Copying Safety and Liveness Properties of Computational Artefacts

Nicola Angius
Department of Cognitive Science
University of Messina, Italy
nicola.angius@unime.it

Giuseppe Primiero
Department of Philosophy
University of Milan, Italy
giuseppe.primiero@unimi.it

Abstract

This paper shows how *safety* and *liveness* properties are not necessarily preserved by different kinds of copies of computational artefacts and proposes procedures to preserve them that are consistent with ethical analyses on software property rights infringement. Safety and liveness are second-order properties that are crucial in the definition of the formal ontology of computational artefacts. Software copies are analysed at the level of their formal models as *exact*, *inexact*, and *approximate* copies, according to the taxonomy in (Angius and Primiero, 2018). First, it is explained how exact copies are the only kind of copies that preserve safety and liveness properties, and how inexact and approximate copies do not necessarily preserve them. Secondly, two model checking algorithms are proposed to verify whether inexact and approximate copies actually preserve safety and liveness properties. Essential properties of termination, correctness, and complexity are proved for these algorithms. Finally, contraction and expansion algorithmic operations are defined, allowing for the automatic design of safety- and liveness-preserving approximate copies. As a conclusion, the relevance of the present logical analysis for the ongoing debates in miscomputation and computer ethics is highlighted.

Keywords. Philosophy of computing, philosophy of information, artefact copy, temporal logic, model checking, software theory change.

1 Introduction

Defining the ontological notion of *copy* between artefacts, and establishing its formal properties, is an open problem in the philosophy of technology (Carrara and Soavi, 2010; Hick and Schmcke, 2016; Tzouvaras et al., 1993). The problem acquires particular significance in the context of the philosophy of information and computing, wherein structural problems are essential in establishing malfunctions of *computational artefacts* (Floridi et al., 2015; Fresco and Primiero, 2013) and because ethical and legal issues arise in determining property rights infringement (Johnson and Miller, 2008; Nissenbaum, 1995). The copy relation can be understood as a logic weakening of the stronger *identity* relation (Angius and Primiero, 2018), the latter being an equivalence relation satisfying Leibniz's law of the indiscernibility of identicals (Noonan

and Curtis, 2018). In particular, not everything which is true of an artefact is necessarily true of its copy. This paper focuses on specific properties of software systems which are relevant for software copies, from a security as well as from an ethical point of view, namely *safety* and *liveness*. The logical analysis on copies carried out in this journal by Angius and Primiero (2018) is here extended to inquire whether safety and liveness properties are preserved by software copies and how to preserve them. This is an especially important task when considering safety-critical systems controlled by software, but also when investigating the limits and possible risks of opacity and bias in replicating AI algorithms (Haibe-Kains et al., 2020; Hutson, 2018).

In formal ontology, traditional approaches to the definition of identity move from the assumption, initially put forward by Frege (1953), that any two objects x and y should be defined identical in terms of a relation $R(x', y')$ holding between objects x' and y' functionally related to x and y.[1] The relation R is usually required to comply with a list of formal constraints, which include equivalence and Leibniz's law (Lombard, 1986; Wiggins, 2001). Angius and Primiero (2018) export this analysis to the case of identity for any two computational artefacts x and y.[2] It is also provided a taxonomy for the copy relation R' for models of computational artefacts distinguishing among *exact*, *inexact*, and *approximate* copies: the specification of the model of y requires, respectively, *all and only*, *all but not only*, or just *some* of the behaviours required by the specification of the model of x. While the *bisimulation* relation is identified as the formal counterpart for the identity relation R holding between the specifications of the models of x and y, the *simulation* relation is then identified as candidate for R'.[3]

Inexact and approximate copy relations are shown to violate some of the formal constraints satisfied by identity, equivalence and Leibniz's law in the first place. Consequently, when y is a copy of x, there are properties of the model of system x which are not properties of the model of system y. Determining which properties are preserved by software copies and which are not may reveal itself fundamental when establishing whether a software copy is preserving the functionalities of the original and whether it constitutes a property rights infringement. In this context, Angius and Primiero (2019) argue that a fixed number of computational traces in the specification of a computational artefact can be copied under the *fair use* doctrine in case they allow to preserve important properties for the system, like safety and liveness.

Computational artefacts are designed and developed as implementing a usually well-defined set of specifications, realising the functional requirements put forward by customers, users, and other stakeholders engaged in the software development process. Formal development methods often require the formalization of specifications in some proper logic, subsequently allowing to check whether a model of the developed system satisfies, or does not satisfy, the advanced specifications. In the context of models of systems working in *safety critical* situations, such as nuclear plants, rockets

[1] For more details on this debate see (Lowe, 1989, 1997).

[2] The analysis in (Angius and Primiero, 2018), along with essential formal preliminaries of temporal logic and model checking techniques, are extensively recollected in section 2 to keep the present contribution self-contained.

[3] Even though behavioural equivalence and preorder can be expressed in different ways than through bisimulation and simulation relations, such as as isomorphism, trace equivalence, and trace inclusion, this paper keeps the formal set-up of Angius and Primiero (2018), being bisimulation and simulation the most studied behavioural equivalence and preorder relations (Sangiorgi, 2011).

controllers, or traffic lights, particularly significant are the specifications formalizing safety and liveness properties (Rushby, 1994).

Safety properties require that anything which is considered "bad" for a particular system will not ever happen. By contrast, liveness properties specify something "good" that is required to happen infinitely often during the running time of the interested system. Safety and liveness properties are often called *second-order* properties in that they are properties of a computational artefacts as many others but which, unlike the other properties, turn out to express something "bad" or something "good" for the artefact. When model checking techniques are employed in formal verification (Clarke et al., 1999), safety and liveness properties are formalized mainly using *Linear Time temporal Logic* (LTL) or its superset *Computational Tree Logic** (CTL*) (Kröger and Merz, 2008). Specific LTL model checking algorithms have been proposed to model check safety or liveness LTL formulas (Biere et al., 2002; Kupferman and Vardi, 2001).

This paper inquiries the extent to which computational artefact copies preserve safety and liveness properties. More specifically, it is analysed whether, assuming that x satisfies, respectively violates, a given set of safety and liveness properties, the latter are preserved, respectively violated, by exact, inexact, and approximate copies of x. It is shown how it immediately follows from well-known theorems in process algebra that inexact and approximate copies do not necessarily preserve safety and liveness properties.

Subsequently, two algorithms to verify safety and liveness properties for, respectively, inexact and approximate copies are provided. Essential properties of termination, correctness, and complexity are proved. The algorithms are adaptations of the LTL model checking algorithms known in the literature to the case of inexact and approximate copies discussed in (Angius and Primiero, 2018). Intent of this paper is thereby that of providing the ethical analysis of Angius and Primiero (2019) on software property rights with procedural means allowing the fair use doctrine in computer ethics to be implementable into actionable rules.

To this aim, the present paper also defines formal design principles that allow one to develop safety- and liveness-preserving inexact or approximate copies. The formal approach on software theory change proposed in (Primiero et al., 2021) is here used to define algorithmic operations leading to an approximate copy of a model preserving safety and liveness properties specified by the original, while copying the minimal amount of allowed behaviours. The last requirement is consistent with the ethical assumption according to which approximate copies should be permitted in case the least number possible of specified behaviours allowing for the preservation of safety or liveness properties is copied (Angius and Primiero, 2019).

The notion of behaviour formally used in this paper recalls the various concepts used in engineering design. Angius and Primiero (2018) closely investigated the various notions proposed in (Chandrasekaran and Josephson, 2000) and suggested that process algebra matches each with different formal aspects. For the present purposes, it is worth noting that the verification of liveness and safety properties requires the evaluation of output variables over intervals of time, thereby specifically matching notion *IV* of that taxonomy.

The paper is structured as follows. Section 2 recalls some formal preliminaries on software specifications in the form of transition systems, the algebraic relations

that such models can entertain, and the taxonomy of the copy relations provided in (Angius and Primiero, 2018). Section 3, after illustrating some straightforward results concerning the fulfilment or violation of safety and liveness properties from exact, inexact, and approximate copies, presents two algorithms to model check inexact and approximate copies against safety and liveness properties. Section 4 defines model-theoretic algorithmic operations to create an approximate copy with respect to specified safety and liveness properties of the copied system. Section 5 concludes by underlining the relevance of the present logical analysis in the formal ontology of computational artefacts for the on-going debates in miscomputation and computer ethics.

2 Formal Preliminaries

Specifications of computational artefacts are here considered as artefact models in terms of abstract machines represented by a finite state transition system TS defined as follows:

Definition 1 (Finite State Transition System)**.** *A finite transition system* $TS = (S, A, T, I, F, AP, L)$ *is a set theoretic structure where:*

- $S = \{s_0, \ldots, s_n\}$ *is a finite set of states;*
- A *is a finite set of transitions labels;*
- $T \subseteq S \times A \times S$ *is a transition relation, each transition having form* $s_i \xrightarrow{\alpha} s_j$*;*
- $I \subseteq S$ *is a set of initial states;*
- $F \subseteq S$ *is a set of final states;*
- AP *is a finite set of state labels;*
- $L : S \to 2^{AP}$ *is a state labelling function.*

Figure 1 depicts a transition system for a microwave control software in the form of an oriented graph, wherein nodes represent labelled states and arrows labelled transitions.[4] State 1 is both an initial state (pointed out by an incoming arrow) and a final state (coloured state). A required behaviour for such a system is expressed as a sequence of functionalities, each satisfied at a state. Such a sequence is realised by a path in the system. For instance, the sequence of functionalities $off, close \to on, close \to on, close, wave \to on, close \to off, close$ is one of the correct behaviours of the microwave control system realised by the path s_1, s_5, s_6, s_5, s_1.

A path of a transition system is accordingly a set of states, starting from some initial state and, in the former case, terminating in a final state. More generally:

Definition 2 (Path)**.** *Given a finite transition system* TS*, a finite path fragment is a finite sequence of states* $s_0, \ldots, s_n$*, such that each* s_i *is the successor of* s_{i-1}*, for all* $0 < i \leq n$ *and* $n \geq 0$*. An infinite path fragment is an infinite sequence of states* $s_0, s_1, \ldots$

[4]The example is inspired by a study case in (Clarke et al., 1999, p. 39).

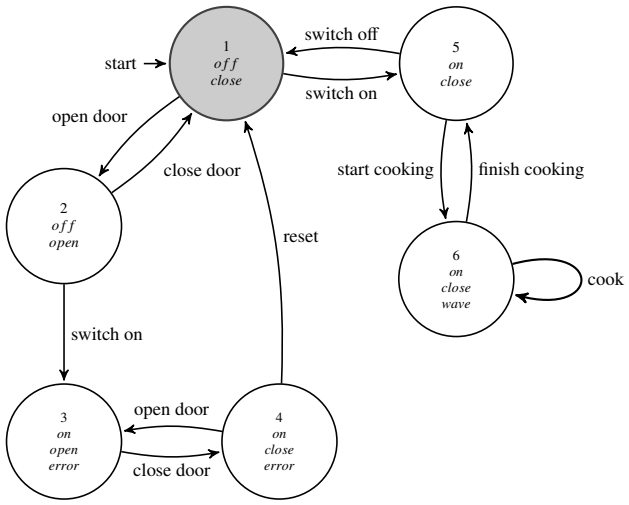

Figure 1: A TS for a microrwave control system.

such that each s_i is the successor of s_{i-1} for all $i > 0$. A path is a path fragment which starts in an initial state $s_i \in I$ and either terminates in a final state $s_j \in F$, or is infinite. The set of paths of a transition system TS is denoted $Path(TS)$.

