Kleene’s three-valued logic and process algebra

Jan A. Bergstra a,b,1, Alban Ponse a,*

a University of Amsterdam, Programming Research Group, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
b Utrecht University, Department of Philosophy, Heidelberglaan 8, 3584 CS Utrecht, The Netherlands

Received 23 December 1997
Communicated by H. Ganzinger

Abstract

We propose a combination of Kleene’s three-valued logic and ACP process algebra via the guarded command construct. We present an operational semantics in SOS-style, and a completeness result. © 1998 Elsevier Science B.V. All rights reserved.

Keywords: Process algebra; Three-valued logic; Guarded command; Design of algorithms; Concurrency; Formal languages

1. Introduction

In considering algorithms or programs in an operational manner, there is ample motivation to include a third truth value next to T (true) and F (false). For some illustrative references, see, e.g., [4,13]. Evaluation of the condition in a conditional construct, such as \( \phi \) in

if \( \phi \) then \( P \) else \( P \),

for some program \( P \) may turn out divergent, or be distinguished as meaningless (e.g., a type clash, or division by zero). In such a case one certainly does not want to consider \( P \) and if \( \phi \) then \( P \) else \( P \) as equal. Typically, the principle of the excluded middle— tertium non datur—is not anymore acceptable. Of course, if \( \phi \) then \( P \) else \( P \) and if \( \neg \phi \) then \( P \) else \( P \) should be considered the same.

In this paper we view process expressions with conditions as a vehicle to describe concurrent algorithms, and consider the question how to deal with a third truth value \( D \), expressing divergence. This value is inspired by Kleene [15], in which it is called undefined, and is used to reason about partial recursive predicates being either undefined, true, or false. We rather use ‘divergence’ instead of ‘undefined’, as for example a type clash in a program is a kind of undefinedness that we want to distinguish from divergence. Naturally, \( \neg D = D \), for divergence in the evaluation of a condition also implies divergence of its negation (cf. \( \phi \) in if \( \phi \) then \( P \) else \( P \) and if \( \neg \phi \) then \( P \) else \( P \)).

We shortly recall the combination of process algebra and logic via the guarded command, an operation which stems from [11], and was introduced in process algebra with two-valued logic in [2] with the following typical laws where \( \phi \rightarrow \_ \) is the guarded command resembling if \( \phi \) then \( \_ \):

\[
\begin{align*}
T & \rightarrow x = x, \\
F & \rightarrow x = \delta, \\
\phi & \rightarrow x + \psi \rightarrow x = \phi \lor \psi \rightarrow x.
\end{align*}
\]

Here + denotes ‘choice’, and \( \delta \) denotes ‘inaction/deadlock’. The constant \( \delta \) is well known in ACP based
approaches [6,7,10], and is axiomatized by
\[ x + \delta = x \quad \text{“inaction is not considered an alternative,} \]
\[ \delta \cdot x = \delta \quad \text{... and is perpetual”}. \]

Here \( \cdot \) represents “sequential composition”. We involve the constant \( D \) with the axiom
\[ D : \rightarrow x = \delta. \]

This preserves the three laws mentioned above in the present three-valued setting. Roughly, the idea is that if evaluation of a condition diverges, there is no point in considering it in the presence of an alternative, whereas it implies deadlock in case there are no alternatives. Now consider the derivations
\[
\begin{align*}
x &= x + \delta \\
   &= T :\rightarrow x + D :\rightarrow x \\
   &= T \lor D :\rightarrow x,
\end{align*}
\]
\[
\begin{align*}
\delta &= \delta + \delta \\
   &= F :\rightarrow x + D :\rightarrow x \\
   &= F \lor D :\rightarrow x.
\end{align*}
\]

Clearly, the interpretation \( D :\rightarrow x = \delta \) leads to the logical consequence
\[ T \lor D = T, \]

and leaves only two options for the definition of \( F \lor D \), namely: \( F \lor D \in \{ F, D \} \). The only reasonable one seems \( F \lor D = D \).\(^2\) So we end up with \( \neg, \lor \) and its dual \( \land \) as defined by the following truth tables:
\[
\begin{array}{c|c|c|c|c|c|c}
\hline
x & \neg x & \lor & T & F & D \\
\hline
T & F & T & T & T \\
F & T & T & F & F \\
D & D & T & F & F \\
\hline
\end{array}
\]

This precisely entails Kleene’s three-valued logic as defined in [15], which we further call \( \mathcal{K}_3 \). (Notice that \( \mathcal{K}_3 \) is not functionally complete: one cannot define \( f \) with \( f(D) = F \) and \( f(v) = T \) for \( v \in \{ T, F \} \).)

