A Theory of Proc-Omata – and a Proof-technique for Parameterized Process-Architectures

Benoît Ballenghien[1][0009−0000−4941−187X] and Burkhart Wolff[2][0000−0002−9648−7663]

[1] LMF, Université Paris-Saclay, Paris, France
benoit.ballenghien@universite-paris-saclay.fr
[2] LMF, Université Paris-Saclay, Paris, France
wolff@lmf.cnrs.fr

Abstract. This work is based on Isabelle/HOL-CSP 2.0, a shallow embedding of the failure-divergence model of denotational semantics proposed by Hoare, Roscoe and Brookes in the eighties. In several ways, HOL-CSP is actually an extension of the original setting in the sense that it admits higher-order processes and infinite alphabets.
In this paper, we present a particular sub-class of CSP processes which we call Proc-Omata, a fantastic beast between processes and functional automata. For this class of processes, particular proof techniques can be applied allowing for reasoning over unbounded families of sub-processes and similar architectural compositions.
We develop the basic theory of deterministic and non-deterministic Proc-Omata, both their relation to conventional CSP processes as well as possible transformation operations on them. As an application of the Proc-Omata theory, we demonstrate the use of so-called compactification theorems that pave the way, for example, to proofs over process rings of arbitrary size.

Keywords: Process-Algebra, Concurrency, Automata, Computational Models, Theorem Proving, Isabelle/HOL

1 Introduction

Communicating Sequential Processes (CSP) is a language to specify and verify patterns of interaction of concurrent systems. Together with CCS and LOTOS, it belongs to the family of *process algebras*. CSP's rich theory comprises denotational, operational and algebraic semantics.

The theory of CSP was first described in 1978 in a book by Tony Hoare [12], but has since evolved substantially [5,6,26]. The denotational semantics of CSP is described by a fully abstract model of behaviour designed to be *compositional*: a process P encompasses all possible behaviours, i. e. sets of *traces* annotated by additional information that allow to reason over

– deadlocks (the resulting semantic domain is called *failure semantics* F)

– and additionally livelocks (the *failure/divergence semantics FD*).

Several attempts have been undertaken to formalize this fairly complex theory, notably [7,32,16,21,13]. The presented work here is based on HOL-CSP [31,29,3,4], a shallow embedding of the denotational and operational semantics theory in the proof-assistant Isabelle/HOL. HOL-CSP is in several ways not only a formalization, but a generalization of the original setting:

– the set of traces α *traces* is constructed over an arbitrary type $'\alpha$ in `HOL`, paving the way for dense-time, vector-spaces, etc, [3],
– in generally, `HOL-CSP` attempts to remove finiteness-restrictions, and
– the semantic domain is encapsulated in the type $'\alpha$ *process* belonging to the cpo type class (see Sect. 2.4). Thus, process patterns can be expressed as functions over processes.

In this paper, we present the formal theory of Proc-Omata built on top of HOL-CSP. Proc-Omata are a sub-class of CSP processes, that have an extremely simple process structure but possess a functional automata [19,20] inside which can have an infinite state and communication alphabet. For certain process-patterns such as an i-indexed family of interleaving processes $|||\ i \in\# M.\ P(i)$, it is possible to convert this pattern into a Proc-Omata provided that the $P(i)$ can be converted into Proc-Omata. Since this construction is possible for index-sets M of arbitrary size, this paves the way for proofs of properties such as deadlock- or livelock freeness over process-patterns. The key-instruments of this constructions are a particular form of equations we call *compactification theorems* that we formally prove correct in this paper.

Functional automata consist of a transition function τ coming in two flavors:

1. the deterministic version of τ has type $'\sigma \Rightarrow 'e \Rightarrow '\sigma\ option$, i.e. in a state $s::'\sigma$, given an event $e::'e$, the transition may result in a successor state $s'::'\sigma$ or fail, and
2. the non-deterministic version of τ having type $'\sigma \Rightarrow 'e \Rightarrow '\sigma\ set$ allowing a transition from a state $s::'\sigma$ for an event $e::'e$ to a (possibly empty) set of successor states.

Automata based on a deterministic transition function will be called deterministic and denoted A_d, and non-deterministic (denoted A_{nd}) otherwise.

Now, Proc-Omata have the general form of a CSP process schema:

$$\mu\ X.\ (\lambda\sigma.\ \Box e \in \varepsilon\ A\ \sigma \rightarrow \sqcap \sigma' \in \tau\ A\ s\ e.\ X\ \sigma')$$

where $\tau\ A$ is the transition of automaton A and $\varepsilon\ A$ computes the set of events for which A is *enabled* (ready to make a transition) to a successor state (both in the deterministic and non-deterministic case). If P is a Proc-Omata, then classic CSP theory gives us the definitions for:

1. the set of *traces* $\mathcal{T}\ (P\ \sigma_0)$ from some initial state σ_0,

[3] or even differential equations as in cyber-physical system models [10]

2. and the set of *failures* $\mathcal{F}\ (P\ \sigma_0)$ from some initial state σ_0.

Note that Proc-Omata have no divergences.

Example 1 *Consider the Collatz-Process:*

$$\begin{aligned} \mathit{Collatz}\ x = (\quad & \Box\ x \in \{0{::}\mathit{nat}\} \to \mathit{SKIP}) \\ & \Box\ (\Box\ x \in \mathit{Even} \to \mathit{Collatz}\ (x\ \mathit{div}\ 2)) \\ & \Box\ (\Box\ x \in \mathit{Odd} \to \mathit{Collatz}\ (3{*}x + 1)) \end{aligned}$$

Seen as a symbolic LTS, this process looks like this:

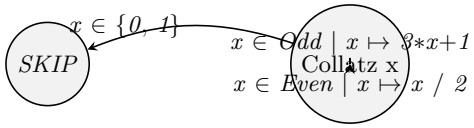

Fig. 1: LTSs for the Copy Buffer example

Presented as a Proc-Omata CA the Collatz process has the following form:

$$\begin{aligned} \tau\ \mathit{CA} = (\lambda\sigma\ e.\ \ & \mathit{if}\ \sigma{=}0 \wedge e{\in}\{0,1\}\ \mathit{then\ Some\ 1} \\ & \mathit{else\ if}\ \sigma{=}0 \wedge \mathit{even\ e\ then\ Some\ 0} \\ & \quad \mathit{else\ if}\ \sigma{=}0 \wedge \mathit{odd\ e\ then\ Some\ 0\ else\ None}) \end{aligned}$$

and where enabledness is defined by $\varepsilon\ \mathit{CA} = (\lambda\sigma.\ \{e \mid \tau\ \mathit{CA}\ \sigma\ e \neq \diamond\})$.

The example above gives rise to a particular proof-methodology: First, we construct for a process P a Proc-Omata and prove that it is equivalent; this proof can be either done by fix-point induction or in finitary cases by model-checking. Second, we apply the above-mentioned compactification theorems over Proc-Omata. Third, we can prove properties over the compactified Proc-Omata by classical invariant reasoning over the state-space of the latter.

In this paper, we proceed as follows: after an introduction into "classic" CSP and our extensions `HOL-CSP` and `HOL-CSPM` in Isabelle/HOL in Sect. 2, we present the core-constructions of this paper: formal definitions of deterministic and non-deterministic Proc-Omata's, a number of basic and evolved theorems over them, the compactification theorems allowing to internalize compositions of Proc-Omata's and the groundwork for inductive proofs, and the consequences for bisimulation proof schemes.

Note that `HOL-CSP`[31], `HOL-CSPM`[3] as well as the novel contribution `HOL-CSP_OpSem` [4] containing the proofs discussed here are published in the Archive of Formal Proofs AFP [4].

[4] For the developer version, see `https://gitlab.lisn.upsaclay.fr/burkhart.wolff/hol-csp2.0/`.

2 Background

2.1 Classic CSP Syntax

At a glance, the syntax of the classical CSP core language reads as follows:

$$P ::= SKIP \mid STOP \mid P \,\Box\, P' \mid P \sqcap P' \mid P \,[\![A]\!]\, P' \mid P \,;\, P' \mid P \setminus A \\ \mid a \to P \mid \Box a \in A \to P(a) \mid \sqcap a \in A \to P(a) \mid Renaming\ P\ g \mid \mu\ X.\ f(X)$$

SKIP signals termination ans *STOP* denotes a deadlock.
Two choice operators are distinguished:

1. the *external choice* -□-, which forces a process "to follow" whatever its context requires,
2. the *internal choice* -⊓-, which imposes on the context of a process "to follow" the non-deterministic choices made.

