Available online at www.sciencedirect.com

ScienceDirect

The Journal of Logic and Algebraic Programming 70 (2007) 222–235

THE JOURNAL OF LOGIC AND ALGEBRAIC PROGRAMMING

www.elsevier.com/locate/jlap

A generalization of ACP using Belnap's logic

Alban Ponse*, Mark B. van der Zwaag

Programming Research Group, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, Netherlands

Abstract

ACP is combined with Belnap's four-valued logic via conditional composition (*if–then–else*). We show that the operators of ACP can be seen as instances of more general, conditional operators. For example, both the choice operator $+$ and δ (deadlock) can be seen as instances of conditional composition, and the axiom $x + \delta = x$ follows from this perspective. Parallel composition is generalized to the binary conditional merge ${}_{\phi}\|_{\psi}$ where ϕ covers the choice between interleaving and synchronization, and ψ determines the order of execution. The instance ${}_{\mathtt{B}}\|_{\mathtt{B}}$ is ACP's parallel composition, where $\mathtt{B}$ (both) is the truth value that models both true and false in Belnap's logic. Other instances of this conditional merge are sequential composition, pure interleaving and synchronous merge. We investigate the expression of scheduling strategies in the conditions of the conditional merge.
© 2006 Elsevier Inc. All rights reserved.

Keywords: Belnap's logic; Conditional composition; Process algebra; ACP

1. Introduction

In 1994, our research group at the University of Amsterdam experienced a revival in the specification of datatypes with divergence, errors and recovery or exception handling. This was triggered by languages such as VDM [13] and an upcoming interest in Java [9]. The first outcome was a paper on a four-valued propositional logic by Bergstra, Bethke and Rodenburg [4]. Consequently it was felt that a combination with ACP [5] via a conditional composition construct (i.e., an *if–then–else* operator) was obvious, and a first paper involving Kleene's three-valued logic [14] was written [7], the idea being that in

if ϕ *then* p *else* q

the condition ϕ may take Kleene's truth value *undefined*. This led to [8] in which the logic of [4] is combined with ACP, to papers in which other non-classical logics were used, and ultimately to the four-valued logic $\mathbf{C}_4$ for ACP with conditional composition ([16] baptized "the logic of ACP"). In [16] we show that this logic (with one, sequential connective) is equivalent to the natural extension of Kleene's three-valued logic with a fourth truth value (which has symmetric connectives). As known from, e.g., [11], this latter logic is in fact Belnap's four-valued logic [2]. In a recent paper based on [16] we report on Belnap's logic with conditional composition as a functional basis [18]. Here, we focus on the generalization of ACP using this logic. An extended abstract of this paper was presented at the APC25 workshop [17].

* Corresponding author. Tel.: +31 20 525 7592; fax: +31 20 525 7490.
E-mail address: alban@science.uva.nl (A. Ponse).

1567-8326/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jlap.2006.08.006

A tricky corner in ACP is the combination of choice and deadlock. One often reads that the process $p + q$ makes a choice between p and q. However, this is not true for $a + \delta$, where a is an action and δ represents deadlock. Indeed, $x + \delta = x$ is an axiom of ACP, and it suggests a descriptive interpretation of the choice. In this paper we show that a prescriptive reading of choice can be obtained via a straightforward correspondence with conditional composition over Belnap's logic, allowing one to explain the nature of choice in process algebra from a logical perspective. Writing

$$p +_{\phi} q$$

for *if ϕ then p else q*, and with truth value B (both) standing for both true and false, and N (none) standing for neither true or false, we define

$$\begin{aligned} p + q &\overset{\text{def}}{=} p +_{\mathtt{B}} q, \\ \delta &\overset{\text{def}}{=} p +_{\mathtt{N}} q. \end{aligned}$$

Secondly, we generalize parallel composition ($\|$) to a conditional operator as well:

$$p\,{}_{\phi}\|_{\psi}\,q$$

is the parallel composition of p and q under conditions ϕ and ψ, as explained in Section 3.3. This will allow us to define several forms of parallel composition. Moreover, we can enforce certain strategies for scheduling.

In Section 2 we review Belnap's logic and conditional composition. In Section 3 we define generalized ACP using the conditional operators. Section 4 contains the completeness proof for generalized ACP. This result is extended to ω-completeness in Section 4.3. In Section 5 we consider ways to express scheduling strategies in the conditions of the conditional merge.

2. Belnap's logic and conditional composition

Belnap's Logic $\mathbf{B}_4$ [2] has truth values B, T, F, and N, where B (both) represents both true and false, T and F are the values true and false, and N (none) represents undefinedness.[1] Negation is defined as an involution (i.e., it satisfies $\neg\neg x = x$) by $\neg\mathtt{B} = \mathtt{B}$, $\neg\mathtt{T} = \mathtt{F}$, $\neg\mathtt{F} = \mathtt{T}$, and $\neg\mathtt{N} = \mathtt{N}$, and conjunction ($\wedge$) and disjunction ($\vee$) are the greatest lower bound and the least upper bound in the distributive lattice

$$\begin{tikzcd} & \mathtt{T} \arrow[dl, dash] \arrow[dr, dash] & \\ \mathtt{B} \arrow[dr, dash] & & \mathtt{N} \arrow[dl, dash] \\ & \mathtt{F} & \end{tikzcd}$$

called the *truth ordering* [11]. This characterization of the logic as a distributive lattice with involution leads directly to a finite and complete equational axiomatization [16].

Now we define an alternative logic $\mathbf{C}_4$ over these truth values that has only one, ternary operator $\_ \lhd \_ \rhd \_$ called *conditional composition*. This operator is defined by

$$\begin{aligned} &x \lhd \mathtt{T} \rhd y = x, \\ &x \lhd \mathtt{F} \rhd y = y, \\ &x \lhd \mathtt{N} \rhd y = \mathtt{N}, \\ &x \lhd \mathtt{B} \rhd y = x \sqcup y, \end{aligned}$$

where $\sqcup$ is the least upper bound of x and y in the lattice

$$\begin{tikzcd} & \mathtt{B} \arrow[dl, dash] \arrow[dr, dash] & \\ \mathtt{T} \arrow[dr, dash] & & \mathtt{F} \arrow[dl, dash] \\ & \mathtt{N} & \end{tikzcd}$$

[1] Belnap motivated B as the result of conflicting outcomes of database queries, and N as the absence of answers.

