Formalizing Errors in CCS with 3-Valued Logic

Alessandro Aldini(✉) and Claudio Antares Mezzina(✉)

University of Urbino Carlo Bo, Urbino, Italy
alessandro.aldini@uniurb.it, claudio.mezzina@uniurb.it

Abstract. Concurrent and distributed systems are often prone to failures. Errors in modeling an agent's behavior can propagate into large interacting systems with unexpected consequences. In this paper, we propose a theory for the process algebra CCS enriched with a formal and explicit representation of errors based on McCarthy's style three-valued logic, which includes the traditional Boolean values and a third error value. In this setting, we formally study how the emergence of local errors may or may not result in propagation, as also emphasized in a real-world case study modeling a distributed microservices architecture.

1 Introduction

Concurrent and distributed systems are fundamental in modern computing, ranging from cloud-based infrastructures to safety-critical applications such as autonomous vehicles and industrial automation. These systems frequently encounter faults due to hardware failures, network disruptions, or software bugs. Process algebras, such as the Calculus of Communicating Systems (CCS) [25], the Algebra of Communicating Processes ACP [4], or π-calculus [27], provide a formal framework to model and reason about concurrent and distributed systems. However, they struggle to accurately represent failures. One of the reasons is that they typically rely on standard two-valued logic (T, F), which is insufficient for describing intermediate or indeterminate states caused by faults. For example, in a distributed database, a node failure does not necessarily mean that the entire system fails; some components may continue to function. Many-valued logics have been proposed to investigate situations in which there are more than two truth values that can have several different interpretations [11]. Kleene used a third value to represent predicates for which no algorithm can decide whether they are true or false [20]. Bochvar studied a logic undefined/meaningless value to a predicate whenever any one of its components is undefined or meaningless [7]. McCarthy proposed an analogous logic [24] with a lazy evaluation of those predicates that may include the third value, which is particularly attractive for computing purposes and used in several programming languages, from Lisp to Haskell [28].

Inspired by these logic paradigms, a process-algebraic treatment of errors is provided in [5]. In this work, ACP is extended with a conditional guard construct

Published by springer Nature Switzerland AG 2025
C. Di Giusto and A. Ravara (Eds.): COORDINATION 2025, LNCS 15731, pp. 30–49, 2025.
https://doi.org/10.1007/978-3-031-95589-1_2

and a three-valued logic, where the third value, M for meaningless, stands for the error value. Additionally, the explicit *error* process μ is introduced, and axiomatized by the following axioms:

$$\mu \cdot x = \mu \tag{1}$$
$$\mu + x = \mu \tag{2}$$
$$\mu \parallel P = \mu \tag{3}$$

where $\cdot$ is the sequential composition of ACP, $+$ and $\parallel$ are the nondeterministic choice and the parallel composition, respectively[1]. As one can see from the above axioms, an error μ suppresses (local) continuation and nondeterminism, thus revealing its infectious nature that propagates also to concurrent processes. While this interpretation is reasonable from a logic point of view (as it basically falls into Bochvar's three-valued logic [7]) and is well motivated in centralized systems, it is also worth investigating alternative interpretations where errors are local and not infectious, which is more adequate in distributed environments. More precisely, while the first axiom (1) appears indisputable – consider a sequential C program that encounters a segmentation fault due to an unusual pointer operation, causing the program to terminate – the other two axioms assume that errors are global and infectious. The axiom (2) equates a program that may fail to a failed one. The axiom (3) equates a program running in parallel with an error to a failed program. However, if we consider two threads or actors running in parallel and one fails, the other should continue to exist. Additionally, in a distributed system, an error may occur in a different location from the other process, thus not directly affecting it. Therefore, to accurately model the behavior of concurrent and distributed programs, in this paper, we reject axioms (2)–(3) and partially retain axiom (1) in CCS style as follows:

$$\mu.P = \mu.\mathbf{0}$$

which has, as its semantic counterpart, the rule

$$\mu.P \xrightarrow{\mu} \mathbf{0}$$

giving the intuition that the error is *local*, and that if a program encounters it, then it fails. Based on these considerations, in this paper, we re-interpret the work of [5] in the setting of CCS and revise the axioms for errors from a distributed programming point of view, by trying to answer the following question:

How can we formally model situations where errors are local rather than global?

To achieve this, we extend CCS with (*i*) a conditional guard operator $\varphi :\rightarrow P$, meaning that if φ holds, then the process P is executed, and with (*ii*) a

[1] The axioms for error in ACP [5] are more articulated, due to the different operators ACP has for parallel composition. Here, we have distilled the meaning of these axioms as if they were using the CCS parallel operator.

special error prefix μ indicating a local error. In this way, we can still retain CCS bisimulation and provide a sound axiomatization of errors. Moreover, we maintain the classical HML-based logical interpretation of bisimulation [19].

The rest of the paper is structured as follows. In Sect. 2, we define the logic for expressing the conditions of the guard statement $\varphi :\rightarrow P$. In Sect. 3, we present the syntax of the extended version of CCS with errors, called CCS+e, and its axiomatization. In Sect. 4, we provide the semantics of CCS+e and we show that the classical notion of bisimulation is a congruence, characterizes the soundness of the axiomatization, and is logically characterized by the HML logic. In Sect. 5, we propose a case study based on a microservices architecture. Finally, in Sect. 6, some conclusions terminate the paper.

2 McCarthy Three-Valued Logic

Given a set of atomic propositions *Prop* (ranged over by $p, q \ldots$) and the set of logical values $\mathcal{T} = \{T, F, M\}$ (true, false, and error), we adopt McCarthy's three-valued propositional logic, which is defined by the set of formulas generated by the grammar:

$$\varphi ::= T \mid F \mid M \mid p \mid \neg\varphi \mid \varphi \wedge \varphi$$

where $p \in \mathit{Prop}$, and with the following truth tables (for clarity, we include also disjunction, which can be defined as $p \vee q ::= \neg(\neg p \wedge \neg q)$):

p	$\neg p$
T	F
F	T
M	M

$\wedge$	T	F	M
T	T	F	M
F	F	F	F
M	M	M	M

$\vee$	T	F	M
T	T	T	T
F	T	F	M
M	M	M	M

We call $\mathcal{L}$ the resulting logic, and we denote with $\varphi, \psi, \ldots$ its formulas. We use the same definition of syntactic substitution and the excluded fourth rule as in [5] to define a complete, inequational evaluation system implied by the truth tables above.[2] We then write $\mathcal{L} \models \varphi = \psi$ to state that $\varphi = \psi$ can be proved in such a system.

We point out that the difference from Bochvar's semantics lies in the interpretation of the role of M, which is infectious in Bochvar's logic. More precisely, in Bochvar's logic, $M \circ p$ and $p \circ M$, with $\circ \in \{\wedge, \vee\}$ and $p \in \{T, F\}$, always give M (the two binary operators are commutative). Instead, note that the two McCarthy binary operators are non-commutative whenever errors come into play. In particular, $F \wedge M$ is different from $M \wedge F$ because in the former case, by applying the McCarthy lazy interpretation, the evaluation of F makes the whole statement immediately false without making it necessary to evaluate the following operand (we can reason symmetrically in the case of $T \vee M$).

[2] A sound and complete axiomatization of the three-valued logic is still an open problem [21] and has been recently faced in [9], which provides an equational basis for McCarthy algebras.

$$P ::= \mathbf{0} \mid \alpha.P \mid \varphi :\rightarrow P \mid P + P \mid P \parallel P \mid P\backslash L \mid C \qquad \textbf{(Processes)}$$
$$\alpha ::= a \mid \overline{a} \mid \tau \mid \mu \qquad \textbf{(Prefixes)}$$

Fig. 1. Syntax of CCS+e

As we will see in the next section, the three-valued logic of [5] relies mainly on Bochvar's operators and leverages the lazy semantics of McCarthy's conjunction only to model the sequential composition of conditional guards. Instead, we completely abandon Bochvar's semantics and the infectious behavior of the error value, in order to give it the desired local interpretation.

3 Syntax of CCS+e

In this section, we extend CCS [25] with the special action representing error and a conditional statement based on the three-valued logic of the previous section. An axiomatization for such an extension is then proposed.