With the prefix operator $a \to P$ which signals a and continues with P (where a is an element of a set Σ of *events*), generalized choices of the form $\Box a{\in}A \to P(a)$ resp. $\sqcap a{\in}A \to P(a)$ are constructed (A is originally a finite set). When events are tagged with *channels*, i. e. $\Sigma = CHANNELS \times DATA$, syntactic sugar like $c?x{\in}A{\to}P(x)$ or $c!x{\in}A{\to}P(x)$ is added; the former reads intuitively as "x is read from channel c" while the latter means "x is arbitrarily chosen from A and sent into c" (where $c \in CHANNELS$ and $x \in DATA$).

The sequential composition $P \,;\, P'$ behaves first like P and, once it has successfully terminated, like P'. $P \setminus A$ consists in hiding the events of the set A. *Renaming* $P\ g$ results in a process in which each event e of P is renamed in $g(e)$. The *Sliding* operator $P \rhd P'$ is defined as $(P \,\Box\, P') \sqcap P'$. The fixed point $\mu\ X.\ f(X)$ operator satisfies $(\mu\ X.\ f(X)) = f\ (\mu\ X.\ f(X))$ (but requires precautions, see Sect. 2.4).

CSP describes all communication with one single primitive: the synchronized product written $P \,[\![A]\!]\, P'$. Note that interleaving $P \,|||\, P'$ stands for $P \,[\![\{\}]\!]\, P'$, whereas the parallel operator $P \,||\, P'$ is a shortcut for $P \,[\![UNIV]\!]\, P'$ ($UNIV$ is the universal set).

2.2 Classic CSP Semantics

The denotational semantics (following [26]) comes in three layers: the *trace model*, the *(stable) failures model* and the *failure/divergence model*.

In the trace semantics model, the behaviour of a process P is denoted by a prefix-closed set of traces, denoted $\mathcal{T}\ P$, similar to the well-known concept of a "language of an automata". Since traces are finite lists and infinite behaviour is therefore represented via the set of approximations, an additional element *tick* (written ✓) is used to represent explicit termination signalized by *SKIP*. Note that, obviously, *tick* should only appear at the end of a trace (traces should be *front-tickFree*).

It is impossible to distinguish external and internal non-determinism in the trace model since the traces of both operators are just the union of their argument traces. To be more discriminant, [5] proposed the failure semantics model,

where traces were annotated with a set of *refusals*, i. e. sets of events a process can *not* engage in. This leads to the notion of a *failure* $(t,\ X) \in \mathcal{F}\ P$ which is a pair of a trace t and a set of refusals X. Consider for example the process $P = (a \rightarrow SKIP) \ \Box\ (a \rightarrow STOP)$. The traces $\mathcal{T}\ P$ will non-deterministically lead to a situation where the process accepts termination (but refuses everything else) or just refuses everything. So, if we assume $\Sigma = \{a, \checkmark\}$, then the traces $\mathcal{T}\ P$ will be $\{[], [a]\}$. The failures $\mathcal{F}\ P$ are then $\{([], \{\{\checkmark\}\}), ([a], \{\Sigma, \{a\}\})\}$ (plus all subsets of the respective refusal sets, which is required for the recursion ordering discussed in Sect. 2.4).

Finally, [5] enriched the semantic domain of CSP with one more element, the set of *divergences* (written $\mathcal{D}\ P$), in order to distinguish deadlocks from livelocks[5]. In the failure divergence model, the semantic domain consists of a pair of failures and divergences, where the latter are traces to situations where livelocks may occur.

In contrast to Hoare Logics and its Hoare Triples, which is a framework to reason over terminating calculations[6], CSP and process refinement are designed to reason over non-terminating calculations. Three classic refinement notions are considered:

1. the trace refinement: $P \sqsubseteq_T Q \equiv \mathcal{T}\ P \supseteq \mathcal{T}\ Q$,
2. the failure refinement: $P \sqsubseteq_F Q \equiv \mathcal{F}\ P \supseteq \mathcal{F}\ Q$, and
3. the failure-divergence refinement $P \sqsubseteq_{FD} Q \equiv \mathcal{F}\ P \supseteq \mathcal{F}\ Q \wedge \mathcal{D}\ P \supseteq \mathcal{D}\ Q$.

It turns out that beyond common protocol refinement proofs and test-problems, many properties such as deadlock or livelock freeness can be expressed via a refinement statement.

2.3 Theories and Locales in Isabelle and HOL

Isabelle is a major interactive proof assistant implementing higher-order logic (HOL). As an LCF style theorem prover, it is based on a small logical core (kernel) to increase the trustworthiness of proofs without requiring — yet supporting — explicit proof objects.

The Isabelle distribution comes with a number of library theories constructed solely from definitional axioms; among them basic data-types for sets, lists, arithmetics, a substantial part of analysis, and — particularly relevant here — Scott domain theory (`HOLCF`) [18] providing a particular type class $'\alpha$::*pcpo*, i.e. the class of types $'\alpha$ for which a least element $\bot$ and a complete partial order -$\sqsubseteq$- is defined.

`HOLCF` provides the concept of *continuity*, the concept of *admissibility*, the fixed point operator $\mu\ x.\ f\ x$ as well as the fixed point induction for admissible predicates. Isabelle's type inference can automatically infer, for example, that if $'\alpha$::*pcpo*, then $('\beta \Rightarrow '\alpha)$::*pcpo*.

A distinguishing feature of Isabelle is the **locale** mechanism, i. e. theories that may be parameterized by types, constant-symbols and local hypotheses

[5] also called *infinite internal chatter* as occurring in processes like $\mu\ x.\ a \rightarrow x \setminus \{a\}$

[6] although this calculation can be extended, see [22].

over them. Since **locale**s may inherit from other **locale**s, they represent a powerful structuring mechanism for orders and algebraic structures very similar to dependent types available in other systems.

2.4 Isabelle/HOL-CSP

`Isabelle/HOL-CSP` is a shallow embedding of CSP in HOL based on the traditional semantic domain described by 9 "axioms" over the three semantic functions $\mathcal{T} :: 'α\ process \Rightarrow 'α\ trace\ set$, $\mathcal{F} :: 'α\ process \Rightarrow 'α\ failure\ set$ and $\mathcal{D} :: 'α\ process \Rightarrow 'α\ trace\ set$:

- the empty trace is always the initial trace and any trace is *front-tickFree*;
- traces of a process are *prefix-closed* and a process can refuse all subsets of refusals;
- any event refused by a process after a trace s must be in a refusal set associated to s;
- the *tick* accepted after a trace s implies that all other events are refused;
- a divergence trace with any suffix is itself a divergence one
- once a process has diverged, it can engage in or refuse any sequence of events.
- a *tick*-ending divergence trace has a *tickFree* divergence trace prefix of maximal length.

More formally, a process P of the type $\Sigma\ process$ should have the following properties:

$$
\begin{aligned}
&([] \in \mathcal{T}\ P \wedge (\forall\, s\ X.\ (s,\ X) \in \mathcal{F}\ P \longrightarrow \textit{front-tickFree}\ s) \wedge\\
&\ (\forall\, s\ t.\ s\ @\ t \in \mathcal{T}\ P \longrightarrow s \in \mathcal{T}\ P) \wedge\\
&\ (\forall\, s\ X\ Y.\ (s,\ Y) \in \mathcal{F}\ P \wedge X \subseteq Y \longrightarrow (s,\ X) \in \mathcal{F}\ P) \wedge\\
&\ (\forall\, s\ X\ Y.\ (s,\ X) \in \mathcal{F}\ P \wedge\\
&\ (\forall\, c.\ c \in Y \longrightarrow s\ @\ [c] \notin \mathcal{T}\ P) \longrightarrow (s,\ X \cup Y) \in \mathcal{F}\ P) \wedge\\
&\ (\forall\, s\ X.\ s\ @\ [\checkmark] \in \mathcal{T}\ P \longrightarrow (s,\ X - \{\checkmark\}) \in \mathcal{F}\ P) \wedge\\
&\ (\forall\, s\ t.\ s \in \mathcal{D}\ P \wedge \textit{tickFree}\ s \wedge \textit{front-tickFree}\ t \longrightarrow s\ @\ t \in \mathcal{D}\ P) \wedge\\
&\ (\forall\, s\ X.\ s \in \mathcal{D}\ P \longrightarrow (s,\ X) \in \mathcal{F}\ P) \wedge (\forall\, s.\ s\ @\ [\checkmark] \in \mathcal{D}\ P \longrightarrow s \in \mathcal{D}\ P))
\end{aligned}
$$

