbb{L}_4$ (cf. (3), (4) and (5)). Now assume $\mathbb{L}_4 \models u = v$. Then, by translation and the completeness of $\mathbb{K}_4$ we have $\mathbb{K}_4 \vdash u' = v'$. So, $\mathbb{L}_4 \vdash (u')^* = (v')^*$. Finally, invariance of our back and forth translation, i.e., $\mathbb{L}_4 \vdash t = (t')^*$, yields $\mathbb{L}_4 \vdash u = v$, as was to be shown. Section 5 is devoted to a detailed (and somewhat long) proof of the completeness of our $\mathbb{L}_4$ axiomatization.

3. Basic Process Algebra

In this section we first introduce a generalization of a simple process algebra. The system $\text{BPA}_5$ (Basic Process Algebra with deadlock) has two
Table 3. The gBPA₃ axioms.

<table>
<thead>
<tr>
<th>(G1)</th>
<th>( x +<em>{\phi &lt; \psi} x \cdot y = (x +</em>{\phi} y) +<em>{\psi} (x +</em>{\chi} y) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(G2)</td>
<td>( (x +<em>{\psi} y) +</em>{\phi} (x' +<em>{\psi} y') = (x +</em>{\phi} x') +<em>{\psi} (y +</em>{\phi} y') )</td>
</tr>
<tr>
<td>(G3)</td>
<td>( x +<em>{\phi} (y +</em>{\phi} z) = (x +<em>{\phi} y) +</em>{\phi} z )</td>
</tr>
<tr>
<td>(G4)</td>
<td>( (x +<em>{\phi} y)z = xz +</em>{\phi} yz )</td>
</tr>
<tr>
<td>(G5)</td>
<td>( (xy)z = x(yz) )</td>
</tr>
<tr>
<td>(GT)</td>
<td>( x +_{T} y = x )</td>
</tr>
<tr>
<td>(GF)</td>
<td>( x +_{F} y = y )</td>
</tr>
<tr>
<td>(GD)</td>
<td>( x +_{D} y = \delta )</td>
</tr>
</tbody>
</table>

Binary operations: alternative composition, or *choice*, and sequential composition. Furthermore, it has a constant \( \delta \) that represents deadlock (or inaction). Both alternative composition and deadlock can be seen as special instances of process algebraic conditional composition. We provide an operational semantics and a complete set of axioms for our generalization of BPA₃ that comprises conditional composition. In Section 4 we also give a generalization of the ACP operations for parallelism.

3.1. The Generalization. We parametrize the alternative composition operation \( (+) \) with \( \mathbb{L}_4 \) terms \( \phi \), hence obtaining the binary operation \( +_{\phi} \) called conditional composition.³ Alternative composition can now be seen as the instance \( +_{c} \) of conditional composition, while \( \delta \) corresponds to \( +_{D} \). Furthermore, we have sequential composition \( (\cdot) \) as usual. We write gBPA₃ for this generalization of BPA₃. For a nonempty finite set \( A \) of action symbols, its terms are generated by the grammar

\[
p ::= a \mid \delta \mid x \mid p +_{\phi} p \mid p \cdot p,
\]

where \( a \) ranges over \( A \), \( x \) ranges over a given set of process variables, and \( \phi \) ranges over the terms of \( \mathbb{L}_4 \). To avoid confusion with process terms, we shall use the letters \( \phi, \psi, \chi \) both for terms and for variables from the logic (recall that in the previous sections we used \( x, y, z \) for proposition variables and \( t, u \) for terms). We may write \( + \) for \( +_{c} \) and we omit the symbol \( \cdot \) from expressions. We let sequential composition bind stronger than conditional composition. The axiom system gBPA₃ consists of the axioms in Table 3. As proof system we use two-sorted equational logic in the following way:

\[
\mathbb{L}_4 \vdash \phi = \psi \quad \text{implies} \quad \text{gBPA}_3 \vdash x +_{\phi} y = x +_{\psi} y,
\]

where \( x, y \) are process variables.

³Recall that the \( \mathbb{L}_4 \) operation \( \triangleleft \) was called conditional composition as well. We now have both a process algebraic and a logical conditional composition. We reserve the notation \( \triangleleft \) for \( \mathbb{L}_4 \).
Next, we give an operational semantics for process-closed process terms, that is, of process terms that do not contain process variables, but that may contain proposition variables. Given the set $A$ of action symbols, we write $P$ for the set of process-closed process terms, and $W$ for the set of valuations for $L_4$ terms (given some set of proposition variables). In Table 4, we give transition rules for the relations

$$\xrightarrow{(\alpha)} P \subseteq (A \times W) \times P,$$

and

$$\xrightarrow{(\alpha)} \sqrt{P} \subseteq P \times (A \times W).$$

The transitions are labelled with an action and a valuation; if

$$p \xrightarrow{a,w} p',$$

then $p$ has the option to execute action $a$ under valuation $w$, and by this execution $p$ evolves into $p'$. The symbol $\sqrt{\cdot}$ is used to indicate successful termination; for example, we have for all $a$ and $w$ that

$$a \xrightarrow{a,w} \sqrt{\cdot}.$$

We proceed with the definition of strong bisimulation equivalence. This definition deviates from the standard definition, because we take valuations into account, so that bisimilar processes have matching action steps for every valuation. A binary relation $R$ on $P$ is a bisimulation if it is symmetric, and whenever $pRq$, then for all $a$ and $w$:

(i) if $p \xrightarrow{a,w} \sqrt{\cdot}$, then $q \xrightarrow{a,w} \sqrt{\cdot};$

(ii) if $p \xrightarrow{a,w} p'$ for some $p'$, then $q \xrightarrow{a,w} q'$ for some $q'$ with $p'Rq'$.

Process-closed process terms $p$ and $q$ are bisimilar, notation $p \equiv q$, if they are related by a bisimulation.

Since bisimilar terms have matching action steps for every possible valuation, we allow the inclusion of (user-defined) propositions in the logic, the
evaluation of which may not be constant throughout the execution of a process. This equivalence may be called \textit{dynamic}, while \textit{static} bisimilarity would be defined as bisimilarity with respect to one, fixed valuation.

The transition rules are in the \textit{panth} format (cf. [82]), from which it follows that bisimilarity is a congruence relation. Furthermore, it is straightforward to verify that the axioms in Table 3 are sound. In the following, we prove that these axioms are complete, that is, that process-closed process terms are bisimilar if and only if they are derivably equal.

Notation. We may write
\[ g\text{BPA}_\delta /\equiv \models t_1(\bar{x}) = t_2(\bar{x}), \]
if \( t_1(\bar{p}) \equiv t_2(\bar{p}) \) for all closed instantiations \( \bar{p} \) of \( \bar{x} \).

\subsection{3.2. Alternative Composition and Guarded Command.} Our claim that alternative composition can be seen as the instance \(+_c\) of conditional composition is supported by showing that the axioms of BPA\(_\delta\) are derivable in gBPA\(_\delta\).

