Effectiveness for Input Output Conformance Simulation iocos

Carlos Gregorio-Rodríguez, Luis Llana, Rafael Martínez-Torres

Effectiveness for input output conformance simulation iocos*

Carlos Gregorio-Rodríguez, Luis Llana, and Rafael Martínez-Torres

Departamento Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Spain
cgr@sip.ucm.es, llana@ucm.es, rmartine@fdi.ucm.es

Abstract. In this paper we continue the study of the input-output conformance simulation (iocos). In particular, we focus on implementation aspects to show that iocos is indeed an interesting semantic relation for formal methods. We address two complementary issues: a) In the context of model based testing (MBT) we present an online, also called on-the-fly, testing algorithm that checks whether an implementation conforms a given specification. Online testing combines test generation and execution and avoids the generation of the complete test suite for the specification. We prove both soundness and completeness of the online algorithm with respect to the iocos relation. b) In the context of formal verification and model checking minimisation a key issue is to efficiently compute the considered semantic relations; we show how the coinductive flavour of our conformance relation iocos makes it appropriate to be cast into an instance of the Generalised Coarsest Partition Problem (GCPP) and thus it can be efficiently computed.

Keywords: Model Based Testing, Online Testing, Simulation Algorithm, Input Output Conformance Simulation, Model Checking Minimisation, Verification, Generalised Coarsest Partition Problem, Formal Methods.

1 Introduction

In a recent paper [12] we introduced the input-output conformance simulation relation (iocos) that refines the classic Tretman's ioco. The work by Tretmans [25] has settled a solid and widespread framework in the Model Based Testing (MBT) community: it offers both offline and online [9] testing algorithms; and there are several model-based test generation tools (e.g. [26,3]) that implement the ioco-testing theory.

From a theoretical point of view, some interesting particularities of the ioco-framework are: behaviours are modelled as labelled transition systems (LTS); quiescent states (see [24]) are considered; implementations should be input-enabled; and the ioco relation is a trace-based semantics, and thus a linear semantics [27].

* Research partially supported by the Spanish MEC projects TIN2009-14312-C02-01 and TIN2012-36812-C02-01

Our iocos approach shares LTS as models, quiescence, and much of the conformance philosophy, while considering a wider behaviour domain not imposing, but allowing, implementations to be input enabled. The substantial difference is that the conformance relation is an input-output simulation (iocos) —a branching semantics [27]— with greater discriminatory power than ioco (see Theorem 1 below). In [12] we presented an offline algorithm that starting from a formal model of the specification produced a test suite to be checked against the possible implementations. We proved the resulting test suite to be sound and exhaustive for the given specification with respect to iocos.

In this paper we present an alternative approach to offline testing that avoids the generation of the whole test suite. When time or space requirements are considered, a more suitable testing approach is the so called online testing. Online testing considers a concrete implementation to be checked against the specification. It combines test generation from the specification model and test execution against the implementation. In this approach only a single step of a test is generated from the model and executed in the implementation; the results of this execution are taken into account to generate the next step in the test, that is again checked against the implementation. In order to show the applicability of our conformance relation, in this paper we define an online testing algorithm for iocos that we prove to be sound and complete.

An essential point of our conformance relation is that it is simulation-based. Simulation is an important notion pervading many fields in computer science (model checking, concurrency theory, formal verification...). It is an active area of research both theoretical (e.g. [2,8,17,10]) and practical (e.g. [5,22]). Regarding practical implementation applications for iocos, it is particularly interesting its use in model checking minimisation ([6,14]) as a technique to overcome the state explosion problem.

The quest for efficient algorithms[1] to compute this relation has been an area of active research in the last years (e.g.[11,5,22]). In [22] you can find an excellent review of the state of the art for simulation algorithms. One of the most outstanding algorithms is the one presented in [11], and subsequently corrected in [28]. It is one of the fastest algorithms and quasi-optimal in space. This algorithm exploits the representation of the simulation problem as a Generalised Coarsest Partition Problem (GCPP). We show in this paper how the conformance relation iocos can be cast into an instance of the GCPP. Therefore it can be efficiently computed using the algorithm in [28].

The paper is organised as follows: in Section 2 we present the essential notation, definitions and results of the iocos-theory used in the rest of the paper. In Section 3, definition and behaviour of our online testing algorithm are explained and we discuss other related online testing algorithms in the literature. Section 4 is devoted to prove that iocos is indeed a relation that can be effi-

[1] As a general overview, we could say that while deciding trace inclusion on finite-state processes is PSPACE-hard [23], the simulation preorder is decidable in polynomial time. Actually, simulation preorder is the coarsest preorder included in trace inclusion with this property.

ciently computed. We use the technique of system transformation to adequate iocos to the problem definition in [11]. Then we prove that this transformation holds all requirements to be computed as a GCPP. Finally in Section 5 we make a summary and advance some future research lines. In order to meet the space requirements, there is an appendix containing the more technical proofs of the paper. This appendix will be available on-line in case of acceptance of the paper.

2 Preliminaries

This section presents the notation used in the paper and reviews the formal framework of the iocos theory introduced in [12,16].

We consider two disjoint finite sets of actions: inputs I, initiated by the environment and annotated with a question mark, $a?, b?, c? \in I$; and outputs O, initiated by the system, and annotated with an exclamation mark, $o!, u!, t! \in O$. In many cases we want to name actions in a general sense, inputs and outputs indistinctly. We will consider the set $L = I \cup O$ and we will omit the exclamation or question marks when naming generic actions, $x, y, z \in L$.

A state with no output actions cannot autonomously proceed, such a state is called *quiescent*. Quiescence is an essential component of the ioco theory. For the sake of simplicity and without lost of generality (see for instance [25,24]), we directly introduce the event of quiescence as a special action denoted by $\delta! \in O$ into the definition of our models.

Definition 1. A *labelled transition system with inputs and outputs* is a 4-tuple $(S, I, O, \rightarrow)$ such that

- S is a set of states or behaviours.
- I and O are disjoint sets of input and output actions respectively. Output action include the quiescence symbol $\delta! \in O$. We define $L = I \cup O$.
- $\rightarrow \subseteq S \times L \times S$. As usual we write $p \xrightarrow{x} q$ instead of $(p, x, q) \in \rightarrow$ and $p \xrightarrow{x}$, for $x \in L$, if there exists $q \in S$ such that $p \xrightarrow{x} q$. Analogously, we will write $p \not\xrightarrow{x}$, for $x \in L$, if there is no q such that $p \xrightarrow{x} q$.
  In order to allow only coherent quiescent systems the set of transitions $\rightarrow$ should also satisfy:
  - if $p \xrightarrow{\delta!} p'$ then $p = p'$. A quiescent transition is always reflexive.
  - if $p \not\xrightarrow{o!}$ for any $o! \in O \backslash \{\delta!\}$, then $p \xrightarrow{\delta!} p$. A state with no (regular) outputs is quiescent.
  - if there is $o! \in O \backslash \{\delta!\}$ such that $p \xrightarrow{o!}$, then $p \not\xrightarrow{\delta!}$. A quiescent state performs no other output action.
- A system is *input-enabled* if at any $s \in S$ for every $a? \in I$ we have $s \xrightarrow{a?}$. □

We denote the set of labelled transition systems with inputs and outputs just as LTS. In general we use $p, q, p', q' \ldots$ for states or behaviours, but also i, i', s and s' when we want to emphasise the concrete role of a behaviours as

implementation or specification. We consider implementations and specifications, or, more generally, behaviours under study, as states of the same LTS.

A trace is a finite sequence of symbols of L. We will normally use the symbol σ to denote traces, that is, $\sigma \in L^*$. The empty trace is denoted by ϵ and we juxtapose, $\sigma_1\sigma_2$, to indicate concatenation of traces. The transition relation of labelled transition systems can naturally be extended using traces instead of single actions, $p \xrightarrow{\sigma} q$. Next we introduce some definitions and notation frequently used in the paper.