\(^2\) By duality, the other option implies \( T \land D = T \), which indeed seems a rather implausible interpretation of \( \land \).

Structure of the paper: In the next section we shortly discuss \( \mathcal{K}_3 \). In Section 3 we combine this extension with ACP. In the next two sections we define an operational semantics and bisimulation equivalence, and we prove a completeness result.

2. Kleene’s three-valued logic with propositions

Consider Kleene’s three-valued logic \( \mathcal{K}_3 \) as introduced in the previous section (cf. [15,3]). An equational specification of \( \mathcal{K}_3 \) follows from [14], and is given in Table 1. As usual, \( \land \) and \( \lor \) are commutative and associative operations. In case we use proposition symbols from set \( P \), we shall write \( \mathcal{K}_3(P) \), and for concise notation we shall identify \( \mathcal{K}_3 \) and \( \mathcal{K}_3(\emptyset) \).

Let \( T^0_3 = \{ T, F, D \} \). In the following we describe a prototypical, generic occurrence of \( D \), starting from considerations that also apply to a two-valued setting. Consider the natural numbers
\[ \omega = \{ 0, S(0), S(S(0)), \ldots \}. \]

and write \( S^0(x) = x \) and \( S^{k+1}(x) = S(S^k(x)) \). Let \( f : \omega \rightarrow T^0_3 \) be some arbitrary function. We define infinitary \( f \)-disjunction, notation \( \lor f \), by
\[ \lor f = f(0) \lor (f \circ S). \]

The recursive definition of \( \lor f \) implies computation of \( f(0), f(S(0)), f(S^2(0)), \ldots \) until \( f(n) = T \) for some value \( n \). In the particular case that for all \( n \in \omega \), \( f(n) = F \), it makes sense to define \( \lor f = D \). We apply this idea in the following example.

Example 2.1. We define equality \( \equiv : \omega \times \omega \rightarrow T^0_3 \) as a binary infix function by
\[
\begin{align*}
0 &\equiv 0 = T, \\
0 &\equiv S(x) = F, \\
S(x) &\equiv 0 = F, \\
S(x) &\equiv S(y) = x \equiv y.
\end{align*}
\]

Next, we define the partial predecessor function \( \text{pprd} : \omega \rightarrow \omega \) using auxiliary function \( g : \omega \times \omega \rightarrow \omega \)
\[
\text{pprd}(x) = g(x, 0),
\]
\[
g(x, y) = \begin{cases} 
  y & \text{if } S(y) \equiv x, \\
  g(x, S(y)) & \text{otherwise}.
\end{cases}
\]
One easily sees that

\[ \text{pprd}(S^{k+1}(x)) \equiv S^k(x). \]

Now consider the case of \( \text{pprd}(0) \). To model its computation, we define an auxiliary predicate \( \text{Aux} \) as follows:

\[ \text{Aux}(x, y, z) \leftrightarrow g(x, y) \equiv z. \]

The recursive definition of \( \text{Aux} \) follows easily from that of \( g \), and falls within \( K_3(\mathbb{P}) \):

\[ \text{Aux}(x, y, z) = (S(y) \equiv x \land y \equiv z) \]

\[ \lor \]

\[ (\neg(S(y) \equiv x) \land \text{Aux}(x, S(y), z)). \]

In particular, \( \text{Aux}(0, 0, z) \) models computation of \( \text{pprd}(0) \). We have

\[ \text{Aux}(0, 0, z) = (S(0) \equiv 0 \land 0 \equiv z) \]

\[ \lor \]

\[ (\neg(S(0) \equiv 0) \land \text{Aux}(0, S(0), z)). \]