Table 1
$\mathbf{C}_4$ axioms

(C1)	$x \triangleleft (u \triangleleft v \triangleright w) \triangleright y = (x \triangleleft u \triangleright y) \triangleleft v \triangleright (x \triangleleft w \triangleright y)$
(C2)	$(x \triangleleft w \triangleright y) \triangleleft v \triangleright (x' \triangleleft w \triangleright y') = (x \triangleleft v \triangleright x') \triangleleft w \triangleright (y \triangleleft v \triangleright y')$
(C3)	$(x \triangleleft w \triangleright y) \triangleleft w \triangleright z = x \triangleleft w \triangleright (y \triangleleft w \triangleright z)$
(C4)	$\mathtt{T} \triangleleft x \triangleright \mathtt{F} = x$
(C5)	$x \triangleleft \mathtt{T} \triangleright y = x$
(C6)	$x \triangleleft \mathtt{F} \triangleright y = y$
(C7)	$x \triangleleft \mathtt{N} \triangleright y = \mathtt{N}$
(C8)	$x \triangleleft \mathtt{B} \triangleright y = y \triangleleft \mathtt{B} \triangleright x$
(C9)	$x \triangleleft \mathtt{B} \triangleright \mathtt{N} = x$
(C10)	$\mathtt{B} \triangleleft \mathtt{B} \triangleright x = \mathtt{B}$

called the *information (*or *knowledge) ordering* [4,11]. Conditional composition has an operational, sequential reading: in $x \triangleleft y \triangleright z$, first y is evaluated, and depending on the outcome, possibly x and/or z. In Table 1 we give a complete set of axioms for $\mathbf{C}_4$ (see [16]).

The logics $\mathbf{B}_4$ and $\mathbf{C}_4$ have exactly the same expressiveness, that is, their operators can be defined in terms of each other. Using $\mathtt{B}, \mathtt{T}, \mathtt{F}$ we have

$$\neg x = \mathtt{F} \triangleleft x \triangleright \mathtt{T}, \tag{1}$$

$$x \wedge y = (y \triangleleft x \triangleright \mathtt{F}) \triangleleft \mathtt{B} \triangleright (x \triangleleft y \triangleright \mathtt{F}), \tag{2}$$

$$x \vee y = (\mathtt{T} \triangleleft x \triangleright y) \triangleleft \mathtt{B} \triangleright (\mathtt{T} \triangleleft y \triangleright x), \tag{3}$$

and, vice versa, using $\mathtt{N}$,

$$x \triangleleft y \triangleright z = (x \wedge y) \vee (z \wedge \neg y) \vee (x \wedge z \wedge \mathtt{N}) \vee (y \wedge \neg y \wedge \mathtt{N}).$$

Hence the two logics can be considered “the same”, but with a different functional basis. We refer to [18] for a further exposition.

3. Generalized ACP

We generalize ACP's operators for alternative composition and parallel composition to conditional operators, where the conditions are propositions in the four-valued logic introduced in Section 2.

3.1. Conditional composition

We first look at Generalized Basic Process Algebra with deadlock (GBPA$_\delta$). It is parametrized by a non-empty set A of actions symbols, and has the binary operators sequential composition $\cdot$, and conditional composition $+_\phi$, where ϕ ranges over the terms of the logic $\mathbf{C}_4$. The composition

$$p +_\phi q$$

is read as *if ϕ then p else q*. Finally it has the constant δ for deadlock. Process terms p 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 non-empty, countably infinite set V of process variables, and ϕ ranges over the terms of the logic $\mathbf{C}_4$. We use x, y, z to range over the variables in V, and p, q, r to range over process terms. Sequential composition binds strongest and the symbol $\cdot$ is often omitted from terms (so we may write pq for $p \cdot q$). The axiom system GBPA$_\delta$ consists of the axioms in Table 2. For the proof system (equational logic) we adopt the rule

$$\mathbf{C}_4 \vdash \phi = \psi \;\Rightarrow\; \text{GBPA}_\delta \vdash x +_\phi y = x +_\psi y.$$

Next, we give an operational semantics for *process-closed* terms, that is, for process terms that do not contain process variables. The conditions may be open $\mathbf{C}_4$ terms. Let $\mathbb{P}$ be the set of process-closed terms, let W be the set of valuations for $\mathbf{C}_4$ terms, and let $A \times W$ be the set of transition labels. The transition rules are given in Table 3, where a transition with label a, w models the execution of action a under valuation w.

Table 2
GBPA_δ axioms

(G1)	$x +_{\phi \lhd \psi \rhd \chi} y = (x +_\phi y) +_\psi (x +_\chi y)$
(G2)	$(x +_\psi y) +_\phi (x' +_\psi y') = (x +_\phi x') +_\psi (y +_\phi y')$
(G3)	$x +_\phi (y +_\phi z) = (x +_\phi y) +_\phi z$
(G4)	$(x +_\phi y)z = xz +_\phi yz$
(G5)	$(xy)z = x(yz)$
(G6)	$x +_{\mathtt{T}} y = x$
(G7)	$x +_{\mathtt{F}} y = y$
(G8)	$x +_{\mathtt{N}} y = \delta$

Table 3
Transition rules ($a \in A$, $w \in W$)

$$a \xrightarrow{a,w} \surd \qquad \frac{p \xrightarrow{a,w} \surd}{pq \xrightarrow{a,w} q} \qquad \frac{p \xrightarrow{a,w} p'}{pq \xrightarrow{a,w} p'q}$$

$$\frac{p \xrightarrow{a,w} \surd,\ w(\phi) \in \{\mathtt{B},\mathtt{T}\}}{p +_\phi q \xrightarrow{a,w} \surd} \qquad \frac{p \xrightarrow{a,w} \surd,\ w(\phi) \in \{\mathtt{B},\mathtt{F}\}}{q +_\phi p \xrightarrow{a,w} \surd}$$

$$\frac{p \xrightarrow{a,w} p',\ w(\phi) \in \{\mathtt{B},\mathtt{T}\}}{p +_\phi q \xrightarrow{a,w} p'} \qquad \frac{p \xrightarrow{a,w} p',\ w(\phi) \in \{\mathtt{B},\mathtt{F}\}}{q +_\phi p \xrightarrow{a,w} p'}$$

We define (strong) bisimulation as usual:

Definition 1. A binary relation R on $\mathbb{P}$ is a *bisimulation* if it is symmetric, and whenever pRq, then
- $p \xrightarrow{a,w} \surd$ implies $q \xrightarrow{a,w} \surd$, and
- $p \xrightarrow{a,w} p'$ for some p', implies $q \xrightarrow{a,w} q'$ for some q' with $p'Rq'$.