Definition 2. Let $(S, I, O, \rightarrow) \in LTS$, and $p \in S$, we define:

1. $p \,\mathsf{after}\, \sigma = \{p' \mid p' \in S,\ p \xrightarrow{\sigma} p'\}$, the set of states after the trace σ.
2. $\mathsf{outs}(p) = \{o! \mid o! \in O,\ p \xrightarrow{o!}\}$, the outputs of a state p (it may include δ!).
3. $\mathsf{ins}(p) = \{a? \mid a? \in I,\ p \xrightarrow{a?}\}$, the set of inputs of a state p. □

A behaviour is deterministic when for any $x \in L$, if $p \xrightarrow{x} p_1$ and $p \xrightarrow{x} p_2$ then $p_1 = p_2$; or equivalently the set $p \,\mathsf{after}\, \sigma$ is always empty or a singleton. While some models prevent non determinism we assume and allow all kinds of non-deterministic behaviour both in specifications and implementations.

Next we recall the simulation-based formal definition of iocos.

Definition 3. Let $(S, I, O, \rightarrow) \in LTS$, we say that a relation $R \subseteq S \times S$ is a iocos-relation if and only if for any $(p, q) \in R$ the following conditions hold:

1. $\mathsf{ins}(q) \subseteq \mathsf{ins}(p)$
2. $\forall a? \in \mathsf{ins}(q)$ if $p \xrightarrow{a?} p'$ then $\exists q' \in S$ such that $q \xrightarrow{a?} q' \wedge (p', q') \in R$.
3. $\forall o! \in \mathsf{outs}(p)$ if $p \xrightarrow{o!} p'$ then $\exists q' \in S$ such that $q \xrightarrow{o!} q' \wedge (p', q') \in R$.

We define the *input-output conformance simulation* (iocos) as the union of all iocos-relations (the biggest iocos-relation). □

Additional details on iocos rationale and examples appear in [12], where it is also proved that iocos is a strictly finer relation than ioco.

Theorem 1. [12] Let $(S, I, O, \rightarrow) \in LTS$; then $\mathsf{iocos} \subseteq \mathsf{ioco}$. That is, for any $p, q \in S$, whenever we have $p \,\mathsf{iocos}\, q$ it is also true that $p \,\mathsf{ioco}\, q$. □

3 An online testing algorithm for iocos

The main goal of this section is to prove iocos as a suitable conformance relation to be used in online testing. As usual, the online testing algorithm merges test generation and execution into a single interactive process. In order to understand the notation and some ideas and key concepts behind Algorithm 1, next we recall the iocos theory of testing: test definition and test execution.

Definition 4. A *test* is a syntactical term defined by the following BNF:

$$T = \text{✗} \mid \text{✓} \mid T_1 \oplus T_2 \mid T_1 + T_2 \mid x;T \qquad \text{where } x \in L$$

We denote the set of tests as $\mathcal{T}$. □

As usual in MBT, the environments we want to model should be able to respond at any moment to any possible output of the implementation under test. That is, tests like $a?;T_{a?}$ will not be accepted as *valid* tests, but should be completed into tests like $a?;T_{a?} + \sum_{o!\in O} o!;T_{o!}$. For the sake of simplicity we use $\sum_{i\in\{1,\ldots,n\}} T_i$ as a shortcut for $T_1 + \cdots + T_n$.

Particularly interesting is that we consider two kind of choices in the tests: the one corresponding to the $+$ operator, with conjunctive semantics, and the $\oplus$ operator with a disjunctive meaning. To define how tests interacts with behaviours and what is the result of the execution of that experiment, we follow Abramsky's ideas in [1] and use a predicate to define the outcomes of the interaction between a test and the behaviour or implementation being tested.

Definition 5. Let $(S, I, O, \rightarrow) \in LTS$, $s \in S$, $a? \in I$, and $o! \in O$, we inductively define the predicate $\mathsf{pass} \subseteq S \times \mathcal{T}$ as follows:

$$
\begin{array}{l}
s \;\mathsf{pass}\; \text{✗} = false \\
s \;\mathsf{pass}\; \text{✓} = true \\
s \;\mathsf{pass}\; o!;T_{o!} = \begin{cases} true & \text{if } o! \notin \mathsf{outs}(s) \\ \bigwedge\{s' \;\mathsf{pass}\; T_{o!} | s \xrightarrow{o!} s'\} & \text{otherwise} \end{cases} \\
s \;\mathsf{pass}\; a?;T_{a?} = \begin{cases} false & \text{if } a? \notin \mathsf{ins}(s) \\ \bigwedge\{s' \; pass \; T_{a?} | s \xrightarrow{a?} s'\} & \text{otherwise} \end{cases} \\
s \;\mathsf{pass}\; T_1 + T_2 = s \;\mathsf{pass}\; T_1 \wedge s \;\mathsf{pass}\; T_2 \\
s \;\mathsf{pass}\; T_1 \oplus T_2 = s \;\mathsf{pass}\; T_1 \vee s \;\mathsf{pass}\; T_2
\end{array}
$$

□

For the sake of convenience the pass predicate is inductively defined over the whole set of tests, with a simpler structural formulation, while at the end we will only be interested in valid tests.

In [12] we defined an algorithm (Definition 6 below) that starting from a given specification s produced a test suite $\mathcal{T}(s)$ of valid tests that characterised the specification with respect to the iocos conformance relation as stated in Theorem 2 below.

Definition 6. Let $(S, I, O, \rightarrow) \in LTS$ and $p \in S$. We denote with $\mathcal{T}(p)$ the set of valid tests from p by applying a finite number of recursive applications of one of the following non-deterministic choices:

1. $T = \text{✓} \in \mathcal{T}(p)$.
2. If $a? \in \mathsf{ins}(p)$, then $T \in \mathcal{T}(p)$ where

$$T = a?;\bigoplus\{Tp_{a?} \mid p \xrightarrow{a?} p_{a?}\} + \sum_{o!\in\mathsf{outs}(p)} o!;\bigoplus\{T_{p_{o!}} \mid p \xrightarrow{o!} p_{o!}\} \; + \sum_{\substack{o!\in O \\ o!\notin\mathsf{outs}(p)}} o!;\text{✗}$$

3. If $\mathsf{ins}(p) = \varnothing$ then $T \in \mathcal{T}(p)$ where

$$T = \sum_{o!\in\mathsf{outs}(p)} o!;\bigoplus\{T_{p_{o!}} \mid p \xrightarrow{o!} p_{o!}\} + \sum_{\substack{o!\in O \\ o!\notin\mathsf{outs}(p)}} o!;\text{✗}$$

In all cases the tests T_p are chosen non-deterministically from the set $\mathcal{T}(p)$, $T_\delta(p) = \checkmark$ if $p \xrightarrow{\delta!}$, and $T_{\delta!}(p) = \text{✗}$ otherwise. □

Theorem 2. [12](***Completeness***) Let $(S, I, O, \rightarrow) \in LTS$ and $p, q \in S$, $T \in \mathcal{T}(p) : q \,\mathsf{pass}\, T$ iff $q \,\mathsf{iocos}\underline{\mathsf{s}}\, p$. □

Online Algorithm Offline generation of test leads to a massive test suite.[2] This is not surprising given that the test suite should be able to prove and disprove the conformance with any *possible* implementation. Online testing, on the contrary, considers a concrete implementation under test (IUT) and, therefore, the testing process can be guided by the mutual interaction.

The online testing algorithm for $\mathsf{iocos}\underline{\mathsf{s}}$ is defined in Algorithm 1. This algorithm somehow merges the test generation and execution (Definition 5 and 6), and it considers only the necessary continuations by taking into account the current state of the specification and the last response from the implementation.

