The Logic of Identity and Copy for Computational Artefacts

Nicola Angius
Department of History, Human Sciences, and Education
University of Sassari, Italy
nangius@uniss.it

Giuseppe Primiero
Department of Computer Science
Middlesex University, London, UK
g.primiero@mdx.ac.uk

Abstract

Defining identity for entities is a longstanding logical problem in philosophy, and it has resurfaced in current investigations within the philosophy of technology. The problem has not yet been explored for the philosophy of information, and of Computer Science in particular. This paper provides a logical analysis of identity and copy for computational artefacts. Identity is here understood as the relation holding between an instance of a computational artefact and itself. By contrast, the copy relation holds between two distinct computational artefacts. We distinguish among exact, inexact and approximate copies. We use process algebra to provide suitable formal definitions of these relations, using in particular the notion of bisimulation to define identity and exact copies, and simulation for inexact and approximate copies. Equivalence is unproblematic for identical computational artefacts at each individual time and for inexact copies; we will examine to which extent the formal constraints on identity criteria discussed in the literature are satisfied by our approach. As for inexact and approximate copy, they are intended as a weakening of the identity relation in that equivalence and other constraints on identity are violated. The proposed approach also suggests a computable treatment of identity and copy checking.

Keywords. Philosophy of Information; Philosophy of Computer Science; Identity Criteria; Copy; Process Algebra.

1 Introduction

Since Frege, the problem of identity of the informational content of two sentences is a fundamental one in the philosophy of language and information. The more recent debate in the philosophy of technology has exploited this problem for the analysis of

technical artefacts. The problem of identity for technical artefacts can be rephrased as the one concerning the informational content of sentences describing their functional properties.

Computational artefacts are informational systems defined at several levels of abstraction for which the problem of identity has not been investigated within the philosophy of computer science (Turner and Angius, 2017). Furthermore, the closely related notion of copy assumes a special significance in the context of computational artefacts which are particularly subject to replication, inducing legal, ethical, and technical issues.

Available analyses for identity and copy of technical artefacts are largely conceptual and have typically an informal nature. This paper contributes to the philosophical debate on identity and copy by providing a rather formal analysis of those concepts for computational artefacts. The examination of identity and copy for computational artefacts is here carried out at the specification level. This allows us to make use of process algebra to understand the formal relations holding between two specifications for distinct artefacts.

The paper is structured as follows. The problem of identity of natural objects in analytic ontology is introduced in Section 2 together with the identity criteria that entities of a given kind are required to satisfy in oder to be considered objects. It is then shown how the problem has been reconsidered in the philosophy of technology in terms of the so-called problem of ontological respectability of artefacts. After introducing some formal preliminaries in the representation and specification of computational artefacts in section 3, a taxonomy of identity and copy relations, that are of significance in the examination of computational artefacts, is provided in section 4. This paper in particular distinguishes among three different sub-categories of copy, namely exact copies, inexact copies, and approximate copies. Section 5 introduces the formal relation of bisimulation on states to capture a suitable identity relation satisfying an identity criterion for computational artefacts. It is shown how, besides being an equivalence relation, bisimulation on states satisfies all of the formal constraints on identity criteria for natural objects. The close relation of bisimulation between state transition systems is used, in section 6, to capture the notion of exact copies, which is shown to satisfy all the constraints satisfied by the identity relation for computational artefacts. The relation of simulation is then used to formalize the notions of inexact copy and approximate copy. Inexact copy is shown here to be a weaker relation than identity, in that it is not symmetric and it violates some additional constraints. And the approximate copy relation is finally shown to be weaker than identity and inexact copy in that it violates transitivity while it preserves symmetry. We conclude in Section 7 to illustrate further developments and applications of the present formal framework.

2 Identity and Copy: from Analytic Ontology to the Philosophy of Technology

The notion of identity in contemporary analytic ontology arises in connection with the problem of defining what an *object* is. Whereas Frege (1892), and the tradition follow-

ing it (Quine, 1948; Wright, 1983), argued that an object is anything that is associated with interpretations of terms and variables, others, such as (Lowe, 1998; Quine, 1990; Wiggins, 2001), argue that an object is anything satisfying defined *identity criteria*. Such a metaphisical approach (Lowe, 1997) on the definition of object is also involved in closely related metaphysical problems. For instance, *counting* objects falling under a given sortal concept (such as counting cats in a room) presupposes the identification of identity criteria for that sortal, so that each object is counted only once, and counting twice, or more, the *same* objects is avoided. Identity is also involved in the interpetation of modal predicates using possible worlds semantics; for instance, $\exists x(\Diamond Px)$ is true if and only if there is an x and there is at least one accesible world with an individual *identical* to x having property P (Lewis, 1986). And indentity is also involved in mereology when dealing with the problem of whether the composition of the parts of a whole is *identical* or not with the whole (Baxter, 1988; Lewis, 1991).

According to the 'classical view' on identity (Noonan and Curtis, 2017), identity is the reflexive, symmetric, and transitive relation that each object has with itself and with nothing else, and such that it satisfies Leibniz's Law, i.e. the principle of the indiscernibility of identical. According to Leibniz's Law, if x and y are two identical objects, then, whatever is true of x is also true of y. Much of the philosophical reflection on identity turned around the problem of establishing identity crtieria. The problem of identity criteria was initially put forward by Frege (1953) and can be splitted into (1) an epistemic question, (2) an ontological question, and (3) a semantic question on identity (Carrara et al., 2014). According to this vew, any kind K of objects determines an identity criterion for objects of kind K. Given a kind K and objects x and y of kind K:

1. How can one know that $x = y$?
2. In what consists the identity relation $x = y$ between x and y?
3. When do x and y have the same interpretation?

Whereas the epistemic question (1) demands for a procedure to *decide* indentity questions, the ontological question (2) refers to properties that objects of the same kind must share in order to be identical. Frege (1953) defined the identity criterion for the identity relation $x = y$ in terms of a relation $R(x', y')$ holding between objects x' and y' different from, but functionally related to, x and y. For instance, identity of directions was defined in terms of parallelism between lines in the following way:

$$\forall x \forall y((Line(x) \wedge Line(y)) \rightarrow (Direction(x) = Direction(y) \leftrightarrow Parallel(x, y)))$$

The identity criterion for directions specified by Frege can be generalized with the following formula, wherein f is a functional expression (Lowe, 1989):

$$\forall x \forall y(f(x) = f(y) \leftrightarrow R(x, y)) \tag{1}$$

Identity criteria of the form of equation 1 have been called by Williamson (1990) two-level criteria, in that the equivalence relation $R(x, y)$ defines an identity relation

between objects $f(x)$ and $f(y)$ distinct from x and y. By contrast, one-level identity criteria define an identity relation $x = y$ between two objects x and y of kind K by means of an equivalence relation $R(x, y)$ holding just between x and y, according to the following formula:

$$\forall x \forall y((x, y \in K) \rightarrow ((x = y) \leftrightarrow R(x, y))) \tag{2}$$

As an example, consider the following identity criterion proposed by Davidson (1969) for events:

$$\begin{gathered}\forall x \forall y((Event(x) \wedge Event(y)) \rightarrow \\ (x = y \leftrightarrow \forall z(Event(z) \rightarrow \\ ((Cause(x, z) \leftrightarrow Cause(y, z)) \wedge (Cause(z, x) \leftrightarrow Cause(z, y)))))\end{gathered}$$

Here if x and y are two objects of the kind events, they are identical if and only if they cause, and are caused by, any distinct event z.

Some argue that two-level criteria are more appropriate to define identity of *abstract* objects, while one-level criteria are more suitable for *concrete* objects (Noonan and Curtis, 2017). Some others argue that it is always possible to reduce a two-level criterion to a one-level criterion so that one can dispense with two-level criteria (Lowe, 1997). For both reasons, this paper will focus only on one-level identity criteria for computational artefacts.

The equivalence relation R in equation 2 can be used to define both *synchronic* and *diachronic* identity. Synchronic identity of an object x is the identity of x with itself at a given time t. Diachronic identity is the identity of object x at time t with x at any time $t' > t$. In this paper we will focus in particular on the former and only provide remarks for an analysis of the latter.

Independently of whether synchronic or diachronic identity is considered, relation R is required, besides equivalence, to comply with different *constraints* (Lombard, 1986; Wiggins, 2001). Following Carrara and Giaretta (2001), this paper considers the following list of constraints:

Non-vacuousness. R must refer to properties that are relevant for defining identity between objects x and y of a given kind K, and such that they be not trivially satisfiable by x and y. For instance, being parallel is a relevant property to define sameness of directions of two lines, while being 5 centimeters long is not. Significant properties of this sort have been called by Lombard (1986) *determinables*, and objects satisfying them *determinates*.

Informativeness. As stated above, it is kind K that determinates the identity criterion for objects of kind K. Nevertheless, it is required that relation R in the identity criterion is informative with respect to K, in the sense that R should not specify tautological properties.

Partial Exclusivity. Besides being non-vacuous, relation R for kind K should specify determinables such that the determinates satisfying them are only objects of kind K. In other words, R should not appear in the identity criterion for objects of kind different than K and each kind of objects should have its own distinct identity criterion.

Minimality. R should include the smallest set of determinables being both necessary and sufficient to determiny identity of any two objects of kind K. Minimality ensures that superfluous determinables are not considered.

Non-circularity. R must not make reference to the identity relation.

Non-totality. It is required that $R \subset K \times K$, that is, that not all object pairs in K satisfy the identity relation defined by R.

K-maximality. R should be the maximal equivalence relation defining identity of objects of some kind K. Given two K objects x and y, for any different relation R' defining identity for K objects, it should hold that $R'(x, y) \rightarrow R(x, y)$ but $\neg(R(x, y) \rightarrow R'(x, y))$.

Uniqueness. In addition, R must be unique with respect to K, that is, for any R', also $\neg(R'(x, y) \rightarrow R(x, y))$ holds. In other words, R is unique with respect to K when there are neither wider, nor narrower relations than R for K objects.

Equivalence. R is required to be reflexive, symmetric, and transitive.

Congruence. If $R(x, y)$, then Leibniz's Law holds: all properites satisfied by x are all and only the properties satisfied by y.

The problem of defining identity of technical artefacts is almost as old as the problem of identity for natural kinds. It traces back to the second book of Aristotelian *Physica* and the distinction between things that 'exists by nature' and 'artificial products'. Aristotle argued that natural objects can be distinguished from technical artefacts in that the latter lack *form*. Whereas natural objects, such as water, earth, and animals, change according to principles that are inner to them, artefacts change according to principles inner to the artefacts' material or to human intervention. Paradigmatic is the case, known till antiquity, of the ship of Theseus, i.e. the ship that Theseus used to reach Crete and defeat the Minotaur. Athenians wanted to retain the ship and preserved it by gradually replacing worn planks by new planks. The question is whether the ship retained by the Athenians was the *same* ship of Theseus.