To express and reason about model behaviours, sequence of labels are considered, one for each state in the path; to reason about model properties, requirements on those sequences are examined, defined as infinite words in the set $(2^{AP})^{\omega}$.[5] Properties of the model are thus requirements over such set. This results in the following standard definitions of *trace* and *property*:

Definition 3 (Trace)**.** *Given a finite transition system TS, the trace of a finite path fragment $s_0, \ldots, s_n$ is the sequence of its labels $L(s_0), \ldots, L(s_n)$. The trace of an infinite path fragment $s_0, s_1, \ldots$ is the sequence of its labels $L(s_0), L(s_1) \ldots$. Traces of transitions systems are* words *over the alphabet 2^{AP}.*

Definition 4 (Property)**.** *A linear-time property T over the set AP is defined as $T \subseteq (2^{AP})^{\omega}$.*

Property specifications holding true of a transition system (as a system specification) are usually expressed in terms of *temporal logic* formulas (Kröger and Merz, 2008). Specifically, safety and liveness formulas, under the focus of the present paper, are often formalised using *Linear Time Logic* (LTL) (Sistla, 1994). The reason is that LTL formulas implicitly quantify over all the computational paths starting from the state wherein the formula holds true. Safety and liveness properties are exactly properties holding in all the paths of a software model, that is, they hold true at any initial

[5] $(2^{AP})^{\omega}$ is the set of infinite words given as infinite concatenations of words in 2^{AP}.

state.[6]

This paper will concentrate on CTL^* temporal logic, given by the union set of LTL and *Computation Tree Logic* (CTL). Safety and liveness properties can well be expressed in CTL^*, either as LTL formulas or in the universal fragment $\forall CTL^*$ in which LTL formulas can be embedded.[7]

CTL^* formulas are composed of atomic propositions from the set AP, Boolean connectives, *temporal operators*, and *path quantifiers*. Temporal operators specify "when" the property holds along the computational path; there are five basic temporal operators:

- **X** ("Next"): a property will hold in the following state;
- **F** ("Finally"): a property will eventually hold along the path;
- **G** ("Globally"): a property always holds along the path;
- **U** ("Until"): this is a binary operator; given two properties, the operator states that the second property will hold at a given state and that the first property holds until that state, that is, in the preceding states of the path;
- **R** ("Release"): it is the dual operation of **U**, that is, it indicates that the second property will hold up to and including the first state where the first property holds.

Path quantifier $\forall$ specifies that a property holds in all paths; path quantifier $\exists$ that the property holds just in some (at least one) paths. CTL^* distinguishes between *state formulas* and *path formulas*. Intuitively, state formulas are formulas that hold true in a given state, while path formulas hold true along a given path.

The syntax for CTL^* well formed formulas is defined as:

Definition 5 (Syntax of CTL^* formulas)**.** *Given the set AP of atomic propositions, a state formula is introduced by the following rules:*

- *if $p \in AP$, then p is a state formula;*
- *if f and g are state formulas, then $\neg f$, $f \wedge g$, $f \vee g$, $f \rightarrow g$ are state formulas;*
- *if f is a path formula, then $\forall f$ and $\exists f$ are state formulas.*

A path formula is introduced by the two additional rules:

- *if f is a state formula, then f is also a path formula;*
- *if f and g are path formulas, then $\neg f$, $f \vee g$, $f \wedge g$, $f \rightarrow g$, $\mathbf{X}f$, $\mathbf{F}f$, $\mathbf{G}f$, $f\mathbf{U}g$, and $f\mathbf{R}g$ are path formulas.*

[6] It would be possible, at this point, to introduce the standard notion of *behaviour* of a system in terms of trace execution, and to reduce safety and liveness to properties holding for all behaviours. Instead, the paper proceeds with the definitions of such properties on traces and of relations between systems satisfying them, in order to make explicit the link to the conceptual notion of copy.

[7] See, for instance, theorem 6.83 in Baier and Katoen (2008).

The satisfiability relation for CTL^* formulas is inductively defined as:[8]

Definition 6 (Satisfaction of CTL^* formulas)**.** *Given a TS, an atomic formula p, state formulas f_1, f_2, and path formulas g_1, g_2, the satisfaction relation in TS of CTL^* formulas is defined as follows:*

$TS, s \models p$	*iff*	$p \in L(s)$;
$TS, s \models \neg f_1$	*iff*	$TS, s \not\models f_1$;
$TS, s \models f_1 \wedge f_2$	*iff*	$TS, s \models f_1$ *and* $TS, s \models f_2$;
$TS, s \models f_1 \vee f_2$	*iff*	$TS, s \models f_1$ *or* $TS, s \models f_2$;
$TS, s \models \forall g_1$	*iff*	*for all paths* π *starting from* s, $TS, \pi \models g_1$;
$TS, s \models \exists g_1$	*iff*	*there exists a path* π *starting from* s *such that* $TS, \pi \models g_1$;
$TS, \pi \models f_1$	*iff*	$TS, s_0 \models p$ *where* $s_0 \in \pi$;
$TS, \pi \models \neg g_1$	*iff*	$TS, \pi \not\models g_1$
$TS, \pi \models g_1 \wedge g_2$	*iff*	$TS, \pi \models g_1$ *and* $TS, \pi \models g_2$;
$TS, \pi \models g_1 \vee g_2$	*iff*	$TS, \pi \models g_1$ *or* $TS, \pi \models g_2$;
$TS, \pi \models \boldsymbol{X} g_1$	*iff*	$TS, \pi_1 \models g_1$;
$TS, \pi \models \boldsymbol{F} g_1$	*iff*	*there exists* $k \geq 0$ *such that* $TS, \pi_k \models g_1$;
$TS, \pi \models \boldsymbol{G} g_1$	*iff*	*for all* $k \geq 0$, $TS, \pi_k \models g_1$;
$TS, \pi \models g_1 \boldsymbol{U} g_2$	*iff*	*there exists* $k \geq 0$ *such that* $TS, \pi_k \models g_2$ *and* $TS, \pi_j \models g_1$ *for all* $0 \leq j < k$;

Notice here that, as underlined above, state formulas are evaluated in a state while path formulas are evaluated in a path. Indeed, state formulas can quantify over paths, that is, they express whether the formula holds true in at least one path or in all paths starting from the state satisfying the state formula itself. By contrast, path formulas assume that time is linear, i.e. that each state has only one successor; they can therefore be evaluated along a single path.

Many of the properties of interest of the microwave control software in Figure 1 can be formalized using CTL^* formulas. One may for instance require that if the oven door is opened while it is *on*, the oven displays an *error* message. This basic functionality can be expressed by the formula $\mathbf{G}(on \wedge open \rightarrow error)$ requiring that all the states ($\mathbf{G}$) labelled with *on* and *open* be also labelled with *error*.

Some of the property specifications that are required to hold in the transition system can be recognized as properties of safety and liveness (Alpern and Schneider, 1987). A safety property specifies that something "bad" will never happen. Accordingly, a safety property is violated by a finite word including a "bad" prefix wherein the "bad" event happens and such that no infinite word starting from the "bad" prefix is able to satisfy the given safety property. By considering linear-time properties of this sort as infinite words subset of $(2^{AP})^\omega$, one may formally define a safety property as in (Baier and Katoen, 2008, p. 112):

Definition 7 (Safety property)**.** *A safety property ψ is a linear time property such that for all infinite words $\sigma \in (2^{AP})^\omega \setminus \psi$ there exists a finite prefix $\hat{\sigma}$ of σ such that $\psi \cap \{\sigma' \in (2^{AP})^\omega \mid \hat{\sigma}$ is a finite prefix of $\sigma'\} = \emptyset$.*

One may require that the *wave* function does not activate while *on* and *open* hold true (indeed, an *error* message should rather be displayed according to the former

[8] In what follows index n in π_n expresses the n^{th} state in π starting from an initial state s_0 in the path.

property specification). This property can be formalized by the CTL^* formula $\mathbf{G}(\neg(on \wedge open \wedge wave))$, requiring that for all the states of the system ($\mathbf{G}$), it never is the case ($\neg$) that $on \wedge open \wedge wave$ holds true at the same time. The latter property is a safety property in that it specifies that something "bad" will never happen. Here the "bad" unsafe situation is one in which microwaves propagate outside the insulated oven. $\mathbf{G}(\neg(on \wedge open \wedge wave))$ is often called *invariant* in that it is required that the predicate logic formula $\neg(on \wedge open \wedge wave)$ holds in every state of the system. A violation of this safety property would consist in a finite word containing a bad prefix, namely one at which $(on \wedge open \wedge wave)$ holds.

By contrast, liveness properties require that something "good" will happen infinitely often (Alpern and Schneider, 1985). Consequently, liveness properties are violated by infinite runs. According to this view, there must exist a finite prefix which can be extended to an infinite word satisfying the liveness property (Baier and Katoen, 2008, p. 6):

Definition 8 (Liveness Property)**.** *Property ϕ is a liveness property if for all finite words $w \in (2^{AP})^*$ there exists an infinite word $\sigma \in (2^{AP})^\omega$ such that $w\sigma \in \phi$.*[9]

A "good" and desired circumstance for the microwave control software is one in which the oven finally starts and keeps cooking. Consequently, the property specification $\mathbf{FG}(wave)$ specifies a liveness property. The formula requires that it will be eventually ($\mathbf{F}$) reached a state such that all reachable states from there ($\mathbf{G}$) satisfy *wave*. This implies that *wave* holds infinitely often. For this reason, the CTL^* formula $\mathbf{FG}(wave)$ is usually called a *persistence* property. As it can be seen from the state transition system in Figure 1, *wave* holds infinitely often when it holds in a state which is reachable from some initial state and when that state belongs to a (self)loop.

Once some of the advanced property specifications have been recognised to be properties of safety and liveness, one may well make use of specific model-checking algorithms conceived to verify them (instead of standard CTL^* model-checking algorithms). The literature offers many algorithms of this sort, see for instance (Biere et al., 2002; Kupferman and Vardi, 2001; Podelski and Rybalchenko, 2003). In this paper, the approach in Baier and Katoen (2008) for model checking regular safety and liveness properties is followed.

In the case of safety properties, the first step is the introduction of a non deterministic finite automaton accepting the language defined by the set of bad prefixes of a given safety property. Recall that a non deterministic finite automaton is defined as follows:

Definition 9 (Nondeterministic Finite Automaton)**.** *A nondeterministic finite automaton on finite words is given by the five-tuple $\mathcal{A} = (\Sigma, Q, \Delta, Q_0, F)$ such that:*

- Σ *is an alphabet;*
- Q *is a finite set of states;*
- $\Delta : Q \times \Sigma \times Q$ *is a nondeterministic transition relation;*
- Q_0 *is a set of initial states;*

[9] $(2^{AP})^*$ is the set of finite words arising in 2^{AP}.

- $F \subseteq Q$ *is a set of accepting states.*

A finite *word* $v \in \Sigma^*$ for a finite automaton $\mathcal{A}$ is a sequence $v = \alpha_1, \alpha_2, \ldots, \alpha_n$ of symbols of the alphabet; a *run* of $\mathcal{A}$ over v is a finite sequence of states $q_0, q_1, \ldots, q_n$ such that $q_0 \in Q_0$ and for all $0 \leq i < n$ it holds that $q_i \xrightarrow{\alpha_{i+1}} q_{i+1}$. Word v is an input for $\mathcal{A}$ and $\mathcal{A}$ is said to read v. A run $q_0, q_1, \ldots, q_n$ is said to be accepting when $q_n \in F$; a finite word $v \in \Sigma^*$ is accepted by A if there is an accepting run for v. The set of finite words in Σ^* accepted by $\mathcal{A}$ is called the accepted language of $\mathcal{A}$ and is denoted by $L(\mathcal{A})$. Since safety properties can be defined as sets of infinite words over 2^{AP}, bad prefixes, by being finite, form a language of finite words with alphabet $\Sigma = 2^{AP}$.