By \( T \land x = x \) and \( S(x) \equiv 0 = F \), it follows that

\[ \text{Aux}(0, 0, z) = (S(0) \equiv 0 \land 0 \equiv z) \]

\[ \lor \]

\[ (S^2(0) \equiv 0 \land S(0) \equiv z) \]

\[ \lor \]

\[ (S^3(0) \equiv 0 \land S^2(0) \equiv z) \]

\[ \lor \]

\[ \ldots \]

so, if \( f = \lambda x . (S(x) \equiv 0 \land x \equiv z) \), we find

\[ \text{Aux}(0, 0, z) = \lor f. \]

Furthermore, we have for each \( n \) that \( f(n) = F \) by axiom \( S(x) \equiv 0 = F \). Hence

\[ \text{Aux}(0, 0, z) = D, \]

and thus \( g(0, 0) \equiv z = D \). The assumption that

\[ \text{pprd}(0) = g(0, 0) \]

can be computed to some value \( z \) leads to value \( D \) of the predicate modeling its computation, irrespective of \( z \). This motivates the following definitions:

\[ \text{pprd}(0) = D, \]

\[ \omega_D = \omega \cup \{D\}, \]

so \( \text{pprd} : \omega \rightarrow \omega_D \). In order to integrate this example with process algebra, we extend the domains of all defined functions to \( \omega_D \) by taking

\[ S(D) = D, \]

\[ D \equiv x = x \equiv D = D, \]

\[ \text{pprd}(D) = D. \]

We continue with this example after having combined \( K_3(\mathbb{P}) \) with process algebra.

3. Process algebra with \( K_3(\mathbb{P}) \)

In the left column of Table 2 we present a slight modification of \( ACP(A, y) \), the Algebra of Communicating Processes [6,7,10]. Here \( A \) is a set of atomic actions, and \( y \) a communication function that is commutative and associative. We take \( y \) total on \( A \times A \rightarrow A_3 \), where \( A_3 = A \cup \{\delta\} \), and the communication merge (commutative (CMC) (by which (CM6) and (CM9), the symmetric variants of (CM5) and (CM8) [10], become derivable). In the right column additional axioms on pre-abstraction (\( t_I \), i.e., renaming of all actions in \( I \) to action \( I \)), and guarded command are listed, where \( \phi \) is taken from \( K_3(\mathbb{P}) \). These axioms are parameterized by action set \( A_I = A \cup \{x\} \). We mostly suppress the...
in process expressions, and brackets according to the following rules: \( \cdot \) binds strongest, \( 
rightarrow \) binds stronger than \( \| \), \( | \), all of which in turn bind stronger than \( + \).

We use

\[
\text{ACP}_D(A_t, \gamma, \mathcal{P})
\]

both to refer to this axiom system and the signature thus defined. We write

\[
\text{ACP}_D(A_t, \gamma, \mathcal{P}) + \mathcal{K}_3(\mathcal{P}) \vdash x = y
\]
or shortly \( \vdash x = y \), if \( x = y \) follows from the axioms of \( \text{ACP}_D(A_t, \gamma, \mathcal{P}) \) and \( \mathcal{K}_3(\mathcal{P}) \). The following derivabilities turn out to be useful:

\[\begin{array}{ll}
(\text{A1}) & x + (y + z) = (x + y) + z \\
(\text{A2}) & x + y = y + x \\
(\text{A3}) & x + x = x \\
(\text{A4}) & (x + y)z = xz + yz \\
(\text{A5}) & (xy)z = x(yz) \\
(\text{A6}) & x + \delta = x \\
(\text{A7}) & \delta x = \delta \\
(\text{CF1}) & a \parallel b = \gamma(a, b) \quad \text{if } a, b \in A_t \\
(\text{CF2}) & a \parallel \delta = \delta \\
(\text{CM1}) & x \parallel y = (x \parallel x + y \parallel x) + x \parallel y \\
(\text{CM2}) & a \parallel x = ax \\
(\text{CM3}) & ax \parallel y = a(x \parallel y) \\
(\text{CM4}) & (x + y) \parallel z = x \parallel z + y \parallel z \\
(\text{CMC}) & x \parallel y = y \parallel x \\
(\text{CM5}) & ax \parallel b = (a \parallel b)x \\
(\text{CM7}) & ax \parallel by = (a \parallel b)(x \parallel y) \\
(\text{CM8}) & (x + y) \parallel z = x \parallel z + y \parallel z \\
(\text{D1}) & \partial_H(a) = a \quad \text{if } a \notin H \\
(\text{D2}) & \partial_H(a) = \delta \quad \text{if } a \in H \\
(\text{D3}) & \partial_H(x + y) = \partial_H(x) + \partial_H(y) \\
(\text{D4}) & \partial_H(xy) = \partial_H(x)\partial_H(y)
\end{array}\]

\[
\begin{array}{ll}
(\text{GT}) & T \rightarrow x = x \\
(\text{GF}) & F \rightarrow x = \delta \\
(\text{GD}) & D \rightarrow x = \delta \\
(\text{GC1}) & \phi \rightarrow x + \psi \rightarrow x = \phi \lor \psi \rightarrow x \\
(\text{GC2}) & \phi \rightarrow x + \phi \rightarrow y = \phi \rightarrow (x + y) \\
(\text{GC3}) & (\phi \rightarrow x)y = \phi \rightarrow xy \\
(\text{GC4}) & \phi \rightarrow (\psi \rightarrow x) = \phi \land \psi \rightarrow x \\
(\text{GC5}) & \phi \rightarrow x \parallel y = \phi \rightarrow (x \parallel y) \\
(\text{GCD}) & \phi \rightarrow x \parallel \psi \rightarrow y = \phi \land \psi \rightarrow (x \parallel y) \\
(\text{T1}) & t_1(\phi \rightarrow x) = \phi \rightarrow t_1(x) \\
(\text{T2}) & t_1(\phi \rightarrow t_1(x)) = \phi \rightarrow t_1(x) \\
(\text{T3}) & t_1(x + y) = t_1(x) + t_1(y) \\
(\text{T4}) & t_1(xy) = t_1(x)t_1(y)
\end{array}\]

**Lemma 3.1.**

1. \( \text{ACP}_D(A_t, \gamma, \mathcal{P}) + \mathcal{K}_3(\mathcal{P}) \vdash \phi \rightarrow \delta = \delta. \)
2. \( \text{ACP}_D(A_t, \gamma, \mathcal{P}) + \mathcal{K}_3(\mathcal{P}) \vdash \phi \rightarrow x = \phi \lor D \rightarrow x. \)

**Proof.** As for (1), \( \phi \rightarrow \delta = \delta \rightarrow \delta + T \rightarrow \delta = \delta \lor T \rightarrow \delta = \delta. \)

As for (2), \( \phi \rightarrow x \rightarrow x = \phi = \phi \rightarrow x + D \rightarrow x = \phi \lor D \rightarrow x. \)

We end this section by using the functions defined in Example 2.1 in a process algebraic setting.
Example 3.2. Recall the data type \( o \), and consider the following counter-like process with parameter in \( o \):

\[
C(x) = r(up) \cdot C(S(x)) + r(down) \cdot C(pprd(x)) + r(\text{set-zero}) \cdot C(0) + x \equiv 0 \Rightarrow r(\text{is-zero}) \cdot C(x).
\]

Here, action \( r(up) \) models “receive command to increase”, action \( r(down) \) represents “receive command to decrease”, action \( r(\text{set-zero}) \) can be used to reset the counter to \( C(0) \), and action \( r(\text{is-zero}) \) indicates that the counter value equals 0. We find:

\[
\begin{align*}
C(D) &= r(up) \cdot C(D) + r(down) \cdot C(D) + r(\text{set-zero}) \cdot C(0), \\
C(0) &= r(up) \cdot C(S(0)) + r(down) \cdot C(D) + r(set-zero) \cdot C(0) + r(is-zero) \cdot C(0), \\
C(S^k(0)) &= r(up) \cdot C(S^{k+2}) + r(down) \cdot C(S^k(0)) + r(set-zero) \cdot C(0).
\end{align*}
\]

Clearly, this modeling is preferred to the case in which \( pprd \) is replaced by \( prd : \omega \rightarrow \omega \) with \( prd(0) = 0 \) and \( prd(S(x)) = x \), which mixes up the number of \( r(down) \) and \( r(up) \) actions in the case of \( C(0) \).

