A Theory of Proc-Omata—and Proof Methods for Process Architectures

Benoît Ballenghien and Burkhart Wolff(✉)

LMF, Université Paris-Saclay, Paris, France
benoit.ballenghien@universite-paris-saclay.fr, 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 terminating and non-terminating Proc-Omata, both their relation to conventional CSP processes as well as possible transformation operations on them. As an application of 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 · CSP

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 by Tony Hoare, and detailed in a book in 1985 [15], but has since evolved substantially [7,8,27]. 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*).

C. Anutariya and M. M. Bonsangue (Eds.): ICTAC 2024, LNCS 15373, pp. 1–18, 2024.
https://doi.org/10.1007/978-3-031-77019-7_16

Several attempts have been undertaken to formalize this fairly complex theory, notably [16,19,24,32]. The presented work here is based on HOL-CSP [3,4,6,29,31], 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:

- type 'a trace is constructed over an arbitrary type 'a in HOL, paving the way for events carrying dense-time, vector-spaces, etc.[1],
- in general, HOL-CSP attempts to remove finiteness-restrictions, and
- the semantic domain is encapsulated in the type 'a process belonging to the class of *complete partial orders* (cpo's). Process patterns are functions in higher-order logic (HOL), and thus first-class citizens.

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 [22,23] inside which can have an infinite state and communication alphabet. For certain process-patterns such as an i-indexed family of interleaving processes ||| i ∈# M. P i, it is possible to convert this pattern into a Proc-Omaton 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; non-terminating and potentially terminating ones.
Now, a Proc-Omaton has the general form of a CSP process schema:

$$\mu\ \mathrm{X}.\ (\lambda\sigma.\ \Box \mathrm{e} \in \varepsilon\ \mathrm{A}\ \sigma \rightarrow \mathrm{F}\ (\tau\ \mathrm{A}\ \sigma\ \mathrm{e})\ \mathrm{X})$$

where μ is the recursion operator over process functions (here: parameterized over an internal state σ), $\Box$ a choice-operator ranging over a set of events, τ A is the transition function of the automaton A, ε A σ computes the set of events for which A is *enabled* (ready to make a transition) in the state σ and the function F depends on whether A is non-terminating or potentially terminating (see Sect. 3 and Sect. 4). In all cases, the resulting fixed point is a function of type '$\sigma \Rightarrow$ 'e process. When P is a Proc-Omaton, classic CSP theory gives us from some state σ the definitions for the set of *traces* $\mathcal{T}$ (P σ), the set of *failures* $\mathcal{F}$ (P σ) and the set of *divergences* $\mathcal{D}$ (P σ). The latter is always empty: Proc-Omata have no divergences.

At first glance, one might think that this concept is too restrictive to be useful in practice. A closer look reveals that the contrary is actually the case, as the following example illustrates:

Example 1. *Consider the Collatz Process:*

$$\begin{aligned}\text{Collatz} \equiv \mu\ \mathrm{X}.\ (\lambda \mathrm{n}.\quad & (\Box \mathrm{x} \in \{0, 1\} \cap \{\mathrm{n}\} \rightarrow \mathrm{SKIP}) \\ & \Box\ (\Box \mathrm{x} \in (\mathrm{Even} - \{0\}) \cap \{\mathrm{n}\} \rightarrow \mathrm{X}\ (\mathrm{x}\ \mathrm{div}\ 2)) \\ & \Box\ (\Box \mathrm{x} \in (\mathrm{Odd} - \{1\}) \cap \{\mathrm{n}\} \rightarrow \mathrm{X}\ (3 * \mathrm{x} + 1)))\end{aligned}$$

[1] Or even differential equations as in cyber-physical system models [11].

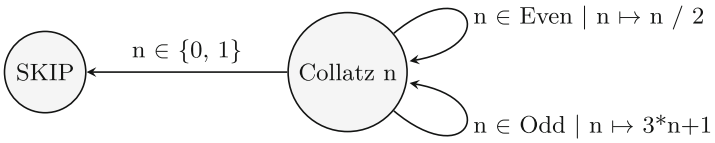

Fig. 1. The Collatz Function seen as Process

Note that the Collatz function is unknown to be terminating; a standard definition in HOL as a recursive function is therefore out of reach. This example shows, however, that it is perfectly possible to represent the Collatz process as a recursive HOL-CSP process, since HOL-CSP is built for modeling potentially non-terminating computations. Figure 1 represents its behaviour as a symbolic Labelled Transition System (LTS), which has the format of an extended finite state machine (EFSM) where the underlying Proc-Omaton is potentially terminating (see Sect. 4). The resulting Proc-Omaton transition definitions are a tedious but direct translation of the symbolic LTS above:

$$\tau\ \text{Collatz\_A}\ \lozenge\ \text{e} = \lozenge$$

$$\tau\ \text{Collatz\_A}\ \lfloor \text{n} \rfloor\ \text{e} = (\text{if n} = \text{e then if n} \in \{0, 1\} \text{ then } \lfloor \lozenge \rfloor \text{ else if even n then } \lfloor \lfloor \text{n div } 2 \rfloor \rfloor \text{ else } \lfloor \lfloor 3 * \text{n} + 1 \rfloor \rfloor \text{ else } \lozenge)$$

where $\lozenge$ *and* $\lfloor\_\rfloor$ *are a notation for* None *and* Some *of the* 'a option*-type.*

The example above gives rise to a particular proof-methodology which is depicted in Fig. 2 and Fig. 3. First, we construct a Proc-Omaton and prove that it is equivalent to the initial process; this *conversion* proof can be done via fixed point induction (see Theorem 2) or sometimes by model-checking. Second, we apply the aforementioned compactification theorems over Proc-Omata which trades so to speak the complexity of the underlying LTS of the process into the complexity of the data space of the automaton. Third, we can prove properties over the compactified Proc-Omaton by classical invariant reasoning.