Subsequently, given a nondeterministic finite automaton $\mathcal{A}_\psi$ accepting the bad prefixes of a safety property ψ, and a transition system TS to be verified against ψ, it is checked whether TS and $\mathcal{A}_\psi$ intersect. This is achieved by considering whether the product automaton $TS \times \mathcal{A}_\psi$ satisfies an invariant $\mathbf{G}\chi$, where χ is a formula derived from the accepting states of $\mathcal{A}_\psi$.[10] Let us first recall the definition of the general product automaton $TS \times \mathcal{A}$; the model checking procedure is then sketched in Algorithm 1.[11]

Definition 10 (Product of a finite transition system and a nondeterministic finite automaton)**.** *Given a transition system without final states* $TS = (S, A, T, I, AP, L)$ *and a nondeterministic finite automaton* $\mathcal{A} = (2^{AP}, Q, \Delta, Q_0, F)$*, the product* $TS \times \mathcal{A}$ *is* $(S^*, A, T^*, I^*, AP^*, L^*)$ *where:*

- $S^* = S \times Q$*;*
- A *is the finite state of transition labels of* TS*;*
- T^* *is the smallest relation such that if* $s_i \xrightarrow{a} s_{i+1}$ *and* $q_k \xrightarrow{L(s_{i+1})} q_{k+1}$ *then* $\langle s_i, q_k \rangle \xrightarrow{a} \langle s_{i+1}, q_{k+1} \rangle$*;*
- $I^* = \{\langle s_0, q \rangle \,|\, s_0 \in I \wedge \exists q_0 \in Q_0$ *such that* $q_0 \xrightarrow{L(s_0)} q\}$*;*
- $AP^* = Q$*;*
- $L^* : S \times Q \to 2^Q$ *such that* $L^*(\langle s, q \rangle) = \{q\}$.

A similar approach is taken by the algorithm used to check whether $TS \models \phi$ for a given liveness property ϕ. The set of bad prefixes of ϕ is here accepted by a nondeterministic *Büchi* automaton. Büchi automata are automata that accept infinite words; they have the same structure as finite automata except they recognize words from Σ^ω, where ω indicates an infinite number of repetitions of a finite word. If $inf(p)$ is the set of states that appear infinitely often within a run p of a Büchi automaton $\mathcal{A}$, such run is said to be accepting iff $inf(P) \cap F \neq \emptyset$, that is, when some accepting state is reached in p infinitely often. The model checking algorithm in this case checks whether the product automaton $TS \times \mathcal{A}_\phi$ satisfies a persistent property $\mathbf{FG}\upsilon$. The procedure is sketched in Algorithm 2.

[10] For more details see (Baier and Katoen, 2008).

[11] Please recall that $TS \models \Theta$, for any formula Θ, means that Θ is satisfied by all paths of TS starting from any initial state.

Data: $TS, \psi, \mathcal{A}_\psi, \chi$
Result: *true* if $TS \models \psi$, $false$ otherwise with a counterexample for ψ
initialization;
if $TS \times \mathcal{A}_\psi \models \boldsymbol{G}\chi$ **then**
  return true
else
  Determine a counterexample $\langle s_0, q_1\rangle \ldots \langle s_n, q_{n+1}\rangle$ of $TS \times \mathcal{A}_\psi$ such that $q_{n+1} \in F$;
  return (false, $s_0, \ldots, s_n$)
end

Algorithm 1: Model checking algorithm for $TS \models \psi$.

Data: $TS, \phi, \mathcal{A}_\phi, \upsilon$
Result: *true* if $TS \models \phi$, $false$ otherwise with a counterexample for ϕ
initialization;
if $TS \times \mathcal{A}_\phi \models \boldsymbol{FG}\upsilon$ **then**
  return true
else
  Determine a counterexample $\langle s_0, q_1\rangle, \ldots, \langle s_n, q_{n+1}\rangle, \ldots, \langle s_n, q_{n+1}\rangle$ of $TS \times \mathcal{A}_\phi$ such that $q_{n+1} \in F$;
  return (false, $s_0, \ldots, s_n, \ldots, s_n$)
end

Algorithm 2: Model checking algorithm for $TS \models \phi$.

Before showing, in the next section, the extent to which software copies preserve safety and liveness properties, and how to verify them, it is essential to define what a software copy is. Angius and Primiero (2018) compare computational artefacts formalised at the level of abstraction of their specifications, i.e. of models that abstract away from implementation and linguistic details. Set-theoretical relations of identity, inclusion, and intersection on models, schematised in Figure 2, are used to identify respectively *exact*, *inexact*, and *approximate* copies. The models of two computational artefacts x and y are in an exact copy relation if the specification they express $S(x) = S(y)$ is the same. This means that exact copies are either different implementations of the same system specification in two different high-level programs, or two different implementations of even the same high-level program (and consequently of the same specification). In other words, models of x and y are in an exact copy relation if the specifications of the corresponding systems induce exactly the very same set of behaviours[12] and, consequently, if they satisfy the very same set of formulas. The exact copy relation catches the notion of copying in the sense of duplicating. When models of x and y are in an inexact copy relation, $S(x)$ is a subset of $S(y)$, meaning that $S(y)$ allows all the behaviours allowed by $S(x)$, potentially specifying additional functionalities. An inexact copy is a system model that incorporates another system

[12]It is assumed that a specification corresponds to set of behaviours it prescribes. Consequently, set-theoretic relations holding between specifications, such as inclusion or intersection, are taken to hold also between the corresponding set of behaviours.

EXACT COPY	*INEXACT COPY*	*APPROXIMATE COPY*
$S(x) = S(y)$	$S(x) \subset S(y)$	$S(x) \cap S(y) \neq \emptyset$

Figure 2: Taxonomy of copy relations according to the analysis in Angius and Primiero (2018)

model and develops new functionalities upon it. When y is an approximate copy of x, it is assumed that the intersection of $S(x)$ and $S(y)$ is not void; this implies that $S(y)$ copies just some of the functionalities specified by $S(x)$, again realizing new functional properties. The approximate copy case is the most common one in software development, amounting to the practice of making use, in programming activities, of already encoded object classes that are known to satisfy some desired property specification. Angius and Primiero (2018) formulate logical definitions of exact, inexact, and approximate copy in terms of process algebra relations holding between couples of specifications for deterministic systems.[13] This in turn allows to prove some logic results of interests for the purposes of this paper.

The notion of exact copy entertained by two systems x and y can be logically defined as a *bisimulation* relation between $S(x)$ and $S(y)$. Intuitively, two transitions systems are in a bisimulation relation when each can simulate the other and the two have the same branching structure. Mutual simulation implies that the two specifications satisfy the same set of behavioural properties. Formally, a bismulation between transition systems TS and TS' is defined as follows:

Definition 11 (Bisimulation)**.** *Let $TS = (S, A, T, I, F, AP, L)$ and $TS' = (S', A', T', I', F', AP', L')$ be two finite transition systems, the binary relation $B \subseteq S \times S'$ is a bisimulation relation if and only if:*

1. *for each initial state $s_0 \in I$ there is an initial state $s'_0 \in I'$ such that $(s_0, s'_0) \in B$, and vice versa;*

2. *$(s_0, s'_0) \in B$ if and only if: s_0 and s'_0 have the same label, there is a successor state s'_1 of s'_0 for every successor state s_1 of s_0 such that $(s_1, s'_1) \in B$ and vice versa.*

TS and TS' are in a bisimulation relation, denoted $TS \equiv TS'$, if there exists a bismulationn relation B for (TS, TS').

The transition system in Figure 3 is an exact copy of the one provided in Figure 1. It can be easily checked that the two satisfy conditions 1 and 2 in definition 11, as well as that the two transition systems correspond to the same system specification. Indeed, any path of the first transition system is also a path of the copy, and vice versa. Exact copies are trivially identical and the only admissible distinctions can be obtained by the addition of entirely isomorphic structures to those already existing, as it is the case

[13]For further details on such relations see Fokkink (2013). In the following, these results on copied systems are used, referring the reader to the original (Angius and Primiero, 2018) for the proofs.

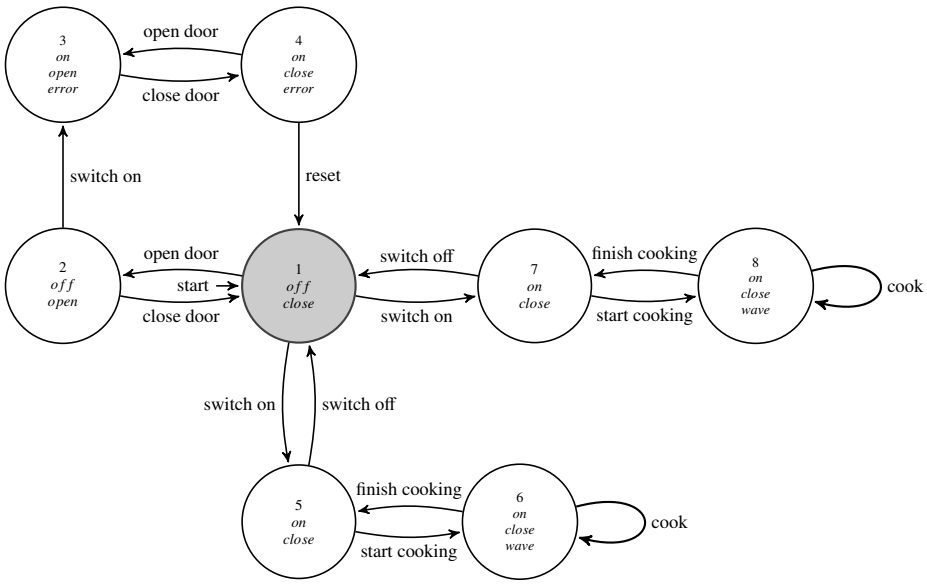

Figure 3: An exact copy of the TS in figure 1.

for path s_1, s_5, s_6, s_5, s_1 in Figure 3. Non-trivial identities are captured by the notions of inexact and approximate copies introduced below.

Hence the copy and the copied system satisfy the very same set of properties. This result can be extended to any CTL^* property in a known process algebra theorem on bismulation and CTL^* equivalence:

Theorem 1 (Bisimulation and CTL^* equivalence)**.** *$TS \equiv TS'$ iff $TS \equiv_{CTL^*} TS'$, i.e. $TS, s_i \vDash g \leftrightarrow TS', s'_i \vDash g$, for any CTL^* formula g.*

Thus, exact copies intended as models that satisfy all and only the same functionalities can be defined on bisimulation:

Theorem 2 (Exact Copy as Bisimulation)**.** *TS' is an exact copy of TS if and only if $TS \equiv TS'$.*

To obtain the weaker notion of inexact and approximate copies, the approach in Angius and Primiero (2018) requires a corresponding weakening of the notion of bisimulation, namely *simulation*, according to the following standard definition:

Definition 12 (Simulation)**.** *Let $TS = (S, A, T, I, F, AP, L)$ and $TS' = (S', A', T', I', F', AP', L')$ be two finite transition systems, the binary relation $R \subseteq S \times S'$ is a simulation relation if and only if:*

1. *for each initial state $s_0 \in I$ there is an initial state $s'_0 \in I'$ such that $(s_0, s'_0) \in R$;*
2. *$(s_0, s'_0) \in R$ if and only if: s_0 and s'_0 have the same label and there is a successor state s'_1 of s'_0 for every successor state s_1 of s_0 such that $(s_1, s'_1) \in R$.*

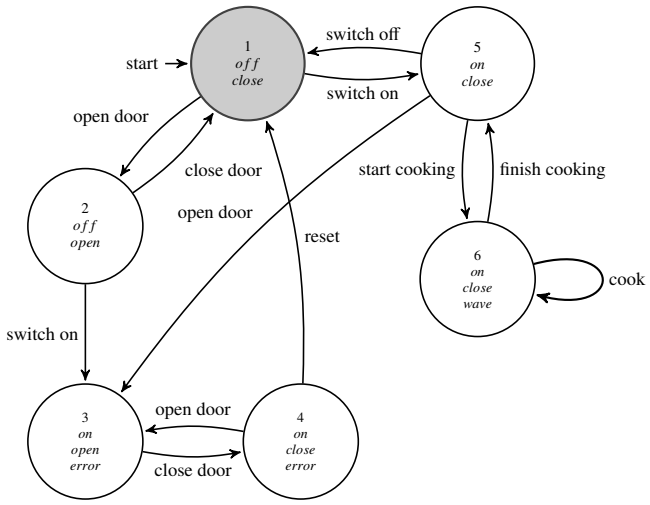

Figure 4: An inexact copy of the TS in figure 1.