Process-closed terms are bisimilar ($\leftrightarrow$) if they are related by a bisimulation. Since bisimilar terms have matching action steps for every valuation, we allow (user-defined) propositions in the logic, the evaluation of which may not be constant throughout the execution of a process. The transition rules are in the *panth* format [19] (if for each ϕ, $+_\phi$ is considered a binary operator), from which it follows that bisimilarity is a congruence. Furthermore, the GBPA_δ axioms are sound, as can be easily proved. Completeness will be proved in Section 4. Bisimulation equivalence of *open* terms, and ω-completeness, are addressed and proven in Section 4.3.

3.2. Alternative composition

Both deadlock and alternative composition can be seen as instances of conditional composition. For deadlock this is obvious by axiom G8. We define alternative composition by $p + q \overset{\text{def}}{=} p +_{\mathtt{B}} q$. If we restrict conditional composition to alternative composition we obtain Basic Process Algebra with deadlock (BPA_δ) [5], that is, by this restriction we characterize exactly the models of BPA_δ (its axioms are collected in Table 4). In particular, the BPA_δ axioms can be derived:

Proposition 2. *Defining* $+$ *as* $+_{\mathtt{B}}$, *the axioms* A1–A7 *in Table* 4 *are derivable in* GBPA_δ.

Proof. Axiom A1 is derived by

$$x +_{\mathtt{B}} y = (y +_{\mathtt{F}} x) +_{\mathtt{B}} (y +_{\mathtt{T}} x) = y +_{\mathtt{F} \lhd \mathtt{B} \rhd \mathtt{T}} x = y +_{\mathtt{B}} x$$

using axioms G6, G7, G1, C8, and C4. Axiom A2 is an instance of G3. Axiom A3:

$$x +_{\mathtt{B}} x = (x +_{\mathtt{T}} y) +_{\mathtt{B}} (x +_{\mathtt{T}} y) = x +_{\mathtt{T} \lhd \mathtt{B} \rhd \mathtt{T}} y = x$$

Table 4
BPA$_\delta$ axioms

(A1)	$x + y = y + x$
(A2)	$(x + y) + z = x + (y + z)$
(A3)	$x + x = x$
(A4)	$(x + y)z = xz + yz$
(A5)	$(xy)z = x(yz)$
(A6)	$x + \delta = x$
(A7)	$\delta x = \delta$

using G6, G1, and $\mathbf{C}_4 \vdash x \lhd \mathtt{B} \rhd x = x$. Axiom A4 is an instance of G4, and A5 occurs here as axiom G5. Axiom A6 can be derived by

$$x +_{\mathtt{B}} \delta = (x +_{\mathtt{T}} y) +_{\mathtt{B}} (x +_{\mathtt{N}} y) = x +_{\mathtt{T} \lhd \mathtt{B} \rhd \mathtt{N}} y = x$$

using G6, G8, G1, and C9. Finally, A7 can be derived using G8 and G4. □

3.3. Generalized parallel composition

GACP (Generalized ACP) is parametrized by a nonempty set A of action symbols, and a binary commutative and associative function $\mid$ on $A \cup \{\delta\}$ which defines which actions communicate (we say that actions a and b communicate if $a \mid b \neq \delta$, and we assume that $a \mid \delta = \delta$ for all a). It extends GBPA$_\delta$ with a generalization ${}_\phi\|_\psi$ of parallel composition, where the condition ϕ covers the choice between interleaving and synchronization, and ψ determines the order of execution, see the explanation below. Furthermore, the axiomatization uses auxiliary operators for generalized left merge ${}_\phi⫦_\psi$ and generalized communication merge ${}_\phi|_\psi$, and there is the encapsulation operator ∂_H, which renames actions from the set $H \subseteq A$ to δ. The axioms of GACP are those of GBPA$_\delta$ together with the axioms in Table 5.

If either of the conditions in

$$p_\phi\|_\psi q$$

has value N, no action is enabled. The other cases:

- If $\phi = \mathtt{T}$, we have pure interleaving (no communication), which may be restricted by ψ. If $\psi = \mathtt{T}$, only the left-hand side p is allowed to execute, and we find that ${}_\mathtt{T}\|_\mathtt{T}$ represents sequential composition. Similarly, if $\psi = \mathtt{F}$ only q may execute, and upon its termination p continues. If $\psi = \mathtt{B}$ both may execute, so ${}_\mathtt{T}\|_\mathtt{B}$ represents nondeterministic interleaving (free merge). For example, we can derive

$$aa_\mathtt{T}\|_\mathtt{F}bb = bbaa, \quad \text{and} \quad a_\mathtt{T}\|_\mathtt{B}b = ab + ba.$$

- Next, $\phi = \mathtt{F}$ means only synchronization (no interleaving). In this case the sequentiality defined by ψ is not used; ${}_\mathtt{F}\|_\diamond$ for $\diamond \in \{\mathtt{B}, \mathtt{T}, \mathtt{F}\}$ defines synchronous merge.
- Finally, $\phi = \mathtt{B}$ means both interleaving and synchronization, as in ACP; again ψ determines the order of non-synchronized actions.

The parallel composition of ACP corresponds to ${}_\mathtt{B}\|_\mathtt{B}$; in this spectrum, it maximizes the number of nondeterministic alternatives, as was the case with alternative composition in GBPA$_\delta$.

Some typical identities:

$$x_\phi\|_\psi y = y_\phi\|_{\neg\psi} x,$$
$$x_\phi|_\psi y = y_\phi|_{\neg\psi} x,$$
$$\delta_\phi|_\psi x = \delta.$$

The transition rules are collected in Tables 3 and 6. In Table 6 we use some notational conventions such as $p_\phi\|_\psi\surd = p$ (of course, $\surd$ is not a term) to reduce the number of rules. Bisimilarity is a congruence, and all axioms are sound in the model thus obtained. The parallel composition operators can be eliminated from process-closed terms, so completeness of GACP follows from the completeness of GBPA$_\delta$, which is proved in Section 4.