### 4. Operational semantics

In this section we provide \( ACP_D(A_t, \gamma, \mathcal{P}) \) with an operational semantics. Of course this semantics depends on interpretations of the propositions occurring in a process expression.

Assume a (non-empty) set \( \mathcal{P} \) of proposition symbols, and let \( w \) range over the values (interpretations) \( \mathcal{V} \) of \( \mathcal{P} \) in \( \mathbb{T}^k \). In the usual way we extend \( w \) to \( \mathbb{K}_3(\mathcal{P}) \):

- \( w(c) \triangleq c \) for \( c \in \{T, F, D\} \),
- \( w(\neg \phi) \triangleq \neg (w(\phi)) \),
- \( w(\phi \land \psi) \triangleq w(\phi) \land w(\psi) \) for \( \land \in \{\land, \lor\} \).

It follows that if

\[ \models w(\phi) = w(\psi) \]

for all \( w \in \mathcal{V} \), then \( \models \phi = \psi \), and thus \( \vdash \phi = \psi \).

In Table 3 we give axioms and rules that define transitions

\[ \begin{align*}
&\vdash_{w,a} \models \mathcal{X} \mathcal{A} \mathcal{P} \mathcal{D}(A_t, \gamma, \mathcal{P}) \times \mathcal{A} \mathcal{P} \mathcal{D}(A_t, \gamma, \mathcal{P}) \\
&\vdash_{w,a} \mathcal{X} \mathcal{A} \mathcal{P} \mathcal{D}(A_t, \gamma, \mathcal{P})
\end{align*} \]

for all \( w \in \mathcal{V} \) and \( a \in A_t \). Transitions characterize under which interpretations a process expression defines the possibility to execute an atomic action, and what remains to be executed (if anything, otherwise \( \mathcal{X} \) symbolizes successful termination). So, a process expression either resembles deadlock (\( \delta \)), or defines outgoing transitions with labels taken from \( \mathcal{V} \times A_t \).

The axioms and rules in Table 3 yield a structured operational semantics (SOS) based on the work described by Groote and Vaandrager in [12]. In particular, this SOS satisfies the so-called path-format (see Baeten and Verhoef [9]), going with the following notion of bisimulation equivalence:

**Definition 4.1.** Let \( B \subseteq \mathcal{A} \mathcal{P} \mathcal{D}(A_t, \gamma, \mathcal{P}) \times \mathcal{A} \mathcal{P} \mathcal{D}(A_t, \gamma, \mathcal{P}) \). Then \( B \) is a bisimulation if for all \( P, Q \) with \( P \not\equiv Q \) the following conditions hold for all transitions \( P \vdash_{w,a} Q \):  

1. \( \forall P' (P \vdash_{w,a} P' \Rightarrow \exists Q' (Q \vdash_{w,a} Q' \land P'BQ')) \)
2. \( \forall Q' (Q \vdash_{w,a} Q' \Rightarrow \exists P' (P \vdash_{w,a} P' \land P'BQ')) \)
3. \( P \vdash_{w,a} Q \iff Q \vdash_{w,a} Q \)

Two processes \( P, Q \) are bisimilar, notation

\[ P \equiv Q. \]

if there exists a bisimulation \( B \) containing the pair \((P, Q)\).