TS is simulated by TS', denoted $TS \leq TS'$, if there exists a simulation relation R for (TS, TS').

Figure 4 depicts a inexact copy of the microwave control software of Figure 1. The copying model is able to simulate all the behaviours of the copied model and it adds additional behaviours, namely those corresponding to all the paths that include the added transition from state 5 to state 3. Accordingly, the simulation relation does not imply CTL^* equivalence. However, the following results can be proven for the universal fragment $\forall CTL^*$ and the existential fragment $\exists CTL^*$ of CTL^*:

Theorem 3 (Simulation and $\forall CTL^*$ equivalence)**.** *$TS \leq TS'$ iff $TS', s_i' \vDash g \rightarrow TS, s_i \vDash g$, for any $\forall CTL^*$ formula g.*

Theorem 4 (Simulation and $\exists CTL^*$ equivalence)**.** *$TS \leq TS'$ iff $TS, s_i \vDash g \rightarrow TS', s_i' \vDash g$, for any $\exists CTL^*$ formula g.*

These results allow to provide an appropriate formal counterpart to the notion of inexact copy:

Theorem 5 (Inexact Copy as Simulation)**.** *TS' is an inexact copy of TS if and only if $TS \leq TS'$.*

Finally, a further weakening is required to express approximate copies as relations that satisfy a non-empty intersection of behaviours. To do so, the notion of simulation quotient can be used. First, let us recall that the simulation relation also holds between a transition system and itself:

Definition 13 (Simulation as a Relation on States)**.** *Given a TS, a simulation on states of TS is a binary relation $\leq \subseteq S \times S$ such that for all $(s_i, s_j) \in \leq$*

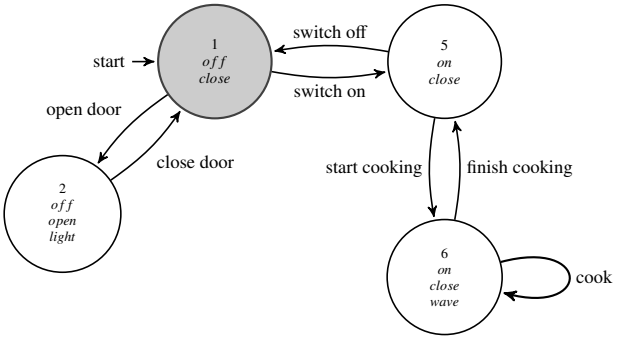

Figure 5: An approximate copy of the TS in figure 1

1. *s_i and s_j have the same label, and*
2. *if there is a successor state s'_i of s_i, then there is a successor s'_j of s_j such that $(s'_i, s'_j) \in \leq$.*

Then the simulation quotient system can be defined as the quotient set for the simulation relation over a transition system. In other words, the simulation quotient system $TS/\leq$ of a transition system TS is a transition system defined by the set of paths that TS is able to simulate over its states. Accordingly, the simulation quotient system $TS/\leq$ amounts to the set of paths of TS.

Following Angius and Primiero (2018), y is an approximate copy of x when y manifests some of, and potentially more than, the behaviours of x, while also x is in the same relation to y. This relation is captured precisely as follows:

Theorem 6 (Approximate copy)**.** *TS' is an approximate copy of TS, denoted by $TS \approx TS'$ iff $\exists \pi' \in TS'/\leq$ such that $\pi' \leq TS$.*

In other words, TS' is an approximate copy of TS when there is a non-empty set of paths in a simulation quotient system of TS' that TS is able to simulate. This is equivalent to saying that there is a non-empty set of paths of TS' that are also paths of TS. Figure 5 shows an approximate copy of the TS in figure 1; the copy shares path s_1, s_5, s_6, s_5, s_1 with the original system model and adds the new path s_1, s_2, s_1 for the added functionality *light*.

3 Safety and Liveness Properties in Copied Structures

An immediate question arising from the previous analysis, and concerned with the specific case of high-order properties of safety and liveness, is whether one can guarantee their preservation in copied structures. In Section 3.1, some immediate results in this direction are illustrated. While these are trivially obtained from the well-known notions in process algebra recalled in section 2, their formulation is here crucial both for the conceptual analysis of the relation of copy and for the examination, in computer ethics,

of what constitutes an intellectual property right infringement. In Section 3.2, a model checking algorithm for verifying safety properties in inexact copies is formulated. An algorithm to check liveness properties in inexact copies is presented in Section 3.3. Finally, in Section 3.4, the analysis is extended to checking liveness and safety for approximate copies. The logical analysis supplied in these sections is aimed to show how traditional algorithms for the formal verification of safety and liveness properties developed in theoretical computer science can well be adapted to the ontological notion of copy in philosophy of technology and of information.

3.1 Safety and liveness in exact and inexact copies

Let us start from exact copies; one would expect that exact copies, by being a duplication of some original system, preserve both safety and liveness properties. Indeed, transition systems which are in a bismulation relation preserve *any* property:

Corollary 1 (Safety and liveness of exact copies)**.** *If TS' is an exact copy of TS then $TS', s'_i \vDash \psi \leftrightarrow TS, s_i \vDash \psi$ and $TS', s'_i \vDash \phi \leftrightarrow TS, s_i \vDash \phi$ for any safety property ψ and liveness property ϕ.*

Proof. By theorem 2 if TS' is an exact copy of TS then $TS \equiv TS'$. Any safety property ψ or liveness property ϕ can be expressed as a CTL^* formula. By theorem 1, bisimulation implies CTL^* equivalence. □

Inexact copies include all the behaviours of the copied systems and combine them with new behaviours. It can be easily proved that any safety and liveness property satisfied by the coping model is also satisfied by the copied model:

Corollary 2 (Safety and Liveness of inexact copies)**.** *If TS' is an inexact copy of TS then $TS', s'_i \vDash \psi \rightarrow TS, s_i \vDash \psi$ and $TS', s'_i \vDash \phi \rightarrow TS, s_i \vDash \phi$ for any safety property ψ and liveness property ϕ.*

Proof. By theorem 5, if TS' is a inexact copy of TS then $TS \leq TS'$. We rely on the known result stating that for a transition system TS, a LTL formula θ, and for each transition system state $s \in TS$, $TS, s_i \models \theta$, according to the LTL semantics, if and only if $TS, s_i \models \forall\theta$ according to the CTL^* semantics. Then any safety property ψ or liveness property ϕ can be expressed in $\forall CTL^*$. Finally, theorem 3 guarantees that if $TS \leq TS'$ then $\forall CTL^*$ formulas are preserved from right to left. □

This corollary provides another expected result and yet of scarce relevance for software development: one would like to know about the opposite direction, that is, about the properties of TS that are also of TS', provided that $TS \leq TS'$. One first result in this direction concerns the violation of safety and liveness:

Corollary 3 (Violation of safety and liveness for inexact copies)**.** *If TS' is an inexact copy of TS then $TS, s_i \vDash \neg\psi \rightarrow TS', s'_i \vDash \neg\psi$ and $TS, s_i \vDash \neg\phi \rightarrow TS', s'_i \vDash \neg\phi$ for any safety property ψ and liveness property ϕ.*

Proof. Any negation of a safety property ψ or liveness property ϕ can be expressed in $\exists CTL^*$, that is, as a negation of a $\forall CTL^*$ formula. By theorem 4, $\exists CTL^*$ formulas are preserved from left to right if and only if $TS \leq TS'$. □

This theorem specifies that if a computational artefact model fails to satisfy a given safety or liveness property, any inexact copy would do so as well. Indeed, any path violating the property would be simulated by the copying model. On the other hand, if some model satisfies the specified second order property, all the paths fulfilling that property will be kept by the inexact copy; however, additional paths may well constitute a counterexample for the desired safety or liveness property.

The conclusion is that in case $TS \models \psi$ or $TS \models \phi$, a inexact copy TS' *may or may not* satisfy them. Seemingly, the only thing to do is here to model check whether $TS', s'_i \vDash \psi/\phi$ using the automata-based model-checking algorithms for safety properties or liveness properties. However, one may well take advantage of the known fact that $TS, s_i \vDash \psi/\phi$ in order to reduce that state space of TS' to be checked against ψ or ϕ. Indeed, one already knows that the state space of TS' that is also of TS does fulfill ψ or ϕ. One therefore needs to model check only that portion of state space of TS' which is not in TS. The next subsection covers the case of safety properties only and formulates a model checking algorithm that, receiving inputs TS, TS', and a safety property ψ, returns an answer "true" $TS', s'_i \vDash \psi$ or "false" $TS', s'_i \nvDash \psi$ together with, in the latter case, a counterexample (see Algorithm 3). The algorithm avoids the state explosion problem for checking $TS', s'_i \vDash \psi$. This argument runs entirely similar for the case of liveness properties in inexact copies, formulated in section 3.3.

3.2 Checking safety in inexact copies

Algorithm 3, presented below, first checks whether $TS' \leq TS$, which one knows it does not hold true, and non-simulative states in TS' which, consequently, are not states of TS are used to define a smaller transition system TS^*; it is then checked whether TS^* satisfies ψ. If it does, it is concluded that also TS' does, insofar as the portion of state space TS' not in TS^* is the state space of TS, already known to satisfy ψ. In case TS^* violates ψ, it is concluded that also TS' does, in that TS' contains paths, the ones in TS^*, violating ψ.

In order to obtain the set of states of TS' that are not states of TS, a modified version of the algorithm proposed in (Baier and Katoen, 2008, pp. 521-527) for computing the simulation quotient system is formulated. While the algorithm in Baier and Katoen (2008) is used to verify whether two transition systems entertain a simulation order relation, the algorithms here proposed are rather aimed at checking whether a system, which is known to be a copy, preserves a safety or liveness property of the copied system.

In the algorithm, the disjoint union set of the two finite structures $TS' \oplus TS$ (provided that a disjoint union set $S' \uplus S$ is given in the sum) is considered as input; the simulation quotient system of $TS' \oplus TS$ is computed taking note of the states of TS' that do not hold a simulation relation with states of TS.[14] Initially all the state couples in $S' \uplus S$ having the same label are put in the variable $\mathcal{R}$. Subsequently, for every state couple in $\mathcal{R}$, it is checked whether its successor state couple is still in $\mathcal{R}$. If it is, the algorithm keeps checking the other successor state couple. Once the algorithm finds a

[14]It should be noted here that the simulation quotient system of $TS' \oplus TS$ is finite and can be computed in finite time, ensuring termination as shown in theorem 8 below. For more details on the finiteness of the simulation quotient system $TS' \oplus TS$ see (Baier and Katoen, 2008, 521).