Let A be a set of visible action names and $Act = A \cup \{\overline{a} \mid a \in A\} \cup \{\tau\} \cup \{\mu\}$, where τ represents an internal, unobservable action and μ denotes the error action. The syntax of CCS+e is represented in Fig. 1, where $L \subseteq A \cup \{\overline{a} \mid a \in A\}$. In the following, we will use $\pi, \pi', \ldots$ to range over $A \cup \{\overline{a} \mid a \in A\} \cup \{\tau\}$.

Process terms given by the P productions extend classical CCS with one new process: the *conditional* guard $\varphi :\rightarrow \_$. Let us explain briefly all the operators. Term $\mathbf{0}$ represents the idle, terminated process; $\alpha.P$ represents a prefixed process, i.e., a process that has to perform the action α before evolving into P. As we will see when commenting on the semantics, this is not true for the error prefixed process $\mu.P$, which evolves into the terminated process $\mathbf{0}$. A prefix, or an action, can be an input a, an output $\overline{a}$, a silent or internal action τ, and an error action μ. The process $P + Q$ represents a nondeterministic choice between P and Q, that is, a process that can be P or Q. The process $P \parallel Q$ represents the parallel composition of P and Q, where, as usual, actions can be executed asynchronously by the two processes or can synchronize (generating an internal action τ) whenever they are an input a and a corresponding output $\overline{a}$. Some actions in a process P can be forbidden, and this is represented by the process $P\backslash L$, where L is the set of restricted actions. C represents a process constant, and to this end, we assume the existence, for each C, of a constant definition $C ::= P$. This is used to model recursive processes.

The intuition behind the conditional guard operator $\varphi :\rightarrow P$ is that it acts like the command `if` φ `then` P:

$$\texttt{if } \varphi \texttt{ then } P ::= \varphi :\rightarrow P$$

where the evaluation of φ may lead to three different values. If φ evaluates to T, then the process behaves like P. If φ evaluates to F, then the process behaves

like **0**. If φ evaluates to M, then an error is generated and, again, the process behaves like **0**. Also, one can derive an `if-then-else` operator in this way:

$$\texttt{if } \varphi \texttt{ then } P \texttt{ else } Q ::= (\varphi :\rightarrow P) + (\neg\varphi :\rightarrow Q)$$

This is possible due to the fact that if φ evaluates to M, then, according to McCarthy's logic, we have that $M = \neg M$.

3.1 Axiomatization of CCS+e

We base our axioms system – see Table 1 – on the classical equational theory for CCS, by extending it with some ideas inherited from [5]. However, differently from [5], we do not have axioms like

$$\mu.P + Q = \mu.P \tag{M0}$$

$$\varphi_1 :\rightarrow P + \varphi_2 :\rightarrow P = (\varphi_1 \vee \varphi_2) :\rightarrow P \tag{G0}$$

Both of them would imply that errors suppress nondeterminism. In particular, if we accept M0, by virtue of the expansion law mapping parallel composition to alternative choice and prefix, we would have that local errors infect (and suppress the continuation of) the whole global system. In other words, if P causes an error in $P \parallel Q$, then also Q would fail, independently of the mutual relation between P and Q. In the case of G0, on the right-hand side, we have that if φ_1 evaluates to M and φ_2 evaluates to T, then $(\varphi_1 \vee \varphi_2)$ evaluates to M, thus suppressing P, while on the left-hand side, the nondeterministic choice is not expected to suppress the potential execution of P.[3]

Apart from the exceptions above, most of the axioms can be inherited from the classical axiomatization of CCS and from [5] in the case of the conditional statement. In particular, we emphasize the axiom introducing the error action:

$$M :\rightarrow P = \mu.\mathbf{0} \tag{G3}$$

to state that the error-value M represents a guard that, in a conditional statement, causes the execution of the error action and the subsequent termination.

However, errors cannot be treated as a normal prefix because of the local interpretation we give to them. In fact, as emphasized above, errors should not propagate in the setting of concurrent processes. We show through the following example that the classical expansion law for parallel composition would not obey our interpretation of local error.

[3] In [5], G0 is based on Bochvar's interpretation of $\vee$. Note that the suppression of nondeterminism in this axiom occurs regardless of whether one uses Bochvar's or McCarthy's semantics, or any other deterministic three-valued semantics for this operator.

Example 1. Consider the classical CCS expansion law and the derivation:

$$\mu.P \parallel a.\mathbf{0} = \mu.(P \parallel a.\mathbf{0}) + a.(\mu.P \parallel \mathbf{0}) = \mu.\mathbf{0} + a.\mu.P$$

which ends up in a process that may fail or do an action a and then have an internal error. Instead, we would expect that if the left-hand process does terminate with an error, then the right-hand one should not fail as well without executing the action a. Note that a local error could still have a negative impact on other concurrent processes, e.g., in the case where the left-hand process is $\mu.\bar{a}.P$, which would inhibit the synchronization between the two processes.

This example reveals that we have to consider the environment in which an error occurs because we must guarantee the continuation of concurrent processes even in the presence of a local error that should not affect their execution. In particular, in the example above, the failure of the left-hand process should not impact its environment represented by the right-hand process. To this aim, we decorate μ with a subscript denoting the environment in which it occurs, by stating as default the condition

$$\mu.P = \mu_{\mathbf{0}}.P \tag{M1}$$

Hence, we extend the syntax of α with the production $\alpha ::= \mu_P$. For analogous reasons, we also extend the conditional operator in the same way by introducing the term $\varphi :\rightarrow_Q P$, with the default condition

$$\varphi :\rightarrow P = \varphi :\rightarrow_{\mathbf{0}} P \tag{G4}$$

As we will see, the environment of an error will be managed when dealing with the parallel composition operator in the expansion law, as emphasized in the following motivating example.

Example 2. Revisiting Example 1, we would expect a derivation like the following one:

$$\mu.P \parallel a.\mathbf{0} = \mu_{\mathbf{0}}.P \parallel a.\mathbf{0} = \mu_{\mathbf{0}\parallel a.\mathbf{0}}.\mathbf{0} + a.\mu_{\mathbf{0}}.P = \mu_{a.\mathbf{0}}.\mathbf{0} + a.\mu_{\mathbf{0}}.P$$

where $\mu_{a.\mathbf{0}}.\mathbf{0}$ denotes that after the local error of the left-hand process, the environment can still behave as $a.\mathbf{0}$, while $a.\mu_{\mathbf{0}}.P$ denotes that after the local action a, the local error of the left-hand process occurs, suppressing any continuation, as its reference environment is $\mathbf{0}$.

We now complete the discussion on the remaining axioms of Table 1. Axioms C1-C4 are the classical axioms for the choice operator, while axioms P1-P3 are the classical axioms for the parallel operator.

Axioms G1 and G2 specify that the conditional guard T is transparent and the conditional statement guarded by F collapses to the terminated process $\mathbf{0}$. Axiom G5, inherited from [5], states that the choice between terms guarded by the same formula can be postponed after the evaluation of the formula. Axiom

G6 is about the sequential composition of conditional guards and provides the motivation for using McCarthy's semantics for conjunction. In particular, if φ_1 turns out to be false in $\varphi_1 :\rightarrow (\varphi_2 :\rightarrow P)$, then the process terminates regardless of the valuation of φ_2. The same behavior is guaranteed by $(\varphi_1 \wedge \varphi_2) :\rightarrow P$ by virtue of McCarthy's truth table of $\wedge$. Hence, we can say that for the evaluation of formulas, we adopt lazy semantics.

Example 3. By using the `if-then-else` construction of the previous section, we can now show how a complex conditional statement with nesting is managed through the axioms of CCS+e:

$$\begin{aligned} &\texttt{if } \varphi_1 \texttt{ then } P_1 \texttt{ else if } \varphi_2 \texttt{ then } P_2 \texttt{ else } P_3 ::= \\ &(\varphi_1 :\rightarrow P_1) + (\neg\varphi_1 :\rightarrow (\varphi_2 :\rightarrow P_2 + \neg\varphi_2 :\rightarrow P_3)) =_{G5} \\ &\varphi_1 :\rightarrow P_1 + \neg\varphi_1 :\rightarrow \varphi_2 :\rightarrow P_2 + \neg\varphi_1 :\rightarrow \neg\varphi_2 :\rightarrow P_3 =_{G6} \\ &\varphi_1 :\rightarrow P_1 + (\neg\varphi_1 \wedge \varphi_2) :\rightarrow P_2 + (\neg\varphi_1 \wedge \neg\varphi_2) :\rightarrow P_3 \end{aligned}$$

We point out that a counterpart of G5 does not hold for the parallel composition operator, as shown in the following example.