According to [9], bisimilarity is a congruence relation. It is not difficult to establish with induction on the size of terms that in the bisimulation model thus obtained all equations of Table 2 are true. Hence we conclude:

**Lemma 4.2.** The system \( \mathcal{A} \mathcal{P} \mathcal{D}(A_t, \gamma, \mathcal{P}) + \mathbb{K}_3(\mathcal{P}) \) is sound with respect to bisimulation:

\[ \text{for all } P, Q \in \mathcal{A} \mathcal{P} \mathcal{D}(A_t, \gamma, \mathcal{P}), \quad \mathcal{A} \mathcal{P} \mathcal{D}(A_t, \gamma, \mathcal{P}) + \mathbb{K}_3(\mathcal{P}) \vdash P = Q \implies P \equiv Q. \]
Table 3
Transition rules in path-format.

<table>
<thead>
<tr>
<th>$a \in A_t$</th>
<th>$a \xrightarrow{w,a} \checkmark$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\cdot, \parallel$</td>
<td>$x \xrightarrow{w,a} \checkmark \quad x \xrightarrow{w,a} x'$</td>
</tr>
<tr>
<td></td>
<td>$x \cdot y \xrightarrow{w,a} y \quad x \cdot y \xrightarrow{w,a} x'y$</td>
</tr>
<tr>
<td></td>
<td>$x \parallel y \xrightarrow{w,a} y \quad x \parallel y \xrightarrow{w,a} x' \parallel y$</td>
</tr>
<tr>
<td>$++, \parallel$</td>
<td>$x \xrightarrow{w,a} \checkmark \quad x \xrightarrow{w,a} x'$</td>
</tr>
<tr>
<td></td>
<td>$x + y \xrightarrow{w,a} \checkmark \quad x + y \xrightarrow{w,a} x'$</td>
</tr>
<tr>
<td></td>
<td>$y + x \xrightarrow{w,a} \checkmark \quad y + x \xrightarrow{w,a} x'$</td>
</tr>
<tr>
<td></td>
<td>$x \parallel y \xrightarrow{w,a} y \quad x \parallel y \xrightarrow{w,a} x' \parallel y$</td>
</tr>
<tr>
<td></td>
<td>$y \parallel x \xrightarrow{w,a} y \quad y \parallel x \xrightarrow{w,a} y \parallel x$</td>
</tr>
<tr>
<td>$\mathbin{</td>
<td>}, \mathbin{\parallel}$</td>
</tr>
<tr>
<td></td>
<td>$x \mathbin{</td>
</tr>
<tr>
<td>(Communication)</td>
<td>$x \xrightarrow{w,a} x' \quad y \xrightarrow{w,b} \checkmark \quad a \mathbin{</td>
</tr>
<tr>
<td></td>
<td>$x \mathbin{</td>
</tr>
<tr>
<td></td>
<td>$x \mathbin{</td>
</tr>
<tr>
<td>$\partial H$</td>
<td>$x \xrightarrow{w,a} \checkmark \quad \partial H(x) \xrightarrow{w,a} \partial H(x')$ if $a \notin H$</td>
</tr>
<tr>
<td></td>
<td>$x \xrightarrow{w,a} x' \quad \partial H(x) \xrightarrow{w,a} \partial H(x')$ if $a \notin H$</td>
</tr>
<tr>
<td>$t_1$</td>
<td>$x \xrightarrow{w,a} \checkmark \quad \text{if } a \notin I$</td>
</tr>
<tr>
<td></td>
<td>$t_1(x) \xrightarrow{w,a} t_1(x') \quad \text{if } a \notin I$</td>
</tr>
<tr>
<td></td>
<td>$x \xrightarrow{w,a} x' \quad \text{if } a \in I$</td>
</tr>
<tr>
<td></td>
<td>$t_1(x) \xrightarrow{w,a} t_1(x') \quad \text{if } a \in I$</td>
</tr>
<tr>
<td>$\Rightarrow$</td>
<td>$x \xrightarrow{w,a} \checkmark \quad \phi \Rightarrow x \xrightarrow{w,a} \checkmark \quad \text{if } w(\phi) = \top$</td>
</tr>
<tr>
<td></td>
<td>$x \xrightarrow{w,a} x' \quad \phi \Rightarrow x \xrightarrow{w,a} x' \quad \text{if } w(\phi) = \top$</td>
</tr>
</tbody>
</table>
5. Completeness

In this section we prove completeness of \( \text{ACP}_D(A_t, \gamma, \mathcal{P}) + \mathbb{K}_3(\mathcal{P}) \), i.e.,

\[ P \equiv Q \iff \text{ACP}_D(A_t, \gamma, \mathcal{P}) \vdash P = Q. \]

Our proof is based on a representation of process expressions for which bisimilarity implies derivability in a straightforward way.