Data: $TS' \oplus TS, \mathcal{A}_\psi$, and χ
Result: 'true' if $TS' \models \psi$, 'false' otherwise with a counterexample for ψ
initialization;
$\mathcal{R} := \{(s_i, s_j) \in TS' \oplus TS | L'(s_i) = L(s_j)\}$;
$\Pi := \emptyset$;
while $\mathcal{R}$ *is not a simulation* **do**
  for $s_i, s_j \in \mathcal{R}$ **do**
    for $s'_i \in Post(s_i), s'_j \in Post(s_j)$ **do**
      if $(s'_i, s'_j) \notin \mathcal{R}$ **then**
        $\mathcal{R} := \mathcal{R} \setminus \{(s_i, s_j)\}$;
        $\Pi := \{(s_i, s_j)\}$
      end
    end
  end
end
Define a transition system TS^* s.t. $TS^* = \{(\Pi, A^*, T^*, I^*, F^*, AP^*, L^*)\}$;
if $TS^* \times \mathcal{A}_\psi \models \boldsymbol{G}\chi$ **then**
  return true
else
  Determine a counterexample $\langle s_0, q_1\rangle \dots \langle s_n, q_{n+1}\rangle$ of $TS^* \times \mathcal{A}_\psi$ such that $q_{n+1} \in F$;
  return (false, $s_0, \dots, s_n$)
end

Algorithm 3: Model checking algorithm for $TS' \models \psi$ when $TS \leq TS'$.

couple not in $\mathcal{R}$, the preceding state couple is removed from $\mathcal{R}$ and is put into variable Π. The loop terminates once $\mathcal{R}$ results to be a simulation relation, that is, when all non simulative states have been removed from $\mathcal{R}$. The algorithm afterwards defines a reduced transition system TS^* over the state set Π and such that sets A', T', I', F', AP', L' in TS' are adapted consequently in $A^*, T^*, I^*, F^*, AP^*, L^*$ over Π. In the same vein of Algorithm 1, Algorithm 3 finally checks whether the product of the reduced transition system TS^* with $\mathcal{A}_\psi$ satisfies the invariant $\mathbf{G}\chi$.

If $TS^* \times \mathcal{A}_\psi \models \mathbf{G}\chi$, the algorithm returns output 'true', meaning that the inexact copy TS' satisfies safety property ψ. If not, a counterexample is determined in $TS^* \times \mathcal{A}_\psi$ and 'false' is outputted together with the corresponding counterexample in TS' showing a violation of ψ in TS'.

Standard properties for Algorithm 3 are here considered:

Theorem 7 (Partial Correctness of Algorithm 3)**.** *On termination, Algorithm 3 returns the answer 'false' if and only if* $TS' \not\models \psi$.

Proof. $\Rightarrow$: The answer 'false' is outputted only if:

- a state in Π is obtained that belongs to TS' but not to TS;
- there is at least one state

$$\langle s_i, q_{i+1}\rangle$$

in one path belonging to the intersection set $TS^* \times \mathcal{A}_\psi$ such that $s_i \not\models \chi$, where $\mathcal{A}_\psi$ is the automaton accepting the bad prefixes for ψ.

If $\langle s_i, q_{i+1}\rangle \not\models \chi$, then, by definition, the intersection $TS^* \times \mathcal{A}_\psi$ is not void and hence there is a path of TS^* that violates ψ. Since, by definition, all paths of TS^* are paths of TS', also TS' violates ψ.

$\Leftarrow$: If TS' fails to satisfy ψ, then there is at least one state s_i of TS' not in TS, since we are assuming that $TS \models \psi$. Accordingly, $\Pi \neq \emptyset$ and $\Pi \ni s_i \not\models \psi$. Because there is a state in Π violating ψ, then the intersection $TS^* \times \mathcal{A}_\psi$ cannot be empty, since $\mathcal{A}_\psi$ is the automaton accepting the language defined over all bad prefixes of ψ. If this is the case, since invariants hold in all states of a system, there must be one state $\langle s_i, q_{i+1}\rangle$ in in $TS^* \times \mathcal{A}_\psi$ such that $\langle s_i, q_{i+1}\rangle \not\models \chi$. Finally, when such a state is identified, a path containing such state in in $TS^* \times \mathcal{A}_\psi$ is isolated and the answer 'false' together with the corresponding counterexample in TS' are returned as outputs. □

Theorem 8 (Termination of Algorithm 3)**.** *Algorithm 3 terminates for $TS \leq TS'$.*

Proof. Termination of the *while* loop is considered in the following.

$\mathcal{R}$ is defined as the set of state couples (s_i, s_j) in the union set $TS' \oplus TS$ that have the same labels. According to Definition 13, $TS' \oplus TS$, as any other transition system, simulates itself. Therefore, by computing all simulation states in $\mathcal{R}$ and dismissing from $\mathcal{R}$ all non-simulation states one obtains, in a finite number of steps, a simulation order. In particular, this *while* loop takes at most time in $O(|\mathcal{R}|)$, where $|\mathcal{R}|$ denotes the state space of $\mathcal{R}$. One does not have to consider here the state space $|S'| \uplus |S|$ since only states with the same labels have to be travelled when computing simulation order. □

Theorem 9 (Time complexity of Algorithm 3)**.** *The worst time complexity of Algorithm 3 is in $O((|\mathcal{R}| \cdot |\mathcal{A}_\psi|) \cdot (1 + |\chi|) + M)$, where $M = \{|T' \cup T| \cdot |\Delta|\}$.*

Proof. Algorithm 3 first looks for states in TS' which are not states of TS and puts such states in Π. In the worst case, there is only one such a state which is tracked down only after travelling the whole state space in $|\mathcal{R}|$. This would therefore require time $O(|\mathcal{R}| + |T' \cup T|)$ where $|T' \cup T|$ is the length of the union set of transitions in $TS' \uplus TS$ needed to travel states in $\mathcal{R}$. Once TS^* is determined, an intersection with $\mathcal{A}_\psi$ is obtained to check whether the intersection set is empty. Supposing all states of TS' except one are not in TS, the intersection operation is bounded by $O(|\mathcal{R}^*| \cdot |\mathcal{A}_\psi| + |T' \cup T| \cdot |\Delta|)$, where $|\mathcal{R}^*|$ is the length of the state space of $\mathcal{R}$ minus the set of states in TS' which are also states of TS, and $|\Delta|$ is the transition set of automaton $\mathcal{A}_\psi$. Finally, property χ has to be model checked, in the worst case, for each state in $O(|\mathcal{R}^*| \cdot |\mathcal{A}_\psi|)$. This therefore requires time in $O(|\mathcal{R}^*| \cdot |\mathcal{A}_\psi|) \cdot (1 + |\chi|))$: it is proportional to 1 in case χ is atomic and to the length of χ otherwise. Therefore, the overall time complexity is in $O((|\mathcal{R}| \cdot |\mathcal{A}_\psi|) \cdot (1 + |\chi|) + M)$, calling $M = \{|T' \cup T| \cdot |\Delta|\}$ the intersection of transitions in $TS' \uplus TS$ and in $\mathcal{A}_\psi$. □

Data: $TS' \oplus TS, \mathcal{A}_\phi$, and υ
Result: 'true' if $TS' \models \phi$, 'false' otherwise with a counterexample for ϕ
initialization;
$\mathcal{R} := \{(s_i, s_j) \in TS' \oplus TS | L'(s_i) = L(s_j)\}$;
$\Pi := \emptyset$;
while $\mathcal{R}$ *is not a simulation* **do**
  for $s_i, s_j \in \mathcal{R}$ **do**
    for $s'_i \in Post(s_i), s'_j \in Post(s_j)$ **do**
      if $(s'_i, s'_j) \notin \mathcal{R}$ **then**
        $\mathcal{R} := \mathcal{R} \setminus \{(s_i, s_j)\}$;
        $\Pi := \{(s_i, s_j)\}$
      end
    end
  end
end
Define a transition system TS^* s.t. $TS^* = \{(\Pi, A^*, T^*, I^*, F^*, AP^*, L^*)\}$;
if $TS^* \times \mathcal{A}_\phi \models \boldsymbol{FG}\upsilon$ **then**
  return true
else
  Determine a counterexample $\langle s_0, q_1\rangle, \ldots, \langle s_n, q_{n+1}\rangle, \ldots, \langle s_n, q_{n+1}\rangle$ of $TS^* \times \mathcal{A}_\phi$ such that $q_{n+1} \in F$;
  return (false, $s_0, \ldots, s_n, \ldots, s_n$)
end

Algorithm 4: Model checking algorithm for $TS' \models \phi$ when $TS \leq TS'$.

3.3 Checking liveness in inexact copies

A dual process needs to be defined for checking that an inexact copy model satisfies a liveness property. Algorithm 4 proceeds as before: it looks for states of TS' not being in a simulation relation with states of TS and it checks whether those states satisfy a liveness property ϕ, i.e. whether the product of a reduced transition system TS^*, built over states in Π belonging to TS' but not to TS, with an automaton accepting a language for the violation of ϕ satisfies, in this case, a persistent $\mathbf{FG}\upsilon$. The specific circumstance is here represented by the relevant state satisfying property υ and belonging to a loop transition to itself, or rather not. If the persistent is falsified, the corresponding explored path in TS' is returned as a counterexample together with a 'false' answer.

Also for this case, relevant properties of the algorithm can be proved.

Theorem 10 (Partial Correctness of Algorithm 4)**.** *On termination, Algorithm 4 returns the answer 'false' if and only if $TS' \not\models \phi$.*

Proof. $\Rightarrow$: The answer 'false' is outputted in case:

- a state in Π is obtained that belongs to TS' but not to TS;
- there is at least one state $\langle s_i, q_{i+1}\rangle$ belonging to the intersection $TS^* \times \mathcal{A}_\phi$ such that $\langle s_i, q_{i+1}\rangle \not\models \upsilon$;

- $\langle s_i, q_{i+1} \rangle$ is a successor state of itself, that is, $\langle s_i, q_{i+1} \rangle$ belongs to a self loop or to a loop.

If $\langle s_i, q_{i+1} \rangle \not\models \upsilon$, and if $\langle s_i, q_{i+1} \rangle$ belongs to a (self) loop, then by definition, the intersection $TS^* \times \mathcal{A}_\phi$ is not void and hence there is a path of TS', corresponding to the identified one in TS^*, that violates ϕ.

$\Leftarrow$: If TS' fails to satisfy ϕ, there is at least one state s_i of TS' not in TS, since it is assumed that $TS \models \phi$. Accordingly, $\Pi \neq \emptyset$ and $\Pi \ni s_i \not\models \phi$. Because there is a state in Π violating ϕ, then the intersection $TS^* \times \mathcal{A}_\phi$ cannot be empty, since $\mathcal{A}_\phi$ is the automaton accepting all bad prefixes of ϕ as a language, and TS^* include that state violating ϕ. If this is the case, by definition, there is a persistent which is not satisfied by at least one state in $\Pi \times \mathcal{A}_\phi$. When a state violating υ is identified, and the violating state $\langle s_i, q_{i+1} \rangle$ belongs to a self loop or to a loop, answer 'false' is given by Algorithm 4 as output. □

Theorem 11 (Termination of Algorithm 4)**.** *Algorithm 2 terminates for $TS \leq TS'$.*

Proof. The *wile* loop in Algorithm 4 is defined over the same conditions of the corresponding *while* loop in Algorithm 3. Accordingly, the proof for termination is the same. □

Theorem 12 (Time complexity of Algorithm 4)**.** *The worst time complexity of Algorithm 4 is in $O((|\mathcal{R}| \cdot |\mathcal{A}_\phi|) \cdot (1 + |\upsilon|) + M)$, where $M = \{|T' \cup T| \cdot |\delta|\}$.*

Proof. Time complexity of Algorithm 4 is the same of algorithm 3. This follows from the fact that Algorithm 4, once found state $\langle s_i, q_{i+1} \rangle$ failing to satisfy υ, checks whether $\langle s_i, q_{i+1} \rangle$ is either a successor state of itself or it belongs to a loop. In order to ascertain whether any of these is the case, the state space in $TS^* \times \mathcal{A}_\phi$ still needs to be travelled. Resources needed to do so are therefor still in $O((|\mathcal{R}| \cdot |\mathcal{A}_\phi|) + M)$. □

3.4 Safety and liveness in approximate copies

Let us now consider approximate copies. If TS' is an approximate copy of TS, the latter simulates a subset of the paths of the former, call it P, given by the intersection of the path set of the two structures. The only result that can be proved for approximate copies is that if TS' is a copy of TS, then the safety and liveness properties satisified by TS are satisfied also by P.