The core of `HOL-CSP` is to encapsulate this wishlist into a type definition. This is achieved by 1) defining the pair of failures and divergences $\Sigma\ process_0$ via $(\Sigma^{\checkmark}\ list \times (\Sigma^{\checkmark})\ set)set \times (\Sigma^{\checkmark})set$ (where $\Sigma^{\checkmark} = \Sigma \uplus \{\checkmark\}$), 2) by turning the above wishlist into a data-constraint *is-process* of type $\Sigma\ process_0 \Rightarrow bool$, and 3) by establishing an isomorphism between the subset of $\Sigma\ process_0$'es satisfying *is-process* via the-specification construct:

$$\textbf{typedef}\ 'α\ process = \{P :: 'α\ process_0\ .\ is\text{-}process\ P\}$$

Subsequently, we provide definitions for each CSP operator in terms of $\Sigma\ process_0$; these definitions formalize directly the textbook [26]. Finally, we prove that each operator preserves the *is-process*-invariant. The preservation even holds for arbitrary (possibly infinite) sets A in the generalisations $\Box x{\in}A \rightarrow P(x)$ resp. $\sqcap x{\in}A \rightarrow P(x)$. Note that both use higher-order abstract syntax and have the type $'α\ set \Rightarrow ('α \Rightarrow 'α\ process) \Rightarrow 'α\ process$.

A major problem prevails: how to give semantics to the fixed point operator? This is achieved by turning the denotational domain of CSP into a Scott complete partial order (cpo) [27], which provides semantics for the fixed point operator $\mu\ x.\ f(x)$ under the condition that f is continuous wrt. this partial ordering. Since the natural ordering - $\sqsubseteq_{FD}$ - is too weak for this purpose, Roscoe and Brookes [23] proposed a complete *process ordering* $P \sqsubseteq Q$ which is stronger, i. e. $P \sqsubseteq Q \Longrightarrow P \sqsubseteq_{FD} Q$, and yet ensures completeness at least for general read and write operations. It is based on the concept *refusals after* a trace s (defined as $\mathcal{R}_a\ P\ s \equiv \{X \mid (s,\ X) \in \mathcal{F}\ P\}$):

$P \sqsubseteq Q \equiv \mathcal{D}\ Q \subseteq \mathcal{D}\ P \wedge (\forall\, s.\ s \notin \mathcal{D}\ P \longrightarrow \mathcal{R}_a\ P\ s = \mathcal{R}_a\ Q\ s) \wedge \mathit{min\text{-}elems}\ (\mathcal{D}\ P) \subseteq \mathcal{T}\ Q$

Theorem 1 (Continuity) *Almost all HOL-CSP operators $\otimes$ are continuous wrt. -$\sqsubseteq$-, i. e.:*

$$\mathit{cont}\ f \Longrightarrow \mathit{cont}\ g \Longrightarrow \mathit{cont}(\lambda x.\ (f\ x) \otimes (g\ x))$$

Based on the lemma that $\sqsubseteq_{FD}$ is admissible for the fixed point induction, when f is continuous we have an induction rule of the following form.

Theorem 2 (Fixed-point Inductions) *For cont f, we have:*

$$C\ (\bot) \sqsubseteq_{FD} Q \Longrightarrow (\bigwedge x.\ C\ (x) \sqsubseteq_{FD} Q \Longrightarrow C(f\ x) \sqsubseteq_{FD} Q) \Longrightarrow C\ (\mu\ X.\ f\ X) \sqsubseteq_{FD} Q$$

Proposition 1 (CSP-Algebra) *HOL-CSP provides about 200 rules derived from the denotational semantics, be it monotonicities or equalities, which were called the "axioms" in the literature. We show here only the small collection used in the subsequent example proof:*

$(\forall\, y.\ c\ y \in S) \Longrightarrow c?x \to P\ x\ [\![S]\!]\ c?x \to Q\ x = c?x \to (P\ x\ [\![S]\!]\ Q\ x)$
$(\forall\, y.\ c\ y \in S) \Longrightarrow \mathit{inj}\ c \Longrightarrow c!a \to P\ [\![S]\!]\ c?x \to Q\ x = c!a \to (P\ [\![S]\!]\ Q\ a)$
$d\ a \notin S \Longrightarrow (\bigwedge y.\ c\ y \in S) \Longrightarrow d!a \to P\ [\![S]\!]\ c?x \to Q\ x = d!a \to (P\ [\![S]\!]\ c?x \to Q\ x)$
$d \in S \Longrightarrow (\bigwedge y.\ c\ y \notin S) \Longrightarrow d \to P\ [\![S]\!]\ c?x \to Q\ x = c?x \to (d \to P\ [\![S]\!]\ Q\ x)$
$d\ a \notin C \Longrightarrow c \in C \Longrightarrow c \to Q\ [\![C]\!]\ d!a \to P = d!a \to (c \to Q\ [\![C]\!]\ P)$
$\forall\, y.\ c\ y \notin B \Longrightarrow\ c?x \to P\ x \setminus B = c?x \to (P\ x \setminus B)$
$\forall\, y.\ c\ y \notin B \Longrightarrow c!a \to P \setminus B = c!a \to (P \setminus B)$
$c\ a \in B \Longrightarrow c!a \to P \setminus B = P \setminus B$ *etc.*

The theories `HOL-CSP` and `HOL-CSPM` [3] also add a number of extensions of the original language. This includes for the binary operators $P\ [\![A]\!]\ P'$, $P\ ;\ P'$, $P\ \Box\ P'$, $P\ \sqcap\ P'$, the generalizations $[\![S]\!]i \in\#\ M.\ P(i)$, $|||\ i \in\#\ M.\ P(i)$, etc. Roscoe's operators *Interrupt* $P \triangle P'$ and *Throw* (exception handler) $P\ \Theta\ a \in A.\ P'(a)$ have also been included since they come in handy in some of the more general constructions. Finally, [29] proposed another refinement ordering, the trace-divergence ordering $P \sqsubseteq_{DT} Q \equiv\ P \sqsubseteq_T Q \wedge \mathcal{D}\ Q \subseteq \mathcal{D}\ P$, which has surprisingly benign properties wrt. operational semantics and which is relevant for applications [10].

2.5 A Model and Sample Proof in HOL-CSP

Of course, proving refinements is not done by unfolding the definitions in the denotational semantics. Instead, the predominant proof technique is merely fixed point induction via Theorem 2, application of the algebraic rules of Proposition 1 as well as the monotonicity rules which are a consequence of Theorem 1. We demonstrate this with the paradigmatic `CopyBuffer` example, where we model a protocol *COPY* ("received data on channel *left* will eventually be copied into channel *right*") and an implementing *SYSTEM* which transfers the data from some *SEND*-component into some *REC*-component using an internal channel *mid* where *REC* acknowledges each data-package via a signal on the internal *ack*-channel.

The formalisation of these model-elements proceeds as follows. The events were defined by the inductive data-type introducing the channels:

$$\textbf{datatype}\ 'a\ channel = left\ 'a \mid right\ 'a \mid mid\ 'a \mid ack$$

Note that this definition leaves open what data is actually transmitted. A synchronisation set *SYN* is defined via $\{e \mid \exists x.\ e = mid\ x\} \cup \{ack\}$. The process *COPY* of type $'a\ channel\ process$ is defined by $\mu\ x.\ left?xa \rightarrow right!xa \rightarrow x$, the process *SEND* by $\mu\ x.\ left?xa \rightarrow mid!xa \rightarrow ack \rightarrow x$ and the process *REC* by $\mu\ x.\ mid?xa \rightarrow right!xa \rightarrow ack \rightarrow x$. The latter two are wired together to the process *SYSTEM* via $SYSTEM \equiv SEND\ [\![SYN]\!]\ REC \setminus SYN$.

Now we ask the question: *does SYSTEM implement the protocol COPY*? This can be rewritten as the following refinement problem : $COPY \sqsubseteq_{FD} SYSTEM$. Unfolding *COPY* and applying Theorem 2 yields the two subgoals:

1. $\bot \sqsubseteq_{FD} (SEND\ [\![SYN]\!]\ REC \setminus SYN)$
2. $\bigwedge x.\ x \sqsubseteq_{FD} (SEND\ [\![SYN]\!]\ REC \setminus SYN) \Longrightarrow$
   $left?a \rightarrow right!a \rightarrow x \sqsubseteq_{FD} (SEND\ [\![SYN]\!]\ REC \setminus SYN)$

where the former is trivial and the latter represents the induction step. If we unfold once *SEND* and *REC* and apply the reduction rules of Proposition 1, this results in:

$$left?a \rightarrow right!a \rightarrow x \sqsubseteq_{FD} left?a \rightarrow right!a \rightarrow (SEND\ [\![SYN]\!]\ REC \setminus SYN)$$

Applying the monotonicity rules resulting from Theorem 1 we can reduce this goal to the induction hypothesis $x \sqsubseteq_{FD} (SEND\ [\![SYN]\!]\ REC \setminus SYN)$.