Example 4.

$$(\varphi :\rightarrow P) \parallel (\varphi :\rightarrow Q) = \varphi :\rightarrow (P \parallel Q)$$

is not an axiom of our system because the left-hand process would raise two errors if φ evaluates to M, while the right-hand process would raise only one error under the same assumption.

Axioms H1–H5 deal with the restriction operator in the classical way. Note that H2 applies to non-error actions only and that H4 states that the restriction operator is transparent with respect to the conditional guard in a way analogous to the other operators.

Axiom E1 extends the classical expansion law of CCS by treating the error prefix and the guarded statements in such a way as to consider the reference environment properly. Note that in $P_1 \parallel P_2$, the two processes are represented in normal form as summations[4] of conditional, prefix-guarded terms, to which any sequential process term can be reduced.[5] The rule establishes how, when expanding the parallel operator, the environments associated with the error actions and the conditional guards are updated. Note that, if an error is local to process P_1 and its own reference environment is P'_{1_k} (and no matter the continuation is), then, if executed in the context $\_ \parallel P_2$, it causes local termination and changes its environment to $P'_{1_k} \parallel P_2$. We can reason symmetrically for process P_2. An analogous solution is adopted to manage the conditional statements, as they

[4] I, K, J, H are finite indexing sets (if empty, the summation gives $\mathbf{0}$).

[5] It is sufficient to consider the following interesting cases. If P is in normal form, then, by virtue of axiom G1, the term $\alpha.P$ can be rewritten as the term $T :\rightarrow \alpha.P$, which is in normal form. Analogously, by applying repeatedly G5 and G6, the term $\varphi :\rightarrow P$ can be turned into normal form, similarly as seen in Example 3. Finally, the case of the restriction operator is handled via axioms H1–H5.

Table 1. Axioms system $\mathcal{A}$. Gray-boxed axioms represent the differences with respect to the original CCS axiomatization [25].

M1	$\mu.P = \mu_{\mathbf{0}}.P$
C1	$P + \mathbf{0} = P$
C2	$P + Q = Q + P$
C3	$P + (Q + R) = (P + Q) + R$
C4	$P + P = P$
P1	$P \parallel \mathbf{0} = P$
P2	$P \parallel Q = Q \parallel P$
P3	$P \parallel (Q \parallel R) = (P \parallel Q) \parallel R$
G1	$T :\rightarrow P = P$
G2	$F :\rightarrow P = \mathbf{0}$
G3	$M :\rightarrow P = \mu.\mathbf{0}$
G4	$\varphi :\rightarrow P = \varphi :\rightarrow_{\mathbf{0}} P$
G5	$\varphi :\rightarrow P + \varphi :\rightarrow Q = \varphi :\rightarrow (P + Q)$
G6	$\varphi_1 :\rightarrow (\varphi_2 :\rightarrow P) = (\varphi_1 \wedge \varphi_2) :\rightarrow P$
H1	$\mathbf{0}\backslash L = \mathbf{0}$
H2	$(\pi.P)\backslash L = \mathbf{0}$ if $\pi \in L$
H3	$(\alpha.P)\backslash L = \alpha.(P\backslash L)$ *if* $\alpha \notin L$
H4	$(\varphi :\rightarrow P)\backslash L = \varphi :\rightarrow (P\backslash L)$
H5	$(P + Q)\backslash L = (P\backslash L) + (Q\backslash L)$
E1	$P_1 \parallel P_2 = \sum_{i\in I} \varphi_{1_i} :\rightarrow_{Q_{1_i}\parallel P_2} \pi_{1_i}.(P_{1_i} \parallel P_2) +$ $\sum_{j\in J} \varphi_{2_j} :\rightarrow_{P_1\parallel Q_{2_j}} \pi_{2_j}.(P_1 \parallel P_{2_j}) +$ $\sum_{i\in I, j\in J, \pi_{1_i}=a\wedge\pi_{2_j}=\overline{a}} (\varphi_{1_i} \wedge \varphi_{2_j}) :\rightarrow_* \tau.(P_{1_i} \parallel P_{2_j}) +$ $\sum_{k\in K} \psi_{1_k} :\rightarrow_{Q'_{1_k}\parallel P_2} \mu_{P'_{1_k}\parallel P_2}.\mathbf{0} +$ $\sum_{h\in H} \psi_{2_h} :\rightarrow_{P_1\parallel Q'_{2_h}} \mu_{P_1\parallel P'_{2_h}}.\mathbf{0}$ if $P_1 = \sum_{i\in I} \varphi_{1_i} :\rightarrow_{Q_{1_i}} \pi_{1_i}.P_{1_i} + \sum_{k\in K} \psi_{1_k} :\rightarrow_{Q'_{1_k}} \mu_{P'_{1_k}}.P''_{1_k}$ and $P_2 = \sum_{j\in J} \varphi_{2_j} :\rightarrow_{Q_{2_j}} \pi_{2_j}.P_{2_j} + \sum_{h\in H} \psi_{2_h} :\rightarrow_{Q'_{2_h}} \mu_{P'_{2_h}}.P''_{2_h}$

could generate errors that must take into account the reference environment. The case of synchronization deserves special treatment. Indeed, the resulting τ action must occur if and only if the two conditions guarding the respective input and output actions evaluate to true. In all other cases, at least one of the two synchronizing actions is not available (due to a local error or to a false local

Table 2. Transition rules of CCS+e

$$(\text{ERR}_{\mathbf{0}})\ \mu.P \xrightarrow{\mu} \mathbf{0} \qquad (\text{ERR})\ \mu_P.Q \xrightarrow{\mu} P \qquad (\text{ACT})\ \pi.P \xrightarrow{\pi} P$$

$$(\text{CHO}_l)\ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad (\text{CHO}_r)\ \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$(\text{PAR}_l)\ \frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q} \qquad (\text{PAR}_r)\ \frac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'}$$

$$(\text{SYN}_l)\ \frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\overline{a}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'} \qquad (\text{SYN}_r)\ \frac{P \xrightarrow{\overline{a}} P' \qquad Q \xrightarrow{a} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

$$(\text{RES})\ \frac{P \xrightarrow{\alpha} P'}{P\backslash L \xrightarrow{\alpha} P'\backslash L} \quad \alpha \notin L \qquad (\text{CONST})\ \frac{C ::= P \qquad P \xrightarrow{\alpha} P'}{C \xrightarrow{\alpha} P'}$$

$$(\text{CONDT})\ \frac{[\![\varphi]\!]_w = T \quad P \xrightarrow{\alpha} P'}{\varphi :\rightarrow_{\square} P \xrightarrow{\alpha} P'} \quad \text{where } \square \text{ is empty, any } Q \in \mathcal{P}, \text{ or } *$$

$$(\text{CONDM}_{\mathbf{0}})\ \frac{[\![\varphi]\!]_w = M}{\varphi :\rightarrow P \xrightarrow{\mu} \mathbf{0}} \qquad (\text{CONDM})\ \frac{[\![\varphi]\!]_w = M}{\varphi :\rightarrow_Q P \xrightarrow{\mu} Q}$$

condition) and, therefore, the impossibility of executing the τ action must not produce any effect. This interpretation will be considered by the semantic rule for the conditional statement in the special case $:\rightarrow$ is decorated with $_*$.

To conclude, we use the notation $\vdash P = Q$ to denote that $P = Q$ derives from the axioms system $\mathcal{A}$ of Table 1 and by following the classical rules of equational deduction [18] and the additional rule

$$(\text{EQUIV})\ \frac{\mathcal{L} \models \varphi = \psi}{\mathcal{A} \vdash \varphi :\rightarrow P = \psi :\rightarrow P}$$

4 Semantics of CCS+e

In this section, we present the semantics of CCS+e together with a notion of bisimulation and related properties. In the following, we call $\mathcal{P}$ the set of process terms generated by the grammar of CCS+e extended with the decorated versions of μ and $:\rightarrow$.