Algorithm 1 starts with a specification, an IUT, and a desired number of iterations (parameters s, IUT and $maxIter$ in function TE). The testing process will continue until a ✗ verdict is found, indicating IUT does not conform s, or until the number of iterations has been reached without a faulty behaviour found in IUT.

According to Definition 5 the testing process will yield a ✗ verdict on these cases: a mandatory input is rejected by IUT or an unexpected output by IUT is registered by the tester. In Algorithm 1, the first situation is solved in the first conditional inside the first **case** statements in function TE_{REC} (lines 16 and 17). Let us note that to check the enabled actions in IUT (line 16), is equivalent to check condition 1 in Definition 3 ($\mathsf{iocos}\underline{\mathsf{s}}$).

Otherwise, if the action that the test chooses to offer is actually enabled in the IUT, the stimulus is sent to the implementation (line 18) and then condition 2 in Definition 3 ($\mathsf{iocos}\underline{\mathsf{s}}$) should be tested: at least one of the descendant tests must be passed by the current state of the implementation, otherwise the final return statement outside the loop will propagate the ✗ verdict. The *copy* clause (line 19) —theoretically essential, see [1,21]— is used to check every descendant test against the same state of the implementation. For software artifacts, and even embedded systems that can be easily replicated, this is indeed a feasible operation.

In a similar way, if we focus on output actions, conditional in lines 28 and 29 detects unexpected outputs from the implementation. Otherwise, if an output received from the IUT is acceptable, the algorithm has to proceed and check condition 3 in Definition 3 ($\mathsf{iocos}\underline{\mathsf{s}}$) as described previously.

Algorithm 1 is non-deterministic and the three cases in the **choice** statement (line 13) are arbitrarily chosen in every run of the function TE_{REC}. The *reset* case introduces the possibility of breadth exploration of the IUT, while the other two choices produce a (one step) deep exploration.

[2] Of course, there is a very interesting line of research in MBT that tries to reduce the size of the test suite while keeping a *good* coverage.

Algorithm 1 Online Testing Algorithm for ioco$_s$

```
1: function TE(s, iut, maxIter)
2:     continue ← ✓
3:     numIter ← maxIter
4:     while numIter > 0 ∧ continue == ✓ do
5:         continue, numIter ← TE_REC(s, iut, numIter)
6:         if continue == ✓ then
7:             reset iut
8:     return continue
9: function TE_REC(s, iut, numIter)
10:    if numIter = 0 then
11:        return ✓, numIter
12:    else
13:        choice
14:            case action do                ▷ Offers an input to the implementation
15:                choice a ∈ ins(s)
16:                if a? is not enabled in iut then
17:                    return ✗, numIter
18:                send a? to iut
19:                iut_0 ← copy(iut)
20:                for s' ∈ s after a? do
21:                    iut ← copy(iut_0)
22:                    continue, numIter ← TE_REC(s', iut, numIter − 1)
23:                    if continue == ✓ then
24:                        return ✓, numIter
25:                return ✗, numIter
26:            case wait do              ▷ Waits for an output from the implementation
27:                wait o! from iut
28:                if s after o! = ∅ then
29:                    return ✗, T
30:                iut_0 ← copy(iut)
31:                for s' ∈ s after o! do
32:                    iut ← copy(iut_0)
33:                    continue, numIter ← TE_REC(s', iut, numIter − 1)
34:                    if continue == ✓ then
35:                        return ✓, numIter
36:                return ✗, numIter
37:            case reset do                 ▷ Resets implementation and restart
38:                return ✓, maxIter
39:
```

There are in the literature of MBT two essential works we can relate Algorithm 1 with. In [9] de Vries and Tretmans presented an online algorithm for ioco that is non-deterministic and even termination is one of the non-deterministic choices. In [15] Larsen et al. describe an online algorithm for an ioco-based conformance relation extended with real time and considering a concrete environment, namely $\mathsf{rtioco_e}$. Algorithm 1 is more similar in form to that on [15] and share with it the *reset* clause and the explicit use of the number of iterations. We do not consider explicit environments but the most general possible environment.[3] As in [9,15], quiescence detection can be implemented with a timeout.

In [29] you can find a practical online testing algorithm that is implemented in the MBT tool developed at Microsoft Research called Spec Explorer. The framework and the approach are somehow different: they use interface automaton as the specification model and assume the implementation to be in the domain of the specifications. In this work soundness and completeness are not even mentioned.

Soundness and Completeness The rest of this section is devoted to prove that Algorithm 1 is indeed sound and complete.

In [9] correctness of the online algorithm for ioco is stated under fairness assumptions (*weather conditions* [19]) in the non-deterministic choices. In [15] online algorithm for $\mathsf{rtioco_e}$ is proved to be sound and complete assuming the classic *test hypothesis*,[4]fairness in the randomisation of algorithm choices, and deterministic IUT.

In our model, we allow both implementations and specifications to behave non-deterministically and, as usual, we assume fairness in the non-deterministic choices and the *test hypothesis* ([9,15]). Test hypothesis assumes that any IUT can be modelled in the domain of LTS.

Theorem 3. Let $(S, I, O, \rightarrow)$ be a LTS, let $i, s \in S$, and $n \in \mathbb{N}$. If the function call $\mathrm{TE}(s, i, n)$ in Algorithm 1 returns a ✗, then $i \not\mathsf{ioco}_S s$.

Proof. The proof proceed by induction on the parameter n of $\mathrm{TE}(s, i, n)$, following the ideas commented in the algorithm explanation. □

Asserting completeness implies that, for any faulty implementation of a given specification s, the online testing process in Algorithm 1 should eventually yield a ✗ result, that is, the algorithm should drive the computation through a test passed by the specification and failed by the implementation. Let us note that a key point to prove the completeness is to assure that all states in the specification and implementation should be inspected. This is where the fairness requirement is needed: every infinitely often eligible action is eventually executed. To ensure this possibility one of the choices of the algorithm is to reset the IUT and to

[3] To introduce environments might lead to a reduction of the search-space because this additional knowledge can be used to further restrict the testing process.

[4] The behaviour of IUT can be described in the model domain. Only the existence is assumed, not a concrete and known instance.

restart the algorithm (lines 37 and 38). Moreover, assuming the fairness hypothesis also implies that the number of times required to ensure that all states have been checked is not known a priori. So the number of iterations appearing in Proposition 1 and Theorem 1 cannot be determined at the beginning of the execution.

Proposition 1. Let $(S, I, O, \rightarrow)$ be a *LTS*, let $i, s \in S$, and let $T \in \mathcal{T}(s)$ be a test such that $s \;\mathsf{pass}\; T$ and $i \;\mathsf{pa\not{s}s}\; T$. Then there exist $n, n' \in \mathbb{N}$ such that $\mathrm{TE}_{\mathrm{REC}}(s, i, n)$ returns $\text{✗},\ n'$.