The problem of the distinction between natural obejcts and technical artefacts is put forward, by contemporary philosophy of technology, in terms of the so-called problem of *ontological respectability* of artefacts (Wiggins, 2001): artefacts are ontologically respectable, i.e. they are on a par with natural objects, if and only if they are able to satisfy identity criteria. Wiggins (2001) defines an artefact kind K as composed of all those artefacts satisfying a given set of functional requirements. And two artefacts x and y of kind K are said to be identical if and only if they satisfy the same sub-set of functional requirements. However, it is known that artefacts correctly instantiating their functional requirements may start, at a certain point, to malfunction, that is, they may start, under usage, violating one or more requirements. Consequentely, transitivity of relation R in equation 2 may not hold. Artefact x at time t may still be said to be identical to artefact x' at time t' even though x' starts malfunctioning. The same can be said for artefacts x' and x'' at time t''. However there may be a time t^* at which artefact x^* is not identical to x insofar as too many requirements are being violated by x^*. Identity criteria for technical artefacts thus violate the equivalence constraint. For the same reason they violate the congruence constraint, in that it may be the case that not all that is true of artefact x is still true of artefact x^*.

Wiggins' critique highlights a potential limitation to any approach using functions

to determine identity criteria. This includes, in principle, an understanding of computational artefacts in terms of their functional requirements. Below, starting in Section 3, we provide a description of computational artefacts based on their layered ontology, which includes the designer's intention, its translation to a functional specification, and their implementation: this exposes our analysis to a critique similar to Wiggins'. To dispel any doubt on this matter, let us briefly consider two main approaches to functional analysis in engineering:

- *Functional Representation*: this approach defines a function in a device-centric sense; it starts from a description of the intended function independently of how the function is accomplished, followed by a description of the structure of the device in terms of its components' composition, finally matching the device to the function by offering a process description, see Chandrasekaran (1994);
- *Functional Basis*: this approach aims at constructing a functional design of an artefact; it starts from a generic input/output relationship, followed by its decomposition in terms of sub-functions, inductively generating a function flow and then the artfeact's process in terms of a model and a structure, finally combined in a design language, see Stone and Wood (1999).

The approach followed below for computational artefacts resembles the Functional Representation approach. We provide a functional description not just as a characterization of a function, but rather as a description of the artefact involved in the execution of such a function. As computational artefacts present multiple realizability of their functional description (several languages and architectures can implement the same functional requirements), this cannot be done in terms of the device's components composition. Instead, we do so by formulating functions in terms of *behaviours*, clarifying in Section 3 in which sense are such behaviours intended. For the purposes of this section, it is just essential to note the following: on the one hand, the notion of behaviour used here allows us to side-step Wiggins' critique, in that behaviours provide a low-level treatment of malfunctions, hence not falling in the same problem that functions have; on the other hand, while computational artefacts can in principle be understood as I/O black boxes, the use of behaviours to characterise functional representations of such artefacts allows us to avoid a functional basis interpretation.

In contrast to natural objects, the notion of *copy* is proper of technical – and hence specifically of computational – artefacts, but also of artworks (Goodman, 1968). Except for some analyses (Carrara and Soavi, 2010; Hick and Schmcke, 2016; Tzouvaras et al., 1993), there is not a well-established and accepted analysis of the notion of copy in the philosophy of technology. Tzouvaras et al. (1993) understand copy as an equivalence relation holding between two artefacts x and y such that x and y are *functionally interchangeable*, that is, such that each can replace the other one in a given system, by playing the same functional role. Carrara and Soavi (2010) critizise Tzouvaras'definition in that it does not take into account the intentional aspect of copying. Copying means reproducing an artecfact taken as a model: if y is a copy of x, the latter is the model of the former and it cannot be said that, at the same time, x is a copy of y. It follows that the copy relation is neither reflexive, nor symmetric, and hence it is not an equivalence relation.

Transitivity does not apply either, at least in those cases in which copying means reproducing an artefact taken as a model: if x is the model for the copy y and y is also taken as a model for copy z, it does not hold that x is the model of z. Also consider that, if x is used as a model to produce y, y is supposed to presents structural similarities or resemblances with x. If x' resembles x, and x'' resembles x', there will be a point in which some x^* cannot be said to resemble x. There are cases, though, in which copying consists of creating a 'perfect copy', such as when duplicating digital artefacts, in which copying is transitive. Perfect copies allowed by the producer of the original artefacts are callled by Carrara and Soavi (2010) 'replicas', this being the case, for instance, of the copies of a piece of software released by a software company.

This paper contributes to fill the gap in the analysis of the notion of copy for computational artefacts by defining it as a weaker relation than identity. To do so, we first define identity for compuatational artefacts, that is, a candidate for relation R in equation 2 and show that it is an equivalence relation complying with all the constraints for the identity criteria. Secondly, we consider candidates for the copy relation R' in equation 3 below, defining when $x \leftsquigarrow y$, that is, when a computational artefacts y is a copy of a distinct artefact x:

$$\forall x \forall y((x, y \in K) \rightarrow ((x \leftsquigarrow y) \leftrightarrow R'(x, y))) \tag{3}$$

The proposed candidates for relation R' in equation 3 are shown, besides violating equivalence, not to comply with all constraints for indentity.

Preliminarily, let us introduce some formal notions concerning the representation and the specification of computational artefacts.

3 Formal preliminaries for Computational Artefacts and their Specifications

Technical artefacts are artefacts, i.e. human made systems, built with the specific purpose of fulfilling some *functions*, in contrast with other kind of artefacts, such as artworks, wich are not supposed to implement functions. It follows that technical artefacts can be defined on the basis of both *functional properties*, dealing with the functions to be fulfilled, and *structural properties*, concerning the phisical properties of the artefacts that allow it to accomplish the intended funtions (Kroes, 2009; Meijers, 2000). Functions usualy reflect intentions of both designers and users (Kroes, 2012) and they can be multiply-realised by different artefacts[1]. What is often called the *dual nature* of technical artefacts (Kroes and Meijers, 2006) is also reflected in their design process: stakeholders' requirements are ususally translated into a set of *functional requirements* on the basis of which a set of *design specifications* are advanced, flowing into a final *bluprint* of the artefacts (Franssen et al., 2015).

Computational artefacts can be described in terms of functional and structural properties, and their design process associated with that of technical artefacts. However, the

[1]Issuess arise here in connection with the problem of how functions, *qua* human intentions, can constrain the structural substratum (Franssen et al., 2015). Addressing this problem requires a proper theory of functions (Kroes, 2009; Vermaas and Houkes, 2003)

nature of computational artefacts cannot be fully understood by the dual ontology approach of Kroes and Meijers (2006). This is due to the role *programs* play in the design process (Floridi et al., 2015). Functional requirements are tranlsated into *program specifications* which are in turn implemented by high-level language programs. For this reason, programs themselves can be understood as technical artefacts providing design specifications for the lower structural levels (Turner, 2014). Also, programs can instantiate specifications in several ways, thereby adding a further level of multiple realizability in the design process.

Accordingly, computational artefacts are characterised by a more layered ontology requiring a description at several levels of abstraction (LoA) (Floridi, 2008; Fresco and Primiero, 2013). Primiero (2016) summarises the informational description of computational systems provided by the philosophy of computer science, distinguishing among:

- Intention
- Specification
- Algorithm
- High-level programming language instructions
- Assembly/machine code operations
- Execution.

Our approach makes use of this layered analysis to investigate the issue of identity and copies by providing a formal analysis at the level of Specifications. This will also cascade through the lower levels. Identity and Copy are formalised as relations between two entities x, y belonging to some kind K, according to equations 2 and 3 above. Following Wiggins (2001) in the functional definition of artefacts kinds, K is here taken to be a class of programs implementing a well-defined set of algorithms (computable functions) and the entities x, y of interest are computational artefacts implementing those functions. For brevity, we will refer to a computational artefact as P, to its specification as $S(P)$, to its implementation in a programming language as $L(P)$, and to its physical realization as $I(P)$. As mentioned above, the analysis of the notions of identity and copy for computational artefacts is here addressed at the level of $S(P)$. As such, kind $K = \{P, P', P'' \ldots\}$ is given as the class of programs each correctly implementing a defined set of specifications in the class $\{S(P), S(P)', S(P'')'' \ldots\}$ and such that all specifications defining K be the realization of some common Intention. For instance, K may be identified with the class of text editors; all specifications defining K are specifications of text editors, and members of K are editors each satisfying the proper specifications. It makes sense to question whether a text editor $y \in K$ is a copy of editor $x \in K$. In order to do so, we first define the identity relation for the members of K.

We consider formal specifications $S(P)$ as state transition systems defined as follows:

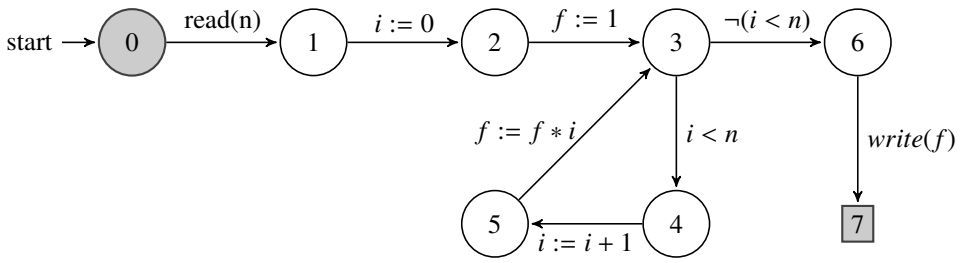

Figure 1: A Transition System for the factorial function

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 set of finite transitions labels;*
- $T \subseteq S \times A \times A$ *is a transition relation;*
- $I \subseteq S$ *is a set of initial states;*
- $F \subseteq S$ *is a set of final states;*
- AP *is a finite set of states labels;*
- $L : S \rightarrow 2^{AP}$ *is the function labelling states.*

Let us start with an easy example showing why it is convenient to focus on $S(P)$ when reasoning about identity and copy of computational artefacts. Consider a program P to compute the factorial of any integer n.[2] At the $S(P)$ level, such program can be presented as a transition system, see Figure 1. This formal specification $S(P)$ presented by a TS can then be offered at the $L(P)$ level in several formats: for example, as a Pascal implementation (Figure 2), or as a program written in C (Figure 3). The program from Figure 3 can in some sense be considered a copy of the program from figure 2, even though their high-level programming language instructions are different, while their specification is the same.

Actual executions of the two programs above correspond to paths in the TS presented in Figure 1, which list the states the computation goes through, according to the following definition of path:[3]:

[2]For this example see (Middelburg, 2016, pp.8-9).

[3]See (Baier and Katoen, 2008, p.96).

```
PROGRAM factorial(input,output);
  VAR i,n,f:0..maxint;
    BEGIN
    read(n);
    i:=0; f:=1;
      WHILE i<n DO
      BEGIN i:=i+1; f:= f*i
      END;
    write(f)
END
```

Figure 2: A program to compute the factorial in PASCAL

```
int main()
{
  int i, n, f = 1;
  scanf("%i", &n);
  for (i=1; i<=n; i++)
    f = fact*i;
  return 0;
}
```

Figure 3: A program to compute the factorial in C

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

A path in the TS from Figure 1 is therefore an execution of the program with given values (see Figure 4; here multiple labels on the edges indicate corresponding executions of the related sub-path for computing the factorial of 2).