Corollary 4 (Safety and liveness in approximate copies)**.** *If $TS \approx TS'$ then $TS \models \psi \rightarrow P \models \psi$ and $TS \models \phi \rightarrow P \models \phi$ for any safety property ψ or liveness property ϕ.*

Proof. If $TS \approx TS'$, then, by Theorem 6, $P \leq TS$. By Theorem 3, any $\forall CTL^*$ property satisfied by TS is also satisfied by P, including any safety or liveness property, known to be expressible in $\forall CTL^*$. □

From Theorem 4 it follows that TS', as an approximate copy of TS, does not necessarily preserve the safety and liveness properties fulfilled by TS. Indeed, only a subset of the paths TS' does preserve safety and liveness, namely P, and the remaining

paths *may* or *may not* fulfill them. As for inexact copies, TS' as an approximate copy needs to be model checked. One can here restrict the state space to be travelled by the model checking algorithm to those states of TS' which are not in P. This can be done by adapting Algorithm 3 for checking safety properties, and algorithm 4 for liveness properties, so as to take respectively inputs $TS' \oplus P, \mathcal{A}_\psi, \chi$ and $TS' \oplus P, \mathcal{A}_\phi, \upsilon$. In this way, the algorithms will isolate states of TS' which are not in P and, for each of those states, it will be checked whether it satisfies the safety or the liveness property.

4 Developing safety- and liveness-preserving systems

The issue of software evolution is a topic of extensive research, see e.g. (Buckley et al., 2005; Chapin et al., 2001; Ernst et al., 2009) for some recent approaches, also in relation to liveness properties (Bloem et al., 2014, 2010) and with the aim of enforcing resilience (Monperrus, 2017). The issue is less explored in view of the notion of functionalities preservation under copying. Additionally, the ethical examination in (Angius and Primiero, 2019) proposes to consider approximate copies of computational artefacts working in safety-critical situations as legally allowed when preserving safety and liveness properties.

The previous section showed that approximate copy models may or may not preserve the safety and liveness properties fulfilled by some copied model. This section now focuses on the principles that, if implemented, would guide the development of safety- and liveness-preserving systems. The section will concentrate on the design principles for the development of a system implementation S' as an approximate copy of a system implementation S with respect to a defined set of safety properties $\psi_1, \cdots, \psi_n$ and a defined set of liveness properties $\phi_1, \cdots, \phi_n$.

Following the approach on software theory change of Primiero et al. (2021), given a computational artefact S as a language implementation of a set of safety properties $\Psi := \{\psi_1, \cdots, \psi_n\}$, a set of liveness properties $\Phi := \{\phi_1, \cdots, \phi_n\}$, and a set of first order properties $F := \{f_1, \cdots, f_n\}$ with a given priority relation $<$ among the corresponding formulas, a model $\mathbf{S}$ of S is formulated as a transition system TS whose states satisfy any formula in S. Recall that a system TS satisfies a functionality $x_i := \{f_i \mid \psi_i \mid \phi_i\}$ if and only if x_i is expressed by a formula p of CTL* and $TS \vDash p$, according to Definition 6. Priority between functionalities satisfied by a system TS can be understood as a preference order on corresponding states:

Definition 14 (Priority between functionalities)**.** *$x_i < x_j$ for any two properties of S iff $\exists \pi_i, \pi_j \in TS$ such that respectively $s_i \in \pi_i, s_i \vDash x_i$ and $s_j \in \pi_j, s_j \vDash x_j$ and $s_i \xrightarrow{a} s_j$.*

Let us consider the TS depicted in figure 1. It is a model of the system defined, among others, by the following CTL^* formulas:[15]

[15]Some of these formulas are not, strictly speaking, satisfied by the TS in figure 1 in that the TS also allows for "unfair" user operations, such as infinitely often opening and closing the door or switching the oven on and off. The TS does satisfy all the listed formulas if corresponding *fairness constraints* are assumed (Clarke et al., 1999, 40-41).

$\mathbf{FG}(off \wedge close)$;
$\mathbf{G}(off \rightarrow \mathbf{F}(on))$;
$\mathbf{G}(on \rightarrow \mathbf{F}(off))$;
$\mathbf{G}(close \rightarrow \mathbf{F}(open))$;
$\mathbf{G}(open \rightarrow \mathbf{F}(close))$;
$\mathbf{G}(on \wedge open \rightarrow error)$;
$\mathbf{G}(on \wedge close \wedge error \rightarrow \mathbf{F}(off \wedge close))$;
$\mathbf{FG}(wave)$;
$\mathbf{G}(\neg(on \wedge open \wedge wave))$.

Among these formulas one may recognize the safety property $\mathbf{G}(\neg(on \wedge open \wedge wave))$ and two liveness properties, namely $\mathbf{FG}(wave)$ and $\mathbf{FG}(off \wedge close)$. The two liveness properties have priority:

$$\mathbf{FG}(off \wedge close) < \mathbf{FG}(wave);$$

the first order formulas have priority:

$$\mathbf{G}(off \rightarrow \mathbf{F}(on)) < \mathbf{G}(close \rightarrow \mathbf{F}(open)) <$$
$$\mathbf{G}(on \wedge open \rightarrow error) < \mathbf{G}(on \wedge close \wedge error \rightarrow \mathbf{F}(off \wedge close)) <$$
$$\mathbf{G}(open \rightarrow \mathbf{F}(close)) < \mathbf{G}(on \rightarrow \mathbf{F}(off)).$$

To each formula defining the system realized by the TS of Figure 1 corresponds a set of paths in TS. For instance, to the formula $\mathbf{G}(off \rightarrow \mathbf{F}(on))$ corresponds a set of paths including path s_1, s_5, s_1, path s_1, s_2, s_3, s_4, s_1, and path s_1, s_5, s_6, s_5, s_1. Considering paths ending in the final state, all paths satisfying $\mathbf{G}(off \rightarrow \mathbf{F}(on))$ thereby satisfy $\mathbf{G}(on \rightarrow \mathbf{F}(off))$. This explains the priority relation given above for the two formulas. Semantically, the priority $\mathbf{G}(off \rightarrow \mathbf{F}(on)) < \mathbf{G}(on \rightarrow \mathbf{F}(off))$ can be seen, according to Definition 14, from the fact that there exists a $\pi_i = \pi_j =_{def} s_1, s_5, s_6, s_5, s_1$ and there exist two states, namely s_1 and s_5, such that $s_1 \xrightarrow{\text{switch on}} s_5$, $s_1 \models off \rightarrow \mathbf{F}(on)$, and $s_5 \models on \rightarrow \mathbf{F}(off)$. The design process leading from S to the system S' to be developed is here understood model-theoretically as the process leading from $\mathbf{S}$ to $\mathbf{S}'$.

4.1 Isolating Safety and Liveness Properties

The first step of the process is to isolate the safety and liveness specifications from $\mathbf{S}$ into its contracted version $\mathbf{S}^{\circ}$, such that the latter includes all the safety and liveness properties of interest, and it removes all the first order properties which are not allowed by the copying operation. In the best version, i.e. where all first order functionalities are removed, and all safety and liveness properties are preserved,

$$\mathbf{S}^{\circ} =_{def} \{S \setminus \{F\}\}.$$

Any contraction operation $(\mathbf{S})^{-}_{f_i}$ with respect to a first order functionality f_i should be safe with respect to the safety and reliability formulas one would like to preserve for $\mathbf{S}'$. Therefore, one should be careful to ascertain that there is not any $s_i \models f_i$ and

$f_i < \phi_i \mid \psi_i$, i.e. any state which satisfies a property object of contraction and which also allows a transition to a state satisfying a liveness or safety property. In principle, one cannot avoid contracting f_i under the ethical assumption that only safety and reliability properties can be copied. This paper argues that, in case f_i implies a safety or a liveness property, one should be allowed to exclude f_i from contraction on S. From the model-theoretic perspective adopted here this corresponds to keeping in $\mathbf{S}^\circ$ only those states that, besides satisfying f_i, satisfy (or lead to states satisfying) ψ_i or ϕ_i. This is meaningful in that, according to the ethical analysis in (Angius and Primiero, 2019), there must be a threshold of the number of paths that approximate copies can copy without incurring into property rights infringement.

Semantically, a contraction operation $(\mathbf{S})^-_{f_i}$ amounts to removing all labels involved in f_i from states in TS; this determines a collapse of states with the same remaining labels together with a collapse of the involved transitions. Accordingly, removing labels is paramount to removing states, in that two states s_i, s_j are the same state iff $L(s_i) = L(s_j)$ and are either initial states or are reachable from the same states. For f_i a functionality object of contraction from S, the set of states of TS satisfying f_i are first defined:

Definition 15. $\Xi = \mathbf{S}(f_i)$ *where* $S(f_i) = \{s_i \mid s_i \vDash f_i\}$

The set of states Ξ_i which can be object of contraction without risk to functionalities satisfied by another state s_j, e.g. where $s_j \models \phi \mid \psi$ is then inductively defined:

Definition 16.

$$\Theta_0 = \Xi$$

$$\Xi_i = \{s_i \mid s_i \in \Theta_i \textit{ and } \neg\exists s_j \textit{ s.t. } s_i \xrightarrow{a} s_j\}$$

$$\Theta_{i+1} = \Theta_i \setminus \Xi_i$$

$$k_0 = min\{k \mid S_i \setminus \bigcup_{i=0}^{k} \Xi_i \nvDash f_i\}$$

Here Θ_i is defined by taking first into account the whole set of states Ξ, and at each step i it is determined the set of states Ξ_i which must be removed to safely contract with respect to f_i. Hence, k is the least number of Ξ_i containing the states that have to be safely removed from S in order to obtain a safe contracted transition system TS°. A TS° for $\mathbf{S}^\circ$ is therefore defined as:

Definition 17 (Safely contracted transition system)**.** *Given a transition system* $TS = (S, A, T, I, F, AP, L)$ *for* $\mathbf{S}$*, a transition system* $TS^\circ = (S^\circ, A^\circ, T^\circ, I^\circ, F^\circ, AP^\circ, L^\circ)$ *for* $(\mathbf{S})^-_{f_i}$ *is defined as follows:*

- $S^\circ = S \setminus \bigcup_{i=0}^{k} \Xi_i$;
- $A^\circ \subseteq A$;
- $T^\circ \subseteq S^\circ \times A \times S^\circ$;
- $I^\circ \subseteq I$;

- $F^\circ \subseteq S^\circ$;
- $AP^\circ \subseteq AP$;
- $L^\circ : S^\circ \to 2^{AP^\circ}$.

Suppose one would like to do without the *error* functionality. In terms of the contraction operation just defined, this means contracting formulas $\mathbf{G}(on \wedge open \to error)$ and $\mathbf{G}(on \wedge close \wedge error \to \mathbf{F}(off \wedge close))$. The contraction determines a new priority relation among the remaining formulas, namely:

$$\mathbf{G}(off \to \mathbf{F}(on)) < \mathbf{G}(close \to \mathbf{F}(open)) < \mathbf{FG}(wave) <$$

$$\mathbf{G}(\neg(on \wedge open \wedge wave)) <$$

$$\mathbf{G}(open \to \mathbf{F}(close)) < \mathbf{G}(on \to \mathbf{F}(off)) < \mathbf{GF}(off \wedge close)$$

The TS in figure 6 is a contraction of the TS in figure 1 with respect to the *error* functionality, that is, with respect to formulas $\mathbf{G}(on \wedge open \to error)$ and $\mathbf{G}(on \wedge close \wedge error \to \mathbf{F}(off \wedge close))$. One can easily check that contracting the two formulas is safe with respect to the safety and liveness properties realized by the uncontracted TS. The label *error* was first removed from all states of the TS in figure 1, causing a collapse of states s_4 and s_5 both labelled with $on, close$ after contraction. In other words, state s_4 was deleted. State s_4 in figure 1 had two outgoing transitions, one towards state s_3 and one towards state s_1. Both states satisfy the safety and liveness properties of interest. However, the contraction is safe in that s_1 and s_3 keep on being reachable after removal of state s_4. By contrast, contracting formulas $\mathbf{G}(close \to \mathbf{F}(open))$ and $\mathbf{G}(open \to \mathbf{F}(close))$, for the *open* and *close* function, would not be safe with respect to the liveness property $\mathbf{FG}(off \wedge close)$. Among other states, state s_1 would change into a new state labelled by off. The old state s_1 had two outgoing transitions, one towards s_2 and one towards s_5, both of which would be re-labelled by off and collapse with state s_1. States s_2 and s_5 satisfied $\mathbf{FG}(off \wedge close)$ in that outgoing transitions to the old state s_1 were given. Clearly, after contraction no state s_i such that $s_i \models off \wedge close$ is reachable anymore.