Table 5
Axioms for conditional merge ($a, b \in A,\ H \subseteq A$)

(GM1)	$x\,{}_\phi\!\parallel_\psi y = (x\,{}_\phi\!\mathbin{\lfloor\!\lfloor}_\psi y +_\psi y\,{}_\phi\!\mathbin{\lfloor\!\lfloor}_{\neg\psi} x) +_\phi (x\,{}_\phi\!\mid_\psi y +_\psi y\,{}_\phi\!\mid_{\neg\psi} x)$
(GM2)	$a\,{}_\phi\!\mathbin{\lfloor\!\lfloor}_\psi x = ax$
(GM3)	$ax\,{}_\phi\!\mathbin{\lfloor\!\lfloor}_\psi y = a(x\,{}_\phi\!\parallel_\psi y)$
(GM4)	$(x +_\phi y)\,{}_\psi\!\mathbin{\lfloor\!\lfloor}_\chi z = x\,{}_\psi\!\mathbin{\lfloor\!\lfloor}_\chi z +_\phi y\,{}_\psi\!\mathbin{\lfloor\!\lfloor}_\chi z$
(GM5)	$a\,{}_\phi\!\mid_\psi b = a \mid b$
(GM6)	$a\,{}_\phi\!\mid_\psi bx = (a \mid b)x$
(GM7)	$ax\,{}_\phi\!\mid_\psi b = (a \mid b)x$
(GM8)	$ax\,{}_\phi\!\mid_\psi by = (a \mid b)(x\,{}_\phi\!\parallel_\psi y)$
(GM9)	$(x +_\phi y)\,{}_\psi\!\mid_\chi z = x\,{}_\psi\!\mid_\chi z +_\phi y\,{}_\psi\!\mid_\chi z$
(GM10)	$z\,{}_\phi\!\mid_\psi (x +_\chi y) = z\,{}_\phi\!\mid_\psi x +_\chi z\,{}_\phi\!\mid_\psi y$
(GD1)	$\partial_H(a) = a \quad \text{if } a \notin H$
(GD2)	$\partial_H(a) = \delta \quad \text{if } a \in H$
(GD3)	$\partial_H(x +_\phi y) = \partial_H(x) +_\phi \partial_H(y)$
(GD4)	$\partial_H(xy) = \partial_H(x)\partial_H(y)$

Table 6
Transition rules ($a, b, c \in A$, $w \in W$, p', q' range over $\mathbb{P} \cup \{\surd\}$, and we let $p\,{}_\phi\!\parallel_\psi \surd = p$, $\surd\,{}_\phi\!\parallel_\psi p = p$, $\surd\,{}_\phi\!\parallel_\psi \surd = \surd$, and $\partial_H(\surd) = \surd$)

$$\frac{p \xrightarrow{a,w} p',\ w(\phi)\in\{\mathtt{B},\mathtt{T}\},\ w(\psi)\in\{\mathtt{B},\mathtt{T}\}}{p\,{}_\phi\!\parallel_\psi q \xrightarrow{a,w} p'\,{}_\phi\!\parallel_\psi q}$$

$$\frac{p \xrightarrow{a,w} p',\ w(\phi)\in\{\mathtt{B},\mathtt{T}\},\ w(\psi)\in\{\mathtt{B},\mathtt{F}\}}{q\,{}_\phi\!\parallel_\psi p \xrightarrow{a,w} q\,{}_\phi\!\parallel_\psi p'}$$

$$\frac{p \xrightarrow{a,w} p',\ q \xrightarrow{b,w} q',\ a|b=c,\ w(\phi)\in\{\mathtt{B},\mathtt{F}\},\ w(\psi)\in\{\mathtt{B},\mathtt{T},\mathtt{F}\}}{p\,{}_\phi\!\parallel_\psi q \xrightarrow{c,w} p'\,{}_\phi\!\parallel_\psi q'}$$

$$\frac{p \xrightarrow{a,w} p',\ q \xrightarrow{b,w} q',\ a|b=c}{p\,{}_\phi\!\mid_\psi q \xrightarrow{c,w} p'\,{}_\phi\!\parallel_\psi q'} \qquad \frac{p \xrightarrow{a,w} p'}{p\,{}_\phi\!\mathbin{\lfloor\!\lfloor}_\psi q \xrightarrow{a,w} p'\,{}_\phi\!\parallel_\psi q}$$

$$\frac{p \xrightarrow{a,w} p',\ a\notin H}{\partial_H(p) \xrightarrow{a,w} \partial_H(p')}$$

4. Completeness

We prove that the axiom system GBPA$_\delta$ is complete with respect to strong bisimulation equivalence.

4.1. Preliminaries

The *guarded command* construct [10] is defined by

$$\phi :\to p \overset{\text{def}}{=} p +_\phi \delta.$$

It expresses the instruction to execute process p if the condition ϕ 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

$$\delta +_\phi \delta = \delta, \tag{4}$$

which is derived by

$$\delta +_\phi \delta = (x +_{\mathtt{N}} x) +_\phi (x +_{\mathtt{N}} x) = (x +_\phi x) +_{\mathtt{N}} (x +_\phi x) = \delta,$$

using axioms G8 and G2. The following identities can be derived straightforwardly[2]:

[2] Here and in the following we freely use the definition of $+$ as $+_{\mathtt{B}}$, and the axioms A1–A7, which can all be derived (Proposition 2).

$$
\begin{aligned}
x +_\phi y &= \phi :\rightarrow x + \neg\phi :\rightarrow y, && (5)\\
\phi :\rightarrow (x+y) &= \phi :\rightarrow x + \phi :\rightarrow y, && (6)\\
(\phi :\rightarrow x)y &= \phi :\rightarrow xy, && (7)\\
x + (\phi :\rightarrow x) &= x, && (8)\\
\phi :\rightarrow (\psi :\rightarrow x) &= \psi :\rightarrow (\phi :\rightarrow x). && (9)
\end{aligned}
$$