**Definition 5.1.** A process expression \( P \in \text{ACP}_D(A_t, \gamma, \mathcal{P}) \) is a basic term if

\[ P = \sum_{i \in I} \phi_i \rightarrow Q_i, \]

where \( \equiv \) is used for syntactic equivalence, \( I \) is a finite, non-empty index set, \( \phi_i \in \mathbb{K}_3(\mathcal{P}) \), and \( Q_i \in \{ \delta, a, aR \mid a \in A_t, R \) a basic term\}.

**Lemma 5.2.** All process expressions in \( \text{ACP}_D(A_t, \gamma, \mathcal{P}) \) can be proved equal to a basic term.

**Proof.** Standard induction on term complexity. \( \square \)

For \( a \in A_t \) and \( \phi \in \mathbb{K}_3(\mathcal{P}) \), the height of a basic term is defined by

\[
\begin{align*}
  h(\delta) &= 0, \\
  h(a) &= 1, \\
  h(\phi \rightarrow x) &= h(x), \\
  h(x + y) &= \max(h(x), h(y)), \\
  h(a \cdot x) &= 1 + h(x).
\end{align*}
\]

**Lemma 5.3.** If \( P \) is a basic term, there is a basic term \( P' \) with \( \vdash P = P' \). \( h(P') \leq h(P) \), and \( P' \) has either the form

\[ \phi \rightarrow \delta, \]

or the form

\[ \sum_{i \in I} \psi_i \rightarrow Q_i \]

with

(i) for all \( i, j \in I \). \( Q_i \neq \delta \), and \( Q_i, Q_j \in A_t \Rightarrow Q_i \neq Q_j \) if \( i \neq j \),

(ii) for each \( i \in I \) there is \( w \in \mathcal{W} \) such that \( w(\psi_i) = T \),

(iii) for no \( i \in I \) and valuation \( w \), \( w(\psi_i) = F \).

**Proof.** Assume

\[ P \equiv \sum_{i=1}^{n} \phi_i \rightarrow Q_i \]

for some \( n \geq 1 \). By Lemma 3.1(1) we may assume that \( Q_i \neq \delta \) for all \( i \in \{ 1, \ldots, n \} \). With (GC1) we easily obtain that each single action occurs at most once. This proves property (i) of the form (2).

Next we consider all summands from \( P \) for which no valuation makes the condition true. For each such summand \( \phi_i \rightarrow Q_i \) it holds that \( \models \phi_i = \phi_i \land D \), and thus \( \models \phi_i = \phi_i \land D \), by which

\[ \vdash \phi_i \rightarrow Q_i = \phi_i \land D \rightarrow Q_i = \phi_i \rightarrow (D \rightarrow Q_i) = \phi_i \rightarrow \delta = \delta. \]

In case all summands can be proved equal to \( \phi_j \rightarrow \delta \) in this way, we are done. In the other case we obtain

\[ \vdash P = \sum_{i=1}^{k} \phi_i \rightarrow Q_i \]

with \( k \leq n \) (and possibly some rearrangement of indices), and for each \( i \in \{ 1, \ldots, k \} \) there is a valuation \( w \) with \( w(\phi_i) = T \). This proves property (ii), and preserves property (i) for \( P \). Finally we define

\[ \psi_i \equiv \phi_i \lor D \]

\[ P' = \sum_{i=1}^{k} \psi_i \rightarrow Q_i. \]

By Lemma 3.1(2) we obtain

\[ \vdash P = P'. \]

By definition of \( \psi_i \) it follows that \( w(\psi_i) \neq F \) for all \( w, i \), which proves property (iii) for \( P' \). (Properties (i) and (ii) are preserved for \( P' \).) \( \square \)

With these two lemma’s we can prove completeness:

**Theorem 5.4.** The system \( \text{ACP}_D(A_t, \gamma, \mathcal{P}) + \mathbb{K}_3(\mathcal{P}) \) is complete with respect to bisimulation.