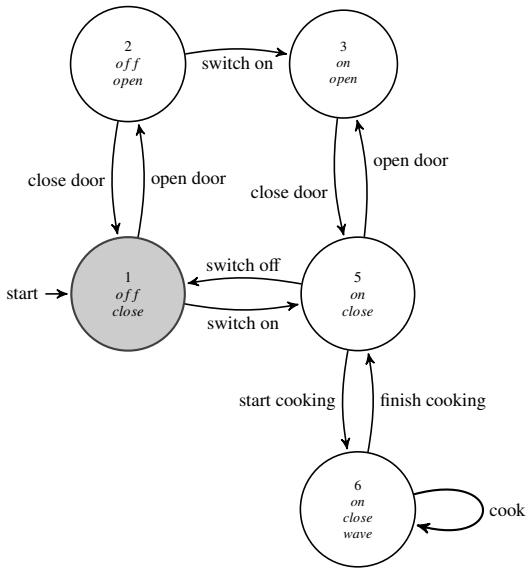

Figure 6: A contraction on the TS in figure 1 with respect to the *error* function and preserving all safety and liveness properties.

Algorithmically, the safe contraction operation is carried out in two steps. First it is computed $(\mathbf{S})^-_{f_i}$ for each f_i of S_i; then, for each contraction, the contracted model is checked and refined in case of unsafe contraction. Algorithm 5 provides a procedure to perform a contraction operation $(\mathbf{S})^-_{f_i}$ that be safe with respect to a general safety or liveness property $\delta_i = \phi_i \mid \psi_i$. This amounts to ascertain that all the remaining states in $(\mathbf{S})^-_{f_i}$ still satisfy, respectively, an invariant or a persistent. The design of the algorithm starts by setting the (minimal) set Δ of safety or liveness properties to be copied from a given TS, the set F of functional properties which are to be checked for preservation in TS°, and a state set Π initially equal to the set of states in TS. For each formula $f_i \in F$ and for each state $s_i \in S$ satisfying f_i, s_i is first included in Ξ_i and removed from Π; subsequently, a contracted transition system TS^* built over the state set Π is considered. In case the TS^* satisfies the safety or liveness property δ_i under consideration, TS^* is taken as the contraction TS°; if TS^* fails to satisfy δ_i, the removed state is included back into the contracted system TS° (and into Π as well).

4.2 Adding new functionalities

The next step is adding new functionalities $g_1 \wedge \cdots \wedge g_n$ into $\mathbf{S}'$. In terms of the formalism advanced in (Primiero et al., 2021), this amounts to an *expansion* operation on $\mathbf{S}^\circ$, namely $(\mathbf{S}^\circ)^+_{g_1 \wedge \cdots \wedge g_n}$. Semantically, an expansion operation $(\mathbf{S}^\circ)^+_{g_i}$ on TS may result in adding a label onto a state of TS, adding a transition to a state of TS, or adding a new state among the states of TS. As shown for the contraction operation on states, the third case can be used to cover both the first and the second case.

Data: TS
Result: TS°
initialization;
$TS := \{(S, A, T, I, F, AP, L)\}$;
$TS^{\circ} := \emptyset$;
$F := \{f_1 < \cdots < f_n\}$;
$\Delta := \{\delta_1 < \cdots < \delta_n\}$;
$\Xi_0 := \emptyset$;
$f_i := f_n$;
$\delta_i := \delta_n$;
$\Pi := S$;
for $f_i \in F, 1 \leq j < n, j := j + 1$ **do**
  for $s_i \in S$ *s.t.* $s_i \vDash f_i, \delta_i \in \Delta, 1 \leq k < n, k := k + 1$ **do**
    $\Xi_i := \Xi_0 \cup s_i$;
    $\Pi := \Pi \setminus \Xi_i$;
    Define a transition system TS^* s.t. $TS^* = \{(\Pi, A^*, T^*, I^*, F^*, AP^*, L^*)\}$;
    if $TS^* \vDash \delta_i$;
     then
      $TS^{\circ} := TS^*$;
    else
      $\Pi := \Pi \cup s_i$;
      $TS^{\circ} := \{(\Pi, A^{\circ}, T^{\circ}, I^{\circ}, F^{\circ}, AP^{\circ}, L^{\circ})\}$;
    end
    $\delta_i := \delta_{n-k}$;
  end
  $f_i := f_{n-j}$;
end
return TS°;

Algorithm 5: Algorithm for safe contraction with respect to a safety or a liveness property $\delta_i = \phi_i \mid \psi_i$.

In order to avoid that adding new functionalities results in a violation of a safety or a liveness property, it is essential to check whether $g_i \models \neg\phi_i$ or $g_i \models \neg\psi_i$, to preserve consistency. For safety this amounts to avoid the case in which $(\mathbf{S}^\circ)^+_{g_i}$ contains states violating a property μ for an invariant $\mathbf{G}\mu$. For liveness this amounts to avoid the case in which $(\mathbf{S}^\circ)^+_{g_k}$ contains paths not going through existing loops satisfying a formula ν for a persistent $\mathbf{FG}\nu$. An ordered consistent expansion ensues as:

Definition 18 (Ordered Consistent Expansion)**.** *An ordered expansion is denoted as*

$$(TS)^+_{f_i} := (TS^\circ \cup S(g_i), T')$$

where

1. $S(g_i) = \{s_i \mid s_i \models g_i\}$
2. $T' = T^\circ \cup \{(s_k, s_i), (s_i, s_j), (s_j, s_l), (s_l, s_k) \mid s_j \in \Xi, s_k \in TS^\circ \setminus \Xi, s_l \models \nu\}$*, given:*
   - $\Xi = TS^\circ \cap S(f_j)$ *and* $g_i \Rightarrow f_j$
   - $s_i \models \mu$

Here the intuition is that an added functionality is introduced in terms of the states that realize it ($s_i \models g_i$) and such that they also satisfy the invariant ($s_i \models \mu$). Those states need to have transitions from states already occurring in the system (s_k, s_i), and to states (also possibly already occurring) which have functionalities that are logically implied by the new functionality (s_k, s_j, such that $s_j \models f_j$). Moreover, the definition accounts for states satisfying the formula for the persistent ($s_l \models \nu$) and belonging to a loop ($(s_k, s_i), ..., (s_l, s_k)$) which is still reachable after the new states have been added.

A TS' for $\mathbf{S}^+_{f_i}$ is therefore defined as

Definition 19. *A transition system* $TS' = (S', A', T', I', F', AP', L')$ *for* $(\mathbf{S})^+_{g_i}$ *is defined as follows:*

- $S' = S^\circ \cup S(g_i)$*;*
- $A' \supset A^\circ$*;*
- $T' \subseteq S' \times A' \times S'$*;*
- $I' \subseteq S'$*;*
- $F' \subseteq S'$*;*
- $AP' \supset AP^\circ$*;*
- $L' : S' \to 2^{AP'}$.

Suppose one would like to add the function *grill* to the KS° of Figure 6. As a first step, a set of property specifications in the form of CTL^* formulas has to be advanced. This may include:

- $\mathbf{FG}(grill)$;

- $\mathbf{G}\neg(wave \wedge grill)$ (supposing one does not want to implement a combo function microwave together with grill);
- $\mathbf{G}\neg(off \wedge grill)$;
- $\mathbf{G}\neg(open \wedge grill)$.

The new formulas have initially to be added to the formula set defining S°. Recall that the latter was given the following priority order:

$$\mathbf{G}(off \rightarrow \mathbf{F}(on)) < \mathbf{G}(close \rightarrow \mathbf{F}(open)) <$$
$$\mathbf{G}(\neg(on \wedge open \wedge wave)) < \mathbf{FG}(wave) <$$
$$\mathbf{G}(open \rightarrow \mathbf{F}(close)) < \mathbf{G}(on \rightarrow \mathbf{F}(off)) < \mathbf{GF}(off \wedge close)$$

The new priority relation is given by:

$$\mathbf{G}(off \rightarrow \mathbf{F}(on)) < \mathbf{G}(close \rightarrow \mathbf{F}(open)) <$$
$$\mathbf{G}(\neg(on \wedge open \wedge wave)) < \mathbf{G}\neg(wave \wedge grill) < \mathbf{G}\neg(off \wedge grill) <$$
$$\mathbf{G}\neg(open \wedge grill) < \mathbf{FG}(wave) \leq \mathbf{FG}(grill) <$$
$$\mathbf{G}(open \rightarrow \mathbf{F}(close)) < \mathbf{G}(on \rightarrow \mathbf{F}(off)) < \mathbf{GF}(off \wedge close)$$

The resulting KS is depicted in figure 7. State 7 is the new added state, satisfying all the newly added formulas, without preventing the expanded KS from satisfying any of the former first order properties and, especially, any of the former safety and liveness properties. Indeed, state 7 satisfies $\neg(on \wedge open \wedge wave)$ and all paths going through it can still reach a loop containing a state satisfying *wave*.

The algorithmic process describing how to construct the new TS with the additional states and transitions is presented in Algorithm 6, receiving as inputs the previously contracted transition system TS° and a priority relation R among the added functionalities and the ones implied by them. The algorithm initializes a set G of added functionalities, a set $S(g_i)$ of states satisfying them, and sets M and N of, respectively, invariants and persistents. For any formula g_i to be expanded, any state s_i satisfying g_i, any formula μ_i for the invariant $\mathbf{G}\mu$, and any formula ν for the persistent $\mathbf{FG}\nu$, the algorithm adds any state s_i that satisfies g_i provided that it also satisfies μ. State s_i is added such that the paths going through it be still able to reach a loop containing a state satisfying ν, as by definition 18.

5 Conclusions

This paper extended the ontological analysis of the copy relation for computational artefacts advanced in (Angius and Primiero, 2018) to ask whether models of exact, inexact, and approximate copies do or do not preserve safety and liveness properties. Exact copies, by being duplicates of the original system, are the only kind of copies that preserve all behavioural properties, safety and liveness included. Inexact and approximate copies do not necessarily preserve the fulfillment of safety and liveness.

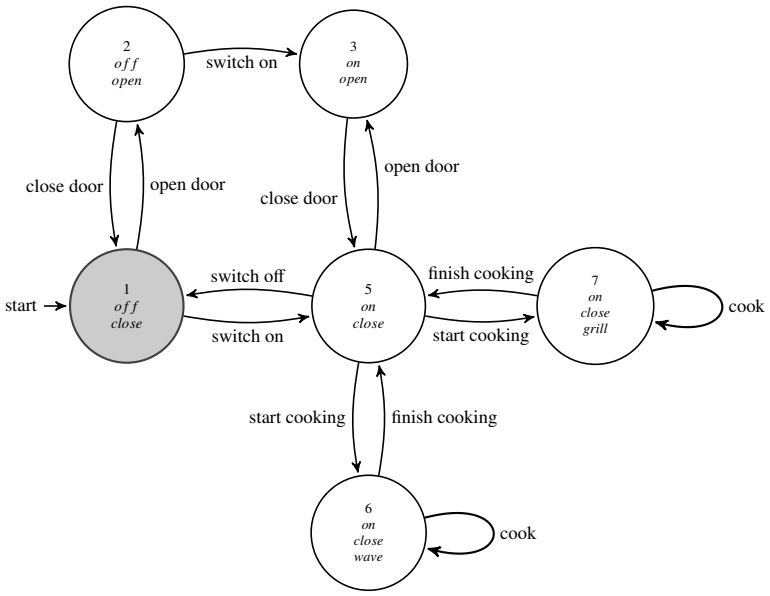

Figure 7: An expansion of the TS in figure 1 with respect to the *grill* function and preserving all safety and liveness properties.