Proof. We make the proof by induction on the depth of T. The case base is when $T = \text{✓}$ that is trivial since $i \;\mathsf{pass}\; T$. According to Definition 6, there are two cases. We are going to consider the test

$$T = a?; \bigoplus\{T_{s_{a?}} \mid s \xrightarrow{a?} s_{a?}\} + \sum_{o! \in \mathsf{outs}(s)} o!; \bigoplus\{T_{s_{o!}} \mid s \xrightarrow{o!} s_{o!}\} \; + \sum_{\substack{o! \in O \\ o! \notin \mathsf{outs}(s)}} o!; \text{✗}$$

The other case is simpler than this one. Since $i \;\mathsf{pa\not{s}s}\; T$, there are three possibilities to make the test fail:

1. $i \;\mathsf{pa\not{s}s}\; a?; \bigoplus\{T_{s_{a?}} \mid s \xrightarrow{a?} s_{a?}\}$. Since we are assuming the *fairness hypothesis*, by choosing an arbitrary high n the choice in line 15 of the algorithm will eventually choose the action $a?$. If $a?$ is not enabled in i, then the algorithm returns ✗ in line 17. Otherwise the algorithm sends $a?$ to the implementation. Again, by the *fairness hypothesis* the implementation, eventually will go to a state i' such that $i' \;\mathsf{pa\not{s}s}\; \bigoplus\{T_{s_{a?}} \mid s \xrightarrow{a?} s_{a?}\}$. Let us name $n_{i'}$ the number of iterations needed to reach this state. On the other hand, since $s \;\mathsf{pass}\; T$, $s' \;\mathsf{pass}\; \bigoplus\{T_{s_{a?}} \mid s \xrightarrow{a?} s_{a?}\}$ for any $s' \in s \;\mathsf{after}\; a?$. So by induction hypothesis for any $s' \in s \;\mathsf{after}\; a?$ there exist $n_{s'} \in \mathbb{N}$ such that $\mathrm{TE}_{\mathrm{REC}}(s', i, n_{s'})$ returns ✗. So by choosing $n_0 \geq= n_{i'} + \sum_{s' \in s\mathsf{after}a?} n_{s'}$, there exists n' such that the $\mathrm{TE}_{\mathrm{REC}}(s, i, n_0)$ returns $\text{✗},\ n'$.
2. $i \;\mathsf{pa\not{s}s}\; \sum_{o! \in \mathsf{outs}(s)} o!; \bigoplus\{T_{s_{o!}} \mid s \xrightarrow{o!} s_{o!}\}$. In this case there is $o! \in \mathsf{outs}(s)$ such that i produces output $o!$ and goes to an state i' such that $i \;\mathsf{pa\not{s}s}\; \bigoplus\{T_{s_{o!}} \mid s \xrightarrow{o!} s_{o!}\}$. By the *fairness hypothesis* the implementation will eventually produce that output and will go to state i'. Let us name $n_{i'}$ the number of iterations needed to reach this state. On the other hand, $s' \;\mathsf{pass}\; \bigoplus\{T_{s_{o!}} \mid s \xrightarrow{o!} s_{o!}\}$ for any $s' \in s \;\mathsf{after}\; o!$. So by induction hypothesis for any $s' \in s \;\mathsf{after}\; o!$ there exist $n_{s'} \in \mathbb{N}$ such that $\mathrm{TE}_{\mathrm{REC}}(s', i, n_{s'})$ returns ✗. So by choosing $n_0 \geq= n_{i'} + \sum_{s' \in s\mathsf{after}o!} n_{s'}$, there exists n' such that the $\mathrm{TE}_{\mathrm{REC}}(s, i, n_0)$ returns $\text{✗},\ n'$.
3. $i \;\mathsf{pa\not{s}s}\; \sum_{\substack{o! \in O \\ o! \notin \mathsf{outs}(s)}} o!; \text{✗}$. In this case there is $o! \in O$ such that i produces output $o!$. By the *fairness hypothesis* the implementation will produce this output an the algorithm will return ✗ in line 29.

□

Since the main function of the algorithm just makes calls to the recursive algorithm as a corollary we get the following result:

Theorem 4. Let $(S, I, O, \rightarrow)$ be a *LTS*, let $i, s \in S$. If $i \not\mathsf{iocos}\, s$ then there exist $n \in \mathbb{N}$ such that the function call $\mathrm{TE}(s, i, n)$ in Algorithm 1 returns ✗. □

4 iocos as a Generalised Coarsest Partition Problem

Milner in [18] introduced the preorder relation called simulation. Simulation equivalence strongly preserves ACTL*, and also strongly preserves LTL and ACTL as sublogics of ACTL* [6]. Both ACTL and LTL are widely used for model checking in practice.

Simulation is a close relative of the well known bisimulation equivalence [20]. Bisimulation equivalence can be fast computed by reducing it to the problem of determining the coarsest partition of a set stable with respect to a given relation [13]. An equivalent result for computing simulation as a generalised coarsest partition problem (GCPP) was given in [11] and corrected in [28]. This algorithm keeps a very good time complexity while its space complexity can be considered minimal.

Although iocos definition is simulation-like, it has particularities inherited from the classic requirements for the conformance relations when dealing with inputs and outputs, and even the use of input-output actions is not symmetric in the definition. Therefore, the applicability of simulation algorithms to compute iocosis not straightforward.

We will show in this section that fortunately we can compute iocos by using any algorithm that solves the GCPP. To achieve this goal, first we will show how to transform the LTS into graphs and partitions as defined in [11,28]. Then we will define the relation g-iocos (graph-iocos) in the transformed LTS that will be equivalent to iocos. Finally, we will formally prove that the transformed LTS holds the conditions of the GCPP.

4.1 Transforming an LTS into a graph

The GCPP in [11] is defined in terms of graphs. These graphs have no labels in the edges. The objective is to transform an *LTS* into a graph without actions and without loosing information. The idea to achieve this is to *encode* the action associated to a transition into the states of the graph. So the states of the associated graph of an *LTS* will be pairs: the first component of the pair refers to a state of the *LTS* whereas the second component is the action needed to reach this state.

One of the details that we have to deal with to transform iocos into a simulation is that in Definition 3 (iocos) the implementation is allowed to introduce unspecified behaviour. We do not need to take into account those states reached by implementation when input actions are not present in the specification. To overcome this situation, in the transformed system, we need to represent these states by adding a sort of new *magic state*, that we denote by the symbol $*$. Such a state has the next property: *every possible implementation* will fulfil it. Moreover, in the transformation we will embed the transition action symbol into

the state itself. Exceptionally, and just for the sake of uniformity, a new action symbol $(\cdot)$ is used for states with no incoming transitions.

Definition 7. Let $\mathcal{L} = (S, I, O, \rightarrow)$ be a labelled transition system. We define its transformed graph as $\mathbb{T}(\mathcal{L}) = (N, \Rightarrow)$ where $N = (S \cup \{*\}) \times (L \cup \{\cdot\})$ and $\Rightarrow$ is defined by the following rules.

$$\frac{s \xrightarrow{y} s'}{(s,x) \Rightarrow (s',y)}, \quad \frac{s \xrightarrow{a?} \not\rightarrow}{(s,x) \Rightarrow (*,a?)}, \quad \frac{}{(*,x) \Rightarrow (*,y)}, \quad \begin{array}{l} s, s' \in S \\ x, y \in L \cup \{\cdot\},\ a? \in I \end{array}$$

The nodes of N will be denoted by the letters n, n_1, n_2, etc. Since these nodes come from states of the original labelled transition system, we annotate the arcs of the graph with the action of the target node for readability reasons. In this way will write $n_1 \stackrel{x}{\Longrightarrow} n_2$ whenever $n_1 \Rightarrow n_2$ and there is $s_2 \in S$ such that $n_2 = (s_2, x)$.

Following the same rationale we adapt the definition of the function ins: $\mathsf{ins}(*, x) = \varnothing$ and $\mathsf{ins}(s, x) = \mathsf{ins}(s)$ if $s \in S$. □

Definition 8. Let $(N, \Rightarrow)$ be the transformed graph of an $\mathcal{L} \in LTS$, a relation $R \subseteq N \times N$ is $\mathsf{g\text{-}iocos}$ *simulation* iff for any $n_1, n_2 \in N$, $(n_1, n_2) \in R$ the following conditions hold:

1. $n_1 = (s_1, x)$, $n_2 = (s_2, y)$ and $x = y$.
2. $\mathsf{ins}(n_2) \subseteq \mathsf{ins}(n_1)$
3. for all $a \in L$, if $n_1 \stackrel{a}{\Longrightarrow} n_1'$ then exists n_2' such that $n_2 \stackrel{a}{\Longrightarrow} n_2'$ and $n_1' R n_2'$.