We proceed as follows. After an introduction to "classic" CSP and our extension HOL-CSP and HOL-CSPM in Isabelle/HOL, we present the core-constructions of this paper: formal definitions of terminating and non-terminating Proc-Omata, a number of basic and advanced theorems over them, and the compactification theorems allowing to internalize compositions of Proc-Omata. Finally we illustrate this with some examples.

HOL-CSP [31] and HOL-CSPM [3] are published in the Archive of Formal Proofs AFP. Wrt. the session HOL-CSP_Proc-Omata see the developer version https://gitlab.lisn.upsaclay.fr/burkhart.wolff/hol-csp2.0/. Note that our formal theories cover also non-deterministic versions of Proc-Omata; however, their detailed presentation is out of scope of this paper due to space limitations.

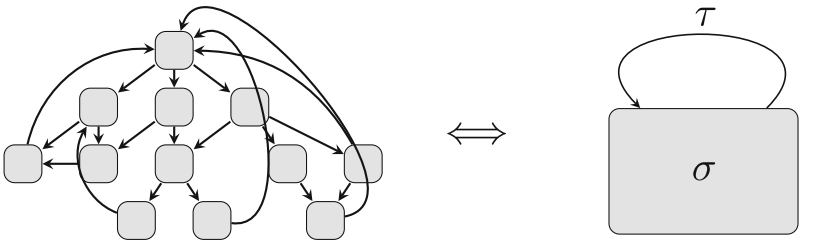

Fig. 2. Conversion of an LTS

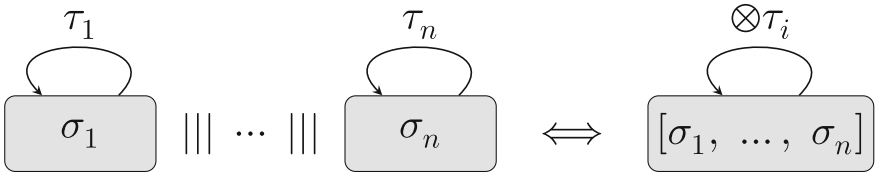

Fig. 3. Compactification

2 Background

2.1 Classic CSP Syntax

At a glance, the fragment of the classic CSP core language we will be using in this paper reads as follows:

P ::= SKIP | STOP | P □ P' | P ⊓ P' | P ⟦A⟧ P' | P ; P' | P \ A
| a → P | □a ∈ A → P a | Renaming P g | μ X. f X

SKIP signals termination and STOP denotes a deadlock.

Two choice operators are distinguished: the *external* one _□_ forces a process "to follow" whatever its context requires and the *internal* one _⊓_ imposes on the context of a process "to follow" the non-deterministic choices made.

The later is generalized to unbounded non-determinism: ⊓a ∈ A. P a where A may be infinite at the price of loosing continuity (which has no incidence here since non-deterministic versions of Proc-Omata are out of scope of this paper).

From the former and the prefix operator a → P which signals a (where a is an element of a set Σ of events) and continues with P, the multi-prefix deterministic choice □a∈A → P a is constructed. When events are tagged with *channels*, i. e.Σ = CHANNELS × DATA, syntactic sugar like c**?**x→P x or c**!**x→P x is added; the former reads intuitively as "x is read from channel c" while the latter means "x is sent into c" (where c ∈ CHANNELS and x ∈ DATA).

The sequential composition P **;** P' behaves first like P and, once it has successfully terminated, like P'. We denote by P \ A the process obtained from P after hiding the events of the set A. Similarly, Renaming P g is a process in which each event e of P was renamed in g(e).

The fixed point μ X. f X operator provides a solution to P = f P (but requires precautions, see Sect. 2.4). The synchronized product P ⟦A⟧ P' is a primitive for all communication. It is abbreviated as P ||| P' when A = $\emptyset$ (interleaving) or P || P' when A = UNIV (parallel), where UNIV :: 'a set is the universal set over the type 'a. Last but not least, the multi-prefix non-deterministic choice appearing in the definition of deadlock freeness (cf Theorem 4) is simply defined as $\sqcap$a ∈ A → P a = $\sqcap$a ∈ A. a → P a.

2.2 Classic CSP Semantics

The denotational semantics (following [27]) 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. Obviously, ✓ should only appear at the end of a trace (i. e. traces are 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, [7] 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) ∈ $\mathcal{F}$ P which is a pair of a trace t and a set of refusals X. Finally, [7] 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[2]. 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.

While Hoare Logics is a framework to reason over terminating calculations, CSP and process refinement are designed to reason over non-terminating ones. Several variants of refinement were considered, but the most important one is 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. Moreover, this is a partial order, thus allowing proofs by double refinement.

2.3 Theories 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

[2] Also called *infinite internal chatter* as occurring in processes like μ x. a → x \ {a}.