The former simulate all the behaviours of the copied artefact, but include additional behaviours; the latter simulate only some behaviours of the original model, including many additional behaviours. This paper showed how only non-simulating computational paths have to be travelled in order to verify the safety and liveness properties of inexact and approximate copies. The two algorithms here proposed do so staying within a polynomial time-complexity. Finally, the paper proposed two algorithmic operations that, at the semantic specification level, allow one to design safety- and liveness-preserving approximate copies. First, by means of a contraction operation all prescribed computations not implying safety and liveness properties are removed. Second, an expansion operation includes, in the contracted model, any desired additional behaviour provided that it does not violate any of the required safety and liveness properties.

It has been shown how the contraction operation, besides removing all states satisfying the formulas to be contracted, reduces the state space of the contracting model by collapsing transitions and states with the same remaining labels. In other words, given a set of safety and liveness formulas, the contraction operation here proposed preserves those formula while keeping the minimal set of paths satisfying them. The algorithmic design principles described here are in line with two philosophically relevant positions. First, such algorithmic design principles avoid disfunctioning copies with respect to safety and liveness properties, as by the definition of negative malfunctioning for software systems provided in (Floridi et al., 2015). Second, the analysis complies with, and extends, the ethical examination in (Angius and Primiero, 2019) concerning the

Data: TS°, R
Result: TS^{+}
initialization;
$TS^{\circ} = (S^{\circ}, A^{\circ}, T^{\circ}, I^{\circ}, F^{\circ}, AP^{\circ}, L^{\circ})$;
$G := \{g_1, \dots g_n\}$;
$g_i := g_1$;
$R := \{f_j \mid f_j > g_i\}$;
$\Xi := TS^{\circ} \cap S^{\circ}(f_j)$;
$S(g_i) := \{s_i \mid s_i \models g_i\}$;
$M := \{\mu_1, \dots, \mu_n\}$;
$N := \{\nu_1, \dots, \nu_m\}$;
$\mu_i := \mu_n$;
$\nu_i := \nu_m$;
$S^{+} := S^{\circ}$;
$T^{+} := T^{\circ}$;
for $g_i \in G, 1 \leq i < n, i := i + 1$ **do**
  for $s_i \in S(g_i)$ **do**
    for $\mu_i \in M, \nu_i \in N, 1 \leq j < n, 1 \leq k < m, j := j + 1, k := k + 1$ **do**
      if $s_i \models \mu_i$;
      then
        $S^{+} := \{S^{\circ} \cup s_i\}$;
        for $f_j > g_i \in R$ **do**
          $T^{+} := T^{\circ} \cup \{(s_k, s_i), (s_i, s_j), (s_j, s_l), (s_l, s_k) \mid s_j \in \Xi, s_k \in TS^{\circ} \setminus \Xi, s_l \models \nu_i\}$;
        end
      end
      $\mu_i := \mu_{n-j}$;
      $\nu_i := \nu_{n-k}$;
    end
  end
  $g_i := g_{i+1}$;
end
Define a transition system TS^{+} s.t. $TS^{+} = (S^{+}, A^{+}, T^{+}, I^{+}, F^{+}, AP^{+}, L^{+})$;
return TS^{+}

Algorithm 6: Algorithm for safe expansion with respect to safety and liveness properties

allowance of approximate copies in the software industry: copying should count as a "fair usage" if not trespassing a threshold defined by the law and if copying is necessary to guarantee important security properties for safety-critical systems. In particular, this paper argued that copying should be allowed not only with respect to safety and liveness properties, but also to those properties that, if removed, would cause a violation of those security properties. Finally, the algorithmic operations here proposed guarantee an *effective legal copying*, that is, procedures that can be implemented to ensure

intellectual property right protection.

Future steps of this research will aim at extending the proposed ontological, epistemological, and formal analyses of the various relations of copy of physical artefacts for probabilistic systems (see e.g. Termine et al. (2021a)) as they are implemented in AI technologies, e.g. in the digital twin vision in industry. Future research will therefore formulate philosophical and algorithmic analyses of properties, such as trustworthiness examined for example in (Termine et al., 2021b), and their verification, to guarantee the reliable transfer between digital and physical artefacts.

Acknowledgements

The authors acknowledge the Research Project ANR-17-CE38-0003-01 (ANR – Agence Nationale de la Recherche) titled "What is a (computer) program: Historical and Philosophical Perspectives". Giuseppe Primiero has been partially funded by the Project PRIN2020 BRIO (2020SSKZ7R) awarded by the Italian Ministry of University and Research (MUR).

References

Alpern, B. and Schneider, F. B. (1985). Defining liveness. *Information processing letters*, 21(4):181–185.

Alpern, B. and Schneider, F. B. (1987). Recognizing safety and liveness. *Distributed computing*, 2(3):117–126.

Angius, N. and Primiero, G. (2018). The logic of identity and copy for computational artefacts. *Journal of Logic and Computation*, 28(6):1293–1322.

Angius, N. and Primiero, G. (2019). Infringing software property rights: Ontological, methodological, and ethical questions. *Philosophy & Technology*, pages 1–26.

Baier, C. and Katoen, J.-P. (2008). *Principles of model checking*. MIT press.

Biere, A., Artho, C., and Schuppan, V. (2002). Liveness checking as safety checking. *Electronic Notes in Theoretical Computer Science*, 66(2):160–177.

Bloem, R., Chatterjee, K., Greimel, K., Henzinger, T. A., Hofferek, G., Jobstmann, B., Könighofer, B., and Könighofer, R. (2014). Synthesizing robust systems. *Acta Informatica*, 51(3):193–220.

Bloem, R., Chatterjee, K., Greimel, K., Henzinger, T. A., and Jobstmann, B. (2010). Robustness in the presence of liveness. In Touili, T., Cook, B., and Jackson, P., editors, *Computer Aided Verification*, pages 410–424, Berlin, Heidelberg. Springer Berlin Heidelberg.

Buckley, J., Mens, T., Zenger, M., Rashid, A., and Kniesel, G. (2005). Towards a taxonomy of software change. *Journal of Software Maintenance and Evolution: Research and Practice*, 17(5):309–332.

Carrara, M. and Soavi, M. (2010). Copies, replicas, and counterfeits of artworks and artefacts. *The Monist*, 93(3):414–432.

Chandrasekaran, B. and Josephson, J. R. (2000). Function in device representation. *Eng. Comput. (Lond.)*, 16(3-4):162–177.

Chapin, N., Hale, J. E., Khan, K. M., Ramil, J. F., and Tan, W.-G. (2001). Types of software evolution and software maintenance. *Journal of Software Maintenance and Evolution: Research and Practice*, 13(1):3–30.

Clarke, E. M., Grumberg, O., Kroening, D., Peled, D., and Veith, H. (1999). *Model checking*. MIT press, Cambridge, MA.

Ernst, N. A., Mylopoulos, J., and Wang, Y. (2009). Requirements evolution and what (research) to do about it. In Lyytinen, K., Loucopoulos, P., Mylopoulos, J., and Robinson, B., editors, *Design Requirements Engineering: A Ten-Year Perspective*, pages 186–214, Berlin, Heidelberg. Springer Berlin Heidelberg.

Floridi, L., Fresco, N., and Primiero, G. (2015). On malfunctioning software. *Synthese*, 192(4):1199–1220.

Fokkink, W. (2013). *Introduction to process algebra*. springer science & Business Media.

Frege, G. (1953). *The Foundations of Arithmetic a Logico-Mathematical Enquiry Into the Concept of Number. English Translation by JL Austin*. Basil Blackwell, Oxford.

Fresco, N. and Primiero, G. (2013). Miscomputation. *Philosophy & Technology*, 26(3):253–272.

Haibe-Kains, B., Adam, G. A., Hosny, A., Khodakarami, F., Waldron, L., Wang, B., McIntosh, C., Goldenberg, A., Kundaje, A., Greene, C. S., et al. (2020). Transparency and reproducibility in artificial intelligence. *Nature*, 586(7829):E14–E16.

Hick, D. H. and Schmcke, R. (2016). *The Aesthetics and Ethics of Copying*. Bloomsbury Academic.

Hutson, M. (2018). Artificial intelligence faces reproducibility crisis. *Science*, 359(6377):725–726.

Johnson, D. G. and Miller, K. (2008). *Computer ethics*. Pearson, New York.

Kröger, F. and Merz, S. (2008). *Temporal Logic and State Systems*. Springer.

Kupferman, O. and Vardi, M. Y. (2001). Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314.

Lombard, L. B. (1986). *Events: A metaphysical study*. Routledge & Kegan Paul Books, London.

Lowe, E. J. (1989). What is a criterion of identity? *The Philosophical Quarterly (1950-)*, 39(154):1–21.

Lowe, E. J. (1997). Objects and criteria of identity. In Hale, B. and Wright, C., editors, *A Companion to the Philosophy of Language*, pages 990–1012. Blackwell Publishers Ltd, Oxford.

Monperrus, M. (2017). Principles of antifragile software. In *Companion to the First International Conference on the Art, Science and Engineering of Programming*, Programming '17, pages 32:1–32:4, New York, NY, USA. ACM.

Nissenbaum, H. (1995). Should I copy my neighbor's software. *Computer Ethics and Social Values. Prentice Hall, Upper Saddle River, NJ*, pages 200–213.

Noonan, H. and Curtis, B. (2018). Identity. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2018 edition.

Podelski, A. and Rybalchenko, A. (2003). Software model checking of liveness properties via transition invariants. *Technical Report MPI-I-2003-2-00, Max-Planck-Institut für Informatik*.

Primiero, G., Raimondi, F., and Chen, T. (2021). A theory of change for prioritised resilient and evolvable software systems. *Synth.*, 198(23):5719–5744.

Rushby, J. (1994). Critical system properties: Survey and taxonomy. *Reliability Engineering & System Safety*, 43(2):189–219.

Sangiorgi, D. (2011). *Introduction to bisimulation and coinduction*. Cambridge University Press.

Sistla, A. P. (1994). Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6(5):495–511.

Termine, A., Antonucci, A., Primiero, G., and Facchini, A. (2021a). Logic and model checking by imprecise probabilistic interpreted systems. In Rosenfeld, A. and Talmon, N., editors, *Multi-Agent Systems - 18th European Conference, EUMAS 2021, Virtual Event, June 28-29, 2021, Revised Selected Papers*, volume 12802 of *Lecture Notes in Computer Science*, pages 211–227. Springer.

Termine, A., Primiero, G., and D'Asaro, F. A. (2021b). Modelling accuracy and trustworthiness of explaining agents. In Ghosh, S. and Icard, T., editors, *Logic, Rationality, and Interaction - 8th International Workshop, LORI 2021, Xi'ian, China, October 16-18, 2021, Proceedings*, volume 13039 of *Lecture Notes in Computer Science*, pages 232–245. Springer.

Tzouvaras, A. et al. (1993). Significant parts and identity of artifacts. *Notre Dame Journal of Formal Logic*, 34(3):445–452.

Wiggins, D. (2001). *Sameness and substance renewed*. Cambridge University Press.