**Proof.** Let \( P_1 \equiv P_2 \). By soundness, we may assume that both \( P_1 \) and \( P_2 \) satisfy the representation format
defined in Lemma 5.3. We proceed by induction on $h = \max(h(P_1), h(P_2))$.

Case $h = 0$. By Lemma 3.1(1), $\vdash P_n = \delta$ for $n = 1, 2$, so $\vdash P_1 = P_2$.

Case $h > 0$. Let $P_n = \sum_{i \in I_n} \psi_{n,i} :\rightarrow Q_{n,i}$ for $n = 1, 2$, so the $P_n$ satisfy form (2) given in Lemma 5.3. Furthermore, we may assume that for all $i \in I_n$, $Q_{n,i} \setminus Q_{n,j}$ for $j \in I_n \setminus \{i\}$. For the case $Q_{n,i} = aR_{n,j}$ and $Q_{n,j} = aR_{n,j}$ this follows by induction: $R_{n,i} \Leftrightarrow R_{n,j}$ implies $\vdash R_{n,i} = R_{n,j}$, so $\vdash aR_{n,i} = aR_{n,j}$, and thus $(GC1)$ can be applied.

Now each summand of $P_1$ can be proved equal to one in $P_2$, and by Lemma 5.3, each such summand yields a transition for a certain $w \in \mathcal{W}$.

- Assume that $P_1 \xrightarrow{\text{w.a.}} \top$ for some $w,a$. Thus $w(\psi_{1,i}) = \top$ for some unique $i \in I_1$. By $P_1 \equiv P_2$, there must be some unique $j \in I_2$ for which $P_2 \xrightarrow{\text{w.a.}} \top$ and $\vdash \psi_{1,i} = \psi_{2,j}$ (the latter derivability follows from Lemma 5.3 and the non-bisimilarity of different summands). Thus $\vdash \psi_{1,i} :\rightarrow a = \psi_{2,j} :\rightarrow a$.

- Assume that $P_1 \xrightarrow{\text{w.a.}} R_{1,i}$ for some $w,a$ and unique $i \in I_1$. Thus $w(\psi_{1,i}) = \top$. By $P_1 \equiv P_2$, there must be some unique $j \in I_2$ for which $P_2 \xrightarrow{\text{w.a.}} R_{2,j}$ and $R_{1,i} \Leftrightarrow R_{2,j}$, and for which $\vdash \psi_{1,i} = \psi_{2,j}$ follows from Lemma 5.3. By induction we find $\vdash R_{1,i} = R_{2,j}$, and therefore $\vdash aR_{1,i} = aR_{2,j}$ and hence $\vdash \psi_{1,i} :\rightarrow aR_{1,i} = \psi_{2,j} :\rightarrow aR_{2,j}$.

By the derivabilities above and symmetry, $\vdash P_1 = P_2$ quickly follows. □

6. Conclusion

The extension of process algebra with guarded command to a setting with Kleene’s three-valued logic seems a modest one, and can be characterized as giving up the principle of the excluded middle, and hence giving up the identity

$$x = \phi :\rightarrow x + \neg\phi :\rightarrow x,$$

but otherwise no surprising identities arise: $D$ and $F$ often play the same role in guarded commands. This matches with the intuition that a process like

$$(D :\rightarrow a) \parallel bc$$
equals bc\delta. The deadlock, caused by a divergence, is postponed until all alternative behaviour has been executed.

We have argued that divergence arises from considerations about partial predicates (cf. [15]), and can be involved in process algebra by $D :\rightarrow x = \delta$. Of course, in the case that the process of evaluation is prominent in the algorithm represented as a process expression, evaluation rather should be modeled as a process (which possibly diverges) than as a condition.

References