For the derivation of (8) we argue as follows:

$$x + (\phi :\rightarrow x) = (x +_{\mathtt{B}} \delta) + (x +_\phi \delta) = x +_{\mathtt{B} \triangleleft \mathtt{B} \triangleright \phi} \delta = x + \delta = x,$$

and for (9) we use (4) 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:

$$\mathtt{B} :\rightarrow x = \mathtt{T} :\rightarrow x = x; \quad \mathtt{F} :\rightarrow x = \mathtt{N} :\rightarrow x = \delta. \qquad (10)$$

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

$$
\begin{aligned}
\phi \vee \psi :\rightarrow x &= \phi :\rightarrow x + \psi :\rightarrow x, && (11)\\
\phi \wedge \psi :\rightarrow x &= \phi :\rightarrow (\psi :\rightarrow x), && (12)\\
\phi \triangleleft \psi \triangleright \chi :\rightarrow x &= \psi \wedge \phi :\rightarrow x + \neg\psi \wedge \chi :\rightarrow x. && (13)
\end{aligned}
$$

These identities can all be derived without difficulty; for example, in the case of (11) we replace the disjunction by its definition (3) and derive that the left-hand side equals

$$\phi :\rightarrow x + \neg\phi :\rightarrow (\psi :\rightarrow x) + \psi :\rightarrow x + \neg\psi :\rightarrow (\phi :\rightarrow x).$$

This term can be derived equal to the right-hand side using (8). For (12), we use (2) 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 (9).

4.2. Completeness proof

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 $\sum_{i\in\emptyset} p_i \equiv \delta$. Note: we write $p \equiv q$ for "term p is defined to be syntactically equal to q".

Definition 3. 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 4. *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 $\mathtt{T} :\rightarrow aq$ by (10). If $p \equiv \delta$, then pq equals the basic term δ by A7. If $p \equiv p_1 +_\phi p_2$, then derive using (5), G4, and (7) that

$$pq = \phi :\rightarrow p_1 q + \neg\phi :\rightarrow p_2 q.$$

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 $p' = p_1 q$ and $p'' = p_2 q$. Using (6) and (12), we derive that pq equals the basic term

$$\sum_i \phi \wedge \psi_i :\rightarrow r_i + \sum_j \neg\phi \wedge \psi_j :\rightarrow r_j.$$

Finally, if $p \equiv p_1 p_2$, then we find by axiom G5 that pq equals $p_1(p_2 q)$. Now we can apply the induction hypothesis twice in succession. □

Lemma 5. *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 $\mathtt{T} :\rightarrow a$ by (10). 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 (5), we find that p equals

$$\phi :\rightarrow p_1' + \neg\phi :\rightarrow p_2'.$$

Using (12) and (6) we get that this term equals the basic term

$$\sum_i \phi \wedge \psi_i :\rightarrow p_i + \sum_j \neg\phi \wedge \psi_j :\rightarrow p_j.$$

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 4. □

Next, we define the *height* of basic terms, that will be used as the basis for the induction in the completeness proof.

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

Lemma 6. *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 $q_i, q_j \in A$, $q_i \not\equiv q_j$ (that is, q_i and q_j are distinct symbols)*,
(iii) *for all $i \in I$, $\vdash \phi_i = \phi_i \wedge \mathtt{B}$*,
(iv) *for all $i \in I$, $\nvdash \phi_i = \mathtt{F}$*.

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 $\psi_i \vee \psi_j :\rightarrow a$ using (11). 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 $\mathtt{B}$ to all conditions ψ: we derive using (10) and (12) that

$$\psi :\rightarrow p_i = \psi :\rightarrow (\mathtt{B} :\rightarrow p_i) = \psi \wedge \mathtt{B} :\rightarrow p_i.$$

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. □

Theorem 7. *All bisimilar process-closed terms are derivably equal.*

Proof. Take bisimilar process-closed terms p_1 and p_2, and assume, without loss of generality (Lemma 5), 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 δ. Next, let $h > 0$. By Lemma 6 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 6. For $k = 1, 2$, we make the following observations.

(a) We may assume that $p_{k,i} \not\leftrightarrow p_{k,j}$ 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 6. Otherwise, let $p_{k,i} \equiv aq$ and $p_{k,j} \equiv ar$ and $q \leftrightarrow r$. 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 $\phi_{k,i} \vee \phi_{k,j} :\rightarrow p_{k,i}$ using (11). 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 6 that either $w(\phi_{k,i}) = \mathtt{B}$ or $w(\phi_{k,i}) = \mathtt{F}$.

(d) For all $i \in I_k$, $w(\phi_{k,i}) = \mathtt{B}$ for at least one $w \in W$, as follows from property (iv) of Lemma 6 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 6 and (c), we find that

$$p_k \xrightarrow{a,w} \surd \quad \text{if and only if} \quad w(\phi_{k,i}) = \mathtt{B},$$

  and, since $p_k \leftrightarrow p_{3-k}$, also $p_{3-k} \xrightarrow{a,w} \surd$ if and only if $w(\phi_{k,i}) = \mathtt{B}$. Using (d), we find that $p_{3-k,j} \equiv a$ for some unique $j \in I_{3-k}$. It follows that $w(\phi_{k,i}) = \mathtt{B}$ if and only if $w(\phi_{3-k,j}) = \mathtt{B}$, and so by (c), we find $\models \phi_{k,i} = \phi_{3-k,j}$ 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

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

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

4.3. ω-Completeness

GBPA$_\delta$ is ω-complete, that is, if all closed instantiations of an equation between open terms are derivable, then the equation is derivable. To prove this, we define an operational semantics for open terms, for which we can use the completeness result for process-closed terms. (But note that ω-completeness does not depend on a particular semantic equivalence.) We obtain this semantics by treating variables as constants: we take again the transition rules presented in Table 3, in which we now let the letter a range over the variables in V as well. In particular it follows that

$$x \xrightarrow{x,w} \surd$$

for all variables $x \in V$ and valuations $w \in W$. Bisimulation semantics is obtained by requiring that both action steps and variable steps are matched:

Definition 8. A binary relation R between (open) process terms is a *bisimulation* if it is symmetric, and whenever pRq, then, for all $\alpha \in A \cup V$, $w \in W$,

- $p \xrightarrow{\alpha,w} \surd$ implies $q \xrightarrow{\alpha,w} \surd$, and
- $p \xrightarrow{\alpha,w} p'$ for some p', implies $q \xrightarrow{\alpha,w} q'$ for some q' with $p'Rq'$.