Commutativity of alternative composition (axiom A1) is derived by
\[
\begin{align*}
x +_c y &= (y +_F x) +_c (y +_T x) \quad \text{(by GT, GF)} \\
&= y +_F <c>_T x \quad \text{(by G1)} \\
&= y +_c x \quad \text{(by LC1, L4)}.
\end{align*}
\]

Associativity of alternative composition (axiom A2) is an instance of axiom G3. Idempotency of alternative composition (axiom A3) can be derived by
\[
\begin{align*}
x +_c x &= (x +_T y) +_c (x +_T y) \quad \text{(by GT)} \\
&= x +_T <c>_T y \quad \text{(by G1)} \\
&= x \quad \text{(by (17), GT)}.
\end{align*}
\]

Right-distributivity of sequential composition over alternative composition (axiom A4) is an instance of axiom G4. Associativity of sequential composition (axiom A5) occurs here as axiom G5. The axiom \( x + \delta = x \) (A6) can be derived by
\[
\begin{align*}
x +_c \delta &= (x +_T y) +_c (x +_D y) \quad \text{(by GT, GD)} \\
&= x +_T <c>_D y \quad \text{(by G1)} \\
&= x \quad \text{(by LC2, GT)}.
\end{align*}
\]

Finally, the axiom \( \delta x = \delta \) (A7) can be derived using axioms GD and G4:
\[
\delta x = (y +_D z)x = yx +_D zx = \delta.
\]

Next, we look at the \textit{guarded command} construct [32], defined by
\[
\phi : \rightarrow x = x +_\phi \delta.
\]
3. Basic Process Algebra

It expresses the instruction to execute process \( x \) if the condition \( \phi \) is satisfied. We use this construct in the next section because it allows a more elegant normal form representation than is possible with conditional composition. Here, we shall prove a number of useful identities concerning the guarded command. We use

\[
\delta + \phi \delta = \delta,
\]

(7)

that is derived by

\[
\delta + \phi \delta = (x +_D x) + \phi (x +_D x) = (x +_D x) +_D (x + \phi x) = \delta,
\]

using axioms GD and G2. The following identities can be derived straightforwardly:

\[
x + y \phi = \phi :\rightarrow x + \neg \phi :\rightarrow y,
\]

(8)

\[
\phi :\rightarrow (x + y) = \phi :\rightarrow x + \phi :\rightarrow y,
\]

(9)

\[
(\phi :\rightarrow x)y = \phi :\rightarrow xy,
\]

(10)

\[
x + (\phi :\rightarrow x) = x,
\]

(11)

\[
\phi :\rightarrow (\psi :\rightarrow x) = \psi :\rightarrow (\phi :\rightarrow x).
\]

(12)

For the derivation of (11) we argue as follows:

\[
x + (\phi :\rightarrow x) = (x +_C \delta) + (x + \phi \delta) = x +_C <_C \phi >_C \delta = x + \delta = x,
\]

and for (12) we use (7) and axiom G2:

\[
(x + \psi \delta) + \phi \delta = (x + \psi \delta) + \phi (\delta + \psi \delta) = (x + \phi \delta) + \psi (\delta + \phi \delta).
\]

Clearly, the following identities are derivable as well:

\[
C :\rightarrow x = T :\rightarrow x = x; \quad F :\rightarrow x = D :\rightarrow x = \delta.
\]

(13)

We see that, as a guard, the truth values C and T have the same behavior, and so do F and D. Consequently, the guarded command has nicer distribution properties over the logical operations than conditional composition:

\[
\phi \lor \psi :\rightarrow x = \phi :\rightarrow x + \psi :\rightarrow x,
\]

(14)

\[
\phi \land \psi :\rightarrow x = \phi :\rightarrow (\psi :\rightarrow x),
\]

(15)

\[
\phi < \psi > \chi :\rightarrow x = \psi \land \phi :\rightarrow x + \neg \psi \land \chi :\rightarrow x.
\]

(16)

These identities can all be derived without difficulty; for example, in the case of (14) we replace the disjunction by its definition (5) and derive that the left-hand side equals

\[
\phi :\rightarrow x + \neg \phi :\rightarrow (\psi :\rightarrow x) + \psi :\rightarrow x + \neg \psi :\rightarrow (\phi :\rightarrow x);
\]

and this term can be derived equal to the right-hand side using (11). For (15), we use (4) and find that the left-hand side equals

\[
\phi :\rightarrow (\psi :\rightarrow x) + \psi :\rightarrow (\phi :\rightarrow x),
\]

so that we can finish the proof using (12).
3.3. Completeness. We prove that the axiom system is complete with respect to strong bisimulation equivalence. In the proof it is convenient to write terms in the basic term format that is defined below. We usually work modulo the associativity and commutativity of alternative composition (axioms A1 and A2). Hence, we let $\sum_{i \in I} p_i$, where $I$ is a finite set of indices, stand for the alternative composition of the processes $p_i$ with $i \in I$; furthermore, we define $\sum_{i \in \emptyset} p_i \equiv \delta$.

Let $A$ be the set of action symbols; then basic terms are terms of the form

$$\sum_{i \in I} \phi_i :\rightarrow p_i,$$

where $p_i \in \{a, aq \mid a \in A, \ q \ \text{a basic term}\}$ for all $i \in I$.

**Lemma 3.1.** For all process-closed terms $p$ and basic terms $q$, the sequential composition $pq$ is derivably equal to a basic term.

**Proof.** We apply induction on the structure of $p$. If $p \equiv a \in A$, then $aq$ equals the basic term $T :\rightarrow aq$ by (13). If $p \equiv \delta$, then $pq$ equals the basic term $\delta$ by A7. If $p \equiv p_1 +_\phi p_2$, then derive using (8), G4, and (10) that

$$pq = \phi :\rightarrow p_1q + \neg \phi :\rightarrow p_2q.$$

It follows from the induction hypothesis that there are basic terms

$$p' \equiv \sum_{i} \psi_i :\rightarrow r_i \quad \text{and} \quad p'' \equiv \sum_{j} \psi_j :\rightarrow r_j,$$

with $p' = p_1q$ and $p'' = p_2q$. Using (9) and (15), we derive that $pq$ equals the basic term

$$\sum_{i} \phi \land \psi_i :\rightarrow r_i + \sum_{j} \neg \phi \land \psi_j :\rightarrow r_j.$$

Finally, if $p \equiv p_1p_2$, then we find by axiom G5 that $pq$ equals $p_1(p_2q)$. Now we apply the induction hypothesis twice in succession. \qed

**Lemma 3.2.** Every process-closed process term $p$ is derivably equal to a basic term.

**Proof.** We apply induction on the structure of $p$. If $p \equiv \delta$, then $p$ equals an empty summation by definition. If $p \equiv a \in A$, then $p$ equals the basic term $T :\rightarrow a$ by (13). If $p \equiv p_1 +_\phi p_2$, then by induction hypothesis there are basic terms

$$p'_1 \equiv \sum_{i} \psi_i :\rightarrow p_i \quad \text{and} \quad p'_2 \equiv \sum_{j} \psi_j :\rightarrow p_j,$$

with $p_1 = p'_1$ and $p_2 = p'_2$. By (8), we find that $p$ equals

$$\phi :\rightarrow p'_1 + \neg \phi :\rightarrow p'_2.$$

Using (15) and (9) we get that this term equals the basic term

$$\sum_{i} \phi \land \psi_i :\rightarrow p_i + \sum_{j} \neg \phi \land \psi_j :\rightarrow p_j.$$
Finally, let \( p \equiv p_1 p_2 \). By induction hypothesis, \( p_2 \) is derivably equal to a basic term, so we can finish this case by application of Lemma 3.1. \( \square \)

Next, we define the \textit{height} of basic terms, that shall be used as the basis for the induction in the completeness proof.

\[
\begin{align*}
    h(a) &= 1, \\
    h(\delta) &= 0, \\
    h(\phi : \rightarrow p) &= h(p), \\
    h(p + q) &= \max(h(p), h(q)), \\
    h(ap) &= 1 + h(p).
\end{align*}
\]

\textbf{Lemma 3.3.} Every basic term \( p \) is derivably equal to a basic term

\[
    q \equiv \sum_{i \in I} \phi_i : \rightarrow q_i,
\]

with the following properties:

(i) \( h(q) \leq h(p) \),

(ii) for all distinct \( i, j \in I \) with \( q_i, q_j \in A \), \( q_i \neq q_j \),

(iii) for all \( i \in I, \models \phi_i = \phi_i \land C \),

(iv) for all \( i \in I, \not\models \phi_i = F \).