Furthermore this proof can be highly automated (reduces to a few lines in Isabelle/Isar). No assumption is made over $'a$, this construction is therefore truly parametric over data, which is in stark contrast to model-checkers for CSP such as [1,28]. Using the fact that functions over processes are continuous, we can specify and analyse, e. g., global variables by $VAR\ \sigma_{init} \equiv (\mu\ x.\ (\lambda\sigma.\ (Read!\sigma \rightarrow x\ \sigma) \Box (Update?\sigma' \rightarrow x\ \sigma')))\ \sigma_{init}$ and other building blocks of concurrent programs like buffers, semaphores and monitors.

3 Deterministic Proc-Omata

3.1 Motivations

Refinements proofs as presented in Sect. 2.5 can quickly become counter-intuitive and fastidious, in particular, if concurrent systems were modeled that arise from iterative architectural compositions.[7] We propose a certain sub-class of CSP processes Proc-Omata to which many processes occurring in practice can be equivalently represented.

We start with another example, a simple counter of two events communicated by the environment:

Example 2 (Counter for integers) *Given two distinct events inc (increase) and dec (decrease), we can define a counter for integers as follows:*

$cnt \equiv \mu\ x.\ (\lambda n.\ (inc \rightarrow x\ (n + 1)) \ \Box\ (dec \rightarrow x\ (n - 1)))$

If inc and dec are of type $'\alpha$, cnt is of type int $\Rightarrow$ $'\alpha$ process, something we could call a higher-order process. In other words, we have defined for each integer a process whose relationship to the others can be illustrated using the following Labelled Transition System (LTS). Note that its non-symbolic presentation makes the state-space infinite.

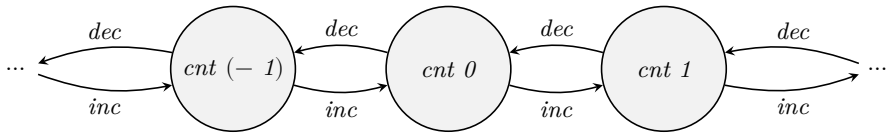

Fig. 2: LTS for the integer counter example

We observe that based on the algebraic properties of *Mprefix* and *Det*, and relying on the fact that *inc* and *dec* are distinct, we can rewrite our counter process as follows:

$cnt =$

$(\mu\ x.\ (\lambda n.\ \Box e \in \{dec, inc\} \rightarrow (\mathsf{if}\ e = inc\ \mathsf{then}\ x\ (n + 1)\ \mathsf{else}\ x\ (n - 1))))$

Note that the fix-point operator acts on a *function* of type *nat* $\Rightarrow$ *nat process*; since *nat process* belongs to the type class *cpo*, this implies that *nat* $\Rightarrow$ *nat process* belongs to the *cpo*-class. Thus, type-inference establishes that the fix-point exists and is — since the argument is continuous — unique. Thus, *cnt* is a process function that is parameterised in the initial state.

[7] In the CSP literature, the synchronous product $P \ [\![S]\!]\ Q$, hiding $P \setminus S$ and renaming were called the *architectural composition operators.*

3.2 Formal Definitions

We capture the intuition of the example above, by a formal definition of our notion of a deterministic *Proc-Omata*: it is a higher-order process canonically associated with a functional automaton [20]. Let us first define a deterministic automaton by a *record* in Isabelle/HOL.

Definition 1 (abstract syntax of a deterministic automaton)

$$\textbf{record}\ (\prime\sigma,\ \prime e)\ A_d = \tau :: \langle \prime\sigma \Rightarrow \prime e \Rightarrow \prime\sigma\ option\rangle$$

Isabelle/HOL records support a limited form of object-orientation; records are *extensible* (i. e. new fields my be added, while theorems established over an extensible record remain valid). We will exploit this feature in the following.

Definition 2 (enabledness) *Let A be an automaton of type $(\prime\sigma,\ \prime e)\ A_d$; we denote its transition function by $\tau\ A$. From the transition function, we derive the following notion:*

$$\varepsilon\ A\ s \equiv \{e \mid \tau\ A\ s\ e \neq None\}$$

Definition 3 (deterministic proc-Omata) *To an automaton A we associate a parametric process that we call a "Proc-Omata". It is a function of type $\prime\sigma \Rightarrow \prime e$ process that we denote by:*
$P[\![A]\!]_d \equiv \mu\ X.\ (\lambda s.\ \Box\ e \in \varepsilon\ A\ s \rightarrow X\ (the\ (\tau\ A\ s\ e)))$

Example 3 (Proc-Omata for the integer counter) *We associate to the process cnt presented in Example 2 the deterministic automaton:*
$A \equiv (\!|\tau = \lambda n\ e.\ if\ e = inc\ then\ Some\ (n + 1 :: int)\ else\ if\ e = dec\ then\ Some\ (n - 1)\ else\ None|\!)$
and prove that $cnt\ n = P[\![A]\!]_d\ n$.

When applying the operational rules of CSP to processes (cite ITP paper), our intuition that process proceeds by transiting between several "states" can be made explicit. In this paper, we use a simpler construction that just requires that they exist and are explicitely accessible.

3.3 Properties of Proc-Omata

As mentioned earlier, a deterministic Proc-Omata can be seen as a way of accessing the states of the process from which we built an automaton. This paves the way for using the underlying automaton for establishing indirectly properties about processes. The following formally proven results constitute bridges between CSP and automata theory.

The first notable thing is that the step function is continuous. We can consequently unfold the fixed point, leading to our first property.

Proposition 2 $P[\![A]\!]_d\ s = \Box\ e \in \varepsilon\ A\ s \rightarrow P[\![A]\!]_d\ (the\ (\tau\ A\ s\ e))$

Theorem 3 (Deterministic Proc-Omata are non terminating) *Using the predicate non-terminating defined in HOL-CSP for processes expressing that no trace is ending normally with ✓, we have:*

$$\textit{non-terminating}\ (P[\![A]\!]_d\ s)$$

Definition 4 (Reachability set of an automaton)

inductive-set $\mathcal{R}_d :: \langle('\sigma, 'e, '\alpha)\ A_d\text{-}scheme \Rightarrow '\sigma \Rightarrow '\sigma\ set\rangle\ (\langle\mathcal{R}_d\rangle)$
for $A :: \langle('\sigma, 'e, '\alpha)\ A_d\text{-}scheme\rangle$ **and** $s :: \langle'\sigma\rangle$
where $\textit{init} : \langle s \in \mathcal{R}_d\ A\ s\rangle$
| $\textit{step} : \langle t \in \mathcal{R}_d\ A\ s \Longrightarrow \textit{Some}\ u = \tau\ A\ t\ e \Longrightarrow u \in \mathcal{R}_d\ A\ s\rangle$

This definition captures the conventional concept of reacheability. It is a necessary prerequisite for some results. Here are two examples: a characterization of deadlock freeness and a computation of the events (the alphabet) of a process.

Theorem 4 (characterization of deadlock freeness)
$\textit{deadlock-free}\ (P[\![A]\!]_d\ s) = (\forall\ t{\in}\mathcal{R}_d\ A\ s.\ \varepsilon\ A\ t \neq \emptyset)$

Theorem 5 (alphabet characterization)
$\textit{events-of}\ (P[\![A]\!]_d\ s) = \bigcup\ (\varepsilon\ A\ `\ \mathcal{R}_d\ A\ s)$

Finally, an interesting yet not too surprising property is that deterministic Proc-Omata correspond to deterministic processes. The latter concept is defined in the CSP theory such that no continuation of a trace can be in a refusals set associated to it, i. e. $\textit{deterministic}\ P \equiv \forall\ s\ e.\ s\ @\ [e] \in \mathcal{T}\ P \longrightarrow (s, \{e\}) \notin \mathcal{F}\ P$. Since such processes are maximal for the $(\sqsubseteq_{FD})$ ordering, we only have to establish the following refinement for proving the equality.