(kernel) to increase the trustworthiness of proofs. The Isabelle distribution comes with a number of library theories constructed solely from definitional axioms; among them theories for sets, lists, arithmetics, and analysis.

A particularly relevant library-theory is HOLCF Scott domain theory [21,28] providing a particular type class for *pointed complete partial orders* 'a, i.e. the class of types 'a which posses a least element $\bot$ and a complete partial order $\_ \sqsubseteq \_$. The type-system uses type-classes to infer automatically that if 'b is a pcpo, then the function space 'a $\Rightarrow$ 'b is also a pcpo.

For types of pcpo, HOLCF provides a theory of *continuity*, the concept of *admissibility*, the *fixed point induction* and the *least fixed point operator* μ x. f x.

2.4 Isabelle/HOL-CSP

Isabelle/HOL-CSP is a shallow embedding of CSP in HOL based on the traditional semantic domain described by nine well-formedness conditions (that we omit here) over the three semantic functions $\mathcal{T}$ :: 'a process $\Rightarrow$ 'a trace set, $\mathcal{F}$ :: 'a process $\Rightarrow$ 'a failure set and $\mathcal{D}$ :: 'a process $\Rightarrow$ 'a trace set expressing well behaviour for processes. The core of HOL-CSP is to encapsulate wellformedness into a type definition. This is achieved by via the specification construct:

typedef 'a process = "{P :: 'a process$_0$. is_process P}"

creating a new type which is isomorphic to the subset of 'a process$_0$'es satisfying the predicate is_process capturing the well-formedness conditions, where 'a process$_0$ is an abbreviation for 'a failure set $\times$ 'a divergence set. Subsequently, we define each CSP operator in terms of 'a process$_0$ and lift them to 'a process by proving the preservation of the is_process-invariant (thus formalizing [27]). 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 'a set $\Rightarrow$ ('a $\Rightarrow$ 'a process) $\Rightarrow$ 'a process.

In order to give semantics to the fixed point operator (and thus access to the theory HOLCF), it is shown that 'a process belongs to the type class pcpo, which also gives higher-order functions over processes a pcpo structure. Recall that Scott domains provides semantics for the fixed point operator μ X. f X only under the condition that f is continuous wrt. a complete partial ordering. Since the natural ordering $\_ \sqsubseteq_{FD} \_$ is too weak for this purpose, Roscoe and Brookes [25] 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 (see session HOL-CSP for technical details). With this ordering, the following core theorem is established:

Theorem 1 (Continuity). *Almost every operator* $\otimes$ *is continuous i. e.:*

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

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 continuous functions* f*, we have:*

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

Proposition 1 (CSP-Algebra). *HOL-CSP provides about 200 rules derived from the denotational semantics, be it monotonicities or equalities, constituting what is often called the "algebraic semantics" in the literature. We show here only the small collection:*

$$P \,\Box\, Q = Q \,\Box\, P \qquad P \,\Box\, P = P \qquad P \,\Box\, (Q \,\Box\, R) = (P \,\Box\, Q) \,\Box\, R$$
$$\Box x{\in}A \cup B \rightarrow P\ x = (\Box a{\in}A \rightarrow P\ a) \,\Box\, (\Box b{\in}B \rightarrow P\ b)$$
$$(\forall y.\ c\ y \in S) \Longrightarrow c?x \rightarrow P\ x \,[\![S]\!]\, c?x \rightarrow Q\ x = c?x \rightarrow (P\ x \,[\![S]\!]\, Q\ x)$$
$$\forall y.\ c\ y \notin B \Longrightarrow c!a \rightarrow P \setminus B = c!a \rightarrow (P \setminus B)$$
$$c\ a \in B \Longrightarrow c!a \rightarrow P \setminus B = P \setminus B \qquad \text{etc.}$$

The theories HOL-CSP and HOL-CSPM [3] also add a number of extensions of the original language. This includes the generalization of synchronization indexed by a multiset M: $[\![S]\!]i \in\#\ M.\ P\ i$, the generalization of sequential composition indexed by a list L: SEQ $l \in@ L.\ P\ i$, etc.

3 Deterministic Proc-Omata

3.1 Motivations

Refinements proofs can quickly become counter-intuitive and fastidious, especially when modeling concurrent systems built from iterative architectural compositions[3]. We propose a certain subclass 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:

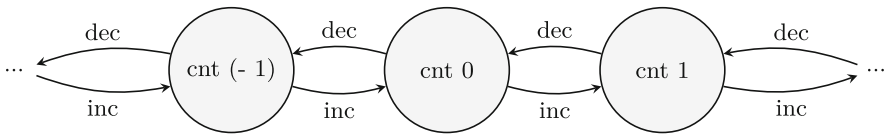

Fig. 4. LTS for the integer counter example

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

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

cnt ≡ μ X. (λn. (inc → X (n + 1)) □ (dec → X (n - 1)))

Note that if inc *and* dec *are of type* 'a*, the fixed point operator acts on a* function *of type* int ⇒ 'a process*; and since* 'a process *belongs to the type class* cpo*, this implies that* int ⇒ 'a process *also belongs to the* cpo *class. Thus, type inference establishes that the least fixed point exists (and by proving that this process has no divergence, one could also establish that such a fixed point is unique) or in other words that* cnt *is well defined. This results in a process function of type* int ⇒ 'a process *that is parameterized in the initial state, something we could call a higher-order process. To rephrase this, we have defined for each integer a process whose relationship to the others can be illustrated by the LTS in Fig. 4. Note that its non-symbolic presentation makes the state-space infinite.*

From algebraic properties of Mprefix and Det in HOL-CSP (cf Proposition 1), and relying on the fact that inc and dec are distinct, we can rewrite our counter process as follows:

cnt = (μ X. (λn. □e∈{dec, inc} → (if e = inc then X (n + 1) else X (n - 1))))

This example gives the intuition of what is a deterministic Proc-Omaton: a fixed point within which there is only one step given by a multi-prefix deterministic choice.