\textbf{Proof.} Starting from \( p \) written

\[
    p \equiv \sum_i \psi_i : \rightarrow p_i,
\]

we first join summands \( \psi_i : \rightarrow p_i \) and \( \psi_j : \rightarrow p_j \) with \( p_i = p_j = a \in A \) to a single summand \( \psi_i \lor \psi_j : \rightarrow a \) using (14). Observe that this does not change the height of the term, so the first property is preserved. The resulting term satisfies property (ii). Then, we add a conjunct \( C \) to all conditions \( \psi \): we derive using (13) and (15) that

\[
    \psi : \rightarrow p_i = \psi : \rightarrow (C : \rightarrow p_i) = \psi \land C : \rightarrow p_i.
\]

The resulting term satisfies property (iii). Observe that this does not change the height of the term, so the first property is preserved. Also, the second property is preserved. Finally, if the condition of one of the summands in the resulting term is derivably equal to \( F \), then that summand can be omitted. The resulting term satisfies property (iv). Also, the other properties are preserved. \( \square \)

\textbf{Theorem 3.4.} All bisimilar process-closed terms are derivably equal.

\textbf{Proof.} Take bisimilar process-closed terms \( p_1 \) and \( p_2 \), and assume, without loss of generality (Lemma 3.2), that they are basic terms. We apply induction on \( h = h(p_1 + p_2) \). First, observe that if \( h = 0 \), then it must be that \( p_1 \) and
$p_2$ are both syntactically equal to $\delta$. Next, let $h > 0$. By Lemma 3.3 we may assume that for $k = 1, 2$, the term

$$p_k \equiv \sum_{i \in I_k} \phi_{k,i} \rightarrow p_{k,i}$$

satisfies the properties (i)–(iv) of Lemma 3.3. For $k = 1, 2$, we make the following observations.

(a) We may assume that $p_{k,i} \not\equiv p_{k,j}$ for all distinct $i, j \in I_k$. If $p_{k,i}$ and $p_{k,j}$ in $A$, then this follows from property (ii) of Lemma 3.3. Otherwise, let $p_{k,i} \equiv aq$ and $p_{k,j} \equiv ar$ and $q \equiv r$. By induction hypothesis, we find that $\vdash q = r$. Hence, the summands $\phi_{k,i} \rightarrow p_{k,i}$ and $\phi_{k,j} \rightarrow p_{k,j}$ could have been joined to the single summand $\phi_{k,i} \lor \phi_{k,j} \rightarrow p_{k,i}$ using (14). This does not increase the height of $p_k$.

(b) We may assume, using idempotency of $+$, that all summands of $p_k$ are unique.

(c) For every $w \in W$ and $i \in I_k$, we have by property (iii) of Lemma 3.3 that either $w(\phi_{k,i}) = C$ or $w(\phi_{k,i}) = F$.

(d) For all $i \in I_k$, $w(\phi_{k,i}) = C$ for at least one $w \in W$, as follows from property (iv) of Lemma 3.3 and (c).

We show that each summand in $p_k$ is derivably equal to a unique summand in $p_{3-k}$. Take an arbitrary $i \in I_k$.

- First, we consider the case $p_{k,i} \equiv a \in A$. By property (ii) of Lemma 3.3 and (c), we find that

$$p_k \xrightarrow{a,w} \sqrt{\top} \quad \text{if and only if} \quad w(\phi_{k,i}) = C,$$

and, since $p_k \equiv p_{3-k}$, also $p_{3-k} \xrightarrow{a,w} \sqrt{\top} \quad \text{if and only if} \quad w(\phi_{k,i}) = C$.

Using (d), we find that $p_{3-k,j} \equiv a$ for some unique $j \in I_{3-k}$. It follows that $w(\phi_{k,i}) = C$ if and only if $w(\phi_{3-k,j}) = C$, and so by (c), we find $\vdash \phi_{k,i} = \phi_{3-k,j}$ and hence $\vdash \phi_{k,i} = \phi_{3-k,j}$. This finishes the case with $p_{k,i} \in A$.

- Next, suppose that $p_{k,i} \equiv aq$. Using (c), we find that

$$p_k \xrightarrow{a,w} q \quad \text{if and only if} \quad w(\phi_{k,i}) = C.$$

Then it follows from $p_k \equiv p_{3-k}$ that $p_{3-k} \xrightarrow{a,w} r$ for some $r$ with $q \equiv r$ if and only if $w(\phi_{k,i}) = C$. By (d), we find that $p_{3-k,j} \equiv ar$ for some unique (using (a)) $j \in I_{3-k}$. It follows that $w(\phi_{k,i}) = C$ if and only if $w(\phi_{3-k,j}) = C$, and so by (c), we have $\vdash \phi_{k,i} = \phi_{3-k,j}$ and hence $\vdash \phi_{k,i} = \phi_{3-k,j}$. Finally, $\vdash p_{k,i} = p_{3-k,j}$, since $q \equiv r$ implies $\vdash q = r$ by induction hypothesis.

$\square$
3.4. Correspondence. We end this section with some reflections on the correspondence between \( \text{gBPA}_\delta \) and \( \text{L}_4 \). Clearly, process algebraic conditional composition and its logical counterpart are quite similar, as becomes apparent when one compares the axioms G1–G3 with the axioms L1–L3, and GT, GF, and GD with LT, LF, and LD, respectively. This correspondence can be expressed as follows:

**Proposition 3.5.** Let \( t_1(\bar{x}, \bar{v}) = t_2(\bar{x}, \bar{v}) \) be a process identity with process variables \( \bar{x} \) and condition variables \( \bar{v} \) in which the only constants are in \( T_4 \) and the only operation is \( +_\phi \), written as \( \triangleleft \phi \). Then

\[
\text{gBPA}_\delta \models t_1(\bar{x}, \bar{v}) = t_2(\bar{x}, \bar{v})
\]

if and only if

\[
\text{L}_4 \models t_1(\bar{x}, \bar{v}) = t_2(\bar{x}, \bar{v}),
\]

where in the latter statement, \( \bar{x} \) also represents condition variables.

Finally, this result implies that \( \text{L}_4 \) (and thus also \( \text{K}_4 \)) characterizes the axiom \( x + \delta = x \) of \( \text{BPA}_\delta \) (by axiom Lc2), and thus the interplay between choice and deadlock from a logical perspective.

### 4. Parallel Composition

We turn to ACP— that is, the supersystem of \( \text{BPA}_\delta \) that includes operations for parallelism, see, e.g., [15, 11, 35]—and discuss a generalization of the remaining operations as well. Following [20, 21], we extend the operational semantics to this setting, and provide a complete set of axioms. Finally, we use the generalized ACP operations to provide an example on the scheduling of parallel components.