Process terms are bisimilar ($\leftrightarroweq$) if they are related by a bisimulation.

The proof of the following lemma was inspired by a proof of a similar property in the setting of Milner's basic CCS in [1].

Lemma 9. *For all (open) terms p and q, if $\sigma(p) \leftrightarroweq \sigma(q)$ for every closed instantiation σ of the process variables in p and q, then $p \leftrightarroweq q$.*

Proof. First, for any process term s, we define the set $Res(s)$ of *residuals* of s inductively as follows:

- $s \in Res(s)$, and
- if $t \in Res(s)$ and $t \xrightarrow{\alpha,w} t'$ for some α, w, then $t' \in Res(s)$.

Take terms p and q. Let n be the total number of symbols (for actions, variables, and operators) occurring in p and q. Take an arbitrary action $a \in A$ (the set A is non-empty by assumption), let a^k for $k \geq 1$ be the sequential composition of k times action a, and let

$$t \stackrel{\text{def}}{=} \left(\sum_{0<i<n} a^i \cdot \delta\right) + a^n.$$

We write $s[x := at]$ for term s under the substitution of the closed term $a \cdot t$ for process variable x. Observation: for all $s \in Res(p) \cup Res(q)$,

$$s \not\leftrightarroweq t, \tag{a}$$

$$s[x := at] \not\leftrightarroweq t, \tag{b}$$

$$s[x := at] \not\leftrightarroweq t \cdot r, \quad \text{for all } r. \tag{c}$$

Assume that

$$\sigma(p) \leftrightarroweq \sigma(q) \quad \text{for all closed instantiations } \sigma. \tag{d}$$

We prove $p \leftrightarroweq q$ by induction on the total number of process variables occurring in p and q. The base case (with p and q process-closed) is trivial.

Inductive step. First observe that it follows from (d) that, for all closed instantiations σ,

$$\sigma(p[x := at]) \leftrightarroweq \sigma(q[x := at]),$$

where x is a process variable occurring in p and/or q. By induction hypothesis we find that

$$p[x := at] \leftrightarroweq q[x := at].$$

We now show that the symmetric closure of the relation

$$R = \{(r, s) \mid r \in Res(p),\ s \in Res(q),\ r[x := at] \leftrightarroweq s[x := at]\}$$

is a bisimulation, and thereby that $p \leftrightarroweq q$.

Take any $(r, s) \in R$. Cases:

- $r \xrightarrow{\alpha,w} \surd$, for some $\alpha \in A \cup V \setminus \{x\}$. Then also $r[x := at]$ has an (α, w)-step to $\surd$, and because $r[x := at] \leftrightarroweq s[x := at]$ it is easy to see that s has an (α, w)-step to $\surd$ as well.
- $r \xrightarrow{x,w} \surd$. Then $r[x := at]$ has an (a, w)-step to t. Because $r[x := at] \leftrightarroweq s[x := at]$, this step must be matched by $s[x := at]$, and it is easy to see that this is possible only if s has an (x, w)-step to $\surd$.
- $r \xrightarrow{\alpha,w} r'$, for some $\alpha \in A \cup V \setminus \{x\}$. Then $r[x := at]$ has an (α, w)-step to $r'[x := at]$, and because $r[x := at] \leftrightarroweq s[x := at]$ it must be that $s[x := at]$ has an (α, w)-step to some $s'[x := at]$ with $r'[x := at] \leftrightarroweq s'[x := at]$.

We find that $s \xrightarrow{\alpha,w} s'$: in case $\alpha \neq a$ this is easy to see. In case $\alpha = a$ observe that the (a, w)-step of $s[x := at]$ cannot be a step leading to a process starting with t resulting from a substitution $x := at$, because then we would violate (b) or (c). So, we find that $r' \in Res(p)$ and $s' \in Res(q)$, so that $(r', s') \in R$.

- $r \xrightarrow{x,w} r'$. Then $r[x := at]$ has an (a, w)-step to $t \cdot r'[x := at]$. Because $r[x := at] \Leftrightarrow s[x := at]$, this step must be matched by $s[x := at]$ with an (a, w)-step to some s^* such that $t \cdot r'[x := at] \Leftrightarrow s^*$. Then it must be (because the branching degree of t can only be matched by a t resulting from a substitution $x := at$) that $s^* = t \cdot s'[x := at]$ for some s' with $s \xrightarrow{x,w} s'$ and $r'[x := at] \Leftrightarrow s'[x := at]$. So $(r', s') \in R$. □

With this semantics for open terms we can prove ω-completeness along the lines of the proof of Theorem 7:

Theorem 10. *GBPA$_\delta$ is ω-complete, i.e., if all closed instantiations of an equation between open terms are derivable, then the equation is derivable.*

Proof. Take terms p and q and assume that $\sigma(p) = \sigma(q)$ is derivable for all closed instantiations σ of the process variables in p and q. Then by soundness of the axioms we find that $\sigma(p) \Leftrightarrow \sigma(q)$ for all σ, and therefore $p \Leftrightarrow q$ by Lemma 9. Now redo the proof of Theorem 7, treating variables as action symbols. This is straightforward.

More examples of this proof technique can be found in [1,12].

5. Scheduling parallel processes

In this section we lift GACP to a setting that admits a form of modal, algebraic reasoning based on $\mathbf{C}_4$. In particular we show how conditional merge can be used to model certain forms of scheduling with help of a so-called history operator. In order to discuss some non-trivial examples, we shall consider (potentially) infinite processes specified with $*$, the binary Kleene star [15]. In process algebra, $*$ is defined by

$$x^*y = x(x^*y) + y.$$

(See also [3].) For example, $a^*\delta$ repeatedly performs a, as follows easily from GBPA$_\delta$ and this defining axiom.