A notion equivalent to the one of path is given by looking at labels of states belonging to it, corresponding to the following notion of trace:[4]

Definition 3 (Trace)**.** *Given a finite transition system TS, the trace of a finite path fragment $\pi = (s_0, \ldots, s_n)$ is the sequence of its labels $trace(\pi) = (L(s_0), \ldots, L(s_n))$. The trace of an infinite path fragment $\pi = (s_0, s_1, \ldots)$ is the sequence of its labels $trace(\pi) = (L(s_0), L(s_1) \ldots)$.*

State transition systems specify all the allowed behaviours of the program implementing the corresponding specification. Individual behaviours are usually formalized by means of temporal logic formulas. Temporal logics (Kröger and Merz, 2008) are formalisms capable of stating how systems evolve over time. Originally they were developed in modal logic in order to express how propositions may change their truth values over time; they extend propositional logic with temporal operators and path quantifiers that allow reference to the ordering of events so as to render them suitable

[4]See (Baier and Katoen, 2008, p.98). Note that an infinite trace over a transition system with terminal states is a trace with a self-loop on its terminal state.

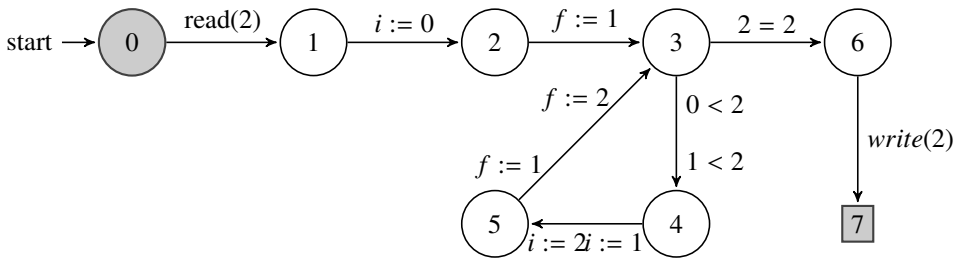

Figure 4: A path in the TS from Figure 1 for $n = 2$

for formalizing specifications of concurrent systems, wherein different processes take place at the same time. When an individual behaviour is executable by a computational system, the transition system corresponding to the latter will satisfy a temporal logic formula expressing that behaviour. CTL (Computational Tree Logic) is the temporal logic which allows for branching time models, i.e. where at each stage different paths are non-deterministically possible, and it is a typical model to analyse properties of software artefacts.

Definition 4 (Satisfaction of CTL formulas)**.** *Given a TS and an atomic formula p, the satisfaction relation in TS of CTL formulas is defined as follows:*

$TS, s_i \models p$	*iff*	$p \in L(s_i)$;
$TS, s_i \models \neg p$	*iff*	$TS, s_i \not\models p$;
$TS, s_i \models p \wedge q$	*iff*	$TS, s_i \models p$ *and* $TS, s_i \models q$;
$TS, s_i \models Xp$	*iff*	*there exists* $s_j \in S$ *s.t.* $s_i T s_j$ *and* $TS, s_j \models p$;
$TS, s_i \models Fp$	*iff*	*for all paths* $\pi = (s_i, s_j, \ldots)$ *such that there exists* $s_k \in \pi$, $TS, s_k \models p$;
$TS, s_i \models Gp$	*iff*	*there exists a path* $\pi = (s_i, s_j, \ldots)$ *such that for all* $s_k \in \pi$, $TS, s_k \models p$;
$TS, s_i \models [pUq]$	*iff*	*there exists a path* $\pi = (s_i, s_j, \ldots)$ *and an index k such that* $TS, s_k \models q$ *and* $TS, s_j \models p$ *for all* $j \leq k$;
$TS, s_i \models \forall p$	*iff*	*for all paths* $\pi = (s_i, s_j, \ldots)$, $TS, \pi \models p$;
$TS, s_i \models \exists p$	*iff*	*there exist a path* $\pi = (s_i, s_j, \ldots)$ *such that* $TS, \pi \models p$;
$TS, \pi \models Xp$	*iff*	$TS, \pi_1 \models p$;
$TS, \pi \models Fp$	*iff*	*there exists* $k \geq 0$ *such that* $TS, \pi_k \models p$;
$TS, \pi \models Gp$	*iff*	*for all* $k \geq 0$, $TS, \pi_k \models p$;
$TS, \pi \models [pUq]$	*iff*	*there exists* $k \geq 0$ *such that* $TS, \pi_k \models q$ *and* $TS, \pi_j \models p$ *for all* $j \leq k$;

The above clauses formulate the satisfaction relation distinguishing between state formulas and path formulas. For the state formulas: a formula p is satisfied at a give state $s_i \in TS$ when p is in the labels of s_i; its contradictory is valid if and only if the satisfaction does not hold; the conjunction of two formulas p, q is satisfied iff each can be individually satisfied; Xp (next) holds at a given state when p will hold at the

following state; Fp (finally) holds at a given state when p will eventually hold at some successive state; Gp (globally) holds at a given state when p will hold at all successive states; pUq (until) holds at a given state when q holds at a given state and p holds in all inbetween states; $\forall p$ holds at a given state s if every path starting from it satisfies p; $\exists p$ holds at a given state s if there is a path starting from it that satisfies p. For the path formulas: Xp holds in a given path when there is an immediate suffix of the path in which p holds; Fp holds in a given path when there exists a non-immediate suffix of the path in which p holds; Gp holds in a given path when for all suffixes of the path, p holds; pUq holds in a given path when there exists a non-immediate suffix of the path in which q holds and for all inbetween suffixes p holds.

Process algebras allows therefore to express a system's functional structure in terms of paths and their traces. A temporal logic formula satisfiable within a given transition system reflects a valid behaviour for the computational artefact interpreted by that system. In the following, starting from Section 4, identity and copy relations are expressed in terms of relations quantified over such behaviours. An important task is the characterization of this formal notion of behaviour with that in use in engineering design. To illustrate such comparison, let us refer to the different meanings of 'behaviour' offered in (Chandrasekaran and Josephson, 2000, p.169):

- *Beh-i*: The value(s), or relations between values, of state variables of interest at a particular instant;
- *Beh-ii*: The value(s), or relations between values, of properties of an object.
- *Beh-iii*: The value(s) of state variables of interest over an interval of time.
- *Beh-iv*: The value(s) of state variable(s) specifically labelled 'output' state variables, either at an instant or over an interval of time.
- *Beh-v*: The values of all the state variables in the object description, either at an instant or over an interval of time.
- *Beh-vi*: The causal rules that describe the values of the variables under various conditions.

When considering the path of a system in terms of a finite fragment of its states and a trace as the corresponding sequence of labels, we are considering both meaning (i) and (ii) above, as we look at values of state variables and relations between them. When we further express these by a temporal logic formula, we add the interval of time parameter required by meaning (iii) above. If the formula of interest refers to a trace including a final state, we are referring to meaning (iv) above. If we are considering all paths of a given transition system, i.e. the set of all possible behaviours, then we are looking at meaning (v) above. The last meaning of behaviour, interpreted in terms of causal rules, can be covered in terms of the implicative relations between states of a given system. In other words, it appears that the use of a definition of behaviour as regulated by temporal logics and process algebra is general enough to provide all the intended standard meaning in engineering design. Note, moreover, that such meanings are not different between identity and the various versions of copy relations: the difference

IDENTITY	*EXACT COPY*	*INEXACT COPY*	*APPROXIMATE COPY*
$S(P) = S(P)$	$S(P) = S(P)$	$S(P) \subset S(P')'$	$S(P) \cap S(P')'$
$L(P) = L(P)$	$L(P) = L(P)$	$L(P) = L(P')$	$L(P) = L(P')'$
$I(P) = I(P)$	$I(P) \neq I(P)'$	$I(P) \neq I(P')'$	$I(P) \neq I(P')'$

Figure 5: Taxonomy of Relations

is determined by the transition systems and the set-theoretical relations between their traces.

4 Defining Identity and Copy Relations

We are now able to associate the specification of a computational artefact $S(P)$ to the corresponding formal representation provided by a TS; different language implementations $L(P), L(P)', L(P)'', \ldots$ may correspond to a single $S(P)$; the physical realization $I(P)$ corresponds to the set of possible paths (or traces) valid for that TS in some given language; and any behaviour prescribed by the $S(P)$ can be expressed by a temporal formula p satisfied by that TS. The relations of identity and copy to be investigated in the following of this paper can now be illustrated in terms of the possible combinations of set-theoretic relations between the three terms $S(P), L(P), I(P)$, as presented in Figure 5.

The first one is the case of *Identity*: it expresses the relation of a computational artefact P with itself at any given time t; in this case there will be only one specification $S(P)$, one language implementation $L(P)$, and one instance of reference $I(P)$. We do not consider here the case of diachronic identity and the corresponding ontological problem of whether a computational artefact is identical to itself at different times. A reason to do so is based on the consideration that a solid analysis of identity and copy for computational artefacts needs to avoid, in the first instance, the problems associated with their unreliability and continuous changes in the execution environment. Therefore, we assume here a principle of computational correctness, according to which any implementation $I(P)$ will always correctly respect the behaviour prescribed by the corresponding $S(P)$. Nonetheless, we suggest here briefly that the behavioural relation of a computational artefact with itself at different points in time, if altered by restriction of functionalities, or addition of (unintended) behaviours, may be considered in a similar vein to what done below for the different relations of copy.[5]

The second case is what we call *Exact Copies*: it refers to two instances $I(P)$ and $I(P)'$ of a computational artefact P with one single specification $S(P)$, installed on two different machines M, N. We then say that $I(P)'$ is an exact copy of $I(P)$ when $I(P)'$ manifests *all and only* the behaviours of $I(P)$.[6] In this case we can in fact distinguish

[5]For an approach to software theory change which can be assumed to model such transformation of specification see Primiero and Raimondi (2015).

[6]This case has been investigated in the Philosophy of Technology literature as the relation of *perfect copies* and replicas, see (Carrara and Soavi, 2010, p.215).

two sub-cases:

1. the two instances $I(P), I(P)''$ share the same implementations $L(P) = L(P)$ in high-level programming languages;

2. the two instances $I(P), I(P)''$ have different implementations $L(P) \neq L(P)'$ in high-level programming languages.

The third case is what we call *Inexact Copies*: it refers to two distinct computational artefacts P and P', with different specifications $S(P)$ and $S(P')'$. We then say that P' is an inexact copy of P when P' manifests *all* the behaviours of P, i.e. such that $S(P) \subset S(P')'$. This relation is known, in the literature in formal methods, as system refinement. Refinements mappings are used since Abadi and Lamport (1991) to prove that a lower-level specification correctly implements a higher-level one in terms of (possibly infinite) state machines specifying safety and liveness requirements.[7] A system S' refines a system S if and only if the behaviours of S are a subset of the behaviours of S'. Refinement can be understood as the process of developing a new system S' out of an old one S, by preserving all the behaviours of the former, and possibly adding new ones. Refinement can be modelled in terms of simulation as first introduced by Milner (1971): informally, a simulation holds between two systems S' and S if we can relate each state of S to a state of S' so that two related states s, s' agree on their observations and every successor of s is related to some successor of s'; accordingly, the relation $S' \leftarrow S$, stating that S refines S' formally corresponds to the statement that S is simulated by S', or else that S' simulates S, (Gorogiannis and Ryan, 2007). In the present work, we use simulation to model the slightly more specific case represented by the notion of inexact copy. In analogy with the case of exact copies, we can distinguish two subcases, on the basis of whether $L(P) = L(P')$ or $L(P) \neq L(P')'$.