3.2 Formal Definitions

We capture the intuition of the example above, by a formal definition of our notion of a deterministic *Proc-Omaton*: it is a higher-order process canonically associated with a functional automaton [23]. Let us first define a deterministic automaton by a *record* in Isabelle/HOL.

Definition 1 (Abstract syntax of a deterministic automaton).

record ('σ, 'e) A_d = τ :: ‹'σ ⇒ 'e ⇒ 'σ option›

This command creates the record named A_d, of type ('σ, 'e) A_d ('σ and 'e being of course polymorphic), and τ is the name of the record's field.

Intuitively 'σ is the type of the states and 'e the type of the transitions.

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 for the extensions). We will exploit this feature in the following.

We now provide the formal definitions of enableness, reachability set and Proc-Omaton associated with an automaton.

Definition 2 (Enableness). *Let* A *be an automaton of type* ('σ, 'e) A_d*; its transition function is accessible via by* τ A*. From this, we derive the following notion:*

$$\varepsilon\ A\ \sigma \equiv \{e \mid \tau\ A\ \sigma\ e \neq \Diamond\}$$

Definition 3 (Reachability set of an automaton).

inductive_set $\mathcal{R}_d$ :: ‹('σ, 'e) A_d ⇒ 'σ ⇒ 'σ set› (‹$\mathcal{R}_d$›)
for A :: ‹('σ, 'e) A_d› and σ :: 'σ
where init : ‹$\sigma \in \mathcal{R}_d$ A σ›
| step : ‹σ' $\in \mathcal{R}_d$ A s ⟹ ⌊σ'⌋ = τ A σ e ⟹ σ' $\in \mathcal{R}_d$ A σ›

Definition 4 (Deterministic Proc-Omata). *To an automaton* A *we associate a parametric process that we call a "Proc-Omaton". It is a function of type* 'σ ⇒ 'e process *that we denote by:*

$$P[\![A]\!]_d \equiv \mu\ X.\ (\lambda\sigma.\ \Box\ e \in \varepsilon\ A\ \sigma \rightarrow X\ (\lceil \tau\ A\ \sigma\ e \rceil))$$

where ⌈y::'a option⌉ *denotes the* x::'a *such that* y = ⌊x⌋ *when* y ≠ ◇*.*

Example 3 (Proc-Omata for the integer counter). *We associate to the process* cnt *presented in Example 2 the deterministic automaton:*

A ≡ (|τ = λn e. if e = inc then ⌊n + 1⌋ else if e = dec then ⌊n - 1⌋ else ◇|)

and prove that cnt n = $P[\![A]\!]_d$ n.

When applying the operational rules of CSP to processes [4], 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 explicitly accessible.

3.3 Properties of Proc-Omata

As mentioned earlier, a deterministic Proc-Omaton 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 (Unfolding). $P[\![A]\!]_d\ \sigma = \Box\ e \in \varepsilon\ A\ \sigma \rightarrow P[\![A]\!]_d\ \lceil \tau\ A\ \sigma\ e \rceil$

Theorem 3 (Termination). non_terminating ($P[\![A]\!]_d\ \sigma$)
where non_terminating *is a predicate over processes expressing that no trace is ending normally with* ✓*.*

The notion of reachability set defined earlier Definition 3 leads to the two following characterizations.

Theorem 4 (Deadlock). deadlock_free $(P[\![A]\!]_d\ \sigma) = (\forall \sigma' \in \mathcal{R}_d\ A\ \sigma.\ \varepsilon\ A\ \sigma' \neq \emptyset)$
where deadlock_free *is a predicate over processes defined as*
$\text{deadlock\_free}\ P \equiv (\mu\ X.\ \sqcap e \in UNIV \rightarrow X) \sqsubseteq_{FD} P$
*expressing that "*P *can always make progress".*

Theorem 5 (Alphabet). events_of $(P[\![A]\!]_d\ \sigma) = \bigcup\ (\varepsilon\ A\ `\ \mathcal{R}_d\ A\ \sigma)$
where events_of P *is the alphabet of the process* P.

Finally, an interesting yet not too surprising property is that deterministic Proc-Omata correspond to deterministic processes. This 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.$\text{deterministic}\ P \equiv \forall s\ e.\ s\ @\ [e] \in \mathcal{T}\ P \longrightarrow (s, \{e\}) \notin \mathcal{F}\ P$.

Since such processes are maximal for the $(\sqsubseteq_{FD})$ ordering, we only have to establish the following refinement for proving the equality.