First, we introduce the interpretation function for the formulas of our three-valued logic. Let w range over the set $\mathcal{W}$ of valuation functions of *Prop* in $\mathcal{T}$. The semantics of any formula φ of the logic $\mathcal{L}$ with respect to w, denoted $[\![\varphi]\!]_w$, is defined as follows:

$$\begin{aligned}
&[\![c]\!]_w = c \quad \textit{if } c \in \{T, F, M\}\\
&[\![p]\!]_w = w(p)\\
&[\![\neg\varphi]\!]_w = \neg[\![\varphi]\!]_w\\
&[\![\varphi \wedge \psi]\!]_w = [\![\varphi]\!]_w \wedge [\![\psi]\!]_w
\end{aligned}$$

Observe that if $[\![\varphi]\!]_w = [\![\psi]\!]_w$ for all valuation functions w, then $\mathcal{L} \models \varphi = \psi$.

Then, the semantics of (P, w), where $P \in \mathcal{P}$ and $w \in \mathcal{W}$, is the least labeled transition system with initial state P that is built by using the rules of the transition relation $\rightarrow_w \subseteq \mathcal{P} \times Act \times \mathcal{P}$ defined in Table 2 (if not ambiguous, we usually omit w). We now describe each rule in detail. The rule ($\textsc{Err}_{\mathbf{0}}$) expresses the blocking behavior of the error prefix, which does not allow any continuation locally. The rule (Err) has the only objective of dealing properly with the decorated versions of μ resulting from the application of the axioms of $\mathcal{A}$. It is at the base of the soundness of the expansion law. Basically, the error prefixed process $\mu_P.Q$ just terminates the local continuation Q by producing the error label and enabling the continuation for the reference environment P. Rule (Act) transforms a prefix π into a label; rules ($\textsc{Cho}_l$) and ($\textsc{Cho}_r$) deal with the nondeterministic choice. Rules ($\textsc{Par}_l$) and ($\textsc{Par}_r$) allow, respectively, the left/right process of a parallel composition to execute without synchronizing with the other process. On the other hand, rules ($\textsc{Syn}_l$) and ($\textsc{Syn}_r$) allow two processes in parallel to synchronize. Rule (Res) deals with the restriction operator: a label can be produced provided that it is not contained in the restricted set L. Rule (Const) deals with process constant definition. In this way, we can model infinite behaviors. Rule (CondT) deals with a conditional statement with true guard, which allows the guarded process term to be executed. Note that the context $\Box$ in $:\!\rightarrow_{\Box}$ is just discarded, as it is used only in case of error in the valuation of the guard. Rules ($\textsc{CondM}_{\mathbf{0}}$) and (CondM) deal with the conditional statement in the case the guard φ evaluates to M. They are, in a sense, the counterparts of ($\textsc{Err}_{\mathbf{0}}$) and (Err), respectively, as they express how the guarded process term can continue after the error occurring in the valuation of the guard. In line with the discussion regarding the expansion law, rule (CondM) does not cover the case $:\!\rightarrow_*$ because this case refers to the potential execution of a τ-action resulting from a synchronization. If such a τ-action cannot be executed because of an error occurring locally, it cannot be replaced by a (global) error action, otherwise we would count twice the same error.

While in [5], the definition of bisimulation must explicitly distinguish between the terminated process and the error process, in our setting, this distinction is not necessary as we treat μ as an action. Hence, we can rely on the classical CCS bisimulation equivalence.

Definition 1 (Bisimulation). *A relation $\mathcal{R}$ on process terms is a bisimulation if for every evaluation $w \in \mathcal{W}$, it holds that for all process terms P, Q with $P\,\mathcal{R}\,Q$, for all $\alpha \in Act$, whenever*

- *$P \xrightarrow{\alpha}_w P'$ implies that $Q \xrightarrow{\alpha}_w Q'$ and $P'\,\mathcal{R}\,Q'$;*
- *$Q \xrightarrow{\alpha}_w Q'$ implies that $P \xrightarrow{\alpha}_w P'$ and $P'\,\mathcal{R}\,Q'$.*

The largest bisimulation is called bisimilarity and noted with $\sim$.

In analogy with CCS, bisimulation equivalence is a congruence as stated by the following theorem.

Theorem 1 (Congruence). *Let $P, Q \in \mathcal{P}$. If $P \sim Q$, then the following hold:*

1. $\alpha.P \sim \alpha.Q$ *for all* $\alpha \in Act$,
2. $\mu_P.R \sim \mu_Q.R'$ *for all* $R, R' \in \mathcal{P}$,
3. $\varphi :\rightarrow_R P \sim \varphi :\rightarrow_R Q$ *for all* $\varphi \in \mathcal{L}$, *and* $R \in \mathcal{P}$ *or empty*,
4. $\varphi :\rightarrow_P R \sim \varphi :\rightarrow_Q R$ *for all* $R \in \mathcal{P}$,
5. $P + R \sim Q + R$ *for all* $R \in \mathcal{P}$,
6. $P \parallel R \sim Q \parallel R$ *for all* $R \in \mathcal{P}$,
7. $P\backslash L \sim Q\backslash L$ *for all* $L \subseteq Act\backslash\{\mu, \tau\}$.

Proof. We show the gray-boxed cases, while the others derive from the standard results for CCS. Assume $\mathcal{R}$ is a bisimulation such that $(P, Q) \in \mathcal{R}$. Regarding case 2, it is sufficient to consider $\mathcal{R}' = \{(\mu_P.R, \mu_Q.R') \mid R, R' \in \mathcal{P}\} \cup \mathcal{R}$ and observe that, by virtue of the rule ERR and of $(P, Q) \in \mathcal{R}$, it is a bisimulation. For case 3, the relation to consider is $\mathcal{R}' = \{(\varphi :\rightarrow_R P, \varphi :\rightarrow_R Q)\} \cup \mathcal{R}$. Then, if $[\![\varphi]\!] = T$, the result derives by virtue of the rule CONDT and of $(P, Q) \in \mathcal{R}$. If $[\![\varphi]\!] = M$ then the result derives by virtue of the rules $\text{CONDM}_{\mathbf{0}}$ or CONDM, depending on R. For case 4, we can reason analogously to the previous case by considering $\mathcal{R}' = \{(\varphi :\rightarrow_P R, \varphi :\rightarrow_Q R)\} \cup \mathcal{R}$. □

We now show that the system presented in Sect. 3.1 (Table 1) is sound with respect to $\sim$ in the case of finite processes (no occurrence of constants). The most interesting case of Table 1 is represented by the expansion law. In Fig. 2, we show an example emphasizing that the classical expansion law would not be sound because of the way the error is treated locally. More precisely, $\mu.b.\mathbf{0} \parallel a.\mathbf{0} \not\sim \mu.(b.\mathbf{0} \parallel a.\mathbf{0}) + a.(\mu.b.\mathbf{0} \parallel \mathbf{0})$ because the first sub-term of the right-hand process causes the suppression of the whole system, while in the left-hand process, the error is local. The same figure shows that, instead, the decorated version of μ allows us to manage properly any error depending on the environment in which it occurs. To this aim, we point out that it is necessary using the rule ERR.

Theorem 2 (Soundness). *Let $P, Q \in \mathcal{P}$ be finite process terms. If $\vdash P = Q$ then $P \sim Q$.*

Proof. Case M1: show that the relation $\mathcal{R} = \{(\mu.P, \mu_{\mathbf{0}}.P) \mid P \in \mathcal{P}\} \cup Id$, where Id is the identity relation, is a bisimulation. Since $\mu.P \xrightarrow{\mu} \mathbf{0}$ (by rule $\text{ERR}_{\mathbf{0}}$) and $\mu_{\mathbf{0}}.P \xrightarrow{\mu} \mathbf{0}$ (by rule ERR), then the thesis follows immediately.

Case G1: the construction of $\mathcal{R}$ is given similarly as above, with the result deriving immediately by virtue of the rule CONDT.