The *minimal history operator* H_0 uses its index to keep track of the number of actions that a process has performed and increases stepwise its index. The knowledge of the history of a process is ‘minimal’ in the sense that this operator only counts the actions that are performed. For example, we find that

$$H_0(abc) \xrightarrow{a} H_1(bc) \xrightarrow{b} H_2(c) \xrightarrow{c} \surd.$$

For $n \in \mathbb{N}$, the minimal history operator is axiomatized by

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

where the occurrence of $+_{H_n(\phi)}$ reflects the ‘lifting’ mentioned above: by defining H_n also on conditions we involve the history of a process in our conditional operators, thus admitting various forms of scheduling in the specification of parallel processes.

Below we discuss some applications using H_n and conditional merge. For this purpose we introduce the following type of modal conditions:

$$\phi ::= c \mid \mathtt{In} \mid \mathtt{P}(\phi) \mid \neg\phi \mid \phi \wedge \phi \quad \text{for } c \in \{\mathtt{B}, \mathtt{T}, \mathtt{F}, \mathtt{N}\},$$

where $\mathtt{In}$ is the assertion which is true for the initial state of a process and false thereafter, and $\mathtt{P}(\phi)$ is the assertion that ϕ is valid in all previous states (i.e., in all states with an action leading to the current state); if there is no such state, then $\mathtt{P}(\phi) = \mathtt{N}$. It is not necessary to define ‘states’ formally as will become clear below where we define the interaction between the minimal history operator and these modal conditions. We have

$$\mathtt{P}(\mathtt{T}) = \neg\mathtt{In} \vee \mathtt{N},$$

$$\begin{aligned}\mathtt{P}(\neg\phi) &= \neg\mathtt{P}(\phi),\\ \mathtt{P}(\phi \wedge \psi) &= \mathtt{P}(\phi) \wedge \mathtt{P}(\psi),\end{aligned}$$

and it is reasonable to define

$$\begin{aligned}\mathtt{P}(\mathtt{B}) &= \mathtt{N} \lhd \mathtt{In} \rhd \mathtt{B},\\ \mathtt{P}(\mathtt{N}) &= \mathtt{N}.\end{aligned}$$

It then follows that $\mathtt{P}$ can be removed from finite expressions except for expressions of the form $\mathtt{P}^k(\mathtt{In})$ for $k > 0$.

The minimal history operator H_n is, for $n \geq 0$, defined on conditions by

$$\begin{aligned}H_n(c) &= c \quad \text{for } c \in \{\mathtt{B}, \mathtt{T}, \mathtt{F}, \mathtt{N}\},\\ H_n(\mathtt{In}) &= \begin{cases}\mathtt{T} & \text{if } n = 0,\\ \mathtt{F} & \text{otherwise,}\end{cases}\\ H_n(\mathtt{P}(\phi)) &= \begin{cases}\mathtt{N} & \text{if } n = 0,\\ H_{n-1}(\phi) & \text{otherwise,}\end{cases}\\ H_n(\neg\phi) &= \neg H_n(\phi),\\ H_n(\phi \wedge \psi) &= H_n(\phi) \wedge H_n(\psi).\end{aligned}$$

With these modal conditions we can specify the scheduling of parallel processes. As a first example consider $H_0(aab_{\mathtt{T}} \|_{\Phi} cd)$, where

$$\begin{aligned}\Phi = \ & \mathtt{In} \vee\\ & (\neg\mathtt{P}(\mathtt{In}) \wedge \mathtt{P}^2(\mathtt{In})) \vee\\ & (\neg\mathtt{P}(\mathtt{In}) \wedge \neg\mathtt{P}^2(\mathtt{In}) \wedge \neg\mathtt{P}^3(\mathtt{In}) \wedge \mathtt{P}^4(\mathtt{In})).\end{aligned}$$

The assertion Φ is true in states where the action history length is 0, 2, or 4, and false otherwise (i.e., $H_n(\Phi) = \mathtt{T}$ if $n \in \{0, 2, 4\}$ and $\mathtt{F}$ otherwise). The history operator in cooperation with Φ schedules the process above as an alternation of steps of aab and cd, beginning with aab:

$$\begin{aligned}H_0(aab_{\mathtt{T}} \|_{\Phi} cd) &= H_0(aab_{\mathtt{T}} \mathbin{⫫}_{\Phi} cd +_{\Phi} cd_{\mathtt{T}} \mathbin{⫫}_{\neg\Phi} aab)\\ &= a \cdot H_1(ab_{\mathtt{T}} \mathbin{⫫}_{\Phi} cd +_{\Phi} cd_{\mathtt{T}} \mathbin{⫫}_{\neg\Phi} ab)\\ &\ \vdots\\ &= acadb.\end{aligned}$$

Note that in this case, the conditional merge excludes communication.

We now consider the scheduling of infinite processes using recursively defined conditions. As an example, let

$$\Phi_{even} = \mathtt{In} \vee \neg\mathtt{P}(\Phi_{even}).$$

Thus Φ_{even} will be true if the action history length is even, and false otherwise. It easily follows that

$$H_0(a^*\delta_{\mathtt{T}} \|_{\Phi_{even}} b^*\delta) = (ab)^*\delta. \tag{14}$$

Of course, Φ_{even} can also be used for finite processes, e.g., $H_0(aab_{\mathtt{T}} \|_{\Phi_{even}} cd) = acadb$.

For an example on scheduling that exploits also communication, consider the parallel composition of the processes

$$\begin{aligned}S &= \left(\sum_d r_1(d) \cdot s_2(d)\right)^* \delta,\\ R &= \left(\sum_d r_2(d) \cdot s_3(d)\right)^* \delta.\end{aligned}$$

The intended scheduling 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. So the requirement of scheduling is that if the action history length modulo 3 is 0, the sender S should execute, if it is 1 a communication $r_2(d) \mid s_2(d)$ should take place, and otherwise the receiver R may perform an action. Therefore we consider $H_0(S_\Theta \|_\Psi R)$ with the conditions Θ and Ψ defined as follows:

$$\Psi = \mathtt{In} \vee (\neg\mathtt{P}(\mathtt{In}) \wedge \neg\mathtt{P}^2(\mathtt{In}) \wedge \mathtt{P}^3(\Psi)),$$
$$\Theta = \mathtt{In} \vee \neg\mathtt{P}(\Psi).$$

So Ψ is true if the action history length is a multiple of 3 and false otherwise, and Θ is false if this length modulo 3 is 1 and true otherwise. In particular, the intended communications (data transmissions) can occur along channel 2 and it is not hard to show that for $k \in \mathbb{N}$,

$$H_{3k}(S_\Theta \|_\Psi R) = \left(\sum_d r_1(d) \cdot (r_2(d) \mid s_2(d)) \cdot s_3(d)\right) \cdot H_{3k+3}(S_\Theta \|_\Psi R).$$

So, we find that $H_0(S_\Theta \|_\Psi R)$ describes the intended scheduling.