The composition

\[
x \phi \parallel \psi \ y
\]

denotes the parallel execution of \( x \) and \( y \) under conditions \( \phi \) and \( \psi \). Here, the condition \( \phi \) covers the choice between interleaving and synchronization, and \( \psi \) determines the order of interleaving and synchronization. We shall see, for example, that the parallel composition operation \( \parallel \) of ACP equals \( c \parallel c \).

The following parametrized auxiliary operations are used in the axiomatization of the generalized parallel composition.

- **Left merge:** \( x \phi \parallel_\psi y \) denotes \( x \phi \parallel_\psi y \) with the restriction that the first action stems from \( x \).
- **Communication merge:** \( x \phi |_\psi y \) denotes \( x \phi |_\psi y \) with the restriction that the first action is a synchronization of both \( x \) and \( y \).
- **Left communication merge:** \( x \phi \llbracket \psi y \) is used to define \( x \phi \llbracket \psi y \).
The Logic of ACP

**TABLE 5. Additional axioms of gACP(A, |); a, b, c ∈ A, H ⊆ A.**

| (C1) | \( a | b = b | a \) |
| (C2) | \( (a | b) | c = a | (b | c) \) |
| (GD1) | \( \partial_H(a) = a \) if \( a \notin H \) |
| (GD2) | \( \partial_H(a) = \delta \) if \( a \in H \) |
| (GD3) | \( \partial_H(x +_\phi y) = \partial_H(x) +_\phi \partial_H(y) \) |
| (GD4) | \( \partial_H(xy) = \partial_H(x)\partial_H(y) \) |
| (GM1) | \( x_\phi |\psi y = (x_\phi |\psi y +_\psi y_\phi |\psi x) +_\phi x_\phi |\psi y \) |
| (GM2) | \( a_\phi |\psi x = ax \) |
| (GM3) | \( ax_\phi |\psi y = a(x_\phi |\psi y) \) |
| (GM4) | \( (x +_\phi y)_\psi |\psi x z = x_\psi |\psi x z +_\phi y_\psi |\psi x z \) |
| (GM5) | \( x_\phi |\psi y = x_\phi |\psi y +_\psi y_\phi |\psi x \) |
| (GM6) | \( ax_\phi |\psi y = a_\phi |\psi (y_\phi |\psi x) \) |
| (GM7) | \( a_\phi |\psi b = a | b \) |
| (GM8) | \( a_\phi |\psi bx = (a | b)x \) |
| (GM9) | \( a_\phi |\psi (x +_\phi y) = a_\phi |\psi x +_\chi a_\phi |\psi y \) |
| (GM10) | \( (x +_\phi y)_\psi |\psi x z = x_\psi |\psi x z +_\phi y_\psi |\psi x z \) |

**Figure 1.** Example; let \( a | b = c \).

Furthermore, we have encapsulation operators \( \partial_H(x) \) for \( H \subseteq A \), that rename atoms in \( H \) to \( \delta \) and distribute over conditional and sequential composition.

A commutative and associative communication function \( | : A \times A \rightarrow A \cup \{\delta\} \) that defines which actions are allowed to be executed synchronously is given (and extended to process terms). The axioms of this generalization of ACP are those of gBPA_δ together with the axioms listed in Table 5. We adopt the convention that \( +_\phi \) binds less strongly than the operations for parallelism, and \( \cdot \) binds most strongly. The resulting axiom system is denoted by gACP, or by gACP(A, |) if we want to make the parameters of the theory explicit.

Observe that the operation \( \psi |_C \) restricts parallel composition to interleaving only, that is, to the so-called free merge, while \( \psi |_\phi \) for \( \phi \in \{C, T, F\} \) defines "synchronous ACP" and \( \psi |_T \) represents sequential composition. For example,
4. Parallel Composition

Table 6. Additional transition rules for gACP(A, |).

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \xrightarrow{a,w} x'/|, \ w(\phi) \in {C, T}, \ w(\psi) \in {C, T}$</td>
<td>$x \phi|\psi y \xrightarrow{a,w} (x'/|) \phi|\psi y$</td>
</tr>
<tr>
<td>$x \xrightarrow{a,w} x'/|, \ w(\phi) \in {C, T}, \ w(\psi) \in {C, F}$</td>
<td>$y \phi|\psi x \xrightarrow{a,w} y \phi|\psi (x'/|)$</td>
</tr>
<tr>
<td>$x \xrightarrow{a,w} x'/|, \ y \xrightarrow{b,w} y'/|, \ a \parallel b = c, \ w(\phi) \in {C, F}, \ w(\psi) \in {C, T, F}$</td>
<td>$x \phi|\psi y \xrightarrow{c,w} (x'/|) \phi|\psi (y'/|)$</td>
</tr>
<tr>
<td>$x \xrightarrow{a,w} x'/|, \ y \xrightarrow{b,w} y'/|, \ a \parallel b = c$</td>
<td>$x \phi|\psi y \xrightarrow{c,w} (x'/|) \phi|\psi (y'/|)$</td>
</tr>
<tr>
<td>$x \xrightarrow{a,w} x'/|, \ x \xrightarrow{a,w} x'/|, \ a \notin H$</td>
<td>$x \xrightarrow{a,w} x'/|, \ a \notin H$</td>
</tr>
</tbody>
</table>

some typical gACP identities are:

$x \phi\|\psi y = y \phi\|\psi x,$

$x \phi\|\psi y = y \phi\|\psi x,$

$\delta \phi\|\psi x = \delta.$

Like in ACP, the parallel composition operations can be eliminated from terms. As an example, we give the terms resulting from the elimination of the parametrized parallel composition in $a \phi\|\psi b$ in Figure 1.

Next, we define an operational semantics; write $P$ for the set of process-closed process terms. We extend the set of transition rules defined in Table 4 with the rules in Table 6. For the notation of these rules, we use the convention that $x'/\|$ and $y'/\|$ range over $P \cup \{\|\}$ (we stress that the symbol $\|$ is not a process term). In order to keep the presentation of the rules short, we also let

$x \phi\|\psi \| \equiv \|\phi\|\psi \| \equiv x, \text{ and } \|\phi\|\psi \| \equiv \partial_H(\|) \equiv \|.$

We stick to bisimulation equivalence as defined in Section 3, and as before it follows that bisimilarity is a congruence for all operations involved. It is not difficult (but tedious) to establish that in the bisimulation model thus obtained
all equations of Table 5 are true. Furthermore, each process-closed process
term over gACP is provably equal to, and thus bisimilar with, a generalized
basic term (see Section 3.3). Hence:

**Theorem 4.1.** The system gACP is complete with respect to bisimulation equivalence.

**Example: The Minimal History Operator.** In the following we provide an example in which the generalized operations are used.\(^4\) The minimal history operator \(H_0\) keeps track of the number of actions that a process has performed since initialization and increases stepwise its index. The knowledge of the history of a process is minimal in the sense that we only count the actions that are performed. For example, we find that

\[
H_0(abc) \xrightarrow{a} H_1(bc) \xrightarrow{b} H_2(c) \xrightarrow{c} \checkmark.
\]

In this section, we shall use the history of processes in the condition parameters of the operations of gACP; hence, we shall be able to program a scheduling mechanism for parallel processes.

Let \(\text{In}\) be the assertion which is true of the initial state of a process and false thereafter. Furthermore, let \(P(\phi)\) be the assertion that \(\phi\) is valid in all the previous states (i.e., the states immediately before the last action); if there is no such state, then \(P(\phi) = \bot\).