Case G2: similarly as above; note that $F :\rightarrow P$ does not enable any transition.

Case G3: similarly as above, with the result deriving immediately by virtue of the rule $\text{CONDM}_{\mathbf{0}}$.

Case G4: show that the relation $\mathcal{R} = \{(\varphi :\rightarrow P, \varphi :\rightarrow_{\mathbf{0}} P) \mid P \in \mathcal{P}\} \cup Id$ is a bisimulation. For both processes, if $[\![\varphi]\!]_w = T$, then by virtue of the rule CONDT

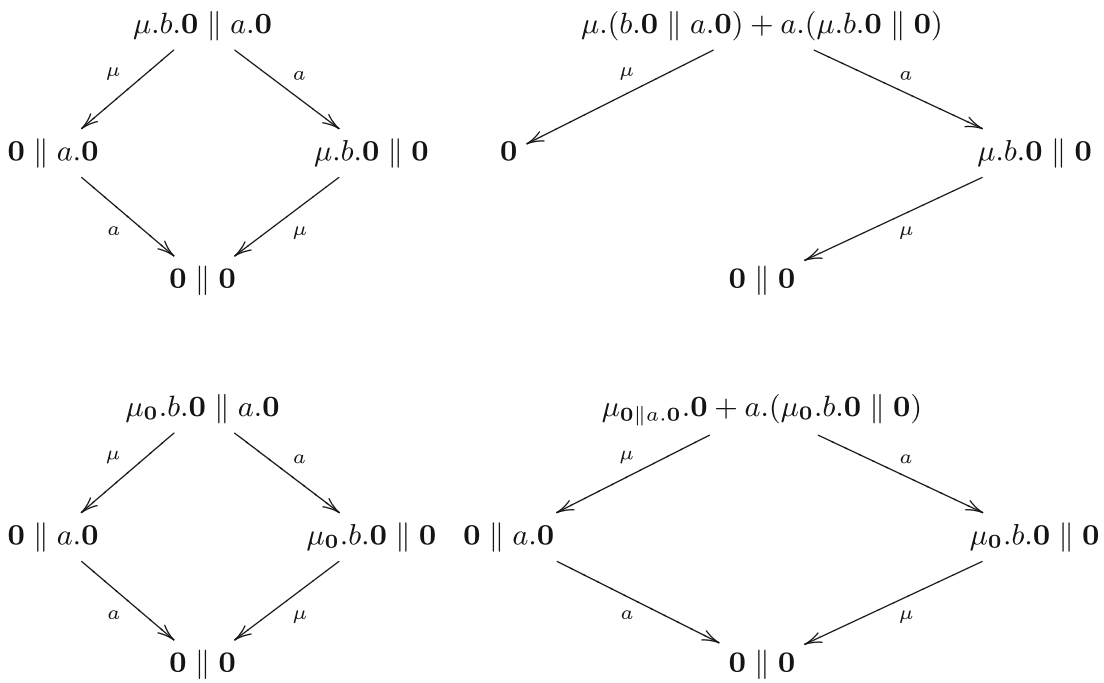

Fig. 2. Wrong (top) and right (bottom) interpretation of expansion law for error prefixes.

the possible transitions are those of P, while if $[\![\varphi]\!]_w = M$, then by virtue of the rules CondM$_{\mathbf{0}}$ and CondM the unique enabled transition executes μ leading to $\mathbf{0}$. The case $[\![\varphi]\!]_w = F$ is trivial. Hence, the thesis follows immediately.

Case G5: similarly as above; the unique interesting case is whenever φ holds. Note that, by virtue of the semantics of the choice operator, the left-hand process executes an α-labeled transition enabled in P (resp., Q) and leading to P' (resp., Q') if and only if the right-hand process so does.

Case G6: similarly as above; there are 9 possible cases when considering the truth values of φ_1 and φ_2 sequentially. For instance, if both evaluate to true, then by virtue of the rule CondT, the left-hand process enables all the transitions enabled by P, and the same holds for the right-hand process by virtue of the semantics of $\varphi_1 \wedge \varphi_2$ and the same rule CondT. All the remaining cases hold as well by virtue of the McCarthy semantics of $\wedge$.

Case H4: take the relation $\mathcal{R} = \{((\varphi :\rightarrow P)\backslash L, \varphi :\rightarrow (P\backslash L)) \mid P \in \mathcal{P}\} \cup Id$ and follow an analogous line of reasoning as in case G4.

Case E1: let us start by considering the behavior of the left-hand process of axiom E1. Given a process term R and the finite set of all transitions outgoing from R, i.e., $\{(R, \alpha_o, R_o) \in \rightarrow_w \mid \alpha_o \in Act, R_o \in \mathcal{P}, \text{with } o \in O\}$ for any finite indexing set O, we use the observation stating that $R \sim \sum_{o \in O} \varphi_o :\rightarrow_{\tilde{R}'_o} \tilde{\alpha}_o.\tilde{R}_o$, where, for each $o \in O$, we have that one of the following cases holds:

1. α_o is a local non-error action, in R it is guarded by φ_o such that $[\![\varphi_o]\!]_w = T$, $\tilde{\alpha}_o = \alpha_o$, $\tilde{R}_o = R_o$, and $\tilde{R}'_o$ does not play any role;

2. $\alpha_o = \tau$, in R it is generated by a synchronization between an action guarded by ψ_1 and by an action guarded by ψ_2 such that $[\![\psi_1]\!]_w = [\![\psi_2]\!]_w = T$, $\varphi_o = \psi_1 \wedge \psi_2$, $\tilde{\alpha}_o = \alpha_o$, $\tilde{R}_o = R_o$, and $\tilde{R}'_o = *$;
3. $\alpha_o = \mu$, in R it is generated by an error action guarded by φ_o such that $[\![\varphi_o]\!]_w = T$, $\tilde{\alpha}_o = \mu_{R_o}$, $\tilde{R}_o = \mathbf{0}$, and $\tilde{R}'_o$ does not play any role;
4. $\alpha_o = \mu$, in R it is generated by an action guarded by φ_o such that $[\![\varphi_o]\!]_w = M$, $\tilde{R}'_o = R_o$ and $\tilde{\alpha}_o.\tilde{R}_o$ does not play any role.

Now, we sketch the proof by noting that in the right-hand process of axiom E1:

- the first two summands cover case 1. (by virtue of the CCS parallel composition semantics and rule CondT) and case 4. (by virtue of the CCS parallel composition semantics and rule CondM), depending on the valuation of the conditional guard;
- the third summand covers case 2. by virtue of the CCS synchronization rules and rule CondT in the case the related conditional guard is true (and does not produce any effect otherwise);
- the fourth and fifth summands cover case 3. (by virtue of the CCS parallel composition semantics and rule Err) and case 4. (by virtue of the CCS parallel composition semantics and rule CondM), depending on the valuation of the conditional guard.

Hence, for each valuation w, the five summands cover all the possible cases, from which the thesis follows.

The result for the rule Equiv is straightforward. All the remaining cases are standard and derive from the soundness result for CCS. □

4.1 Logical Characterization of Bisimulation

Since at the semantics level, the error is represented explicitly by the special action μ, we have seen that a classical notion of bisimulation is sufficient to establish a soundness result for the axiomatization of CCS+e. Similarly, we now show that a slight variant of the Hennessy-Milner logic, denoted HML+e, represents the logical characterization of bisimulation. We point out that the logic we are considering is a standard modal logic.

Definition 2 (HML+e). *The language $\mathcal{L}_{\mathrm{HM}}$ of HML+e is defined by the following grammar:*

$$\phi ::= \top \mid \neg\phi \mid \phi \wedge \phi \mid \langle\pi\rangle\phi \mid M\phi$$

with the satisfaction relation $\models_{\mathrm{HM}}$ (for any finite process term $P \in \mathcal{P}$ and any valuation function w) defined as follows:

- $P, w \models_{\mathrm{HM}} \top$,
- $P, w \models_{\mathrm{HM}} \neg\phi$ *if* $P, w \not\models \phi$,
- $P, w \models_{\mathrm{HM}} \phi_1 \wedge \phi_2$ *if* $P, w \models \phi_1$ *and* $P, w \models \phi_2$,
- $P, w \models_{\mathrm{HM}} \langle\pi\rangle\phi$ *if there exists* P' *such that* $P \xrightarrow{\pi}_w P'$ *and* $P', w \models \phi$,
- $P, w \models_{\mathrm{HM}} M\phi$ *if there exists* P' *such that* $P \xrightarrow{\mu}_w P'$ *and* $P', w \models \phi$.