For $n_1, n_2 \in N$, we say that n_1 $\mathsf{g\text{-}iocos}$ n_2 if there exists a $\mathsf{g\text{-}iocos}$ simulation R such that $(n_1, n_2) \in R$. □

At a first glance, $\mathsf{g\text{-}iocos}$ is defined as a *kind* of ready simulation preorder [4]. Next, we show that computing $\mathsf{g\text{-}iocos}$ over graphs is equivalent to compute iocos on LTS. Proposition 2 describes how to transform a concrete iocos-simulation into a $\mathsf{g\text{-}iocos}$-simulation, while Proposition 3 presents the reciprocal transformation. Finally, Theorem 5 is a corollary of the previous results showing iocos and $\mathsf{g\text{-}iocos}$ to be two different formulations for the same semantics in two different domains.

Proposition 2. Let $\mathcal{L} = (S, I, O, \rightarrow)$, let $\mathbb{T}(\mathcal{L}) = (N, \Rightarrow)$ be its transformed graph, let R be iocos relation $R \subseteq S \times S$, and let $R' \subseteq N \times N$ defined as

$$R' = \big\{((i,a),(s,a)) | (i,s) \in R, \forall a \in L \cup \{\cdot\}\big\} \cup \big\{(i,a)(*,a) |\ a \in L \cup \{\cdot\}, i \in S \cup \{*\}\big\}$$

Then R' is a $\mathsf{g\text{-}iocos}$ simulation.

Proof. Let us take $(n_1, n_2) \in R'$ and check that R' holds the statements of Definition 8.

$n_2 = (*, a)$**.** i.e., we are in the second subset of R'. Trivially we have $\mathsf{ins}(n_1) \supseteq \varnothing = \mathsf{ins}(n_2)$ (statement 2 of Definition 8). As $(*, x) \Rightarrow (*, y)$ for any $x, y \in L$ and $(n_1', (*, a)) \in R'$ for any n_1', then statement 3 of Definition 8 holds.

$n_1 = (*, a)$**.** As R' is defined, $n_2 = (*, a)$, and it is solved as in the previous case.

$n_1 \neq (*, a)$ **and** $n_2 \neq (*, a)$**.** Let us assume that $n_1 = (i, a)$ and $n_2 = (s, a)$. Since R is an iocos simulation, we obtain $\mathsf{ins}(n_1) = \mathsf{ins}((i, a)) = \mathsf{ins}(i) \supseteq \mathsf{ins}(s) = \mathsf{ins}((s, a)) = \mathsf{ins}(n_2)$ (statement 2 of Definition 8).

Now let us take action $x \in L$ such that $n_1 \stackrel{x}{\Longrightarrow} n_1'$. If $x \in O$ then we obtain that there exist i' such that $i \stackrel{x}{\longrightarrow} i'$ and $n_1' = (i', x)$. Since R is a iocos simulation, then there exists s' such that $s \stackrel{a}{\longrightarrow} s'$ and $(s, s') \in R$. Again, by the construction of the transitions $n_2 = (s, a) \stackrel{x}{\Longrightarrow} (s', x) = n_2'$ and $(n_1', n_2') \in R'$ by definition of R'.

If $x \in I$ there are two possibilities: either $x \notin \mathsf{ins}(i)$ or $x \in \mathsf{ins}(i)$. In the first case we have $(i, a) \stackrel{x}{\longrightarrow} (*, a) = n_1'$ by the way transitions are defined. Since R is an iocos simulation $\mathsf{ins}(s) \subseteq \mathsf{ins}(i)$ and therefore $x \notin \mathsf{ins}(s)$. Then $n_2 = (s, a) \stackrel{x}{\Longrightarrow} (*, x) = n_2'$ and $(n_1', n_2') \in R'$. In the second case there exists i' such that $i \stackrel{x}{\longrightarrow} i'$. If $x \in \mathsf{ins}(s)$, since R is an iocos simulation there exists s' such that $s \stackrel{x}{\longrightarrow} s'$ and and $(s, s') \in R$. By the construction of the transitions $n_2 = (s, a) \stackrel{x}{\Longrightarrow} (s', x) = n_2'$ and $(n_1', n_2') \in R'$ by definition of R'. If $x \notin \mathsf{ins}(s)$, then $(s, a) \stackrel{x}{\longrightarrow} (*, x) = n_2'$. Then $(n_1', n_2') \in R'$ by definition of R'

□

Proposition 3. Let $\mathcal{L} = (S, I, O, \rightarrow)$ be a labelled transition system, let $\mathbb{T}(\mathcal{L}) = (N, \Rightarrow)$ be its transformed graph, let $R' \subseteq N \times N$ be a $\mathsf{g\text{-}iocos}$ relation, and let $R \subseteq S \times S$ defined as $R = \{(i, s) | \exists a \in L \cup \{\cdot\}.\ ((i, a), (s, a)) \in R' \wedge i, s \in S\}$. Then R is a iocos simulation.

Proof. Let $(i, s) \in R$. By construction of R, there exists $a \in L$ and $n_1 = (i, a)$ and $n_2 = (s, a)$ such that $(n_1, n_2) \in R'$. Since R' is a $\mathsf{g\text{-}iocos}$ simulation, we have $\mathsf{ins}(i) = \mathsf{ins}((i, a)) = \mathsf{ins}(n_1) \supseteq \mathsf{ins}(n_2) = \mathsf{ins}((s, a)) = \mathsf{ins}(s)$ according to the statement 1 of the definition of iocos. Now let us check the other two conditions:

- Let us consider $b? \in \mathsf{ins}(s)$ such $i \stackrel{b?}{\longrightarrow} i'$, so $(i, a) \stackrel{b?}{\Longrightarrow} (i', b?)$. Since R' is a $\mathsf{g\text{-}iocos}$ simulation there exists $n_2' = (s', b?)$ such that $n_2 \stackrel{b?}{\Longrightarrow} n_2$. Since $b? \in \mathsf{ins}(s)$, $s' \neq *$, so $s' \in S$. Finally, by definition of, R we obtain $(i', s') \in R$.
- The case when $o! \in \mathsf{outs}(i)$ and $i \stackrel{o!}{\longrightarrow} i'$ is similar to the previous one, but in this case $s' \neq *$ because $o! \notin I$.

□

Finally, we obtain the theorem that relates iocos and $\mathsf{g\text{-}iocos}$. This theorem is a corollary of the previous Proposition 2 and 3.

Theorem 5. Let $\mathcal{L} = (S, I, O, \rightarrow)$ be a labelled transition system and $\mathbb{T}(\mathcal{L}) = (N, \Rightarrow)$ its transformed graph. Then, for any $s_1, s_2 \in S$, we have $(s_1, s_2) \in \mathsf{iocos}$ if and only if $((s_1, x), (s_2, x)) \in \mathsf{g\text{-}iocos}\ \ \forall x \in L$. □

4.2 g-iocos as a GCPP problem

To conclude this section we will show that $\mathsf{g\text{-}iocos}$ can be seen as an instance of the GCPP. In order to make the paper self-contained, we are going to reproduce in condensed form definitions from [11].

Definition 9. Let $\mathcal{L} = (S, I, O, \rightarrow)$ be a labelled transition system and $\mathbb{T}(\mathcal{L}) = (N, \Rightarrow)$ its transformed graph.