Though \(P\) is a modality, we have

\[
\begin{align*}
P(\top) &= \neg\text{In} \lor \bot, \\
P(\neg\phi) &= \neg P(\phi), \\
P(\phi \land \psi) &= P(\phi) \land P(\psi),
\end{align*}
\]

and one can set

\[
\begin{align*}
P(C) &= \bot <\text{In} >> C, \\
P(\bot) &= \bot.
\end{align*}
\]

It then follows that \(P\) can be removed from finite expressions except for atoms of the form \(P^n(\text{In})\) for \(n \in \mathbb{N}\).

The minimal history operator \(H_n\) is, for \(n \in \mathbb{N}\), defined on processes by

\[
\begin{align*}
H_n(a) &= a \text{ for } a \in A \cup \{\delta\}, \\
H_n(ax) &= a \cdot H_{n+1}(x) \text{ for } a \in A, \\
H_n(x +_\phi y) &= H_n(x) +_{H_n(\phi)} H_n(y),
\end{align*}
\]

and on conditions (as occurring in the last line above), by

\[
H_n(c) = c \text{ for } c \in T_4.
\]

\(^4\)This example is based on a similar one in [18].
4. Parallel Composition

\[ H_n(\text{In}) = \begin{cases} T & \text{if } n = 0, \\ F & \text{otherwise}, \end{cases} \]

\[ H_0(P(\phi)) = \delta, \]

\[ H_{n+1}(P(\phi)) = H_n(\phi), \]

\[ H_n(\neg \phi) = \neg H_n(\phi), \]

\[ H_n(\phi \land \psi) = H_n(\phi) \land H_n(\psi). \]

As an example, consider

\[ \Phi = \text{In} \lor \]

\[ (\neg \text{P(\text{In})} \land \text{P}^2(\text{In})) \lor \]

\[ (\neg \text{P(\text{In})} \land \neg \text{P}^2(\text{In}) \land \neg \text{P}^3(\text{In}) \land \text{P}^4(\text{In})). \]

The assertion \( \Phi \) is true in states where the action history length is 0, 2, or 4, and false otherwise. We assume that all communications are \( \delta \). Now consider the processes

\[ P = (\Phi \to a)(\Phi \to a)(\Phi \to b), \]

\[ Q = (\neg \Phi \to c)(\neg \Phi \to d). \]

We find that \( H_0(P \parallel Q) \) equals \( acadb \). The history operator in cooperation with \( \Phi \) schedules \( P \parallel Q \) as an alternation of steps, beginning with \( P \).

In process algebra, one often considers potentially nonterminating processes that can be specified with \( * \), the binary Kleene star [61], defined by

\[ x^* y = x(x^* y) + y. \]

(See also [12].) In particular, \( x^* \delta \) repeatedly performs \( x \), as follows easily from the axioms. An obvious question is how to provide scheduling guards for potentially nonterminating processes. This leads us to infinitary propositions, which can be defined by recursion. As an example, let

\[ \Phi_{\text{even}} = \text{In} \lor \neg \text{P(\Phi_{\text{even}})}. \]

Thus \( \Phi_{\text{even}} \) will be true for even step numbers, and it easily follows that

\[ H_0((\Phi_{\text{even}} \to a)^* \delta \parallel (\neg \Phi_{\text{even}} \to b)^* \delta) = (ab)^* \delta. \]

For another example, let

\[ \Psi = \text{In} \lor (\neg \text{P(\text{In})} \land \neg \text{P}^2(\text{In}) \land \text{P}^3(\Psi)). \]

So \( \Psi \) is true if the action history length is a multiple of 3. In order to give a somewhat more real-life example on scheduling, we consider \( \Upsilon \), the "negation" of \( \Psi \), and \( \Phi \), which is true if the action history length modulo 3 is either 0 or 2.
These infinitary propositions can be recursively defined by
\[ \Upsilon = \neg \text{In} \land (P(\text{In}) \lor P^2(\text{In}) \lor P^3(\Upsilon)), \]
\[ \Phi = \text{In} \lor P(\Upsilon). \]

Now consider the processes
\[ S = (\sum_d r_1(d) \cdot s_2(d)) \cdot \delta, \]
\[ R = (\sum_d r_2(d) \cdot s_3(d)) \cdot \delta. \]

The idea is that sender \( S \) receives a datum from some finite domain along channel 1 from the environment and then sends this datum via channel 2, while receiver \( R \) receives data along channel 2 and propagates these along channel 3. Now, using \( \Phi \) and \( \Psi \), the parallel composition of \( S \) and \( R \) can be scheduled in such a way that only communications (data transmissions) can occur along channel 2: it is not hard to show that for \( k \in \mathbb{N} \),
\[ H_{3k}(S \Phi || \Psi R) = \left( \sum_d r_1(d) \cdot (r_2(d) \cdot s_2(d)) \cdot s_3(d) \right) \cdot H_{3k+3}(S \Phi || \Psi R). \]

So, for naturals \( k \), and in particular for \( k = 0 \), we find that \( H_{3k}(S \Phi || \Psi R) \) describes the intended scheduling.

5. Completeness of the Axioms for Conditional Composition

In this section we give a full proof of the completeness of the \( \mathbb{L}_4 \) axioms, as explained in Section 2.4. We start with some useful \( \mathbb{L}_4 \) identities, and then we establish a normal form representation. We suggest a general strategy for proving \( \mathbb{L}_4 \) identities, which is then used to derive the translations of the \( \mathbb{K}_4 \) axioms in \( \mathbb{L}_4 \). Finally we argue that translating a term from \( \mathbb{L}_4 \) to \( \mathbb{K}_4 \), and translating back the result yields a provably equal term, which completes our proof.

5.1. Preliminaries. In Table 7 we recall the axiomatization for \( \mathbb{L}_4 \) given earlier in Table 2. We shall freely use the fact that the binary operation \( \sqcup \) (the abbreviation of \( \text{In} > \text{C} > \text{Out} \)) is idempotent (17), commutative (Lc1), and associative (L3).

**Lemma 5.1.** The following identities are derivable:
\[ x \sqcup x = x, \quad (17) \]
\[ (x \sqcup x') \cdot z = x \cdot y \cdot z \sqcup x' \cdot y \cdot z, \quad (18) \]
\[ x \cdot (y \sqcup y') = z = x \cdot y \cdot z \sqcup x \cdot y' \cdot z, \quad (19) \]
\[ x \cdot y \cdot (z \sqcup z') = x \cdot y \cdot z \sqcup x \cdot y \cdot z'. \quad (20) \]
5. Completeness of the Axioms for Conditional Composition

Table 7. Axioms of L4.

| (L1) | \( x \triangleleft (x' \triangleleft y \triangleright z') \triangleright z = (x \triangleleft x' \triangleright z) \triangleright y \triangleright (x \triangleleft z' \triangleright z) \) |
| (L2) | \( (x \triangleleft y \triangleright z) \triangleright y' = (x' \triangleleft y \triangleright z') = (x \triangleleft y' \triangleright x') \triangleright y \triangleright (z \triangleleft y' \triangleright z') \) |
| (L3) | \( (x \triangleleft y \triangleright x') \triangleright y \triangleright z = x \triangleleft y \triangleright (x' \triangleleft y \triangleright z) \) |
| (L4) | \( T \triangleleft x \triangleright F = x \) |
| (LT) | \( x \triangleleft T \triangleright y = x \) |
| (LF) | \( x \triangleleft F \triangleright y = y \) |
| (LD) | \( x \triangleleft D \triangleright y = D \) |
| (LC1) | \( x \triangleleft C \triangleright y = y \triangleleft C \triangleright x \) |
| (LC2) | \( x \triangleleft C \triangleright D = x \) |
| (LC3) | \( C \triangleleft C \triangleright x = C \) |

Proof. In the case of (17) we derive using axiom LC2 that \( x \triangleleft C \triangleright x \) equals

\[
(x \triangleleft C \triangleright D) \triangleleft C \triangleright (x \triangleleft C \triangleright D)
\]

which is derivably equal to \( x \) by L1, LC3, and LC2. Equations (18) and (20) are derived using (17) and L2. Equation (19) is an instance of axiom L1.