Theorem 6 (FD refinement implies to equality) $P[\![A]\!]_d\ s \sqsubseteq_{FD} P \Longrightarrow P = P[\![A]\!]_d\ s$

Thus, establishing the core of the Proc-Omata theory requires some work, indeed. Now comes the benefit: the general theorems for compactification.

3.4 Compactification of Synchronization

It turns out that the Proc-Omata behave very well in synchronization contexts. The main idea is that given two deterministic automaton A_1 and A_2 of type $('\sigma, 'e)\ A_d$ and a synchronization set E (of type $'e\ set$), we *construct* a new deterministic automaton $A_0\ {}_d{\otimes}[\![E]\!]_{bin}\ A_1$ of type $('\sigma\ list, 'e)\ A_d$ such that

$P[\![A_0]\!]_d\ s_0\ [\![E]\!]\ P[\![A_1]\!]_d\ s_1 = P[\![A_0\ {}_d\otimes[\![E]\!]_{bin}\ A_1]\!]_d\ [s_0, s_1]$
under some assumptions on enableness-independence that we will discuss later.

With an intuitive inductive generalization, we end up with what we call a compactification theorem.

Theorem 7 (Compactification of deterministic Proc-Omata) *Assuming $|\sigma s| = |\sigma A|$ and a generalization of the hypothesis of independence on the enableness, we show:*

$[\![E]\!]\ (s, A) \in \#mset\ (zip\ \sigma s\ \sigma A).\ \ P[\![A]\!]_d\ s = P[\![_d\bigotimes[\![E]\!]\ \sigma A]\!]_d\ \sigma s$

If we say that this generalisation from the binary to the n-ary case is intuitive, this does by no means imply that the underlying proofs are straight-forward; this part of the Proc-Omata-theory took about NNN lines of highly dense proofs in Isabelle/HOL.

The importance of Theorem 7 lies in the fact that an interative synchronization of an arbitrary number of Proc-Omata can be reconstructed into an Proc-Omata, paving the way for invariant proof techniques in combination with what we evoked in Sect. 3.3.

Note that the order in which the Proc-Omata appears is arbitrary, but we have to track the associated state, and this is why the *zip* function seems to appear from nowhere.

Some basic ideas of this result already appeared in [30] from 2020. But instead of one transition function τ, the enableness ε had also to be defined independently (rather than having it as a derived concept). This made the construction more difficult to understand, and obscured the compactification result that was hidden inside a proof for a particular example. Moreover, only the case $E = \emptyset$ and $E = UNIV$ i. e. interleaving and parallelism had been discussed, while the above form of the compactification is suited for arbitrary synchronization sets.

As mentioned earlier, our result is always available as soon as we have the independence assumption on the enableness (we write the binary version here):
$\forall\, t_0 \in \mathcal{R}_d\ A_0\ s_0.\ \forall\, t_1 \in \mathcal{R}_d\ A_1\ s_1.\ \varepsilon\ A_0\ t_0 \cap \varepsilon\ A_1\ t_1 \subseteq E$
This is necessary because otherwise the process resulting of the synchronization would not be deterministic anymore.

4 Non Deterministic Proc-Omata

4.1 Formal Definitions

Instead of a *'σ option*, we may allow several states with the type *'σ set*. The **record** is modified accordingly.

Definition 5 $('\sigma, 'e)\ A_{nd} = \tau :: \langle '\sigma \Rightarrow 'e \Rightarrow '\sigma\ set\rangle$

This is almost the same thing as $('\sigma, 'e)\ A_d$, except that we adapt the definition of enableness (we override the notation):

Definition 6 $\varepsilon\ A\ s \equiv \{e \mid \tau\ A\ s\ e \neq \emptyset\}$

The associated non-deterministic Proc-Omata is also a function of type $'\sigma \Rightarrow\ 'e\ process$ that we denote by $P[\![A]\!]_{nd}$.

Definition 7 $P[\![A]\!]_{nd} \equiv \mu\ X.\ (\lambda s.\ \Box e \in \varepsilon\ A\ s \rightarrow \sqcap s' \in \tau\ A\ s\ e.\ X\ s')$

The only limitation of this is that the *GlobalNdet* operator involved in the definition is not continuous if $\tau\ A\ s\ e$ is not *finite*, and therefore the fixed point only makes sense if $\forall\ s\ e.\ finite\ (\tau\ A\ s\ e)$, which is encapsulated in the predicate *finite-trans A*. This is not only a restriction about Proc-Omata but more generally about the expressiveness of the framework `HOL-CSP` and `HOL-CSPM`, and will be resolved in a future publication.

4.2 Properties of Proc-Omata

We can obtain for non-deterministic Proc-Omata results similar to those obtained for the deterministic case. First of all, unfolding the fixed point is requiring the *finite-trans* assumption.

Proposition 3 $finite\text{-}trans\ A \Longrightarrow P[\![A]\!]_{nd}\ s = \Box\ e \in \varepsilon\ A\ s \rightarrow \sqcap s' \in \tau\ A\ s\ e.\ P[\![A]\!]_{nd}\ s'$

Theorem 8 (Non-deterministic Proc-Omata are non terminating)

$finite\text{-}trans\ A \Longrightarrow non\text{-}terminating\ (P[\![A]\!]_{nd}\ s)$

Of course, the notion of reachability has to be adapted.

Definition 8 (Reachability set of an automaton)

inductive-set $\mathcal{R}_{nd} :: \langle('\sigma, 'e, '\alpha)\ A_{nd}\text{-}scheme \Rightarrow '\sigma \Rightarrow '\sigma\ set\rangle\ (\langle\mathcal{R}_{nd}\rangle)$
for $A :: \langle('\sigma, 'e, '\alpha)\ A_{nd}\text{-}scheme\rangle$ **and** $s :: \langle'\sigma\rangle$
where $init : \langle s \in \mathcal{R}_{nd}\ A\ s\rangle$
| $step : \langle t \in \mathcal{R}_{nd}\ A\ s \Longrightarrow u \in \tau\ A\ t\ e \Longrightarrow u \in \mathcal{R}_{nd}\ A\ s\rangle$

Theorem 9 $finite\text{-}trans\ A \Longrightarrow deadlock\text{-}free\ (P[\![A]\!]_{nd}\ s) = (\forall\ t{\in}\mathcal{R}_{nd}\ A\ s.\ \varepsilon\ A\ t \neq \emptyset)$

Theorem 10 $finite\text{-}trans\ A \Longrightarrow events\text{-}of\ (P[\![A]\!]_{nd}\ s) = \bigcup\ (\varepsilon\ A\ `\ \mathcal{R}_{nd}\ A\ s)$

Actually, any deterministic Proc-Omata can be seen as a non-deterministic one via the injection that maps *None* to $\emptyset$ and *Some s* to $\{s\}$. Therefore, all our proofs (characterizations, compactifications) are done on non-deterministic Proc-Omata and the deterministic versions presented in Sect. 3 are corollaries.

4.3 Compactification of Synchronization

Thanks to the fact that non-deterministic choice distributes on all the CSP operators, and especially synchronization, we achieved to generalize the compactification Theorem 7 as follows.

Theorem 11 (Compactification of deterministic Proc-Omata) *Assuming* $|\sigma s| = |\sigma A|$ *and* $\forall\, A{\in}set\ \sigma A$. *finite-trans A, we show:*

$[\![E]\!]\ (s,\ A){\in}\#mset\ (zip\ \sigma s\ \sigma A).\ \ P[\![A]\!]_{nd}\ s = P[\![_{nd}\bigotimes [\![E]\!]\ \sigma A]\!]_{nd}\ \sigma s$

We get rid here of the independence hypothesis and, since any deterministic Proc-Omata can be seen as a non-deterministic one, this formula is a total generalization of Theorem 7.

5 Terminating Proc-Omata

For keeping the intuition, we hid some details under the hood saying that our automata were only described by a transition function in Definition 1 and Definition 5. Such Proc-Omata are *non-terminating* (cf Theorem 3 and Theorem 8), which prevent sequential composition (either we end with a deadlock, or we don't end at all, but never with *SKIP*). Since this may be useful for defining small blocks of complex architectures, we have extended our definitions.