The correspondence theorem relates bisimilar processes and modal equivalent processes, i.e., processes that satisfy the same set of HML+e formulas. We limit the following result to processes that are finite and image-finite (the image of P, α under the transition relation $\rightarrow_w$ is finite for each w).

Theorem 3 (Logical characterization of $\sim$). *Let $P, Q \in \mathcal{P}$ be finite and image-finite process terms. Then:*

$$P \sim Q \text{ iff for every } w \in \mathcal{W} \text{ and for every } \phi \in \mathcal{L}_{\mathrm{HM}}.\, P, w \models \phi \Leftrightarrow Q, w \models \phi.$$

Proof. Case ($\Rightarrow$). Let $\mathcal{R}$ be a bisimulation including the pair (P, Q). We show the result by induction on the structure of the formulas. The base case is trivial. The only interesting cases are $\langle\pi\rangle\phi$ and $M\phi$. Take an arbitrary valuation w. Suppose that $P, w \models_{\mathrm{HM}} \langle\pi\rangle\phi$ because $\exists P'$ such that $P \xrightarrow{\pi}_w P'$ and $P', w \models_{\mathrm{HM}} \phi$. Hence, by hypothesis, $\exists Q'$ such that $Q \xrightarrow{\pi}_w Q'$ and $P' \sim Q'$. By applying the induction hypothesis, $Q', w \models_{\mathrm{HM}} \phi$, and, therefore, $Q, w \models_{\mathrm{HM}} \langle\pi\rangle\phi$. Hence, no HML+e formula $\langle\pi\rangle\phi$ can distinguish P from Q. By replacing π with μ in the proof, we obtain the result for the case $M\phi$.

Case ($\Leftarrow$). Let $\equiv$ be the modal equivalence relation. We show (by contradiction) that $\equiv$ is a bisimulation itself. By hypothesis, $P \equiv Q$. Take arbitrary action α and valuation w. Assume α is a non-error action π and that there exists P' such that $P \xrightarrow{\pi}_w P'$, but there does not exist Q' such that $Q \xrightarrow{\pi}_w Q'$, with $P' \equiv Q'$. Let $\mathcal{Q}$ be the finite set of processes accessible from Q through a π-labeled transition. $\mathcal{Q}$ is non-empty otherwise $\langle\pi\rangle\top$ would distinguish P from Q. By assumption, for each $Q'' \in \mathcal{Q}$, $1 \leq j \leq |\mathcal{Q}|$, there exists ϕ_j such that $P' \models \phi_j$ and $Q'' \not\models \phi_j$. Hence, it holds that $P \models \langle\pi\rangle \bigwedge_j \phi_j$ (since there is P' such that $P \xrightarrow{\pi}_w P'$ and satisfying $\bigwedge_j \phi_j$), and $Q \not\models \langle\pi\rangle \bigwedge_j \phi_j$, thus contradicting the hypothesis. The same kind of reasoning applies to the symmetric case. Now, if we assume $\alpha = \mu$, the proof is similar to that above. □

5 Case Study: Microservices Architecture

To demonstrate the effectiveness of CCS+e, we apply it to a microservices-based distributed system, where services interact with a load balancer and a database. We show that local failures (e.g., a single server crash) do not force the entire system to fail. The microservice architecture is written in Erlang. It consists of three main components: a load balancer (LB, lines 11–36), two servers able to satisfy the client's requests (S_1 and S_2, lines 38–68), and a database (DB, lines 70–86). Some components may fail in different ways: LB can encounter a network error and stop functioning (lines 28–31); one of the two servers may crash (lines 64–66); the database may corrupt data, impacting clients' requests (lines 81–83). The system behavior is as follows.

The client sends a request to the LB. The LB checks the availability of the network (by querying formula φ_N) and then either forwards the request to a chosen server (true), or informs the client of a failure so the client can retry the request (false), or there is a network error and the LB cannot forward the

request (M). A server, upon receiving a request, checks its state with the formula φ_S. The outcome can be either that it can satisfy the client and connect to the database (true), or it rejects the request, maybe due to too many connections (false) or to an internal crash (M). Depending on the messages received, the client can decide to retry the request. As we can see, if one of the two servers crashes (evaluates to M), the system still keeps working, and the client can still retry its request to the other server. For the sake of simplicity, we are not considering cases in which DB may fail (e.g., $\varphi_{DB} = M$).

```
-module(coordination).
-export([start/0, load_balancer/1, server/2, database/1, client/2]).

start() ->    % Spawn system components
    DB = spawn(?MODULE, database, [healthy]),
    S1 = spawn(?MODULE, server, [db, DB]),
    S2 = spawn(?MODULE, server, [db, DB]),
    LB = spawn(?MODULE, load_balancer, [[S1, S2]]),
    spawn(?MODULE, client, [LB, "data_request"]).

load_balancer(Servers) ->% LOAD BALANCER
  receive
    {request, Client} ->
      % Simulate network conditions using a three-valued logic
      Phi_N = case rand:uniform(3) of
        1 -> error; %Network failure M  -> LB cannot forward requests
        2 -> false; %Connection failed  -> The client can retry
        3 -> true   %Connection success -> Forward request to a server
      end,
      case Phi_N of
        true ->
          Server = lists:nth(rand:uniform(length(Servers)), Servers),
          Server ! {process, Client},
          load_balancer(Servers);
        false ->
          Client ! {network_down}, % Client is informed and may retry
          load_balancer(Servers);
        error ->
            % Network error: Load balancer continues, but cannot forward
            io:format("[LB] Network error, but system remains operational.~n
    "),
            load_balancer(Servers)
      end;
      {error, Reason} ->
        io:format("[LB] Error: ~p. Continuing operations.~n", [Reason]),
        load_balancer(Servers)
    end.

server(Type, DB) ->% SERVER
  receive
    {process, Client} ->
      % Simulate server failure using three-valued logic
      Phi_S = case rand:uniform(3) of
        1 -> error;  % Server crashes
        2 -> false;  % Server rejects request (too busy)
        3 -> true    % Server processes request
      end,
      case Phi_S of
        true ->
        DB ! {query, self()},
        receive
          {response, Data} ->
            io:format("[Server] Received response: ~p~n", [Data]),
            Client ! {ok, Data},
            server(Type, DB);
          {fail, corrupted} ->
```

```
56              io:format("[Server] DB corrupted. Returning fallback response.~n
    "),
57              Client ! {service_unavailable},
58              server(Type, DB)
59            end;
60          false ->
61              io:format("[Server] Request rejected (too busy).~n"),
62              Client ! {service_unavailable},
63              server(Type, DB);
64          error ->
65              io:format("[Server] Unexpected crash. Terminating.~n"),
66              exit(server_failure)
67        end
68    end.
69
70 database(Status) ->% DB
71   receive
72     {query, Server} ->
73       Phi_DB = case Status of % Simulate database state
74         healthy -> true;
75         corrupted -> false  % database corrupted need to recover
76       end,
77       case Phi_DB of
78         true ->
79           Server ! {response, "Data from DB"},
80           database(Status);
81         false ->
82           Server ! {fail, corrupted},
83           database(Status)
84       end;
85       {set_status, NewStatus} -> database(NewStatus)
86     end.
87
88 client(LB, Request) ->% CLIENT
89   io:format("[Client] Sending request: ~p~n", [Request]),
90   LB ! {request, self()},
91   receive
92     {ok, Response} ->
93       io:format("[Client] Received response: ~p~n", [Response]);
94     {network_down} ->
95       io:format("[Client] Network is down. Retrying...~n"),
96       timer:sleep(100),
97       client(LB, Request);
98     {service_unavailable} ->
99       io:format("[Client] Server unavailable. Retrying...~n"),
100      timer:sleep(100),
101      client(LB, Request)
102    end.
```