The fourth case is what we call *Approximate Copies*: it refers to two distinct computational artefacts P and P', with different specifications $S(P), S(P')'$. We then say that P' is an approximate copy of P when P' manifests *some* of the behaviours of P, i.e. $S(P) \cap S(P')'$. Analogously to the case of exact and inexact copies, also for approximate copies we distinguish between approximate copies sharing the same high-level programming language implementation $L(P) = L(P')$ and approximate copies having different implmentations $L(P) \neq L(P')'$.

In the next section we use process algebra to provide suitable formal definitions of the four relations defined above. Consequently, copy relations will be examined only considering the logic properties they satisfy, without taking into consideration the non-formal properties that nonetheless define them, including their intentional character. Indeed, it follows from the definition of exact copy above that it is an equivalent relation in that it clearly is reflexive, symmetric, and transitive. And, as it will be shown in the details of sections 6.5 and 6.7, inexact copy is a reflexive, asymmetric, and transitive relation, whereas approximate copy is reflexive, symmetric, but intransitive. However, according to Carrara and Soavi (2010), exception made for replicas, copy is an irreflexive, asymmetric, and non-transitive relation.

[7] See section 6.3 for a definition of safety and liveness properties.

As highlighted in section 2, the reason at the basis of the non-equivalence of copy lies in the intentional aspect of such relation. Whether exact copies are replicas or not properly depends on that intentional character of copies, that is, it depends on whether copying is allowed by the producer. In case it is not, it can be said that exact copy, *qua* logic relation, is reflexive, symmetric, and transitive, but they are irreflexive, asymmetric, and non-transitive if the intentional aspect of creating illegal copies of an artefact is considered. The same holds for inexact and approximate copies: the former is reflexive, asymmetric, and transitive, and the latter reflexive, symmetric, and intransitive, only *qua* logic relations. Whereas an analysis based on logic properties seems to be indespensable when a taxonomy of the copy relations for computational artefacts is to be provided, the intentional aspect of copy cannot be ignored when ethical and legal aspects of copying are considered. In particular, given two artefacts x and y for which $R'(x, y)$, identifying the model and the copy between x and y is necessaty when determing copyright infringement.

5 Logical Definition of Identity for Computational Artefacts

Process Algebra Theory (Fokkink, 2000), a formal theory for concurrent processes interpreted over process graphs (like a TS), is apt to formalize the notion of equivalence between structures. From a general point of view, two structures are said to be equivalent if and only if they can execute the same strings of transitions. In order to compare or relate two TSs, certain binary relations between states, called implementation relations, are introduced. There are several kinds of implementation relations: some, called strong relations, impose very strict constraints; others, called weak relations, impose more relaxed constraints. Two typical implementation relations are the bisimulation equivalence and the simulation preorder, which are two strong relations.

The strongest equivalence relation is bisimulation; this notion requires not only that two structures be able to execute the same transitions, but also that they have the same branching structure and thus can simulate each other. Intuitively, bisimulation between two structures requires that every step of one structure be matched by one step of the other structure and vice versa. Bisimulation equivalence is a mutual, stepwise simulation.

Definition 5 (Bisimulation)**.** *Given two labelled transition systems TS and TS' on a finite set of states S, a bisimulation between them denoted as $TS \equiv TS'$ occurs when:*

1. *for each initial state s_0 in TS there is an initial state s'_0 in TS' that establishes a bisimulation relation $\equiv (s_0, s'_0)$, and vice versa;*

2. *$\equiv (s_0, s'_0)$ holds if and only if s_0 and s'_0 have the same label and there is a successor state s'_1 of s'_0 in TS' for each successor state s_1 of s_0 in TS such that $\equiv (s_1, s'_1)$.*

In other words, a bisimulation between two states occurs when they have the same label and there is a successor state in the second structure for each successor state in the

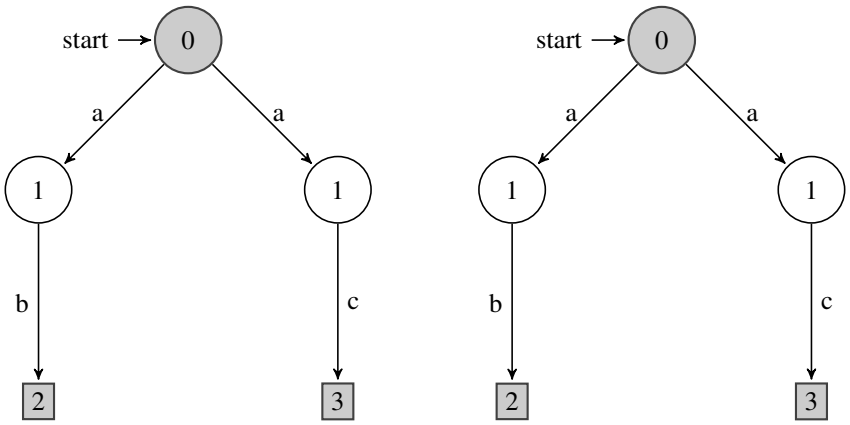

Figure 6: Two bisimilar TSs.

first structure, and vice versa, such that a bisimuation relation between the successors holds. A bisimulation between two structures occurs when for each initial state of one structure there is an initial state of the second structure that establishes a bisimulation relation. Bisimulation is an equivalence relation, that is, it is a reflexive, symmetric and transitive relation.

Bisimulation implies a number of properties over the related transition systems. For our purposes, it is essential to recall how the relation of bisimulation is defined over paths and temporal formulas: the former allows to compare identical behaviours, the latter identical properties. Let us start from defining identical paths:[8]

Lemma 1 (Bisimulation on Paths)**.** *Given two bismulation equivalent transition systems TS and TS', if $\equiv (s_i, s'_i)$, then for each (finite or infinite) path $\pi = (s_i, s_j, \ldots)$ of TS there exists a path $\pi' = (s'_i, s'_j, \ldots)$ of the same length in TS' and $\equiv (s_k, s'_k)$ for all k, and viceversa.*

Recall that paths correpond to the transitions through labels indicated by traces, hence the following result holds:[9]

Theorem 1 (Bisimulation implies trace equivalence)**.** *Two transition systems TS and TS', defined over the same set of atomic proposition AP, are said to be trace equivalent, if $traces_{AP}(TS) = traces_{AP}(TS')$, where $traces(TS) = \bigcup_{s \in I} traces(s)$ denotes the set of traces starting at initial states of TS. It holds that: $TS \equiv TS'$ implies that $traces_{AP}(TS) = traces_{AP}(TS')$.*

Bisimulation corresponds to equivalence of properties, as expressed by valid temporal formulas being satisfied by the corresponding structures. To intrduce this known result, recall that CTL^* is the superset of CTL where formulas containing temporal operators need not to be directly preceded by a quantifier.

[8] See (Baier and Katoen, 2008, p.454).
[9] See (Baier and Katoen, 2008, p.456).

Theorem 2 (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.*

Bisimulation expresses therefore exactly that two structures have identical behaviours and satisfy identical structures. In order to be exploited to express identity of behaviour of a computational artefact with itself, we need to define bisimulation as a relation on its states, as follows:[10]

Definition 6 (Bisimulation as a Relation on States)**.** *Given a TS, a bisimulation on states of TS denoted by $\equiv_{TS}$ is a binary relation $\equiv \subseteq S \times S$ such that for all $\equiv (s_i, s_j)$*

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

We argue that a computational artefact P is identical to itself insofar as a bisimulation equivalence can be defined on the TS corresponding to $S(P)$. This is shown in Figure 6 by two bisimilar TSs which are one the duplication of the other, as in a structure related to itself. In other words, we propose to identify relation R in equation 2 with $\equiv_{TS}$.

Definition 7 (Identity as Bisimulation on States)**.** *TS is identical to itself in that the relation $\equiv_{TS}$ of bisimulation on its states can be defined.*

We understand identity at the level of specification as sameness of prescribed behaviours. The set of temporal properties corresponding to such behaviours is the set:

$$G := \{g \mid TS, s_i \vDash_{CTL^*} g\}$$

for each $s_i \in TS$, where TS is the formal translation of the specification $S(P)$ for the computational artefact of interest. If the specification modelled by TS satisfies at each given time the same set of properties as expressed by G, then $TS \equiv_{CTL^*} TS$; then by Theorem 2 it holds $TS \equiv TS$, which means $\equiv_{TS}$. Conversely, the relation $\equiv_{TS}$ is a subset of $S \times S$ satisfied by all the pairs that are related by a transition in TS; this identifies exactly all behaviours of TS, hence is by definition behaviourally identical to the specification of TS.

5.1 Constraints on Identity Criteria

We analyse here the constraints for identity satisfied by $\equiv_{TS}$.

Non-vacuousness. The bisimulation on states relation $\equiv_{TS}$ is a significant, non-vacuous, determinable for the evaluation of identity of computational artefacts, as explicated in Definition 7: a computational artefact is identical to itself in that a bismulation on the states of the artfact's specification can be defined.

[10] See (Baier and Katoen, 2008, p.456).

Informativeness. $\equiv_{TS}$ contributes to specifying the nature of K in that bisimulation is preserved over different modes of presentation of the same structure, as given a TS, several bisimilar graphs can be used to represent it, including unwinding and duplication.[11] This means that given two different graphs, bisimulation allows one to ensure that the two graphs are bisimilar graphs for the same specification TS of a given program P.

Partial Exclusivity. K is defined as the set of objects that instantiate a defined set of computable functions which are here expressed as TSs, and bisimulation is an algebraic relation holding between them. Notice that TSs are cyclic graphs able to represent non-ending executions characterizing the so-called reactive software systems. It should be noted that also many physical processes can be represented by means of directed graphs, but a-cyclic ones, including causal processes (Pearl, 2009). Accordingly, bisimulation cannot be applied to non-computational processes. These processes are therefore excluded by our analysis and bisimulation is not a trivializing relation.

Minimality. $\equiv_{TS}$ is the only one determinable induced by our identity criterion for computational artefacts. Definition 7 ensures that $\equiv_{TS}$ is a both necessary and sufficient condition, as it associates two analytic concepts to one another.

Noncircularity $\equiv_{TS}$ does not presuppose identity. Indeed, given any two distinct computational artefacts P and P', if $S(P) = S(P')$ then $S(P) \equiv S(P')$. This is the case of exact copies presented below.

Non-totality. Given kind $K = \{P, P', \ldots, P^{\circ}\}$ of objects each satisfying a set of specifications in $S = \{S(P), S(P'), \ldots, S(P^{\circ})^{\circ}\}$, the relation $\equiv_{TS}$ is a subset of $K \times K$, namely:

$$\equiv_{TS} =_{def} \{(S(P), S(P)), (S(P')', S(P')'), \ldots, (S(P^{\circ})^{\circ}, S(P^{\circ})^{\circ})\}$$

Hence all other pairs in $K \times K$ are not included by the relation $\equiv_{TS}$.