5.1 Formal Definitions

Actually our **record**s (deterministic and non-deterministic) have an additional field: $\mathcal{S}_F$, which is the set of final states. Formally, we have:

Definition 9 $('\sigma,\ 'e)\ A_d = \tau :: \langle '\sigma \Rightarrow 'e \Rightarrow '\sigma\ option\rangle\ \mathcal{S}_F :: \langle '\sigma\ set\rangle$

Definition 10 $('\sigma,\ 'e)\ A_{nd} = \tau :: \langle '\sigma \Rightarrow 'e \Rightarrow '\sigma\ set\rangle\ \mathcal{S}_F :: \langle '\sigma\ set\rangle$

Morally when a final state is reached, we end the fixed point with *SKIP*. Formally:

Definition 11 $P_{SKIP}[\![A]\!]_d = (\mu\ X.\ (\lambda s.\ if\ s \in \mathcal{S}_F\ A\ then\ SKIP\ else\ \Box e{\in}\varepsilon\ A\ s \rightarrow X\ (the\ (\tau\ A\ s\ e))))$

Definition 12 $P_{SKIP}[\![A]\!]_{nd} = (\mu\ X.\ (\lambda s.\ if\ s \in \mathcal{S}_F\ A\ then\ SKIP\ else\ \Box e{\in}\varepsilon\ A\ s \rightarrow \sqcap s' \in \tau\ A\ s\ e.\ X\ s'))$

These definitions are of course generalizations of what we have in Sect. 3.2 and in Sect. 4.1, which are special cases when $\mathcal{S}_F\ A = \emptyset$. The fixed points can be unfolded in the same way as in Proposition 2 and Proposition 3.

5.2 Properties of Proc-Omata

These new versions of Proc-Omata enjoy almost the same properties as the previous ones, except that we now have to consider the possibility of terminating with *SKIP*. We will only write here some non-deterministic results, knowing that deterministic ones are straightforward corollaries thanks to the injection.

Theorem 12 *finite-trans* $A \Longrightarrow$ *non-terminating* $(P_{SKIP}[\![A]\!]_{nd}\ s) = (\mathcal{S}_F\ A \cap \mathcal{R}_{nd}\ A\ s = \emptyset)$

Theorem 13 $[\![$*finite-trans* A; *fin-states-not-enabled* $A]\!]$
$\Longrightarrow$ *deadlock-free*$_{SKIP}$ $(P_{SKIP}[\![A]\!]_{nd}\ s) = (\forall\, t{\in}\mathcal{R}_{nd}\ A\ s.\ t \in \mathcal{S}_F\ A \vee \varepsilon\ A\ t \neq \emptyset)$

The last one requires some explanation. *deadlock-free*$_{SKIP}$ P is a predicate on P defined as $(\mu\ x.\ (\sqcap xa{\in}UNIV \rightarrow\ x) \sqcap SKIP) \sqsubseteq_{FD} P$ and telling that P is either always making progress, either terminating with *SKIP*. In the other hand, *fin-states-not-enabled* A stipulates that $\varepsilon\ A\ s = \emptyset$ as soon as $s \in \mathcal{S}_F\ A$, which is intuitive but needs to be assumed since it is not a consequence of our formalization.

However, the most important result is probably the fact that we have managed to extend the compactification of synchronization to such Proc-Omata.

Theorem 14 $[\![|\sigma s| = |\sigma A|;\ \forall\, A{\in}$*set* $\sigma A.$ *finite-trans* $A;$
$\forall\, A{\in}$*set* $\sigma A.$ *fin-states-not-enabled* $A]\!]$
$\Longrightarrow [\![E]\!]\ (s,\ A){\in}\#$*mset* $(zip\ \sigma s\ \sigma A).\ \ P_{SKIP}[\![A]\!]_{nd}\ s = P_{SKIP}[\![_{nd}\bigotimes [\![E]\!]\ \sigma A]\!]_{nd}\ \sigma s$

5.3 Compactification of Sequential Composition

The extension of our formalization for allowing terminating Proc-Omata was actually motivated by the fact that *SKIP* is the neutral element for the sequential composition. In other words we have $P\ ;\ SKIP = P$ and $SKIP\ ;\ P = P$. This makes it possible to split big architectures into smaller sub-components, which can potentially be Proc-Omata. But why should we only consider the binary case? For the *MultiSeq* operator, we proved a compactification theorem in the same philosophy as Theorem 14.

Theorem 15 $[\![\sigma A \neq [];\ |\sigma s| = |\sigma A|;\ \forall\, A{\in}$*set* $\sigma A.$ *finite-trans* $A;$
fin-states-not-enabled $(last\ \sigma A)]\!]$
$\Longrightarrow$ *SEQ* $(s,\ A){\in}@zip\ \sigma s\ \sigma A.\ P_{SKIP}[\![A]\!]_{nd}\ s = P_{SKIP}[\![_{nd}\bigotimes ;\ \sigma A]\!]_{nd}\ \sigma s$

6 Examples

6.1 Dining Philosophers

A good illustration of the power of Proc-Omata reasoning is the Dining Philosophers example, first introduced by Dijkstra in 1965 and reformulated to the present form by Hoare [12].

Let is assume that N philosophers are dining around a round table, each one sharing his right and left fork with his right and left neighbour respectively. Of course, to be able to eat, a philosopher needs its two forks, which may result in a deadlock. This will become clearer with the formalization in `HOL-CSP`.

We first start by a **datatype** for the channels.

Definition 13
datatype *dining-event* = *picks* (*phil*:*nat*) (*fork*:*nat*)
| *putsdown* (*phil*:*nat*) (*fork*:*nat*)

Then we can define a right-handed philosopher and a fork as fixed points.

Definition 14 $RPHIL\ i \equiv \mu\ x.\ picks\ i\ i \rightarrow picks\ i\ ((i - 1)\ mod\ N) \rightarrow putsdown\ i\ ((i - 1)\ mod\ N) \rightarrow putsdown\ i\ i \rightarrow x$

Definition 15 $FORK\ i \equiv \mu\ x.\ (picks\ i\ i \rightarrow putsdown\ i\ i \rightarrow x) \ \square\ (picks\ ((i + 1)\ mod\ N)\ i \rightarrow putsdown\ ((i + 1)\ mod\ N)\ i \rightarrow x)$

Now, the philosophers are interleaved, so are the forks, and the resulting global process is obtain by parallelizing philosophers and forks.

Definition 16 (Architecture of Dining Philosophers)

$RPHILS \equiv |||\ P{\in}\#mset\ (map\ RPHIL\ [0..{<}N]).\ \ P$
$FORKS \equiv |||\ P{\in}\#mset\ (map\ FORK\ [0..{<}N]).\ \ P$
$RDINING = FORKS\ ||\ RPHILS$

Reasoning about the possibility of deadlock for such a system can be difficult: you do not really know where to start. But if we can find automata so that our processes may be written in terms of deterministic Proc-Omata, we inherit the powerful theorems seen in Sect. 3.

This can be achieved for the forks and the right-handed philosophers. With the appropriate definitions we prove $FORK\ i = P[\![fork\text{-}A\ i]\!]_d\ 0$ and $RPHIL\ i = P[\![rphil\text{-}A\ i]\!]_d\ 0$. Finally, using the compactification Theorem 7, we obtain a big Proc-Omata for *RDINING*.

Theorem 16 $RDINING = P[\![_d\bigotimes [\![\emptyset]\!]\ (map\ rphil\text{-}A\ [0..{<}N])\ _d\otimes [\![UNIV]\!]_{bintuple}\ _d\bigotimes [\![\emptyset]\!]\ (map\ fork\text{-}A\ [0..{<}N])]\!]_d\ (replicate\ N\ 0,\ replicate\ N\ 0)$

We can then show that (*replicate N 1*, *replicate N 1*) is reachable from the initial state (*replicate N 0*, *replicate N 0*), and that in this state our Proc-Omata has its enableness empty. This corresponds to the situation where each philosopher has picked his right fork, thus no left fork is available. With the characterization Theorem 4, we have proven that *RDINING* is not deadlock free.

We can correct this by introducing a left-handed philosopher, who picks his left fork before his right one as follows.

Definition 17 $LPHIL0 \equiv \mu\ x.\ picks\ 0\ (N - 1) \rightarrow picks\ 0\ 0 \rightarrow putsdown\ 0\ 0 \rightarrow putsdown\ 0\ (N - 1) \rightarrow x$

Definition 18 (Corrected architecture of Dining Philosophers)

$PHILS \equiv |||\ P{\in}\#add\text{-}mset\ LPHIL0\ (mset\ (map\ RPHIL\ [1..{<}N]))).\ \ P$
$DINING = FORKS\ ||\ PHILS$

Once compactified, a proof by invariant shows that the enableness in every reachable state is never empty, which results in the following theorem.