The above Erlang code is modeled as the following CCS+e process:

$$\begin{aligned}
\textsc{Sys} &= (\mathrm{C} \parallel \mathrm{LB} \parallel \mathrm{S}_1 \parallel \mathrm{S}_2 \parallel \mathrm{DB})\backslash L \\
\mathrm{LB} &= lb.(\texttt{if } (\varphi_N)_{\mathrm{LB}} \texttt{ then } \overline{s_1}.\mathrm{LB} + \overline{s_2}.\mathrm{LB} \texttt{ else } \overline{nd}.\mathrm{LB}) \\
\mathrm{S}_i &= s_i.(\texttt{if } (\varphi_S) \texttt{ then } \overline{db}.(ok.\overline{resp}.\mathrm{S}_i + crash.\overline{su}.\mathrm{S}_i) \texttt{ else } \overline{su}.S_i) \\
\mathrm{DB} &= db.(\texttt{if } (\varphi_{DB}) \texttt{ then } \overline{ok}.\mathrm{DB} \texttt{ else } \overline{crash}.\mathrm{DB}) \\
\mathrm{C} &= \overline{lb}.(nd.\mathrm{C} + su.\mathrm{C} + resp.\overline{done}.\mathbf{0})
\end{aligned}$$

such that $\texttt{if } (\varphi)_R \texttt{ then } P \texttt{ else } Q ::= (\varphi :\rightarrow_R P) + (\neg\varphi :\rightarrow_R Q)$

where L includes all the actions of the specification above but $\overline{done}$. LB takes a request from the client and, depending on condition φ_N, it acts as follows: if

$\varphi_N = T$, it forwards the request to one of the servers ($\overline{s}_1 + \overline{s}_2$); if $\varphi_N = F$, it tells the client that the network is down (action nd); if $\varphi_N = M$, it crashes and starts over, while the request is lost. Once one of the servers receives a request from LB, depending on the condition φ_S it acts as follows: if $\varphi_S = T$, it contacts DB, awaits a response, and then forwards it to the client; if $\varphi_S = F$, it tells the client that the service is temporarily unavailable (action su); if $\varphi_S = M$, the server crashes locally, not affecting the other processes.

Let us consider a computation in which the contacted server fails (φ_N is true while φ_S for the chosen server evaluates to M):

$$\begin{aligned}\textsc{Sys} &\xrightarrow{\tau} \big((nd.\mathrm{C} + su.\mathrm{C} + resp.\overline{done}.\mathbf{0}) \parallel \texttt{if } (\varphi_N)_{\mathrm{LB}} \cdots \parallel \mathrm{S}_1 \parallel \mathrm{S}_2 \parallel \mathrm{DB}\big)\backslash L \\ &\xrightarrow{\tau} \big((nd.\mathrm{C} + su.\mathrm{C} + resp.\overline{done}.\mathbf{0}) \parallel \mathrm{LB} \parallel \texttt{if } (\varphi_S) \cdots \parallel \mathrm{S}_2 \parallel \mathrm{DB}\big)\backslash L \\ &\xrightarrow{\mu} \big((nd.\mathrm{C} + su.\mathrm{C} + resp.\overline{done}.\mathbf{0}) \parallel \mathrm{LB} \parallel \mathbf{0} \parallel \mathrm{S}_2 \parallel \mathrm{DB}\big)\backslash L\end{aligned}$$

Note that now the client deadlocks (the same would also hold in the case the local condition φ_N evaluates to M) because, as an intended feature, the server's local error does not propagate by causing the client's failure. Usually, in distributed systems, this kind of situation is faced using timeouts[6]. Anyhow, the above behavior does not prevent the system from operating with other clients. For example, in the following version of the system:

$$\textsc{Sys}' = (\mathrm{C}_1 \parallel \mathrm{C}_2 \parallel \mathrm{LB} \parallel \mathrm{S}_1 \parallel \mathrm{S}_2 \parallel \mathrm{DB})\backslash L$$

with $\mathrm{C}_1 = \mathrm{C}_2 = \mathrm{C}$, and by reproducing the execution from above, we have:

$$\textsc{Sys}' \xrightarrow{\tau}\xrightarrow{\tau}\xrightarrow{\mu} \big((nd.\mathrm{C}_1 + su.\mathrm{C}_1 + resp.\overline{done}.\mathbf{0}) \parallel \mathrm{C}_2 \parallel \mathrm{LB} \parallel \mathbf{0} \parallel \mathrm{S}_2 \parallel \mathrm{DB}\big)\backslash L$$

The reached system, call it $\textsc{Sys}_e$, can still operate because C_2 request can be accepted by LB and S_2[7], assuming that all the conditions are evaluated to true. In particular, from $\textsc{Sys}_e$, we can have the following reduction:

$$\begin{aligned}&\textsc{Sys}_e \xrightarrow{\tau}\xrightarrow{\tau}\xrightarrow{\tau}\xrightarrow{\tau}\xrightarrow{\tau} \\ &\qquad \big((nd.\mathrm{C}_1 + su.\mathrm{C}_1 + resp.\overline{done}.\mathbf{0}) \parallel \overline{done}.\mathbf{0} \parallel \mathrm{LB} \parallel \mathbf{0} \parallel \mathrm{S}_2 \parallel \mathrm{DB}\big)\backslash L \\ &\xrightarrow{\overline{done}} \big((nd.\mathrm{C}_1 + su.\mathrm{C}_1 + resp.\overline{done}.\mathbf{0}) \parallel \mathbf{0} \parallel \mathrm{LB} \parallel \mathbf{0} \parallel \mathrm{S}_2 \parallel \mathrm{DB}\big)\backslash L\end{aligned}$$

We can also describe the above computation via the following $\mathcal{L}_{\mathrm{HM}}$ formula:

$$\textsc{Sys}', w \models_{\mathrm{HM}} \langle\tau\rangle\langle\tau\rangle M\langle\tau\rangle\langle\tau\rangle\langle\tau\rangle\langle\tau\rangle\langle\tau\rangle\langle\overline{done}\rangle\top$$

where w is the valuation respecting the conditions described above. Regarding the behavior of LB, we can verify the property that if it fails ($[\![\varphi_N]\!]_w = M$), it starts over again:

$$\mathrm{LB}, w \models_{\mathrm{HM}} \langle lb\rangle M\langle lb\rangle\top$$

[6] In the Erlang code of the Client, it is sufficient to add the `after T` clause in the `receive` statement.

[7] For the sake of simplicity, we have chosen $\mathrm{C}_1 = \mathrm{C}_2$ hence there could be a case in which the deadlocked C_1 intercepts the response for C_2. This can be easily avoided by using indexed channels, like nd_i, $resp_i$, and nu_i, where i is the index of the client.

6 Conclusions

In this paper, we have investigated the addition of an explicit error action to the theory of CCS. To this end, we employed a three-valued logic for the valuation of conditional statements. This work is inspired by previous work on ACP [5] and differs from it in the interpretation of the infectious behavior of the error, which in our setting is intended to be local. The effects of our assumptions can be viewed on the axioms that characterize CCS+e. In particular, we recall the importance of the expansion law for the parallel composition operator, as it is compliant with the idea that exploring any possible interleaving is the most adequate approach for testing [6] and that the occurrence of errors should not prune any of the significant branches.

As a future work, we plan to investigate an alternative interpretation of the error between our approach and [5]. The idea is to model errors that are causally infectious, i.e., an error should infect just those processes causally related to it. This would require devising a causal semantics for CCS+e in line with the semantics for causal-consistent reversibility [22,26]. It is also worth comparing our approach with alternative ways of dealing with failures. Indeed, there are several works explicitly modeling and handling errors in process algebras [1,8,12,23]. Basically, we differ from them by the fact that we model the error via a three-valued logic and that, in many cases, they do not provide any axiomatization.