K-maximality. $\equiv_{TS}$ is maximal with respect to K in that it is the *only* relation over K that for all $S(P)$ with $P \in K$, returns all the pairs that satisfy identity.

Uniqueness. $\equiv_{TS}$ is unique with respect to K, since every other relation which can be taken to satisfy an identity criterion cannot be considered independent of bisimulation, i.e. it can be at most equivalent to $\equiv$. Trace inclusion and CTL* equivalence being some cases in point. One admissible alternative to evaluate the identity relation of computational artefacts that is also independent from bisimulation may be observational equivalence. However, supposing that two artefacts, while observed for a finite lapse of time, manifest the very same behaviours, one cannot conclude that the two artefacts are necessarily identical, as one cannot argue for generalizations on the basis of testing.

Equivalence. $\equiv_{TS}$ is an equivalence relation.[12]

Congruence. $\equiv_{TS}$ is congruent in that it implies CTL^* equivalence: whatever is true of a TS according to a CTL^* state formula is true of any bisimulation euqivalent TS, including itself.[13]

[11] For these definitions, see e.g. (Clarke et al., 1999, p.172).

[12] See (Baier and Katoen, 2008, Lemma 7.8, p.453).

[13] See (Baier and Katoen, 2008, Definition 7.17, part 2, p. 468).

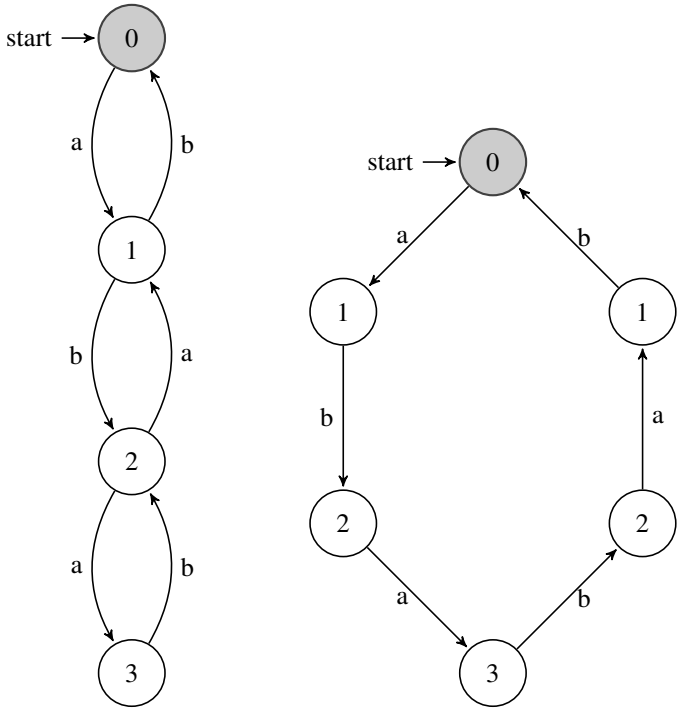

Figure 7: Two distinct bisimilar TSs.

6 Logical Definition of Copies for Computational Artefacts

6.1 Exact Copies

For the formal translation of the notion of exact copy, we still rely on the relation of bisimulation. According to this definition, two computational artefact P, P' are exact copies if a bisimulation equivalence can be defined between the corresponding TS, TS' for the relevant $S(P), S(P')'$. In other words, we propose to identify relation R' in equation 3 for exact copies with $\equiv$. Note how bisimulation is not only definable bewteen two structures where one is a *duplication* of the other, but also where identical behaviours are present among distinct structures, see Figure 7.

Theorem 3 (Exact Copy as Bisimulation)**.** *TS' is an exact copy of TS if and only if $TS \equiv TS'$.*

Proof. From left to right: consider the set of properties of TS as expressed by the set of formulas

$$G := \{g \mid TS, s_i \vDash_{CTL*} g\}$$

for each $s_i \in TS$, where TS is the formal translation of the specification $S(P)$ for the first computational artefact of interest; and the set of properties of TS' as expressed by the set of formulas

$$G' := \{g' \mid TS', s'_i \vDash_{CTL^*} g'\}$$

for each $s'_i \in TS'$, where TS' is the formal translation of the specification $S(P')'$ for the second computational artefact of interest. Then, if $G = G'$, then $TS \equiv_{CTL^*} TS'$ and by Theorem 2 it holds $TS \equiv TS'$.

From right to left: the relation $TS \equiv TS'$ is a subset of $S \times S'$ satisfied only by the pairs that are related by a transition in TS as well as in TS'; this identifies exactly all behaviours of *both* TS and TS', hence by definition it expresses behavioural identity of the specifications of TS and TS', which as such can be called exact copies. □

6.2 Exact Copy Constraints

We analyse here the constraints for exact copies satisfied by ≡.

Non-vacuousness. Theorem 3 shows that the bisimulation relation $TS \equiv TS'$ is a non-vacuous determinable to establish whether TS' is or is not an exact copy of TS.

Informativeness. ≡ contributes to specifying the nature of K in that, given a TS representing $I(P)$ and TS' representing $I(P)'$, then for every computable function f valid according to K, if $TS \equiv TS'$ then $TS \models g$ iff $TS' \models g$ for any CTL^* formula g expressing a property satisfied by a function f.

Partial Exclusivity. The same observation as for Identity Criteria applies here.

Minimality. The same observation as for Identity Criteria applies here.

Noncircularity Bisimulation does not presuppose the relation of being an exact copy, indeed bisimulation also holds for identity, where $I(P) = I(P)'$.

Non-totality. Given $I(P)$ and $I(P)'$ which are exact copies, they are instances of the same specification $S(P)$ and hence still satisfying only one pair in

$$\equiv =_{def} \{(S(P), S(P)), (S(P')', S(P')'), \ldots, (S(P^\circ)^\circ, S(P^\circ)^\circ\}$$

and hence it still is a subset of $K \times K$.

K-maximality. Given $I(P)$ and $I(P)'$ which are exact copies, they are instances of the same specification $S(P)$ and hence still satisfying only one pair in

$$\equiv =_{def} \{(S(P), S(P)), (S(P')', S(P')'), \ldots, (S(P^\circ)^\circ, S(P^\circ)^\circ\}$$

which is the maximal partition over K returning all the identical specifications for exact copies.

Uniqueness. ≡ is unique wrt K since every other relation which can be taken to satisfy an exact copy criterion cannot be considered independent of bisimulation. Trace inclusion and CTL equivalence being some cases in point.

Equivalence. Assuming correctness of each exact copy with respect to the identical specification, ≡ is an equivalence relation.[14]

Congruence. ≡ is congruent in that, given a TS representing the specification determining executions of $I(P)$ and a TS' representing the specification determining executions of $I(P)'$, TS and TS' are CTL* equivalent.[15]

This analysis shows that bisimulation, both defined as $\equiv_{TS}$ for identity and as ≡ for exact copies, satisfies all constraints on identity illustrated in the literature.

6.3 Simulation

An equivalence relation is a binary relation that must be reflexive, transitive and symmetric; if a binary relation between structures is only reflexive and transitive but not symmetric it generates a preorder. Whereas bisimulation is an equivalence relation requiring two bisimilar states to exhibit identical stepwise behaviors, the simulation relation is a preorder requiring only that one state be able to mimic all stepwise behaviors of the simulated state but not vice versa. This means that the simulating structure might perform transitions that are not performed by the simulated structure. The simulation relation can be formally defined in the following way:

Definition 8 (Simulation)**.** *Given two labelled transition systems TS and TS' on a finite set of states, TS' is said to simulate TS denoted by $TS \leq TS'$ when*

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

In Figure 8, we have a relation $TS \leq TS'$: the structure TS' on the right is able to mimic all the stepwise behaviours of the strucure TS on the left, in particular: path $\pi = (s_0, s_1, s_2)$ of TS is allowed by transitions a, b of TS', and path $\pi = (s_0, s_1, s_3)$ of TS is allowed by transitions a, c of TS'. The two systems are not in a bisimulation relation in that TS is not able to similarly mimic all behaviours of TS': in particular, at state denoted s_1 in TS, there is no undeterministic choice to move to state denoted s_2 or to a state denoted s_3, but only deterministically to either one of such states. Note that it is essential here to specify whether the structures of reference is finite, as we are doing here by denoting states s_2, s_3 as final.

As for bisimulation, the simulation relation on finite structures implies a number of properties at the level of behaviours and properties. Let us start by considering the considerably weakened implication to path simulation:[16]

Lemma 2 (Simulation on Paths)**.** *Given two transition systems TS and TS' such that $TS \leq TS'$, if $\leq (s_i, s'_i)$, then for each (finite or infinite) path $\pi = (s_i, s_j, \ldots)$ of TS there exists a path $\pi' = (s'_i, s'_j, \ldots)$ of the same length in TS' and $\leq (s_k, s'_k)$ for all k.*

[14] See (Baier and Katoen, 2008, Lemma 7.8, p.453).
[15] See (Baier and Katoen, 2008, Definition 7.17, part 2, p. 468).
[16] See (Baier and Katoen, 2008, p.504).

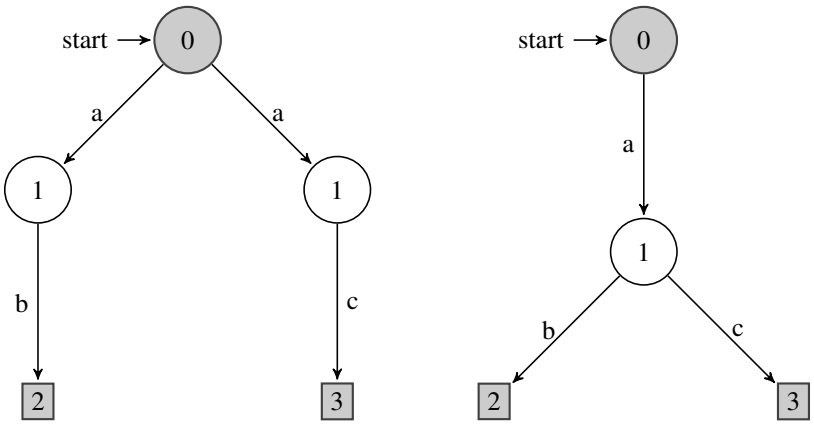

Figure 8: TS' on the right simulates TS on the left

Note that, as expected, here the existence of identical paths is only induced in one direction, i.e. from the existence in the simulated structure to existence in the simulating structure. Accordingly, this implies trace inclusion:[17]

Theorem 4 (Simulation implies trace inclusion)**.** *Two transition systems TS and TS', defined over the same set of atomic proposition AP, are said to satisfy trace inclusion, if $traces_{AP}(TS) \subseteq traces_{AP}(TS')$, where $traces(TS) = \bigcup_{s \in I} traces(s)$ denotes the (finite) set of traces starting at initial states of TS. It holds that: $TS \leq TS'$ implies that $traces_{AP}(TS) \subseteq traces_{AP}(TS')$.*

As the relation of trace inclusion presented above concerns *finite* traces, the properties of relevance are *safety properties* of the system. Let us see. If one defines the universal fragment of CTL^*, denoted $\forall CTL^*$, as the temporal logic obtained by considering only universally quantified formula in CTL^*, then:[18]

Theorem 5 (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.*

As an example, consider that in TS' from Figure 8, the state labelled by s_0 satisfies the formula $G1$ (where we take here the name of the state to express a corresponding propositional variable), which simulates the behaviour of the state labelled by s_0 in TS. Also, the state on the left branch labelled by s_1 in TS satisfies $X2$, while the state labelled by s_1 in TS' does not. Because $\forall CTL^*$ describes properties that are quantified over all possible behaviors, and since every behavior of TS' is a behavior of TS, every $\forall CTL^*$ formula satisfied by TS' must also hold true in TS. However, when a $\forall CTL^*$ formula is not satisfied in TS', it may or may not be satisfied in TS. It is crucial that the behaviour inclusion of the simulated structure in the simulating structure concerns universally quantified formulas, i.e. expressing safety properties.