Theorem 6 (Deterministic equality). $P[\![A]\!]_d\ \sigma \sqsubseteq_{FD} P \Longrightarrow P = P[\![A]\!]_d\ \sigma$

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 $('\sigma\ \text{list}, 'e)\ A_d$ such that
$P[\![A_0]\!]_d\ \sigma_0\ [\![E]\!]\ P[\![A_1]\!]_d\ \sigma_1 = P[\![A_0\ {}_d\otimes[\![E]\!]_{bin}\ A_1]\!]_d\ [\sigma_0, \sigma_1]$
under some assumptions on enableness independence that we will discuss later. We omit the formal definition of this binary operator here (which is essentially a translation of the synchronization behavior into the automaton product), but we draw the reader's attention to the choice of $'\sigma$ list instead of $'\sigma \times '\sigma$. This is so that we can inductively generalize the product, leading to what we call a compactification theorem.

Theorem 7 (Compactification of Synchronization). *Assuming* $|\sigma s| = |\sigma A|$ *and a generalization of the hypothesis of independence on the enableness:*

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

If we say that this generalization 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 1500 lines of definitions and dense proofs in Isabelle/HOL, in short because synchronization gives rise to many cases to be dealt with during the proof by double refinement.

The importance of Theorem 7 lies in the fact that an iterative synchronization of an arbitrary number of Proc-Omata can be reconstructed into a Proc-Omaton, 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 $\mathrm{E} = \emptyset$ and $\mathrm{E} =$ 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):

$\forall\, \sigma_0' \in \mathcal{R}_d\ \mathrm{A}_0\ \sigma_0.\ \forall\, \sigma_1' \in \mathcal{R}_d\ \mathrm{A}_1\ \sigma_1.\ \varepsilon\ \mathrm{A}_0\ \sigma_0' \cap \varepsilon\ \mathrm{A}_1\ \sigma_1' \subseteq \mathrm{E}$

This is necessary if we want to remain with deterministic automata, because otherwise the synchronization would not be a deterministic process anymore.

4 Potentially Terminating Proc-Omata

In this section, we discuss a variant of Proc-Omata as introduced in Sect. 3, which were non_terminating automata. They may deadlock, but never gracefully terminate with SKIP. However, since this may be useful for numerous applications, we have extended our definitions to cope with this kind of situation. We will concentrate a few key-results of the deterministic case.

4.1 Formal Definitions

Actually our records have an additional field: $\mathcal{S}_F$, the set of final states.

Definition 5 (Abstract syntax of a deterministic automaton). *We add an additional field in the record with the concept of a final state, i. e. a state representing that* SKIP *has been reached:*

$$\text{record } (\text{'}\sigma, \text{'e})\ \mathrm{A}_d = \tau :: \langle \text{'}\sigma \Rightarrow \text{'e} \Rightarrow \text{'}\sigma\ \text{option}\rangle\ \mathcal{S}_F :: \langle \text{'}\sigma\ \text{set}\rangle$$

Definition 6 (Deterministic SKIP-Proc-Omata). *The process scheme corresponding to a SKIP-Proc-Omaton reads as follows:*

$$\mathrm{P}_{SKIP}[\![\mathrm{A}]\!]_d \equiv \mu\ \mathrm{X}.\ (\lambda\sigma.\ \text{if } \sigma \in \mathcal{S}_F\ \mathrm{A} \text{ then SKIP else } \Box \mathrm{e} \in \varepsilon\ \mathrm{A}\ \sigma \rightarrow \mathrm{X}\ \lceil \tau\ \mathrm{A}\ \sigma\ \mathrm{e} \rceil)$$

Note that the Collatz process shown in Example 1 can be expressed as a Proc-Omaton of this variant. This definition naturally generalizes Sect. 3.2, which is a special case $\mathcal{S}_F\ \mathrm{A} = \emptyset$. Such a fixed point can be unfolded in the same way as in Proposition 2.

4.2 Properties of Potentially Terminating 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 termination with SKIP. The counterparts of Theorem 3, Theorem 6 and Theorem 4 are as follows:

$$\text{non\_terminating } (\mathrm{P}_{SKIP}[\![\mathrm{A}]\!]_d\ \sigma) = (\mathcal{S}_F\ \mathrm{A} \cap \mathcal{R}_d\ \mathrm{A}\ \sigma = \emptyset)$$

$$\mathrm{P}_{SKIP}[\![\mathrm{A}]\!]_d\ \sigma \sqsubseteq_{FD} \mathrm{P} \Longrightarrow \mathrm{P} = \mathrm{P}_{SKIP}[\![\mathrm{A}]\!]_d\ \sigma$$

$$\text{fin\_states\_not\_enabled A} \Longrightarrow$$
$$\text{deadlock\_free}_{SKIP}\ (\mathrm{P}_{SKIP}[\![\mathrm{A}]\!]_d\ \sigma) = (\forall\, \sigma' \in \mathcal{R}_d\ \mathrm{A}\ \sigma.\ \sigma' \in \mathcal{S}_F\ \mathrm{A} \vee \varepsilon\ \mathrm{A}\ \sigma' \neq \emptyset)$$

The latter theorem requires some explanation. $\text{deadlock\_free}_{SKIP}$ P is a predicate on P meaning that P is either always making progress, either terminating with SKIP. For establishing this result we need an additional assumption: fin_states_not_enabled A stipulating that ε A $\sigma = \emptyset$ as soon as $\sigma \in \mathcal{S}_F$ A.