1. A *partition pair* is a tuple $\langle \mathcal{S}, \precsim \rangle$ such that $\mathcal{S} = \{\alpha, \beta, \dots\}$ is a partition of N and $\precsim$ is a reflexive and acyclic relation on $\mathcal{S}$.
2. Let $\mathcal{S}$ be a partition and let $n \in N$, we write $[n]_{\mathcal{S}}$ as the unique element of $\mathcal{S}$ that contains n.
3. Let $\alpha, \beta \in S$, we write $\alpha \Rightarrow_{\exists} \beta$ iff $\exists a \in \alpha$ such that $a \Rightarrow g$ for some $b \in \beta$. We write $\alpha \Rightarrow_{\forall} \beta$ iff $\forall a \in \alpha$ we have $a \Rightarrow b$ for some $b \in \beta$.
4. We say that $\langle S, \precsim \rangle$ is *stable* with respect to $\Rightarrow$ iff

$$\forall \alpha, \beta, \gamma \in \mathcal{S} \quad \text{if } (\alpha, \beta) \in \precsim \wedge \alpha \Rightarrow_{\exists} \gamma \text{ then } \exists \epsilon \in \mathcal{S} : \quad (\gamma, \epsilon) \in \precsim \wedge \beta \Rightarrow_{\forall} \epsilon$$

5. Let $\langle S_1, \precsim_1 \rangle$ and $\langle S_2, \precsim_2 \rangle$ be partition pairs. We say that $\langle S_1, \precsim_1 \rangle$ is a refinement of $\langle S_2, \precsim_2 \rangle$, written $\langle S_1, \precsim_1 \rangle \sqsubseteq \langle S_2, \precsim_2 \rangle$, if S_1 is finer than S_2, i.e., $(\forall \alpha \in S_1 \exists \alpha' \in S_2 : \ \alpha \subseteq \alpha')$ and $\precsim_1 \subseteq \precsim_2 (S_1)$ where $\precsim_2 (S_1)$ is the induced relation on S_1 by $\precsim_2$: $(\alpha, \beta) \in \precsim_2 (S_1)$ iff there exist $\alpha', \beta' \in S_2$ such that $\alpha \subseteq \alpha'$, $\beta \subseteq \beta'$ and $(\alpha', \beta') \in \precsim_2$.
6. Let us consider a pair $\langle S, \precsim \rangle$, the Generalised Coarsest Partition Problem (GCPP) for $\langle S, \precsim \rangle$ consists in finding a partition pair $\langle S_0, \precsim_0 \rangle$ such that: a) $\langle S_0, \precsim_0 \rangle \sqsubseteq \langle S \precsim \rangle$, b) $\langle S_0, \precsim_0 \rangle$ is stable respect to $\Rightarrow$, and c) $\langle S_0, \precsim_0 \rangle$ is maximal fitting a) and b).

□

Now we are ready to set up our key concepts in order to embed iocos computation into GCPP problem via a $\mathsf{g\text{-}iocos}$ reduction.

Definition 10. Let $\mathcal{L} = (S, I, O, \rightarrow)$ be a labelled transition system and $\mathbb{T}(\mathcal{L}) = (N, \Rightarrow)$ its transformed graph. We write $\equiv_{\mathsf{g\text{-}iocos}}$ for the kernel of $\mathsf{g\text{-}iocos}$, that is $(a, b) \in \equiv_{\mathsf{g\text{-}iocos}}$ iff $(a, b) \in \mathsf{g\text{-}iocos}$ and $(b, a) \in \mathsf{g\text{-}iocos}$.

Since $\equiv_{\mathsf{g\text{-}iocos}}$ is an equivalence relation, it induces the partition $N/\equiv_{\mathsf{g\text{-}iocos}}$. In this partition we can define $\precsim_{\mathsf{g\text{-}iocos}}$ as the natural relation induced by $\mathsf{g\text{-}iocos}$ in $N/\equiv_{\mathsf{g\text{-}iocos}}$, namely, for $n_1, n_2 \in N$, $([n_1]_{\mathsf{g\text{-}iocos}}, [n_2]_{\mathsf{g\text{-}iocos}}) \in \precsim_{\mathsf{g\text{-}iocos}}$ iff $(n_1, n_2) \in \mathsf{g\text{-}iocos}$. □

Let us note, that $\precsim_{\mathsf{g\text{-}iocos}}$ is reflexive and acyclic in $N/\equiv_{\mathsf{g\text{-}iocos}}$, so $\langle N/\equiv_{\mathsf{g\text{-}iocos}}, \precsim_{\mathsf{g\text{-}iocos}} \rangle$ is a partition pair. So, the rest of this section is devoted to prove that the partition $\langle N/\equiv_{\mathsf{g\text{-}iocos}}, \precsim_{\mathsf{g\text{-}iocos}} \rangle$ can be solved with any algorithm that solves the GCPP. In order to do it we need to define an initial partition pair $\langle \Omega, \preccurlyeq \rangle$ such that $\langle N/\equiv_{\mathsf{g\text{-}iocos}}, \precsim_{\mathsf{g\text{-}iocos}} \rangle \sqsubseteq \langle \Omega, \preccurlyeq \rangle$.

Definition 11. Let $\mathcal{L} = (S, I, O, \rightarrow)$ be a labelled transition system and $\mathbb{T}(\mathcal{L}) = (N, \Rightarrow)$ its transformed graph.

- We define the partition $\Omega = \{\alpha_1, \alpha_2, \dots \alpha_n\} \subseteq \mathcal{P}(N)$ as follows: $(n_1, n_2) \in \alpha_i$ if and only if $n_1 = (s_1, x)$, $n_2 = (s_2, y)$, $\mathsf{ins}(s_2) = \mathsf{ins}(s_1)$ and $x = y$.
- We define the relation $\preccurlyeq \subseteq \Omega \times \Omega$ as $[n_1]_\Omega \preccurlyeq [n_2]_\Omega$ iff $n_1 = (s_1, x)$, $n_2 = (s_2, y)$, $\mathsf{ins}(s_2) \subseteq \mathsf{ins}(s_1)$ and $x = y$.

□

Lemma 1. Let $\mathcal{L} = (S, I, O, \rightarrow)$ be a labelled transition system and its transformed graph $\mathbb{T}(\mathcal{L}) = (N, \Rightarrow)$. Then $\langle N/\equiv_{\mathsf{g\text{-}iocos}}, \precsim_{\mathsf{g\text{-}iocos}}\rangle \sqsubseteq \langle \Omega, \preccurlyeq\rangle$.

Proof. This lemma is immediate because of the definition of $\mathsf{g\text{-}iocos}$-simulation, kernel $\equiv_{\mathsf{g\text{-}iocos}}$ and the induced partition pair $\langle N/\equiv_{\mathsf{g\text{-}iocos}}, \precsim_{\mathsf{g\text{-}iocos}}\rangle$. □

Lemma 2. Let $\mathcal{L} = (S, I, O, \rightarrow)$ be a labelled transition system and its transformed graph, $\mathbb{T}(\mathcal{L}) = (N, \Rightarrow)$, then the partition pair $\langle N/\equiv_{\mathsf{g\text{-}iocos}}, \precsim_{\mathsf{g\text{-}iocos}}\rangle$ is stable with respect to $\Rightarrow$.

Proof. Let us consider equivalence classes $[n_1]_{\mathsf{g\text{-}iocos}}, [n_2]_{\mathsf{g\text{-}iocos}}, \gamma \in N/\equiv_{\mathsf{g\text{-}iocos}}$ such that $([n_1]_{\mathsf{g\text{-}iocos}}, [n_2]_{\mathsf{g\text{-}iocos}}) \in \precsim_{\mathsf{g\text{-}iocos}}$ and $[n_1]_{\mathsf{g\text{-}iocos}} \Rightarrow_\exists \gamma$. Then there exists $n_1' \in \gamma$ and $x \in L$ such that $n_1 \stackrel{x}{\Longrightarrow} n_1'$. As $(n_1, n_2) \in \mathsf{g\text{-}iocos}$ there exists n_2' such that $n_2 \stackrel{x}{\Longrightarrow} n_2'$ and $([n_1']_{\mathsf{g\text{-}iocos}}, [n_2']_{\mathsf{g\text{-}iocos}}) \in \precsim_{\mathsf{g\text{-}iocos}}$. Let us consider n_2' a *maximal* element with respect to $\mathsf{g\text{-}iocos}$, i.e, for any n_i' such that $(n_1, n_i') \in \mathsf{g\text{-}iocos}$ we have $(n_i', n_2') \in \mathsf{g\text{-}iocos}$.