Theorem 17 *deadlock-free DINING*

6.2 Copy Buffer with a Queue

We just saw an example where Proc-Omata are the ideal tool. Here we would like to present a case where things are not going so well.

We want to consider a buffer with a queue where the data transmitted by the sender is stored until the recipient actually receives it.

We start with a **datatype**.

Definition 19
datatype $'a\ chan = left\ 'a \mid enqueue\ 'a \mid dequeue\ 'a \mid right\ 'a$

Intuitively we define the sender and the recipient, and synchronize them with the queue as follows.

Definition 20 (Definitions for the Copy Buffer with a queue)

$send \equiv \mu\ X.\ left?x \rightarrow enqueue!x \rightarrow X$

$rec \equiv \mu\ X.\ dequeue?x \rightarrow right!x \rightarrow X$

$queue \equiv \mu\ X.\ (\lambda L.\ (enqueue?x \rightarrow X\ (x\ \#\ L)) \Box$
$(if\ L = []\ then\ STOP\ else\ dequeue!last\ L \rightarrow X\ (butlast\ L)))$

$$\mathit{QueueBuffer\text{-}pre}\ L \equiv \mathit{send}\ [\![\mathit{range\ enqueue}]\!]\ (\mathit{queue}\ L\ [\![\mathit{range\ dequeue}]\!]\ \mathit{rec})$$

$$\mathit{QueueBuffer}\ L \equiv \mathit{QueueBuffer\text{-}pre}\ L \setminus \mathit{range\ enqueue} \cup \mathit{range\ dequeue}$$

We can write *send*, *rec* and *queue* as Proc-Omata without much difficulty, and therefore *QueueBuffer-pre* with compactification. As a consequence of Theorem 4, we immediately obtain that *QueueBuffer-pre* is deadlock free. But when it comes to *QueueBuffer*, because of the *Hiding*, we can no longer express it as a Proc-Omata and consequently we lose the characterization with enableness. But actually, using the Proc-Omata obtained for *QueueBuffer-pre*, we can still apply a fixed point induction in which we let the state free. This allows us to only look one step further and prove relatively easily that *QueueBuffer* is deadlock free.

7 Future Work

The previous example underlined the fact that the theory we developed can be useful even when we do not achieve to express the whole system as a Proc-Omata. Indeed, be able to see some sub-components as automata is already enough to simplify induction proofs, derive properties on the alphabet (*events-of*), etc. Actually the step-laws and the distributivity of internal choice allowed us to establish for some operators powerful properties. For example, for *Throw* and *Interrupt*, we have the following theorems.

Theorem 18 *finite-trans* $A \Longrightarrow$
$\mathit{Throw}\ (P_{SKIP}[\![A]\!]_{nd}\ s)\ B\ Q =$
$(\mathsf{if}\ s \in \mathcal{S}_F\ A\ \mathsf{then}\ \mathit{SKIP}$
$\ \mathsf{else}\ \Box e{\in}\varepsilon\ A\ s$
$\quad \rightarrow (\mathsf{if}\ e \in B\ \mathsf{then}\ Q\ e$
$\quad\quad \mathsf{else}\ \sqcap\ s'{\in}\tau\ A\ s\ e.\ \ \mathit{Throw}\ (P_{SKIP}[\![A]\!]_{nd}\ s')\ B\ Q))$

Theorem 19 *finite-trans* $A \Longrightarrow$
$P_{SKIP}[\![A]\!]_{nd}\ s \triangle Q =$
$Q \Box (\mathsf{if}\ s \in \mathcal{S}_F\ A\ \mathsf{then}\ \mathit{SKIP}\ \mathsf{else}\ \Box e{\in}\varepsilon\ A\ s \rightarrow \sqcap\ s'{\in}\tau\ A\ s\ e.\ \ P_{SKIP}[\![A]\!]_{nd}\ s' \triangle Q)$

For the external choice, when the Proc-Omata are non terminating, we can obtain something similar to the compactification.

Theorem 20 $[\![$*finite s-A-set*; $\bigwedge s\ A.\ (s,\ A) \in$ *s-A-set* $\Longrightarrow$ *finite-trans* $A]\!]$
$\Longrightarrow \Box(s,\ A){\in}$*s-A-set*. $\ P[\![A]\!]_{nd}\ s =$
$\quad \Box e{\in}\bigcup(s,\ A){\in}$*s-A-set* $\varepsilon\ A\ s$
$\quad\quad \rightarrow \sqcap\ (s',$
$\quad\quad\quad A){\in}\{uu\text{-} \mid \exists\, s'\ s\ A.$
$\quad\quad\quad\quad\quad uu\text{-} = (s',\ A) \wedge$
$\quad\quad\quad\quad\quad (s,\ A) \in$ *s-A-set* $\wedge\ e \in \varepsilon\ A\ s \wedge s' \in \tau\ A\ s\ e\}.$
$\quad\quad\quad P[\![A]\!]_{nd}\ s'$

As a future work, it should be interesting to extend such properties or even maybe find a way to compactify other operators. This would require modifying the formalism (for example the external choice can be resolved, that is to say some Proc-Omata should disappear) and this is something we did not achieve yet.

Another future work could be to connect our Proc-Omata formalization with a model checker such as Cubicle [9]. Indeed we traduced for example deadlock freeness into a predicate over the reachable states of an automaton which size is finite but unbounded. Such tools are designed to deal with this kind of parameterized systems so the connection should be possible, resulting in the possibility of quickly check for counter examples and even maybe construct invariants that we could use later in a verified Isabelle proof.

Lastly, our formalization lends itself very well to sub-component breakdown. A future project we have already started is to create a library of elementary bricks for which we define and establish the correspondence with Proc-Omata. Such bricks shall be assembled together, becoming bigger Proc-Omata thanks to compactification theorems.

8 Related Work

In the introduction, we claimed that HOL-CSP is arguably the most *comprehensive* formalization of CSP; here, we'd like to substantiate this claim.

The theory of CSP has attracted a lot of interest since the eighties and nineties, both as a theoretical device as well as a modelling language to analyze complex concurrent systems. A wealth of theoretical articles appeared to investigate certain fragments and extensions of the core framework; it is therefore not surprising that attempts to their formalisations have been undertaken with the advent of interactive proof assistants.

Most noteworthy to these attempts is an early CSP trace semantics model in HOL System proposed by [8]. Its successor [7] presented a first failure-divergence semantics for a restricted set of operators and used the notion of a universal (polymorphic) alphabet[8]. Note that [32] tackled already with subtle difficulties concerning *is-process* and ✓.

The tool CSP-Prover [15] — based on a deep embedding of CSP in an Isabelle/HOL theory on the stable failures model — allows for the refinement verification [15] by using some automated support for induction. However, only if a process is divergence-free, its failures are the same as its stable failures. In our view, this is a too strong assumption for both a theory as well as a practical tool.

In the past few years, CSP benefited from a renewed interest with proof assistants. CSP Agda was introduced in 2026 [13] with an implementation quite

[8] Our first attempts for HOL-CSP [32] are based on a extended version of this theory ported to Isabelle/HOL

different from HOL-CSP since it is based on coinductive data types. Only trace and stable failures semantics have been covered so far, and the library of proven laws is fairly modest [14]. In 2020, an operational semantics of CSP in Coq was introduced [11] by a direct definitional approach. The theory covers only trace refinement and a subset of CSP's operators, but offers rather well-developed proof automation for this language fragment close to conventional automata theory.

With respect to all these formalizations in HOL, we would like to remind the importance of the general fact that invariants (like *is-process*) or bridge theorems (like Hoare's $P \rightsquigarrow_e Q \equiv P \sqsubseteq_{FD} (e \rightarrow Q) \,\Box\, P$) do not simply generalise from one fragment to the next, and that features which are well studied in one fragment are not necessarily well-understood in the whole picture. It is our main contribution to provide an integrated formal theory that tackles with the complexities of the necessary generalisations. This involved a revision of the role of the *After* operator in the entire theory.

In the late nineties, research focused on automated verification tools for CSP, most notably on FDR (see [1] for the latest instance). It relies on an operational CSP semantics, that allows for a conversion of processes into labelled transition systems, where the states were normalized by the "axioms" derived from the denotational semantics. For finite event sets, FDR can reduce refinement proofs to bisimulation problems. With efficient compression techniques, state-elimination and factorization by semantic equivalence [25], FDR was successful in analysing some industrial applications. However, such a model checker can never handle infinite cases. Another similar model checking tool [28] implemented some more optimization techniques, such as partial order reduction, symmetric reduction, and parallel model checking, but is also restricted to the finite case. In a way, these tool require for their foundation integrated denotational/algebraic/operational techniques as provided by our theory.