However, the most important result is the fact that we have managed to extend the compactification of synchronization to such variants of Proc-Omata. Admittedly, these rules have more the format of program transformation rules (what they are) which are geared towards mechanization.

Theorem 8 (General Compactification of Synchronization). *The generalized form is technically quite dense, but this is due to the fact that it requires a fairly large number of applicability conditions. We assume:*

- $|\sigma\mathrm{s}| = |\sigma\mathrm{A}|$
- $\forall\, \mathrm{A} \in \text{set}\ \sigma\mathrm{A}.$ fin_states_not_enabled A
- $\forall\, \mathrm{i} < |\sigma\mathrm{A}|.\ \forall\, \mathrm{j} < |\sigma\mathrm{A}|.\ \mathrm{i} \neq \mathrm{j} \longrightarrow \text{det\_indep\_enabl}\ \sigma\mathrm{A}_{[i]}\ \sigma\mathrm{s}_{[i]}\ \mathrm{E}\ \sigma\mathrm{A}_{[j]}\ \sigma\mathrm{s}_{[j]}$.

Thus $[\![\mathrm{E}]\!]\ (\sigma, \mathrm{A}) \in \#\text{mset}\ (\text{zip}\ \sigma\mathrm{s}\ \sigma\mathrm{A}).\ \mathrm{P}_{SKIP}[\![\mathrm{A}]\!]_d\ \sigma = \mathrm{P}_{SKIP}[\![_d\bigotimes [\![\mathrm{E}]\!]\ \sigma\mathrm{A}]\!]_d\ \sigma\mathrm{s}$.

4.3 Compactification of Sequential Composition

The extension of our formalization for allowing potentially terminating Proc-Omata was actually motivated by the fact that SKIP is the neutral element for the sequential composition i. e. 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 8.

Theorem 9 (General Compactification of Sequential composition). *Again, the generalized form is technically quite dense. We assume:*

- $\sigma\mathrm{A} \neq []$
- $|\sigma\mathrm{s}| = |\sigma\mathrm{A}|$
- fin_states_not_enabled (last σA).

Thus SEQ $(\sigma, \mathrm{A}) \in @\text{zip}\ \sigma\mathrm{s}\ \sigma\mathrm{A}.\ \mathrm{P}_{SKIP}[\![\mathrm{A}]\!]_d\ \sigma = \mathrm{P}_{SKIP}[\![_d\bigotimes ;\ \sigma\mathrm{A}]\!]_d\ \sigma\mathrm{s}$.

5 Examples

5.1 Bounded Buffer

Let us first start with an example where conversion is immediate. Using a fixed point we define a bounded buffer such that for every n and L:

BBuf n L =
(n < N) & (input**?**x → BBuf (n + 1) (L @ [x])) □
(0 < n) & (output**!**hd L → BBuf (n - 1) (tl L))

where:

- n is a nat and L an 'a list
- the maximal size N of the buffer is non negative
- input and output are channels
- P & c is just an abbreviation for if c then P else STOP.

We prove that this example can be easily "procomatized" with the following transition function:

λ(n, L) e.
  case e of input x ⇒ if n < N then ⌊(n + 1, L @ [x])⌋ else ◊
  | output x ⇒ if 0 < n ∧ hd L = x then ⌊(n - 1, tl L)⌋ else ◊

From this it is straightforward to conclude that BBuf 0 [] is deadlock_free, and we can more generally use it inside an architecture with other Proc-Omata.

5.2 Dining Philosophers

A good illustration of the power of Proc-Omata reasoning is the paradigmatic Dining Philosophers example, first introduced by Dijkstra in 1965 and reformulated to the present form by Hoare [15]. The problem has been tackled by many model checkers, who routinely solve this problem up to say 15 philosophers before giving up due to state explosion.

The problem assumes that N philosophers—with N an **arbitrary** number—are dining around a round table, each one sharing his right and left fork with his left and right neighbour respectively. However, philosophers need two forks to actually eat, a naive strategy to just grab both forks may therefore result in a deadlock.

In the following, we present a formalization in HOL-CSP. We start to define the *channels* by a datatype:

datatype dining_event = picks nat nat | putsdown nat nat

Based on these events, we construct a number of simple processes representing right-handed philosophers (they pick the fork to the right first), a left-handed philosopher, and forks:

Definition 7 (Basic Processes).

$$\text{RPHIL i} \equiv \mu\ \text{X. picks i i} \rightarrow \text{picks i ((i - 1) mod N)} \rightarrow \text{putsdown i ((i - 1) mod N)} \rightarrow \text{putsdown i i} \rightarrow \text{X}$$

$$\text{LPHIL0} \equiv \mu\ \text{X. picks 0 (N - 1)} \rightarrow \text{picks 0 0} \rightarrow \text{putsdown 0 0} \rightarrow \text{putsdown 0 (N - 1)} \rightarrow \text{X}$$

$$\text{FORK i} \equiv \mu\ \text{X.} \quad (\text{picks i i} \rightarrow \text{putsdown i i} \rightarrow \text{x}) \ \square\ (\text{picks ((i + 1) mod N) i} \rightarrow \text{putsdown ((i + 1) mod N) i} \rightarrow \text{X})$$

Using the architectural operator multi-interleave, philosophers and forks are "wired together" as follows:

Definition 8 (Architecture of right-handed Dining Philosophers).