[17] See (Baier and Katoen, 2008, p.512).
[18] See (Baier and Katoen, 2008, p.517–519).

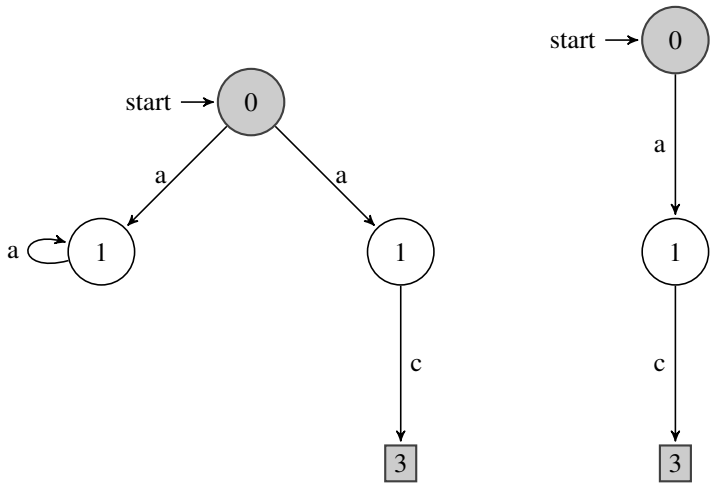

Figure 9: TS' on the right simulates TS on the left

A different situation concerns structures which admit infinite paths, by the presence of loops on states. Consider the two structures in Figure 9. In this case, $TS \leq TS'$, but there is now an infinite path in TS, namely $\pi = \{s_0, s_1, s_1...\}$ such that the sequence of its states' labels 01 is still included in TS'.

Theorem 6 (Simulation implies trace inclusion)**.** *Two transition systems TS and TS', defined over the same set of atomic proposition AP, are said to satisfy trace inclusion, if $traces_{AP}(TS) \subseteq traces_{AP}(TS')$, where $traces(TS) = \bigcup_{s \in I} traces(s)$ denotes the (infinite) set of traces starting at initial states of TS. It holds that: $TS \leq TS'$ implies that $traces_{AP}(TS) \subseteq traces_{AP}(TS')$.*

Let us now consider how this extends to other properties of the structures as expressed by temporal formulas. If one defines the existential fragment of CTL^*, denoted $\exists CTL^*$, as the temporal logic obtained by considering only existentially quantified formula in CTL^*, then:[19]

Theorem 7 (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.*

As an example, consider that in TS' from Figure 9, the state labelled by s_1 satisfies the formula $F3$ (again, using labels of states as variables for propositional formulas), while the state labelled by s_1 in the left branch of TS does not. Because $\exists CTL^*$ describes properties that are quantified over some possible behaviors, and since every behavior of TS is a behavior of TS', some properties satisfied by TS must also hold true in TS'. However, when an $\exists CTL^*$ formula is not satisfied in TS, it may or may not be satisfied in TS'. It is crucial that the behaviour inclusion of the simulated structure

[19] See (Baier and Katoen, 2008, p.520).

in the simulating structure concerns existentially quantified formulas, i.e. expressing *liveness properties*.

6.4 Inexact Copies

We use the simulation relation to characterize the relation of being an inexact copy. A computational artefact P' is an inexact copy of P if the TS' corresponding to $S(P')'$ displays all the universally valid behaviours of the TS corresponding to $S(P)$. We thus propose to identify relation R' in equation 3 for inexact copies with $\leq$.

Theorem 8 (Inexact Copy as Simulation)**.** *TS' is an inexact copy of TS if and only if $TS \leq TS'$.*

Proof. Recall that we understand inexact copy at the level of specification as inclusion of prescribed behaviours. In view of Theorem 5, the set of all universally valid behaviours of the copied structure which are manifested by the copying structure, can be identified by the set of safety properties of the copying structure, namely with

$$\forall G' := \{g' \mid TS', s'_i \vDash_{\forall CTL^*} g'\}$$

Now the following inclusion relation holds:

$$\forall G' \subseteq G := \{g \mid TS, s_i \vDash_{CTL^*} g\}$$

The inclusion above expresses the fact that all safety properties of TS' are properties of TS, but it does not say anything of properties that are not satisfied by TS'.

For the left to right direction: assume that TS' is an inexact copy of TS, in the sense of having all of its universally valid behaviours. This must be more precisely intended as saying that for every property that is always manifested by every behaviour of TS', then such behaviour is manifested by TS as well. Then if the specification modelled by TS' as G' is a subset of G for TS for the safety properties, then $TS', s'_i \vDash g \rightarrow TS, s_i \vDash g$, for any ∀CTL* formula g, then by Theorem 5 it holds $TS \leq TS'$.

For the right to left direction: Assume there is a behaviour expressed as a formula g that is always displayed by every execution of P; then by definition $TS, s \vDash g$ and g is a $\forall CTL^*$ formula. Assume moreover that there is some execution of P' that does not satisfy g, then by definition $\exists s' \in TS'$ such that $TS', s' \vDash \neg g$. By our assumption $TS \leq TS'$ then by Theorem 4 the paths leading to behaviours of TS are paths leading to behaviours of TS', i.e. $traces(TS) \subseteq traces(TS')$. Hence, TS has a state satisfying $\neg g$, and also some path leading to g: therefore, it is not possible that a property satisfied by every behaviour of P be not a property of P', and hence if TS' simulates TS then it is an approximate copy of it.

□

6.5 Inexact Copy Constraints

We analyse here which of the constraints analysed so far are satisfied by inexact copies in terms of the $\leq$ relation.

Non-vacuousness. As above, the non-vacousness of $\leq$ is assured by Theorem 8.

Informativeness. Simulation contributes to specifying the nature of K in that, given TS representing $I(P)$ and TS' representing $I(P')'$, If $TS \leq TS'$ then there is a computable function f such that if $TS' \models g$ then also $TS \models g$, for some $\forall CTL^*$ formula g expressing a safety property satisfied f. Clearly, the informativeness of $\leq$ over K offers more constraints than $\equiv$ over K.

Partial Exclusivity. The same observation as for bisimulation applies to simulation.

Minimality. The same observation as for the Identity and Exact Copy Criteria applies here.

Noncircularity. Simulation does not presuppose the relation of being an inexact copy. Indeed, given two artefacts P, P' and supposing $TS \leq TS'$ (where TS represents P and TS' represents P'), it does not necessarily mean that P' is an inexact copy of P. In the case where P' is an exact copy of P, or P' is identical to P, then $TS \equiv TS$ from which it also follows $TS \leq TS'$. Note however that being an inexact copy will not imply being either an exact copy or an identical artefact.

Non-totality. Simulation is a pre-order and as such is transitive but not symmetric. For any two $S(P), S(P')'$ with $P, P' \in K$, the relation $\leq$ will hold in one direction but not in the other, hence $\leq \subset K \times K$.

K-maximality. P' is an inexact copy of P when P' manifests all but not only the behaviours of P; as such, by definition, P' simulates P. However, one may define a wider relation $P \preceq P'$ defining any additional property of P' which is not a property of P, such as a liveness property on TS'. If $P \preceq P'$ then $P \leq P'$ but not viceversa, hence maximality *fails*.

Uniqueness. Our criterion of interpretation for inexact copies is defined as behavioural inclusion analysed at the level of specifications, such that if P' is an inexact copy of P, then $S(P) \subset S(P')$. This inclusion is analysed at the level of TS*s*, and given two such structures TS, TS' behavioural inclusion corresponds to simulation. Any other relation on structures preserving this criterion is either wider (e.g. bisimulation which implies simulation) or narrower (e.g. $\forall$CTL equivalence which is implied by simulation). As such, no other independent relation for inexact copies exists.

Equivalence. Simulation satisfies reflexivity and transitivity but, as a pre-order, it does not satisfy symmetry, hence it *is not* an equivalence relation.

Congruence. By definition of $P \leq P'$, all that is true of the simulated artefact P will be true of the simulating artefact P' but not viceversa. Hence, congruence *is not* satisfied by simulation.

This analysis shows that the relation holding between inexact copies is weaker than identity, as Maximality, Equivalence and Congruence do not hold.

6.6 Approximate Copy

For the notion of approximate copy, recall that it refers to two distinct computational artefacts P and P', with different specifications $S(P), S(P')'$ such that P' is an approximate copy of P when P' manifests *some* of the behaviours of P, i.e. $S(P) \cap S(P')' \neq \emptyset$. This leaves in turn the possibility that each artefact presents behaviours not included in the other. To formalise this notion, we need to introduce a new set of relations, starting from the definition of simulation as a relation on states of a single TS:[20]

Definition 9 (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 $\leq (s_i, s_j)$*

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 $\leq (s'_i, s'_j)$.*

Recall that the quotient set of a set S, denoted $S/\sim$, is the partition of S with respect to a relation $\sim$, such that two elements $a, b \in S$ belong to the same partition if and only if they satisfy the relation $\sim$ at hand. In particular, we consider the simulation quotient of a TS, denoted by $TS/\leq$ as the partition of a TS according to the simulation relation, i.e. a partition of paths of TS such that a path is in the partition if and only if TS simulates it. Formally:[21]

Definition 10 (Simulation Quotient System)**.** *Given a TS, the simulation quotient transition system $TS/\leq$ is a set theoretic structure $TS/\leq = (S/\leq, \{\tau\}, T_\leq, I_\leq, F_\leq, AP, L_\leq)$ where:*

- *$S/\leq$ is the set of states obtained by partition according to the simulation relation by TS;*
- *$\{\tau\}$ is any set of finite transitions' labels;*
- *$T_\leq$ is the quotiented transition relation such that if $T_{a\in A}(s, s')$ then $T_\tau([s]_\leq, [s']_\leq)$, where A is the original finite set of transitions' labels of TS;*
- *$I_\leq = \{[s]_\leq \mid s \in I\}$ is the set of states obtained by the partition according to simulation by TS on its initial states, where I is the original finite set of initial states of TS;*
- *$F_\leq = \{[s]_\leq \mid s \in F\}$ is the set of states obtained by the partition according to simulation by TS on its final states, where F is the original finite set of final states of TS;*
- *AP is a finite set of states' labels;*
- *$L_\leq$ is the function labelling states such that $L_\leq([s]_\leq) = L(s)$.*

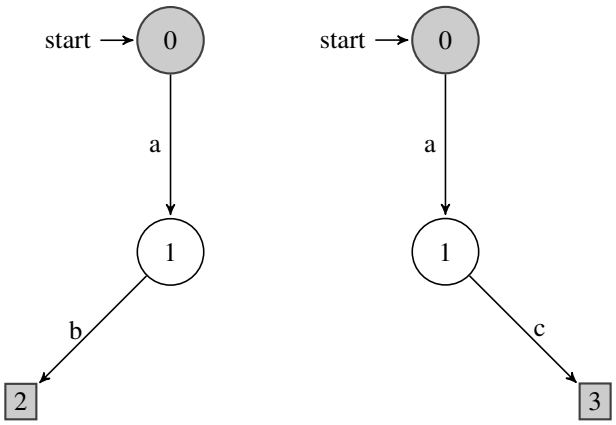

Figure 10: Elements of $TS'/\leq$ from Figure 8, π on the left and π' on the right.