Attempts to find characterizations of processes to generalise finite results to infinite ones by *data-independence* [17,2,24], a variant of parametric model-checking, have seen only a limited success. Roscoe developed a data independent technology to verify security protocols modelled with CSP/FDR, which allows the node to call infinite fresh values for nonces, thus infinite sequence of operations [24]. An extension of this work was proposed in [2] using the script language CSP_M. However, in their works, even though each agent in the security protocol can perform infinite number of operations, the number of agent entities remains finite. HOL-CSP satisfies the need to parameterization and high-order processes naturally [30] as a consequence of their *pcpo*-type structure. A formalization and theory development of CSP_M has been undertaken [3] but is out of the scope of this paper.

9 Conclusion

After introducing the framework HOL-CSP, we presented a formalization of Proc-Omata, a bridge that allows us to see processes as automata providing powerful proof techniques. We actually presented several motivations and vari-

ants with increasing levels of abstraction before giving an overview of the key results: the compactification theorems. We illustrated this with two examples: the Dining Philosophers where the situation of a parameterized unbounded ring of processes fits our theory perfectly, and the Copy Buffer with a queue where we can still use Proc-Omata on sub-components. This motivates the future works that we presented: tend to generalize the formalism in order to include more operators, and connect external tools for quick counter examples, help in the construction of invariants, etc.

References

1. FDR4 - The CSP Refinement Checker. https://www.cs.ox.ac.uk/projects/fdr/ (2019)
2. An, J., Zhang, L., You, C.: The design and implementation of data independence in the csp model of security protocol. Advanced Materials Research **915-916**, 1386–1392 (04 2014). https://doi.org/10.4028/www.scientific.net/AMR.915-916.1386
3. Ballenghien, B., Taha, S., Wolff, B.: HOL-CSPM - architectural operators for HOL-CSP. Archive of Formal Proofs **2023** (2023), https://www.isa-afp.org/entries/HOL-CSPM.html
4. Ballenghien, B., Wolff, B.: Operational semantics formally proven in hol-csp. Archive of Formal Proofs (December 2023), https://isa-afp.org/entries/HOL-CSP_OpSem.html
5. Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A theory of communicating sequential processes. J. ACM **31**(3), 560–599 (1984)
6. Brookes, S.D., Roscoe, A.W.: An improved failures model for communicating sequential processes. In: Brookes, S.D., Roscoe, A.W., Winskel, G. (eds.) Seminar on Concurrency. pp. 281–305. Springer, Berlin, Heidelberg (1985)
7. Camilleri, A.J.: A higher order logic mechanization of the csp failure-divergence semantics. In: Birtwistle, G. (ed.) IV Higher Order Workshop, Banff 1990. pp. 123–150. Springer, London (1991)
8. Camilleri, A.J.: Mechanizing CSP trace theory in higher order logic. IEEE Trans. Software Eng. **16**(9), 993–1004 (1990)
9. Conchon, S., Goel, A., Krstic, S., Mebsout, A., Zaïdi, F.: Cubicle: A parallel smt-based model checker for parameterized systems - tool paper. In: Madhusudan, P., Seshia, S.A. (eds.) Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings. Lecture Notes in Computer Science, vol. 7358, pp. 718–724. Springer (2012). https://doi.org/10.1007/978-3-642-31424-7_55, https://doi.org/10.1007/978-3-642-31424-7_55
10. Crisafulli, P., Taha, S., Wolff, B.: Modeling and analysing cyber-physical systems in HOL-CSP. Robotics Auton. Syst. **170**, 104549 (2023). https://doi.org/10.1016/J.ROBOT.2023.104549, https://doi.org/10.1016/j.robot.2023.104549
11. da Silva Carvalho de Freitas, C.A.: A theory for communicating, sequential processes in coq (2020), https://api.semanticscholar.org/CorpusID:259373665
12. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1985)
13. Igried, B., Setzer, A.: Programming with monadic csp-style processes in dependent type theory. In: Proceedings of the 1st International Workshop on Type-Driven

Development. p. 28–38. TyDe 2016, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2976022.2976032, https://doi.org/10.1145/2976022.2976032
14. Igried, B., Setzer, A.: Trace and stable failures semantics for csp-agda. arXiv preprint arXiv:1709.04714 (2017)
15. Isobe, Y., Roggenbach, M.: A complete axiomatic semantics for the CSP stable-failures model. In: CONCUR 2006 - Concurrency Theory, 17th International Conference, Bonn, Germany, August 27-30, 2006. pp. 158–172 (2006)
16. Isobe, Y., Roggenbach, M.: Csp-prover: a proof tool for the verification of scalable concurrent systems. Information and Media Technologies **5**(1), 32–39 (2010). https://doi.org/10.11185/imt.5.32
17. Lazic, R.S.: A semantic study of data-independence with applications to the mechanical verification of concurren. Ph.D. thesis, University of Oxford (1999)
18. Müller, O., Nipkow, T., von Oheimb, D., Slotosch, O.: HOLCF = HOL + LCF. j-fp **9**(2), 191–223 (1999). https://doi.org/10.1017/S095679689900341X
19. Nipkow, T.: Verified lexical analysis. In: Grundy, J., Newey, M.C. (eds.) Theorem Proving in Higher Order Logics, 11th International Conference, TPHOLs'98, Canberra, Australia, September 27 - October 1, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1479, pp. 1–15. Springer (1998). https://doi.org/10.1007/BFB0055126, https://doi.org/10.1007/BFb0055126
20. Nipkow, T.: Functional automata. Arch. Formal Proofs **2004** (2004), https://www.isa-afp.org/entries/Functional-Automata.shtml
21. Noce, P.: Conservation of CSP noninterference security under sequential composition. Archive of Formal Proofs (2016), https://www.isa-afp.org/entries/Noninterference_Sequential_Composition.shtml
22. Åman Pohjola, J., Myreen, M.O., Tanaka, M.: A hoare logic for diverging programs. Archive of Formal Proofs (January 2023), https://isa-afp.org/entries/HoareForDivergence.html, Formal proof development
23. Roscoe, A.W.: An alternative order for the failures model. J. Log. Comput. **2**, 557–577 (1992)
24. Roscoe, A.W., Broadfoot, P.J.: Proving security protocols with model checkers by data independence techniques. Journal of Computer Security **7**(1), 147–190 (1999)
25. Roscoe, A.W., Gardiner, P.H.B., Goldsmith, M.H., Hulance, J.R., Jackson, D.M., Scattergood, J.B.: Hierarchical compression for model-checking csp or how to check 1020 dining philosophers for deadlock. In: Tools and Algorithms for the Construction and Analysis of Systems. pp. 133–152. Springer Berlin Heidelberg, Berlin, Heidelberg (1995)
26. Roscoe, A.: Theory and Practice of Concurrency. Prentice Hall (1997)
27. Scott, D.: Continuous lattices. In: Lawvere, F.W. (ed.) Toposes, Algebraic Geometry and Logic. pp. 97–136. Springer, Berlin, Heidelberg (1972)
28. Sun, J., Liu, Y., Dong, J.S., Pang, J.: Pat: Towards flexible verification under fairness. Lecture Notes in Computer Science, vol. 5643, pp. 709–714. Springer (2009)
29. Taha, S., Wolff, B., Ye, L.: The HOL-CSP refinement toolkit. Arch. Formal Proofs **2020** (2020), https://www.isa-afp.org/entries/CSP_RefTK.html
30. Taha, S., Wolff, B., Ye, L.: Philosophers may dine - definitively! In: Dongol, B., Troubitsyna, E. (eds.) Integrated Formal Methods - 16th International Conference, IFM 2020, Lugano, Switzerland, November 16-20, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12546, pp. 419–439. Springer (2020). https://doi.org/10.1007/978-3-030-63461-2_23, https://doi.org/10.1007/978-3-030-63461-2_23

31. Taha, S., Ye, L., Wolff, B.: HOL-CSP Version 2.0. Archive of Formal Proofs (Apr 2019), http://isa-afp.org/entries/HOL-CSP.html
32. Tej, H., Wolff, B.: A corrected failure divergence model for CSP in Isabelle/HOL. In: Fitzgerald, J.S., Jones, C.B., Lucas, P. (eds.) Formal Methods Europe (FME). LNCS, vol. 1313, pp. 318–337. Springer (1997). https://doi.org/10.1007/3-540-63533-5_17