$$\begin{aligned} \text{RPHILS} &\equiv |||\ \text{i}\in\#\ \text{mset}\ [0..<\text{N}].\ \text{RPHIL i} \\ \text{RPHILS'} &\equiv |||\ \text{i}\in\#\ \text{mset}\ [1..<\text{N}].\ \text{RPHIL i} \\ \text{FORKS} &\equiv |||\ \text{i}\in\#\ \text{mset}\ [0..<\text{N}].\ \text{FORK i} \\ \text{RDINING} &= \text{FORKS} \parallel \text{RPHILS} \end{aligned}$$

Reasoning about a potential deadlock for such a system can be difficult: you do not really know where to start in the ring. But it is straightforward to construct for the simple processes of Definition 7 equivalent deterministic Proc-Omata. In more details, this can be achieved for the forks and the right-handed philosophers. With the appropriate definitions we prove FORK i = P⟦fork_A i⟧$_d$ 0 and RPHIL i = P⟦rphil_A i⟧$_d$ 0. Finally, using the compactification Theorem 7, we obtain a big Proc-Omaton equivalent to RDINING.

We show that ([1 ... 1],[1 ... 1]) (both lists length N) is reachable from the initial state ([0 ... 0],[0 ... 0]), and that in this state our big Proc-Omaton 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 deadlocks for any N > 1.

By modifying the construction to DINING = (FORKS || (LPHIL0 ||| RPHILS')) i. e. replacing the first right-handed philosopher by a left-handed one, we can again compactify, and prove by invariant on the Proc-Omaton for DINING that every reachable state is enabled, which results with Theorem 4 in deadlock_free DINING.

We emphasize, again, that these proofs are independent from N and that the fully formalized proofs for e.g. deadlock_free DINING can be found in session HOL-CSP_Proc-Omata.

5.3 Copy Buffer with a Queue

Even if a process is not completely convertible in a Proc-Omaton, the technique may be applicable for crucial sub-problems. Let us consider a copy buffer with a queue where the data transmitted by the sender is stored until the recipient actually receives it. The queue may have arbitrary size *and* be parameterized over an arbitrary type 'a.

Again, we define the channels of the process via a datatype:

```
datatype 'a chan = left 'a | enqueue 'a | dequeue 'a | right 'a
```

Definition 9 (Definitions for a Queue Buffer). *The definition of the Queue-Buffer is as follows:*

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

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

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

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

$$\text{QueueBuffer } L \equiv \text{QueueBuffer\_pre } L \setminus \text{range enqueue} \cup \text{range dequeue}$$

We can write send, rec and queue as Proc-Omata without much difficulty, and treat therefore QueueBuffer_pre with compactification. As a consequence of Theorem 4, we immediately obtain that QueueBuffer_pre is deadlock free by establishing that enableness is always non-empty. Because of the Hiding operator, however, the entire QueueBuffer can not be converted. Fortunately, for the obtained Proc-Omaton for QueueBuffer_pre, we can still apply a fixed point induction where we let the state free variable. This allows us to look only one step ahead and prove relatively easily that QueueBuffer is deadlock free.

6 Related Work

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. Not surprisingly, numerous formalisation attempts have been undertaken with the advent of powerful interactive proof assistants. This ranges from pioneering work in HOL4 [9], fragments of operational semantics as in [24] and [16] and first attempts towards a symbolically working tool [12,17–19]. Our CSP background theory contained in [3,5,6,29,31] represents to

our knowledge the most comprehensive formal treatment of the theory of CSP (cf. [4] for a more in-depth comparison over existing approaches).

Parametric verification attracted a lot of interest, in particular in model-checking communities, in order to overcome the obvious limitations of finite models. This applies for stochastic model-checking [13], linear time model-checking [2], or protocol verifiers [10,14] just to cite a few out of a plethora of publications.

Attempts for parametric model-checking can also be found in the closer related field of process algebras. Notably [1,20,26] attempted to find process characterizations to generalise finite results to infinite ones by *data-independence*. 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 [26]. We'd like to object that our approach, albeit having the apparent drawback to be based on interactive theorem proving, achieves similar or even more general results with finally lesser effort[4].

7 Conclusion

We presented a formalization of Proc-Omata and the cornerstones of their theory. The interest in Proc-Omata lays in the fact that they may serve as a bridge between process algebras and automata theory. They can be seen as a compromise: by conceding a certain expressiveness, we obtain powerful proof techniques by reusing product-automata constructions and their properties. We actually presented several motivations and variants with increasing levels of abstraction before giving an overview of the key results: the compactification theorems geared at process synchronization families.

We illustrated this with three examples: the bounded buffer, where the conversion to Proc-Omaton is straightforward, the Dining Philosophers where the situation of a parameterized unbounded ring of processes fits our theory perfectly, and the queue buffer where Proc-Omata-techniques can at least be used for critical sub-components.

Our construction motivates several lines of future research:

- automated conversion of processes in their Proc-Omata counterpart
- the extension of our theory to non-deterministic Proc-Omata
- generalizations to more operators and process patterns, e. g. interleaving and sequencing for reordering theorems,
- more proof automation by connecting to external tools (model checkers, simulators)
- automated synthesis of invariants as in CEGAR, Cubicle, or Kind 2.

[4] Our version of the random-number generator can be found in [4].