The identities in the following lemma are used below, when we introduce normal forms for L4.

Lemma 5.2. The following identities are derivable:

\[
D \triangleleft x \triangleright D = D,
\]

\[
(x \triangleleft y \triangleright z) = x \triangleleft y \triangleright D \sqcup D \triangleleft y \triangleright z,
\]

\[
x \triangleleft y \triangleright z = z \triangleleft (F \triangleleft y \triangleright T) \triangleright x,
\]

\[
(x \triangleleft z \triangleright D) \triangleleft y \triangleright D = (x \triangleleft y \triangleright D) \triangleleft z \triangleright D,
\]

\[
(y \triangleleft x \triangleright D) \triangleleft x \triangleright D = y \triangleleft x \triangleright D.
\]

Proof. The left-hand side of (21) equals

\[
(y \triangleleft D \triangleright y) \triangleleft x \triangleright (y \triangleleft D \triangleright y)
\]

by axiom LD. Now apply L2 and LD. In the case of (22) we derive by LC1 and LC2 that \( x \triangleleft y \triangleright z \) equals

\[
(x \sqcup D) \triangleleft y \triangleright (D \sqcup z),
\]

which equals the right-hand side by L2. Equation (23) is derived using the axioms L1, LT, and LF. The left-hand side of (24) equals

\[
(x \triangleleft z \triangleright D) \triangleleft y \triangleright (D \triangleleft z \triangleright D)
\]

by (21); this case is finished using L2 and (21). Finally, equation (25) is derived using axiom L3 and (21).
5.2. Normal Forms. We define simple normal forms as follows: the truth values T and F are simple normal forms; if \( t \) is a simple normal form, then \( t \land u \triangleright D \) is a simple normal form for any term \( u \).

A normal form is a least upper bound

\[
\bigsqcup_{i \in I} t_i
\]

of simple normal forms \( t_i \), where \( I \) is a finite set of indices; we define \( \bigsqcup_{i \in \emptyset} t_i = D \).

Every simple normal form is of the form

\[
((\cdots((a \land u_n \triangleright D)\land u_2 \triangleright D)\land u_1 \triangleright D),
\]

where \( a \in \{T, F\} \), for some \( n \geq 0 \). We call the \( u_i \) the guards of the simple normal form. Using equations (24) and (25), we see that the order of the guards can be changed, and that double occurrences of the same guard can be identified. Hence, we shall write these simple normal forms with the set of guards notation

\[
\{u_1, \ldots, u_n\}a.
\]

**Proposition 5.3.** For all terms \( u_1, \ldots, u_n \) and \( a \in \{T, F\} \) we have

\[
\{u_1, \ldots, u_n\}a = a \land (u_1 \land \cdots \land u_n) \triangleright D.
\]

A normal form consists of a T-part and an F-part: it can be written as

\[
\bigsqcup_{i} \alpha_i T \sqcup \bigsqcup_{j} \alpha_j F,
\]

where the \( \alpha_i, \alpha_j \) are finite sets of terms. As an example, we derive a normal form for the variable \( x \) using L4, (22), and (23):

\[
x = T \land x \triangleright F = T \land x \triangleright D \sqcup D \land x \triangleright F = \{x\}T \sqcup \{F \land x \triangleright T\}F,
\]

where the right-hand side is a normal form. The following theorem is a consequence of (26):

**Theorem 5.4.** Every term of \( \mathbb{L}_4 \) is derivably equal to a normal form.

A simple normal form is optimal, if all its guards are either variables or negated variables: a simple normal form \( \alpha T \) or \( \alpha F \) is optimal if every element of \( \alpha \) is either a variable or of the form \( F \land x \triangleright T \) for some variable \( x \), where in the latter case it is called the negation of \( x \). We shall further abbreviate the negation of \( x \) by \( \neg x \). (Of course, the classical negation operation is defined exactly like this, cf. (3).) It is not difficult to prove that every term is derivably equal to an optimal normal form, that is, to a least upper bound of optimal simple normal forms.
Finding optimal normal forms is a straightforward procedure, as is illustrated by (26) and

\[
x < y \triangleright z = x < y \triangleright D \cup z < \neg y \triangleright D
\]

\[
= ([x] T \cup \{\neg x\} F) < y \triangleright D \cup ([z] T \cup \{\neg z\} F) < \neg y \triangleright D
\]

\[
= \{y, x\} T \cup \{y, \neg x\} F \cup \{\neg y, z\} T \cup \{\neg y, \neg z\} F,
\]

\tag{27}

where we used identities (22), (23), (26), and (18).

We present some useful identities concerning normal forms:

**Lemma 5.5.** Let \( \alpha \) and \( \beta \) be finite sets of terms. We can derive the following identities:

\[
x < \alpha T \triangleright y = x < (\alpha \cup \beta) T \triangleright y, \tag{28}
\]

\[
x < \alpha F \triangleright y = D < (\alpha \cup \beta) T \triangleright y, \tag{29}
\]

\[
\beta T < \alpha T \triangleright D = (\alpha \cup \beta) T, \tag{30}
\]

\[
\beta F < \alpha T \triangleright D = (\alpha \cup \beta) F, \tag{31}
\]

\[
D < \alpha F \triangleright y = x < \alpha T \triangleright D. \tag{32}
\]

**Proof.** We prove (28) using induction on \( |\alpha| \). If \( \alpha = \emptyset \), then \( \alpha T = T \) and the identity follows from axiom LT. If \( \alpha = \alpha' \cup \{u\} \), for some \( u \not\in \alpha' \), then

\[
x < \alpha T \triangleright y = x < (\alpha' \cup \{u\}) T \triangleright y \]

\[= (x < \alpha' T \triangleright y) < u \triangleright D \quad \text{(by L1, LD)}
\]

\[= (x < \alpha' T \triangleright D) < u \triangleright D \quad \text{(by IH)}
\]

\[= x < \alpha T \triangleright D.
\]

We prove (30) using induction on \( |\alpha| \). If \( \alpha = \emptyset \), then \( \alpha T = T \) and the identity follows from axiom LT. If \( \alpha = \alpha' \cup \{u\} \), for some \( u \not\in \alpha' \), then

\[
\beta T < \alpha T \triangleright D = \beta T < (\alpha' \cup \{u\}) T \triangleright D
\]

\[= (\beta T < \alpha' T \triangleright D) < u \triangleright D \quad \text{(by L1, LD)}
\]

\[= (\alpha' \cup \beta) T < u \triangleright D \quad \text{(by IH)}
\]

\[= (\alpha \cup \beta) T.
\]

The proofs of the other identities are similar. \(\square\)

**Lemma 5.6** (Absorption). If \( \alpha \) and \( \beta \) are finite sets of terms, and \( \alpha \subseteq \beta \), then we can derive

\[
\alpha T \cup \beta T = \alpha T \quad \text{and} \quad \alpha F \cup \beta F = \alpha F. \tag{Abs}
\]

**Proof.** We derive

\[
\alpha T \cup (\alpha \cup \beta) T = \alpha T < C \triangleright D \cup \alpha T < \beta T \triangleright D
\]

\[= \alpha T < (C \cup \beta T) \triangleright D \quad \text{(by L1)}
\]

\[= \alpha T \quad \text{(by LC3, LC2)}.
\]
The proof of the second part of the lemma is similar using (31).