The lazy treatment of errors we have adopted recalls McCarthy's operator of ambiguous choice [24], which is formalized, e.g., in the setting of π-calculus for different interpretations of divergence [10]. We will further investigate this relationship in future works. Additionally, McCarthy's lazy evaluation of conjunction may be used to model sequential composition in functional programming languages with lazy evaluation (e.g., Haskell). To this end, we plan to extend our framework to a lambda calculus with lazy evaluation and actor-based concurrency (e.g., mailboxes) like Haskell cloud [13]. Once we get such semantics, one could think of reversing it in order to obtain a reversible debugger [17] for an actor-based language with a proper axiomatization of errors. Also, a three-valued characterization of errors for actor languages could be used to synthesize run-time monitors [16] to react to such errors. Other interesting research developments within our framework are the study of the noninterference approach to information flow analysis [14,15], as well as exploring extensions in quantitative settings [2], which would allow us to model and verify the relation between failure of trusted components, error propagation, and performance degradation.

Finally, another important direction for future work involves establishing the completeness result for our axiomatization, specifically that bisimilar processes in normal form can be equated through the axioms of the system $\mathcal{A}$. However, this result requires an axiom like G0 for aggregation purposes [5]. As motivated in Sect. 3.1, we reject the relation induced by G0 between the nondeterministic choice $+$ and the disjunction connective $\vee$, as no deterministic interpretation of $\vee$ in the three-valued logic (neither in the McCarthy semantics nor in the Bochvar semantics) can capture the fact that the error is not supposed to suppress nondeterminism (in essence, $M \vee T$ and $T \vee M$ should output both T and

M, nondeterministically). An operator $\tilde{\vee}$ obeying such nondeterministic semantics is investigated in the setting of a sequent calculus [3], where $\tilde{\vee}$ results from the combination of McCarthy logic and Kleene logic with the aim of modeling critical and non-critical errors. The study of the algebraic properties of this operator and its integration in our framework is left for future work. Together with an equational theory for the three-valued logic, it could pave the way for a sound and complete axiomatization of full CCS+e.

Acknowledgement. This work has been funded by the European Union - NextGenerationEU within the framework of PNRR Mission 4 - Component 2 - Investment 1.1 under the Italian Ministry of University and Research programme PRIN 2022 - grant number 2022SM4XC8 - DeKLA - CUP: F53D23004840006.

References

1. Aceto, L., Hennessy, M.: Termination, deadlock, and divergence. J. ACM **39**(1), 147–187 (1992). https://doi.org/10.1145/147508.147527
2. Aldini, A.: Modeling and verification of trust and reputation systems. Secur. Commun. Netw. **8**(16), 2933–2946 (2015). https://doi.org/10.1002/sec.1220
3. Avron, A., Konikowska, B.: Proof systems for reasoning about computation errors. Stud. Log.: Int. J. Symb. Log. **91**, 273–293 (2009). http://www.jstor.org/stable/40269036
4. Bergstra, J.A., Klop, J.W.: Process algebra for synchronous communication. Inf. Control **60**(1–3), 109–137 (1984). https://doi.org/10.1016/S0019-9958(84)80025-X
5. Bergstra, J.A., Ponse, A.: Bochvar-McCarthy logic and process algebra. Notre Dame J. Formal Log. **39**(4), 464–484 (1998). https://doi.org/10.1305/NDJFL/1039118863
6. Bianchi, F.A., Margara, A., Pezzè, M.: A survey of recent trends in testing concurrent software systems. IEEE Trans. Softw. Eng. **44**(8), 747–783 (2018). https://doi.org/10.1109/TSE.2017.2707089
7. Bochvar, D.A., Bergmann, M.: On a three-valued logical calculus and its application to the analysis of the paradoxes of the classical extended functional calculus. Hist. Philos. Log. **2**(1–2), 87–112 (1981). https://doi.org/10.1080/01445348108837023
8. de Boer, F.S., Coenen, J., Gerth, R.: Exception handling in process algebra. In: Purushothaman, S., Zwarico, A.E. (eds.) NAPAW92. Workshops in Computing, pp. 86–100. Springer, London (1992). https://doi.org/10.1007/978-1-4471-3217-2_6
9. Bonzio, S., John, G.S.: On the structure and theory of McCarthy algebras (2025). https://arxiv.org/abs/2503.10816
10. Carayol, A., Hirschkoff, D., Sangiorgi, D.: On the representation of McCarthy's amb in the π-calculus. Theoret. Comput. Sci. **330**(3), 439–473 (2005). https://doi.org/10.1016/j.tcs.2004.10.005
11. Cobreros, P., Égré, P., Ripley, D., van Rooij, R.: Foreword: three-valued logics and their applications. J. Appl. Non-Classical Log. **24**(1–2), 1–11 (2014). https://doi.org/10.1080/11663081.2014.909631

12. Dragoni, N., Gaspari, M.: An object based algebra for specifying a fault tolerant software architecture. J. Log. Algebraic Methods Program. **63**(2), 271–297 (2005). https://doi.org/10.1016/J.JLAP.2004.05.006
13. Epstein, J., Black, A.P., Jones, S.L.P.: Towards Haskell in the cloud. In: Claessen, K. (ed.) Proceedings of the 4th ACM SIGPLAN Symposium on Haskell, pp. 118–129. ACM (2011). https://doi.org/10.1145/2034675.2034690
14. Esposito, A., Aldini, A., Bernardo, M.: Branching bisimulation semantics enables noninterference analysis of reversible systems. In: Huisman, M., Ravara, A. (eds.) FORTE 2023. LNCS, vol. 13910, pp. 57–74. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-35355-0_5
15. Esposito, A., Aldini, A., Bernardo, M., Rossi, S.: Noninterference analysis of reversible systems: An approach based on branching bisimilarity. Log. Methods Comput. Sci. **21**(1) (2025). https://doi.org/10.46298/lmcs-21(1:6)2025
16. Francalanza, A., Mezzina, C.A., Tuosto, E.: Reversible choreographies via monitoring in erlang. In: Bonomi, S., Rivière, E. (eds.) DAIS 2018. LNCS, vol. 10853, pp. 75–92. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93767-0_6
17. Giachino, E., Lanese, I., Mezzina, C.A.: Causal-consistent reversible debugging. In: Gnesi, S., Rensink, A. (eds.) FASE 2014. LNCS, vol. 8411, pp. 370–384. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54804-8_26
18. Gorrieri, R., Versari, C.: Introduction to Concurrency Theory - Transition systems and CCS. Texts in Theoretical Computer Science. An EATCS Series. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-21491-7
19. Hennessy, M., Milner, R.: On observing nondeterminism and concurrency. In: de Bakker, J., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 299–309. Springer, Heidelberg (1980). https://doi.org/10.1007/3-540-10003-2_79
20. Kleene, S.C.: Introduction to Metamathematics. D. van Nostrand (1952)
21. Konikowska, B.: McCarthy algebras: a model of McCarthy's logical calculus. Fund. Inform. **26**(2), 167–203 (1996). https://doi.org/10.3233/FI-1996-26205
22. Lanese, I., Mezzina, C.A., Stefani, J.: Reversibility in the higher-order π-calculus. Theor. Comput. Sci. **625**, 25–84 (2016). https://doi.org/10.1016/J.TCS.2016.02.019
23. Lanese, I., Vaz, C., Ferreira, C.: On the expressive power of primitives for compensation handling. In: Gordon, A.D. (ed.) ESOP 2010. LNCS, vol. 6012, pp. 366–386. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11957-6_20
24. McCarthy, J.: A basis for a mathematical theory of computation. In: Braffort, P., Hirschberg, D. (eds.) Computer Programming and Formal Systems, Studies in Logic and the Foundations of Mathematics, vol. 26, pp. 33–70. Elsevier (1959). https://doi.org/10.1016/S0049-237X(09)70099-0
25. Milner, R. (ed.): A Calculus of Communicating Systems. Lecture Notes in Computer Science, vol. 92. Springer, Heidelberg (1980). https://doi.org/10.1007/3-540-10235-3
26. Phillips, I., Ulidowski, I.: Reversing algebraic process calculi. J. Log. Algebraic Methods Program. **73**(1–2), 70–96 (2007). https://doi.org/10.1016/J.JLAP.2006.11.002
27. Sangiorgi, D., Walker, D.: The Pi-Calculus - A Theory of Mobile Processes. Cambridge University Press (2001)
28. Turner, D.A.: Some history of functional programming languages. In: Loidl, H.-W., Peña, R. (eds.) TFP 2012. LNCS, vol. 7829, pp. 1–20. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40447-4_1