References

1. An, J., Zhang, L., You, C.: The design and implementation of data independence in the CSP model of security protocol. Adv. Mater. Res. **915–916**, 1386–1392 (2014). https://doi.org/10.4028/www.scientific.net/AMR.915-916.1386
2. André, É., Nguyen, H.G., Petrucci, L., Sun, J.: Parametric model checking timed automata under non-zenoness assumption. In: Barrett, C., Davies, M., Kahsai, T. (eds.) NFM 2017. LNCS, vol. 10227, pp. 35–51. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57288-8_3
3. Ballenghien, B., Taha, S., Wolff, B.: HOL-CSPM - architectural operators for HOL-CSP. Arch. Formal Proofs **2023** (2023). https://www.isa-afp.org/entries/HOL-CSPM.html
4. Ballenghien, B., Wolff, B.: An operational semantics in Isabelle/HOL-CSP. In: Bertot, Y., Kutsia, T., Norrish, M. (eds.) 15th International Conference on Interactive Theorem Proving, ITP 2024. LIPIcs, vol. 309, pp. 29:1–29:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2024). https://doi.org/10.4230/LIPIcs.ITP.2023.29
5. Ballenghien, B., Wolff, B.: An operational semantics in Isabelle/HOL-CSP. In: Bertot, Y., Kutsia, T., Norrish, M. (eds.) 15th International Conference on Interactive Theorem Proving, ITP 2024, 9–14 September 2024, Tbilisi, Georgia. LIPIcs, vol. 309, pp. 7:1–7:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2024). https://doi.org/10.4230/LIPICS.ITP.2024.7
6. Ballenghien, B., Wolff, B.: Operational semantics formally proven in HOL-CSP. Archive of Formal Proofs (2023). https://isa-afp.org/entries/HOL-CSP_OpSem.html
7. Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A theory of communicating sequential processes. J. ACM **31**(3), 560–599 (1984)
8. Brookes, S.D., Roscoe, A.W.: An improved failures model for communicating processes. In: Brookes, S.D., Roscoe, A.W., Winskel, G. (eds.) CONCURRENCY 1984. LNCS, vol. 197, pp. 281–305. Springer, Heidelberg (1985). https://doi.org/10.1007/3-540-15670-4_14
9. 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). https://doi.org/10.1007/978-1-4471-3182-3_9
10. Conchon, S., Delzanno, G., Ferrando, A.: Declarative parameterized verification of distributed protocols via the cubicle model checker. Fundam. Informaticae **178**(4), 347–378 (2021). https://doi.org/10.3233/FI-2021-2010
11. 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
12. da Silva Carvalho de Freitas, C.A.: A theory for communicating, sequential processes in Coq (2020). https://api.semanticscholar.org/CorpusID:259373665
13. Hahn, E.M., Hermanns, H., Wachter, B., Zhang, L.: PARAM: a model checker for parametric Markov models. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 660–664. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_56
14. Hess, A.V., Mödersheim, S.A., Brucker, A.D.: Stateful protocol composition in Isabelle/HOL. ACM Trans. Priv. Secur. **26**(3), 25:1–25:36 (2023). https://doi.org/10.1145/3577020

15. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall Inc., Upper Saddle River (1985)
16. 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, TyDe 2016, pp. 28–38. Association for Computing Machinery, New York (2016). https://doi.org/10.1145/2976022.2976032
17. Igried, B., Setzer, A.: Trace and stable failures semantics for CSP-AGDA. arXiv preprint arXiv:1709.04714 (2017)
18. Isobe, Y., Roggenbach, M.: A complete axiomatic semantics for the CSP stable-failures model. In: CONCUR 2006 - Concurrency Theory, 17th International Conference, Bonn, Germany, 27–30 August 2006, pp. 158–172 (2006)
19. Isobe, Y., Roggenbach, M.: CSP-prover: a proof tool for the verification of scalable concurrent systems. Inf. Media Technol. **5**(1), 32–39 (2010). https://doi.org/10.11185/imt.5.32
20. Lazic, R.S.: A semantic study of data-independence with applications to the mechanical verification of concurren. Ph.D. thesis, University of Oxford (1999)
21. 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
22. Nipkow, T.: Verified lexical analysis. In: Grundy, J., Newey, M. (eds.) TPHOLs 1998. LNCS, vol. 1479, pp. 1–15. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0055126
23. Nipkow, T.: Functional automata. Arch. Formal Proofs **2004** (2004). https://www.isa-afp.org/entries/Functional-Automata.shtml
24. 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
25. Roscoe, A.W.: An alternative order for the failures model. J. Log. Comput. **2**, 557–577 (1992)
26. Roscoe, A.W., Broadfoot, P.J.: Proving security protocols with model checkers by data independence techniques. J. Comput. Secur. **7**(1), 147–190 (1999)
27. Roscoe, A.: Theory and Practice of Concurrency. Prentice Hall, Hoboken (1997)
28. Scott, D.: Continuous lattices. In: Lawvere, F.W. (ed.) Toposes, Algebraic Geometry and Logic. LNM, vol. 274, pp. 97–136. Springer, Heidelberg (1972). https://doi.org/10.1007/BFb0073967
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.) IFM 2020. LNCS, vol. 12546, pp. 419–439. Springer, Cham (2020). 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 (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., Jones, C.B., Lucas, P. (eds.) FME 1997. LNCS, vol. 1313, pp. 318–337. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63533-5_17