Now, a general strategy for proving equations between open terms is to write both sides as (optimal) normal forms, and then apply absorption. As an example, we derive the identity \( x \iff x \iff x = x \):

\[
x \iff x \iff x = \{x\}T \cup \{x, \neg x\}F \cup \{x, \neg x\}T \cup \{\neg x\}F \\
= \{x\}T \cup \{\neg x\}F \\
= x
\]  
(by (27))

(by (Abs))

(by (26)).

5.3. Derivation of the \( K_4 \) Axioms. In this section, we show that the axioms of the logic \( K_4 \) are, after translation to \( L_4 \), derivable from the \( L_4 \) axioms. Together with the proof of the translation invariance presented in the next section, this constitutes a completeness proof for \( L_4 \). The translation from \( K_4 \) to \( L_4 \) is based on (3), (4), and (5), presented in Section 2.3. For convenience, we repeat them here:

\[
\neg x = F \iff x \iff T,  
\]

\[
x \land y = y \iff x \iff F \cup x \iff y \iff F, 
\]

\[
x \lor y = T \iff x \iff y \iff T \iff x \iff y \iff x. 
\]

As before, we write \( \neg x \) for \( F \iff x \iff T \) in the setting of \( L_4 \).

We start with the axioms for negation; these cases are straightforward: first, axiom N0 translates to

\[
F \iff (y \iff x \iff F \cup y \iff x \iff F) \iff T = T \iff \neg x \iff \neg y \iff T \iff \neg y \iff \neg x. 
\]

We find by application of axioms L1 and LF that the left-hand side equals

\[
F \iff (y \iff x \iff F) \iff T \iff F \iff (x \iff y \iff F) \iff T, 
\]

which equals the right-hand side by (23). Axiom N1 translates to

\[
F \iff (F \iff x \iff T) \iff T = x, 
\]

which is derived using axioms L1, LT, LF, and L4. Axiom N2 translates to the identity \( F \iff T \iff T = F \) which is an instance of axiom LT. For axiom N3 we find \( F \iff C \iff T = C \) which can be derived using axioms LC1 and L4. Finally, axiom N4 translates to \( F \iff D \iff T = D \) which is an instance of axiom LD.

Next we turn to the axioms K1–K6. In the cases of the axioms K2–K4, it is not easy to find a “direct” derivation; in these cases we use rewriting to optimal normal forms, after which application of absorption (Abs) yields the required identity. First, axiom K1 translates to an instance of axiom LC1. In the case of axiom K2 we find that the left-hand side translates to

\[
z \iff (x \iff y \iff F \cup y \iff x \iff F) \iff F \iff (x \iff y \iff F \cup y \iff x \iff F) \iff z \iff F, 
\]

while the right-hand side translates to

\[
(y \iff z \iff F \cup z \iff y \iff F) \iff x \iff F \iff x \iff (y \iff z \iff F \cup z \iff y \iff F) \iff F. 
\]
Straightforward computation yields that both sides equal the optimal normal form
\[ \{x, y, z\} T \cup \{-x\} F \cup \{-y\} F \cup \{-z\} F, \]
which finishes this case. The left-hand side of axiom K3 translates to
\[ (T \triangleleft y \triangleright z \cup T \triangleleft z \triangleright y) \triangleleft x \triangleright F \cup x \triangleleft (T \triangleleft y \triangleright z \cup T \triangleleft z \triangleright y) \triangleright F, \]
and the right-hand side translates to
\[ T \triangleleft (x \triangleleft y \triangleright F \cup y \triangleleft x \triangleright F) \triangleright (x \triangleleft z \triangleright F \cup z \triangleleft x \triangleright F) \triangleright (x \triangleleft y \triangleright F \cup y \triangleleft x \triangleright F). \]

Straightforward computation yields that both sides equal the optimal normal form
\[ \{-x\} F \cup \{-y, -z\} F \cup \{x, y\} T \cup \{x, z\} T. \]

Axiom K4 translates to
\[ T \triangleleft x \triangleright (y \triangleleft x \triangleright F \cup x \triangleleft y \triangleright F) \triangleright (y \triangleleft x \triangleright F \cup x \triangleleft y \triangleright F) \triangleright x = x. \]

It is not difficult to derive both sides equal to the optimal normal form \( \{x\} T \cup \{-x\} F \). Axiom K5 translates to \( x \triangleleft T \triangleright F \cup T \triangleleft x \triangleright F = x \), which is derivable using axioms L4 and LT, and identity (17). Finally, axiom K6 translates to \( D \triangleleft C \triangleright F \cup C \triangleleft D \triangleright F = F \), which can be derived using axioms LC1, LC2, and LD.

5.4. Translation Invariance. For a term \( t \) in the \( L_4 \) signature, we write \( t' \) for its translation to \( K_4 \) (by (6)), and for a term \( t \) of \( K_4 \), we write \( t^* \) for its translation to \( L_4 \) (by (3), (4) and (5)). We give a proof of the translation invariance: we show that every term \( t \) of \( L_4 \) is derivably equal to \((t')^*\).

Consider the term \( t \equiv u \triangleleft v \triangleright w \), where \( u \), \( v \), and \( w \) are arbitrary terms. We prove that \((t')^*\) is derivably equal to \( t \) in \( L_4 \) using induction on terms: we assume that \((x')^*\) is derivably equal to \( x \) for \( x = u, v, w \).

First, we translate \( t \) according to (6):
\[ t' = (s_1 \lor s_2) \lor (s_3 \lor s_4), \]
where
\[ s_1 = u' \land v', \]
\[ s_2 = w' \land \neg v', \]
\[ s_3 = (u' \land w') \land D, \]
\[ s_4 = (v' \land \neg u') \land D. \]
Then we translate $t'$ back to $L_4$, and show that the result is derivably equal to $t$. We apply the translation to $L_4$ bottom-up: we first translate the $s_i$ to $L_4$. We find

$$s_1^* = (u' \land v')^*$$

$$= (v')^* \land (u')^* > F \sqcup (u')^* \land (v')^* > F$$

$$= v < u > F \sqcup u < v > F,$$

where we used the induction hypothesis in the last step. Similarly, using the induction hypothesis, we find

$$s_2^* = \neg v < u > F \sqcup w < \neg v > F,$$

$$s_3^* = F \sqcup (w < u > F \sqcup u < w > F) > F,$$

$$s_4^* = D \sqcup (\neg v < v > F \sqcup v < \neg v > F) > F,$$

where $\neg v$ stands for $F < v > T$. Normal forms for the $s_i^*$ terms:

$$s_1^* = \{u, v\} T \sqcup \{u\} F \sqcup \{v\} F,$$

$$s_2^* = \{w, \neg v\} T \sqcup \{\neg w\} F \sqcup \{v\} F,$$

$$s_3^* = \{\neg u\} F \sqcup \{w\} F,$$

$$s_4^* = \{v\} F \sqcup \{\neg v\} F.$$

Now, we compute a normal form for $(s_1 \lor s_2)^*$. We find that

$$(s_1 \lor s_2)^* = T \land s_1^* \lor T \land s_2^* \lor T \land s_2^* \lor s_1^*. $$

We derive

$$T \land s_1^* \lor s_2^* = T \land (\{u, v\} T \sqcup \{\neg u\} F \sqcup \{v\} F) > s_2^*$$

$$= T \land (\{u, v\} T \lor s_2^* \sqcup \{\neg u\} F \lor s_2^* \sqcup \neg v F) > s_2^*$$

(by (28), (29))

$$= T \land (\{u, v\} T \lor s_2^* \lor \{\neg u\} F \lor \neg v F) > \ldots$$

(by (30), (32))

$$= \{u, v\} T \lor \ldots$$

(by (18), (30), (31)).