It is obvious that the elements of a simulation quotient are all the behaviours that a system can simulate and therefore all its own behaviours. As an example, examine Figure 10, showing the elements of $TS'/\leq$ from the TS' in Figure 8, i.e. its paths π, π'.

Consider now a new TS'' such that $\pi \leq TS''$, see Figure 11. According to our intuition, P' is an approximate copy of P when P' manifests some of, and potentially more than, the behaviours of P, while also P is in the same relation to P'. The simulation from Figure 12 instantiates it precisely, as it allows TS'' to have more behaviours than TS, and it also allows TS to have more behaviours than TS'', as the latter simulates only some of its own behaviours.

This relation is captured precisely as follows:

Theorem 9 (Approximate copy)**.** *TS' is an approximate copy of TS, denoted by $TS \approx TS'$ iff $\exists \pi' \in TS'/\leq$ such that $\pi' \leq TS$.*

Proof. Let us denote with $G'/\leq$ the consequence set of $TS'/\leq$, i.e.

$$G'/\leq := \{g' \mid TS'/\leq, \pi' \vDash_{CTL^*} g'\}$$

Then, the relation of approximate copy expressed as a relation between the sets of properties of TS, TS' is given by

$$G \subset G'/\leq := \{g \mid T(S) \vDash_{CTL^*} g\} \subset \{g' \mid TS'/\leq, \pi' \vDash_{CTL^*} g'\}$$

and the Theorem is reformulated by saying that

$$G \subset G'/\leq \text{ iff } \exists \pi' \in TS'/\leq \text{ such that } \pi' \leq TS$$

We now prove this double implication.

[20] See (Baier and Katoen, 2008, p.506).
[21] See (Baier and Katoen, 2008, p.508).

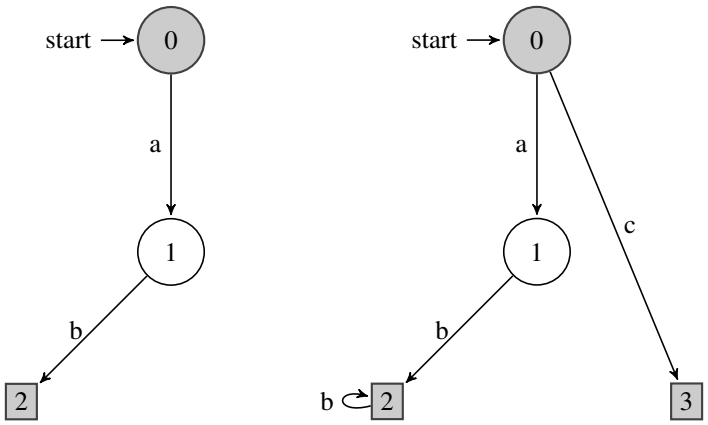

Figure 11: TS'' on the right simulates π on the left.

From left to right: if the specification modelled by TS as G is a proper subset of $G'/\leq$ for $TS'/\leq$, then:

1. $\exists g \in G$ such that $\exists \pi \in TS$ such that $\pi \vDash_{\forall CTL^*} g \wedge \exists \pi' \in TS'/\leq$ such that $\pi' \vDash g$
2. $\exists g' \in G'/\leq$ such that $\exists \pi' \in TS'/\leq$ such that $\pi' \vDash_{\forall CTL^*} g' \wedge \forall \pi \in TS\, \pi \nvDash g'$

Then by Theorem 5, the first condition above presents a witness for $\exists \pi \in TS'/\leq$ such that $\pi \leq TS$.

From right to left: recall that we informally define approximate copies as satisfying two conditions

1. they share some behaviours;
2. they do not share every behaviour (in either direction).

Then, if there is a path in the simulation quotient of TS' which is simulated by TS, then by Definition 8 every state in such a path has an equivalent state in TS. Then this path expresses exactly the common behaviors of TS and TS', satisfying the first of the conditions above. But this does not say anything about behaviours that are not satisfied by TS (which could be satisfied by TS') or about behaviours that are satisfied by TS' (but are not by TS, as the simulation involves only a path of the former). Hence, also the second condition above is satisfied and the two structures are one an approximate copy of the other. □

It should be noted that by the right to the left direction above, $\approx$ is a symmetric relation and thereby also $S(P')'$ simulates a non-empty subset of the simulation partitions of $S(P)$. This also captures the idea, highlighted in the philosophy of technology (for instance in Carrara and Soavi (2010)), that when P' is a copy of P, $I(P')'$ presents some similarities with $I(P)$, that is, the behaviours corresponding to the simulated partitions. Notice however that theorem 9 focuses on prescribed behaviours at the

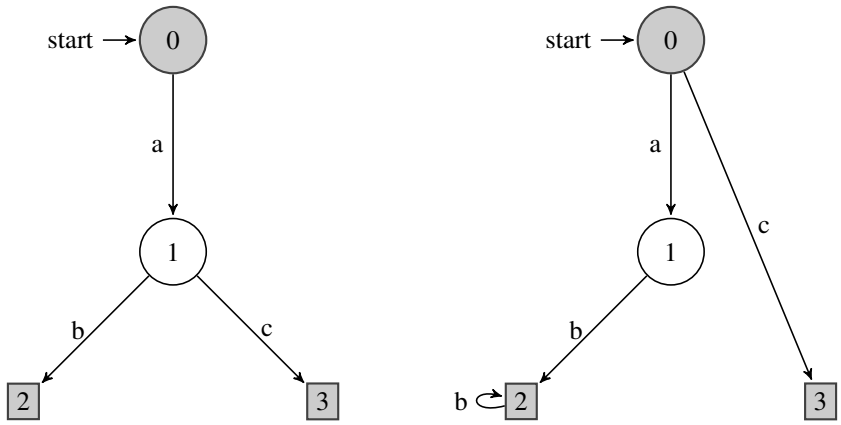

Figure 12: TS'' on the right approximates TS on the left.

formal specification LoA, not on implemented observable behaviours. This allows to avoid common problems connected to the similarity relations, in particular their high context-dependency (Goodman, 1972). Determining whether two computational artefacts are similar on the basis of the similarity of observed behaviours may not always turn to be revealing in that some of the resulting behaviours may depend on contextual factors, especially the operating environment.

6.7 Approximate Copy Constraints

We analyse here which of the constraints for the identity criteria are satisfied by approximate copies in terms of the $\approx$ relation, expressed in turn as simulation between a computational artefact and an element of the simulation quotient of another artefact.

Non-vacuousness. As above, Theorem 9 shows that $\approx$ is a significant determinable to evaluate approximate copies of computational artefacts.

Informativeness. Simulation over a partition contributes to specifying the nature of K in that, given a TS representing $I(P)$ and TS' representing $I(P')'$, if $TS \approx TS'$ then for some computable function f valid according to K, if $TS \approx TS'$ then $TS \models g$ iff $TS' \models g$ for at least one CTL* formula g expressing a property satisfied by the function f. Clearly, the informativeness of $\approx$ over K offers more constraints than $\leq$ over K.

Partial Exclusivity. The same applies as for the inexact copy relation above.

Minimality. The same observations made above for identity, exact and inexact copies applies here.

Noncircularity. Simulation over partition does not presuppose the relation of being an approximate copy. Indeed, given two artefacts P, P' where P' is an exact copy of P, or P' is identical to P, then $P \equiv P'$, which implies $P \leq P'$ which in turn implies $P \approx P'$. Note however that being an approximate copy will not imply being an inexact

copy, which in turn will not imply being either an exact copy or an identical artefact.

Non-totality. The same applies as for the inexact copy relation above.

K-maximality. P' is an approximate copy of P when P' manifests some but not only the behaviours of P; as such, by definition, $P \approx P'$. However, one may define a wider relation defining any additional property of P' which is not a property of P, such as for the case of simulation $P \leq P'$. If $P \leq P'$ then $P \approx P'$ but not viceversa, hence maximality *fails*.

Uniqueness. Our criterion of interpretation for approximate copies is defined as partial behavioural inclusion analysed at the level of specifications, such that if P' is an approximate copy of P, then $S(P) \cap S(P) \neq \emptyset$. This inclusion is analysed at the level of TS, and given such two structures TS, TS' partial behavioural inclusion corresponds to simulation over partition. Any other relation on structures preserving this criterion is wider (e.g. simulation which implies simulation over partition). As such, no other independent relation for approximate copies exists.

Equivalence. Simulation over partition satisfies reflexivity and symmetry. As explained below, simulation over partition does not satisfy transitivity. Hence it *is not* an equivalence relation.

Congruence. By definition of $P \approx P'$, not all that is true of the approximated artefact P will be true of the approximating artefact P', nor viceversa. Hence, congruence *is not* satisfied by simulation over partition.

This analysis shows that, as in the case of the relation of being an inexact copy, the relation of being an approximate copy does not satisify Maximality, Equivalence and Congruence. Approximate copies do not satisfy transitivity or, more precisely, they satisfy intransitivity:

$$\neg\forall x\forall y\forall z((y \leftsquigarrow x) \wedge (z \leftsquigarrow y) \rightarrow (z \leftsquigarrow x)) \tag{4}$$

To see this, reconsider the example from Figure 12, which proves that TS'' on the right is an approximate copy of TS on the left. Now it is not difficult to design a new TS''', of which TS is an approximation, but which is not approximated by TS'', see Figure 13. However, this does not need to be the case, as a similarly easy example show instead a TS'''' approximated by both TS and TS'', see Figure 14.