Next we will prove that $[n_2] \Rightarrow_\forall [n_2']$. Let us consider $n_3 \in [n_2]$. Since they belong to the same $\equiv_{\mathsf{g\text{-}iocos}}$ class, the equivalence kernel makes true $(n_2, n_3) \in \mathsf{g\text{-}iocos}$. Therefore there exists n_3' such that $n_3 \stackrel{x}{\Longrightarrow} n_3'$ and $(n_2', n_3') \in \mathsf{g\text{-}iocos}$. Again, $(n_3, n_2) \in \mathsf{g\text{-}iocos}$, there exists n_2'' such that $n_2 \stackrel{x}{\Longrightarrow} n_2''$ and $(n_3', n_2'') \in \mathsf{g\text{-}iocos}$. Since $\mathsf{g\text{-}iocos}$ is transitive we obtain $(n_2', n_2'') \in \mathsf{g\text{-}iocos}$. Since n_2' was a maximal element then $n_2' = n_2''$ and then $n_3' \in [n_2']$. □

We also need to show that $\langle N/\equiv_{\mathsf{g\text{-}iocos}}, \precsim_{\mathsf{g\text{-}iocos}}\rangle$ is maximal with respect to the points a) and b) of the GCPP. In order to do it, first let us note that any partition pair $\langle S, \precsim\rangle$ induces a natural relation in N that we will denote by $\precsim(N)$.

Definition 12. Let $\mathcal{L} = (S, I, O, \rightarrow)$ be a labelled transition system and $\mathbb{T}(\mathcal{L}) = (N, \Rightarrow)$ its transformed graph, and let $\langle S, \precsim\rangle$ be a partition pair. We define the relation $\precsim(N) \subseteq N \times N$ as $(n_1, n_2) \in \precsim(N)$ iff $([n_1]_S, [n_2]_S) \in \precsim$. □

Second, if $\langle S, \precsim\rangle$ satisfies the points a) and b) of the GCPP for the partition pair $\langle \Omega, \preccurlyeq\rangle$, then the following lemma states that $\precsim(N)$ is a $\mathsf{g\text{-}iocos}$-simulation.

Lemma 3. Let $\mathcal{L} = (S, I, O, \rightarrow)$ be a labelled transition system and $\mathbb{T}(\mathcal{L}) = (N, \Rightarrow)$ its transformed graph, and let $\langle S, \precsim\rangle$ be a refinement of $\langle \Omega, \preccurlyeq\rangle$ such that is stable with respect to $\Rightarrow$. Then $\precsim(N)$ is a $\mathsf{g\text{-}iocos}$-simulation.

Proof. Let us consider $(n_1, n_2) \in N(\precsim)$. Since $\langle S, \precsim\rangle$ is a refinement of $\langle \Omega, \preccurlyeq\rangle$, we obtain $([n_1]_\Omega, [n_2]_\Omega) \in \preccurlyeq$. Then n_1 has the form (s_1, x), and n_2 has the form (s_2, y) with $x = y$ and $\mathsf{ins}(n_2) \subseteq \mathsf{ins}(n_1)$, hence fulfilling 1 of Definition 8.

Now let us consider $x \in L$ and $n_1' \in N$ such that $n_1 \stackrel{x}{\Longrightarrow} n_1'$. By rewriting all those elements at *coarse level* in S we obtain $([n_1]_S, [n_2]_S) \in \precsim$ and $[n_1]_S \Rightarrow_\exists [n_1']_S$. Since $\langle S, \precsim\rangle$ is stable there exists $\delta \in S$ such that $[n_2]_S \Rightarrow_\forall \delta$ and $([n_1']_S, \delta) \in \precsim$. At *discrete level* that means there exists $n_2' \in \delta$ such that $n_2 \Rightarrow n_2'$. Since $\langle S, \precsim\rangle$ is a refinement of $\langle \Omega, \preccurlyeq\rangle$, we obtain $([n_1']_\Omega, [n_2']_\Omega) \in \preccurlyeq$. Since $n_1 \stackrel{x}{\Longrightarrow} n_1'$, then n_1' has the form (s_1', x) (see transformation rules in Definition 7). By the definition of Ω (Definition 11), then n_2' has also the form (s_2', x). Therefore $n_2 \stackrel{x}{\Longrightarrow} n_2'$.

□

Now Since g-iocos$_{\underline{s}}$ is the maximal g-iocos$_{\underline{s}}$-simulation, we obtain that $\precsim (N) \subseteq$ g-iocos$_{\underline{s}}$. From this fact is easy to prove the following lemma.

Lemma 4. Let $G = (N, \Rightarrow)$ be the transformed graph of an *LTS* and let $\langle S, \precsim \rangle$ be a partition pair such that $\precsim (N)$ is a g-iocos$_{\underline{s}}$-simulation. Then $\langle S, \precsim \rangle \sqsubseteq \langle N_{\equiv_{\text{g-iocos}_{\underline{s}}}}, \precsim_{\text{g-iocos}_{\underline{s}}} \rangle$.

Proof. First let us consider $\alpha \in S$ and $a, b \in \alpha$. Since $\precsim$ is reflexive we obtain $(\alpha, \alpha) \in \precsim$ and therefore $(a, b) \in \precsim (N)$ and $(b, a) \in \precsim (N)$. Now considering that $\precsim (N) \subseteq$ g-iocos$_{\underline{s}}$ we obtain $(a, b) \in \equiv_{\text{g-iocos}_{\underline{s}}}$. Therefore all elements of α are in the same equivalence class in $N/\equiv_{\text{g-iocos}_{\underline{s}}}$, so S is a refinement of $N/\equiv_{\text{g-iocos}_{\underline{s}}}$. Now let us consider $\alpha, \beta \in \precsim$, let us consider $a, b \in N$ such that $[a]_S = \alpha$ and $[b]_S = \beta$. Since $\precsim (N) \subseteq$ g-iocos$_{\underline{s}}$, we obtain $(a, b) \in$ g-iocos$_{\underline{s}}$ and $([a]_{\text{g-iocos}_{\underline{s}}}, [b]_{\text{g-iocos}_{\underline{s}}}) \in \precsim_{\text{g-iocos}_{\underline{s}}}$. Since $[a]_S \subseteq [a]_{\text{g-iocos}_{\underline{s}}}$ and $[b]_S \subseteq [b]_{\text{g-iocos}_{\underline{s}}}$, we obtain $([a]_S, [b]_S) \in \precsim_{\text{g-iocos}_{\underline{s}}} (S)$. □

Finally, we obtain the main theorem of this section.

Theorem 6. Let $\mathcal{L} = (S, I, O, \rightarrow)$ be a labelled transition system and $\mathbb{T}(\mathcal{L}) = (N, \Rightarrow)$ its transformed graph, then $\langle N_{\equiv_{\text{g-iocos}_{\underline{s}}}}, \precsim_{\text{g-iocos}_{\underline{s}}} \rangle$ is the solution of the GCPP for the partition pair $\langle \Omega, \preccurlyeq \rangle$.