Similarly, we derive

$$T \land s_2^* \lor s_1^* = \{w, \neg v\} T \lor \{w, u, v\} T \sqcup \{\neg w, \neg u\} F$$

$$\lor \{\neg w, \neg v\} F \lor \ldots$$

Combining these results, we find by application of absorption (Abs) that

$$(s_1 \lor s_2)^* = \{u, v\} T \lor \{\neg u, \neg w\} F$$

$$\lor \{\neg u, v\} F \lor \ldots$$

(33)
For \((s_3 \lor s_4)^*\) we find:
\[
(s_3 \lor s_4)^* = T < s_3^* \triangleright s_4^* \sqcup T < s_4^* \triangleright s_3^*.
\]
We derive
\[
T < s_3^* \triangleright s_4^* = T < (\{u\} \cup \{\neg v\}) \triangleright s_4^*
\]
\[
= T < (\{u\}) \triangleright T \sqcup T < (\{\neg v\}) \triangleright s_4^* \quad (\text{by L1})
\]
\[
= s_4^* < (\{u\}) T \triangleright D \sqcup s_4^* < (\{\neg v\}) T \triangleright D \quad (\text{by (29), (32)})
\]
\[
= ((\{v\} \cup \{\neg v\}) \triangleright (\{u\}) T \triangleright D
\]
\[
\sqcup ((\{v\} \cup \{\neg v\}) \triangleright (\{\neg v\}) T \triangleright D
\]
\[
= \{u, v\} \triangleright (\{u, v\} \triangleright D
\]
\[
\sqcup \{w, v\} \triangleright (\{w, v\} \triangleright D
\]
\[
(\text{by (33))}
\]
A similar derivation yields the same normal form for \(T < s_4^* \triangleright s_3^*\). Hence,
\[
(s_3 \lor s_4)^* = \{u, v\} \triangleright (\{u, v\} \triangleright D
\]
\[
\sqcup \{u, v\} \triangleright (\{u, v\} \triangleright D
\]
\[
(\text{by (34))}
\]
We are now ready to compute \(((s_1 \lor s_2) \lor (s_3 \lor s_4))^*\), which equals \(r_1 \sqcup r_2\) with
\[
r_1 = T < (s_1 \lor s_2) \triangleright (s_3 \lor s_4)^* \quad \text{and} \quad r_2 = T < (s_3 \lor s_4)^* \triangleright (s_1 \lor s_2)^*.
\]
We derive
\[
\quad r_1 = T < \{u, v\} \triangleright (\{u, v\} \triangleright D
\]
\[
\sqcup \{u, v\} \triangleright (\{u, v\} \triangleright D
\]
\[
= \{u, v\} \triangleright (\{u, v\} \triangleright D
\]
\[
\sqcup \{u, v\} \triangleright (\{u, v\} \triangleright D
\]
\[
(\text{by (33))}
\]
Similarly, we find
\[
r_2 = \{u, v\} \triangleright (\{u, v\} \triangleright D
\]
\[
\sqcup \{u, v\} \triangleright (\{u, v\} \triangleright D
\]
\[
(\text{by (34))}
\]
Combining these results we find using absorption (Abs):
\[
((s_1 \lor s_2) \lor (s_3 \lor s_4))^* = \{u, v\} \triangleright (\{u, v\} \triangleright D
\]
\[
\sqcup \{u, v\} \triangleright (\{u, v\} \triangleright D
\]
where the right-hand side is easily shown to be equal to \(u < v \triangleright w\).
6. Conclusions

This article follows a number of papers about the combination of process algebra and non-standard propositional logics, among which [17, 18, 20, 21]. Motivation for this line of research is the characterization of erroneous behavior using propositional logics with non-standard truth values.

In [13], Bergstra, Bethke and Rodenburg introduced a four-valued propositional logic comprising the special values D and M (meaningless) and proposed the ‘information ordering lattice’ where M majorizes T and F, while D is their greatest lower bound. Furthermore, in the spirit of McCarthy [67], these authors introduced special connectives for the sequential interpretation of the usual connectives (instead of directing the evaluation of these, as is done in [67]). In particular, left sequential conjunction, notation \( \land \), can be motivated as providing an interpretation of conjunction with an operational, sequential flavor (for instance suitable to represent lazy, left sequential evaluation of conditions in imperative programming). Finally, the truth value M represents a catastrophic notion of ‘meaningless’; typically, \( x \land M = x \lor M = \neg M = M \), whereas for instance \( F \land M = F \). The truth values D and M can be motivated as covering all types of “errors” that one would want to characterize in error modelling. This four-valued logic, with truth values \{M, T, F, D\}, is combined with process algebra in [18], where a strict correspondence between the truth value D and inaction \( \delta \) is established.

In [20], a five valued logic with truth values \{M, C, T, F, D\} is introduced: in that paper it is observed that Kleene’s partial logic admits another interpretation of the ‘undefined’ value (in our setting: D), namely that of the value C. The question whether this interpretation has any association with a relevant phenomenon, and if so, how the associated truth value can be combined with the values previously distinguished is settled in that paper. (Moreover, the truth value C is added to the information ordering lattice as the least upper bound of T and F, and is majorized by M.) Finally, conditional composition is introduced as a logical operation, making left-sequential conjunction, as well as the associated right-sequential and dual operations, definable:

\[
x \land y = y \triangleleft x \triangleright F.
\]

This article starts from the observation that with a propositional logic over C and D, the more involved primitives of ACP, i.e., choice and inaction, can be characterized via conditional composition (\( +_C \) and \( +_D \), respectively). This justifies the introduction of a four-valued propositional logic over \{C, T, F, D\} with conditional composition, and the idea to call this logic “the logic of ACP”. We studied this logic in detail, and showed that it can either be viewed as a

5In [13], left-sequential and symmetric (or parallel) conjunction both occur in a single logic.  
6 Also in [18], a strict correspondence is established between the truth value M and a process constant \( \mu \) representing chaos (which can be added to ACP).
sequential one ($\mathbb{L}_4$), or as a symmetric one ($\mathbb{K}_4$). The main contributions of this article are: the establishment of the logic $\mathbb{L}_4$ and the demonstration of its equivalence with $\mathbb{K}_4$; the provision of a complete axiomatization for $\mathbb{L}_4$; the definition of normal forms that allows effective equational reasoning in $\mathbb{L}_4$; and the definition of the generalization of ACP with respect to $\mathbb{L}_4$.

A final word about the truth value $C$. We may include proposition letters in our logic and choose to interpret these only as $T$, $F$, or $D$ (thus excluding $C$ from interpretation). Motivation for this choice is that $C$ typically models a situation that is beyond any means of analysis or control (as, e.g., the order of interleaving in a concurrent process), and hence a situation that cannot be referenced by a user-defined proposition. Of course, all our completeness results are preserved when proposition letters are added to one of the logics discussed, and interpretation may follow the consideration raised here.