The setting described in this section provides an abstract level for reasoning about questions how scheduling can be specified. For example, to what extent can scheduling be specified in the components of a parallel composition, and to what extent in some scheduling operator. Such questions are of relevance in the modeling and analysis of multithreading in programming languages as Java [9], and also in theoretical accounts about operators for scheduled interleaving [6]. We give some simple examples. First, consider

$$H_0((\Phi_{even} :\rightarrow a)^*\delta \parallel (\neg\Phi_{even} :\rightarrow b)^*\delta). \tag{15}$$

Observe that in this case, scheduling is steered from within the component processes. Furthermore, observe that if no communications are defined (so $a \mid b = \delta$), the behavior of (15) equals that in (14) (where scheduling was completely determined by the conditional merge). Our next example sketches a naive form of *fair interleaving*: let some large, even number N be given and let the recursive condition Ξ be defined such that

$$H_n(\Xi) = \begin{cases} \mathtt{T} & \text{if } n \bmod N = 0, \\ \mathtt{F} & \text{if } n \bmod N = \frac{N}{2}, \\ \mathtt{B} & \text{otherwise.} \end{cases}$$

Then in e.g. $H_0(a^*\delta_{\mathtt{T}} \|_\Xi b^*\delta)$ it is guaranteed that at each point in execution, each of the component processes gets a turn within N steps.

Finally, we note that the minimal history operator stems from [8], where it supports Minimal History Logic (MHL). In that paper also Action History Logic (AHL) is defined; this logic comprises conditions on the identity of the last action executed and a so-called action history operator. Incorporating such features would admit various forms of *dynamic scheduling* in which actions may influence the control of execution.

6. Conclusion

We have generalized ACP by conditional operators over Belnap's logic: conditional composition characterizes choice and deadlock, and conditional merge models several kinds of parallel composition. In particular we have shown how the truth value B (both) corresponds to nondeterministic choice. Thus we have added a logical perspective on the nature of choice in ACP. Moreover we introduced a setting in which the conditional merge can be used to model and analyze various kinds of parallel scheduling.

Acknowledgments

We thank the referees for their valuable reviews.

References

[1] L. Aceto, W.J. Fokkink, R.J. van Glabbeek, A. Ingólfsdóttir, Axiomatizing prefix iteration with silent steps, Inform. and Comput., 127 (1) (1996) 26–40.

[2] N.D. Belnap, A Useful Four-Valued Logic, in: J.M. Dunn, G. Epstein (Eds.), Modern Uses of Multiple-Valued Logic, D. Reidel, 1977, pp. 8–37.

[3] J.A. Bergstra, I. Bethke, A. Ponse, Process algebra with iteration and nesting, The Computer Journal 37 (4) (1994) 243–258.

[4] J.A. Bergstra, I. Bethke, P.H. Rodenburg, A propositional logic with 4 values: true, false, divergent and meaningless, J. Appl. Non-Classical Logics 5 (2) (1995) 199–218.

[5] J.A. Bergstra, J.W. Klop, Process algebra for synchronous communication, Inform. and Control 60 (1/3) (1984) 109–137.

[6] J.A. Bergstra, C.A. Middelburg, A thread algebra with multi-level strategic interleaving, in: S.B. Cooper, B. Loewe, L. Torenvliet (Eds.), CiE 2005, LNCS, vol. 3526, Springer-Verlag, 2005, pp. 35–48.

[7] J.A. Bergstra, A. Ponse, Kleene's three-valued logic and process algebra, Inform. Process. Lett. 67 (2) (1998) 95–103.

[8] J.A. Bergstra, A. Ponse, Process algebra with four-valued logic, J. Appl. Non-Classical Logics 10 (1) (2000) 27–53.

[9] G. Bracha, et al., The Java Language Specification, 2nd ed., Addison Wesley, 2000.

[10] E.W. Dijkstra, Cooperating sequential processes, in: F. Genuys (Ed.), Programming Languages, Academic Press, New York, 1968, pp. 43–112.

[11] M.C. Fitting, Kleene's three valued logics and their children, Fund. Inform. 20 (1994) 113–131.

[12] R.J. van Glabbeek, A complete axiomatization for branching bisimulation congruence of finite-state behaviours, in: A.M. Borzyszkowski, S. Sokolowski (Eds.), Proceedings 18th Symposium on Mathematical Foundations of Computer Science (MFCS'93), LNCS, vol. 711, Springer-Verlag, 1993, pp. 473–484.

[13] C.B. Jones, Systematic Software Development using VDM, 2nd ed., Prentice-Hall International, Englewood Cliffs, 1990.

[14] S.C. Kleene, On a notation for ordinal numbers, J. Symbolic Logic 3 (1938) 150–155.

[15] S.C. Kleene, Representation of events in nerve nets and finite automata, Automata Studies, Princeton University Press, 1956, pp. 3–41.

[16] A. Ponse, M.B. van der Zwaag, The logic of ACP. Report SEN-R0207, CWI, 2002. Also appeared in: M.B. van der Zwaag, Models and Logics for Process Algebra, Ph.D. thesis, IPA Dissertation Series 2002-11, ISBN 90-5170-636-7, 2002.

[17] A. Ponse, M.B. van der Zwaag, ACP and Belnap's Logic, in: Short Contributions from the Workshop on Algebraic Process Calculi: The First Twenty Five Years and Beyond, Bertinoro, Italy, August 2005. BRICS Notes Series NS-05-3, 2005.

[18] A. Ponse, M.B. van der Zwaag, Belnap's logic and conditional composition, to appear in TCS.

[19] C. Verhoef, A congruence theorem for structured operational semantics with predicates and negative premises, Nordic Journal of Computing 2 (2) (1995) 274–302.