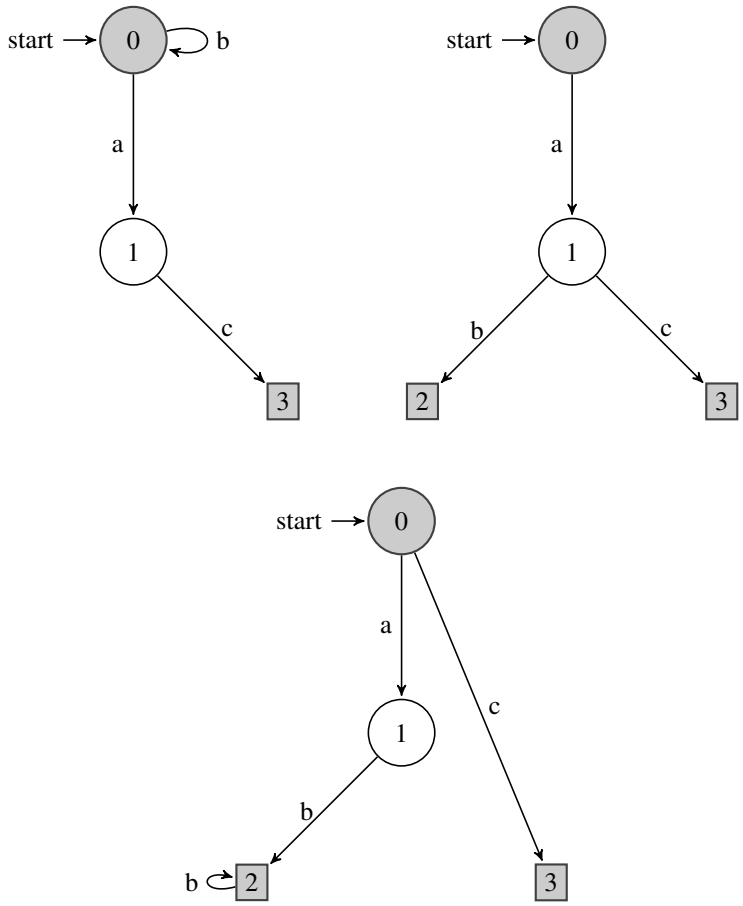

Figure 13: An example of non transitive approximation: TS at the bottom approximates TS'' on the top right, which approximates TS''' on the top left, whil TS does not approximate TS'''.

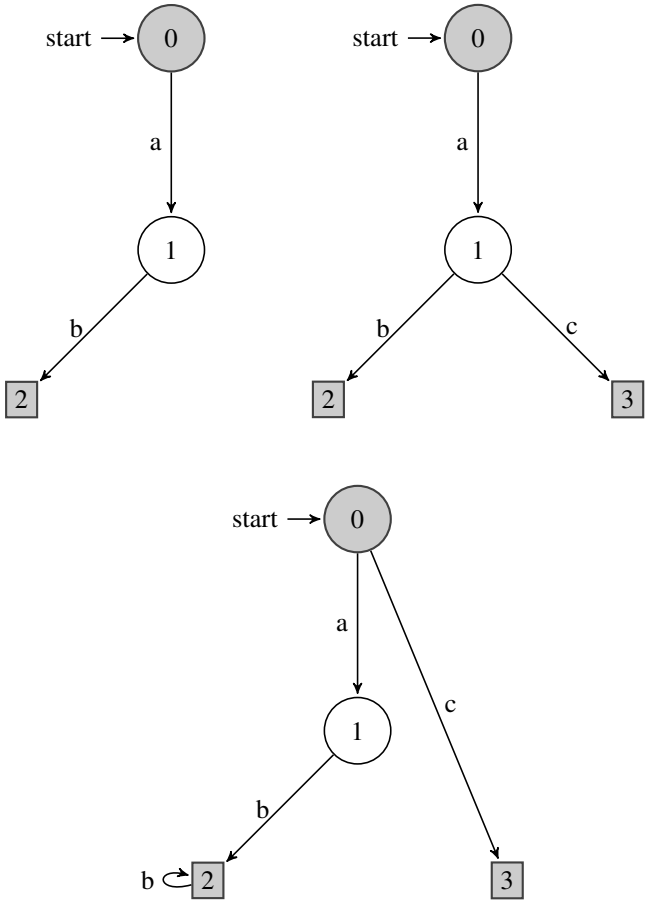

Figure 14: An example of transitive approximation: TS at the bottom approximates TS'' on the top right and TS''' on the top left.

7 Future Developments and Applications

This paper addressed the ontological question on identity criteria put forward by Frege (1953) for computational artefacts. We argued that the identity relation $x = y$ between computational artefacts x and y can be defined in terms of a bisimulation equivalence, in that such formal relation satisfies all the constraints required by identity criteria for natural objects and technical artefacts. The paper also addressed the ontological question with respect to the copy relation between computational artefacts x and y, by defining it in terms of a simulation preorder. The exact-inexact-approximate copy relations have been here understood as progressive weakenings of the identity relation, on the basis of the constraints being satisfied or violated by the different copy relations. The choice of analysing identity of computational artefacts at the formal specification

level turns out to be useful also to address the epistemological question on identity and copy. Indeed, establishing whether $S(x) \equiv S(y)$ or $S(x) \leq S(y)$ for computational artefacts x and y, can be algorithmically checked. It is known that checking bisimulation equivalence or simulation order are polinomyally solvable problems, while checking trace equivalence is a PSPACE-complete problem. Clearly, complexity issues arising from such decision problems should be analysed for non-trivial system specifications and for the relevant behaviours.

Defining a mechanical solution to the problem of whether a given computational artefact is or is not a – approximate or even inexact – copy of an original one is compelling in *computer ethics* (Johnson and Miller, 2008), especially in connection with the issue of defining copyright, or patent infringement. Indeed, defining copies at the program specification level is helpful in those many cases in which infringement has to be evaluated among programs written in different programming languages (Rapaport, 2017).

And the epistemological question for the copy relation is also connected to the problem of determining whether second-order properties, including safety and reliability, are preserved through copies. Our distinction among exact, inexact, and approximate copies, and the corresponding formal relation between bisimulation and CTL^* equivalence, and between simulation and $\forall CTL^*$ equivalence, provides a conceptual and formal framework wherein the problem can be advanced and examined.

Acknowledgements

The initial version of this work has been made possible thanks to a Visiting Professorship granted to Giuseppe Primiero by the University of Sassari in 2017. Both authors are grateful for the opportunity given. The final version of this work has been realised in the context of the Research Project ANR-17-CE38-0003-01 (ANR - Agence Nationale de la Recherche) titled 'What is a (computer) program: Historical and Philosophical Perspectives'.

References

Abadi, M. and L. Lamport (1991). The existence of refinement mappings. *Theoretical Computer Science 82*(2), 253 – 284.

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

Baxter, D. L. (1988). Many-one identity. *Philosophical Papers 17*(3), 193–216.

Carrara, M., S. Gaio, and M. Soavi (2014). Artifact kinds, identity criteria, and logical adequacy. In M. Franssen, P. Kroes, T. Reydon, and P. E. Vermaas (Eds.), *Artefact Kinds: Ontology and the human-made world*, pp. 85– 101. Springer.

Carrara, M. and P. Giaretta (2001). Identity criteria and sortal concepts. In *Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001*, pp. 234– 243. ACM.

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

Chandrasekaran, B. (1994). Functional representation and causal processes. *Advances in Computers 38*, 73–143.

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

Clarke, Jr., E. M., O. Grumberg, and D. A. Peled (1999). *Model Checking*. Cambridge, MA, USA: MIT Press.

Davidson, D. (1969). The individuation of events. In *Essays in honor of Carl G. Hempel*, pp. 216–234. Springer.

Floridi, L. (2008). The method of levels of abstraction. *Minds and machines 18*(3), 303–329.

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

Fokkink, W. (2000). *Introduction to Process Algebra*. Texts in Theoretical Computer Science. An EATCS Series. Springer.

Franssen, M., G.-J. Lokhorst, and I. van de Poel (2015). Philosophy of technology. In E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy* (Fall 2015 ed.). Metaphysics Research Lab, Stanford University.

Frege, G. (1892). On concept and object. *The Frege Reader*, 181–193.

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

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

Goodman, N. (1968). *Languages of art: An approach to a theory of symbols*. Hackett publishing.

Goodman, N. (1972). *Problems and Projects*. Indianapolis: Bobbs-Merrill.

Gorogiannis, N. and M. Ryan (2007). Minimal refinements of specifications in model and termporal logics. *Formal Asp. Comput. 19*(1), 35–62.

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

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

Kroes, P. (2009). Engineering and the dual nature of technical artefacts. *Cambridge Journal of Economics 34*(1), 51–62.

Kroes, P. (2012). *Technical artefacts: creations of mind and matter: a philosophy of engineering design*, Volume 6. Springer Science & Business Media.

Kroes, P. P. and A. A. Meijers (2006). The dual nature of technical artefacts. *Studies in History and Philosophy of Science 37*(1), 1.

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

Lewis, D. (1986). *On the plurality of worlds*. Oxford: Basil Blackwell.

Lewis, D. (1991). *Parts of classes*. Oxford: Basil Blackwell.

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

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 B. Hale and C. Wright (Eds.), *A Companion to the Philosophy of Language*, pp. 990–1012. Oxford: Blackwell Publishers Ltd.

Lowe, E. J. (1998). *The possibility of metaphysics: Substance, identity, and time*. Clarendon Press.

Meijers, A. A. (2000). The relational ontology of technical artifacts. In P. Kroes and A. Meijers (Eds.), *The empirical turn in the philosophy of technology*. Amsterdam: Elsevier.

Middelburg, C. A. (2016). A short introduction to process theory. *CoRR abs/1610.01412*.

Milner, R. (1971). An algebraic definition of simulation between programs. In *Proceedings of the 2Nd International Joint Conference on Artificial Intelligence*, IJCAI'71, San Francisco, CA, USA, pp. 481–489. Morgan Kaufmann Publishers Inc.

Noonan, H. and B. Curtis (2017). Identity. In E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy* (Spring 2017 ed.). Metaphysics Research Lab, Stanford University.

Pearl, J. (2009). *Causality: Models, Reasoning and Inference* (2nd ed.). New York, NY, USA: Cambridge University Press.

Primiero, G. (2016). Information in the Philosophy of Computer Science. In L. Floridi (Ed.), *The Routledge Handbook in the Philosophy of Information*, pp. 90–106. Routledge.

Primiero, G. and F. Raimondi (2015). Software theory change for resilient near-complete specifications. In E. M. Shakshuki (Ed.), *Proceedings of the 6th International Conference on Ambient Systems, Networks and Technologies (ANT 2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015), London, UK, June 2-5, 2015*, Volume 52 of *Procedia Computer Science*, pp. 988–995. Elsevier.

Quine, W. V. (1948). On what there is. *The Review of Metaphysics 2*(1), 21–38.

Quine, W. V. O. (1990). *Pursuit of truth*. Harvard University Press.

Rapaport, W. J. (2017). *Philosophy of computer science*. Draft.

Stone, R. B. and K. Wood (1999). Development of a functional basis for design. *ASME. J. Mech. Des 122*, 359–370.

Turner, R. (2014). Programming languages as technical artifacts. *Philosophy & technology 27*(3), 377–397.

Turner, R. and N. Angius (2017). The philosophy of computer science. In E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy* (Spring 2017 ed.). Metaphysics Research Lab, Stanford University.

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

Vermaas, P. E. and W. Houkes (2003). Ascribing functions to technical artefacts: A challenge to etiological accounts of functions. *The British Journal for the Philosophy of Science 54*(2), 261–289.

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

Williamson, T. (1990). *Identity and discrimination*. Oxford: Basil Blackwell.

Wright, C. (1983). *Frege's conception of numbers as objects*. Aberdeen: Aberdeen University Press.