Proof. The points a) and b) of the GCPP follows from Lemmas 1, and 2. The maximality follows from Lemmas 3 and 4. □

5 Conclusions and Future Work

In this paper we have defined an online algorithm that allows to check if a certain implementation iocos$_{\underline{s}}$-conforms a given specification by interacting with it and without computing any a priori set of tests. Under fairly standard hypothesis —even weaker than in other models— we prove the algorithm to be sound and complete.

We plan to introduce test selection criteria and coverage in the iocos$_{\underline{s}}$-theory. This technique is mainly used in offline testing but we think that implementations of the online algorithm we have presented can also benefit from them to further restrict the search tree.

Since iocos$_{\underline{s}}$ is a branching semantics, it is essential in the online algorithm to make use of the *copy* clause (see for instance [1] for a more elaborated discussion). While this copy capability will exclude some systems to being tested with iocos$_{\underline{s}}$ —essentially unique systems that cannot be replicated— for a vast number of applications, for instance to check software products, it could definitely be a feasible operation. The implementation of our online algorithm is currently under development using cluster computing techniques.

In this paper we have also proved that the conformance relation iocos$_{\underline{s}}$, in spite of its particularities and asymmetry with input and output actions, can

be solved with the GCPP. Actually, we are adapting the mCRL2 toolset [7], that implements one of the best solution to the GCPP [28], to compute iocos and the minimised LTS for a give specification. This technique, frequently used in model checking, allows to reduce the size of the models and therefore the state explosion in any further testing process. Moreover, we plan to investigate the logic preservation properties of iocos that would allow to perform model checking of the intended model specifications and integrate MBT and model checking in the same theory.

Finally, once the ground model has proved to be useful, we plan to improve the expressiveness introducing a syntax language and integrating internal τ actions.

References

1. Samson Abramsky. Observational equivalence as a testing equivalence. *Theoretical Computer Science*, 53(3):225–241, 1987.
2. Luca Aceto, David de Frutos Escrig, Carlos Gregorio-Rodríguez, and Anna Ingolfsdottir. Axiomatizing weak simulation semantics over BCCSP. *Theoretical Computer Science*, To appear, 2013. Available online since 26 March 2013.
3. Axel Belinfante. Jtorx: A tool for on-line model-driven test derivation and execution. In Javier Esparza and Rupak Majumdar, editors, *TACAS*, volume 6015 of *LNCS*, pages 266–270. Springer, 2010.
4. Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, 1995.
5. Peter E. Bulychev, Thomas Chatain, Alexandre David, and Kim Guldstrand Larsen. Efficient on-the-fly algorithm for checking alternating timed simulation. In Joël Ouaknine and Frits W. Vaandrager, editors, *FORMATS*, volume 5813 of *LNCS*, pages 73–87. Springer, 2009.
6. Doron Bustan and Orna Grumberg. Simulation-based minimization. *ACM Trans. Comput. Logic*, 4(2):181–206, April 2003.
7. Sjoerd Cranen, JanFriso Groote, JeroenJ.A. Keiren, FrankP.M. Stappers, ErikP. Vink, Wieger Wesselink, and TimA.C. Willemse. An overview of the mcrl2 toolset and its recent advances. In Nir Piterman and ScottA. Smolka, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 7795 of *LNCS*, pages 199–213. Springer Berlin Heidelberg, 2013.
8. David de Frutos-Escrig, Carlos Gregorio-Rodríguez, Miguel Palomino, and David Romero-Hernández. Unifying the linear time-branching time spectrum of process semantics. *Logical Methods in Computer Science*, 9(2:11):1–74, 2013.
9. René G. de Vries and Jan Tretmans. On-the-fly conformance testing using spin. *STTT*, 2(4):382–393, 2000.
10. Ignacio Fábregas, David de Frutos-Escrig, and Miguel Palomino. Logics for contravariant simulations. In John Hatcliff and Elena Zucca, editors, *FMOODS/FORTE*, volume 6117 of *LNCS*, pages 224–231. Springer, 2010.
11. Raffaella Gentilini, Carla Piazza, and Alberto Policriti. From bisimulation to simulation: Coarsest partition problems. *J. Autom. Reasoning*, 31(1):73–103, 2003.
12. Carlos Gregorio-Rodríguez, Luis Llana, and Rafael Martínez-Torres. Input-output conformance simulation (iocos) for model based testing. In Dirk Beyer and Michele Boreale, editors, *FMOODS/FORTE*, volume 7892 of *LNCS*, pages 114–129. Springer, 2013.

13. Paris C. Kanellakis and Scott A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, 1990.
14. Joost-Pieter Katoen, Tim Kemna, Ivan Zapreev, and DavidN. Jansen. Bisimulation minimisation mostly speeds up probabilistic model checking. In Orna Grumberg and Michael Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *Lecture Notes in Computer Science*, pages 87–101. Springer Berlin Heidelberg, 2007.
15. Kim Guldstrand Larsen, Marius Mikucionis, and Brian Nielsen. Online testing of real-time systems using UPPAAL. In J. Grabowski and B. Nielsen, editors, *FATES*, volume 3395 of *LNCS*, pages 79–94. Springer, 2004.
16. Luis Llana and Rafael Martínez-Torres. Ioco as a simulation. In *3rd Workshop on Formal Methos in the Development of Software, WS-FMDS 2013*, volume to appear of *LNCS*, 2013.
17. Gerald Lüttgen and Walter Vogler. Ready simulation for concurrency: It's logical! *Information and Computation*, 208(7):845–867, 2010.
18. Robin Milner. An algebraic definition of simulation between programs. In *Proceedings 2nd Joint Conference on Artificial Intelligence*, pages 481–489. BCS, 1971. Report No. CS-205, Computer Science Department, Stanford University.
19. Robin Milner. *A Calculus of Communicating Systems*. LNCS 92. Springer, 1980.
20. Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.
21. Pablo Rabanal, Ismael Rodríguez, and Fernando Rubio. Testing restorable systems: formal definition and heuristic solution based on river formation dynamics. *Formal Aspects of Computing*, 25(5):743–768, 2013.
22. Francesco Ranzato. A more efficient simulation algorithm on kripke structures. In Krishnendu Chatterjee and Jiri Sgall, editors, *MFCS*, volume 8087 of *LNCS*, pages 753–764. Springer, 2013.
23. Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond Strong, editors, *STOC*, pages 1–9. ACM, 1973.
24. Gerjan Stokkink, Mark Timmer, and Mariëlle Stoelinga. Talking quiescence: a rigorous theory that supports parallel composition, action hiding and determinisation. In Alexander K. Petrenko and Holger Schlingloff, editors, *MBT*, volume 80 of *EPTCS*, pages 73–87, 2012.
25. Jan Tretmans. Model based testing with labelled transition systems. In R. M. Hierons, J. P. Bowen, and M. Harman, editors, *Formal Methods and Testing*, volume 4949 of *Lecture Notes in Computer Science*, pages 1–38. Springer, 2008.
26. Jan Tretmans and Ed Brinksma. Torx: Automated model-based testing. In A. Hartman and K. Dussa-Ziegler, editors, *First European Conference on Model-Driven Software Engineering*, pages 31–43, December 2003.
27. Rob J. van Glabbeek. *Handbook of Process Algebra*, chapter The Linear Time – Branching Time Spectrum I: The Semantics of Concrete, Sequential Processes, pages 3–99. Elsevier, 2001.
28. Rob J. van Glabbeek and Bas Ploeger. Correcting a space-efficient simulation algorithm. In Aarti Gupta and Sharad Malik, editors, *CAV*, volume 5123 of *LNCS*, pages 517–529. Springer, 2008.
29. Margus Veanes, Colin Campbell, Wolfram Schulte, and Nikolai Tillmann. Online testing with model programs. In Michel Wermelinger and Harald Gall, editors, *ESEC/SIGSOFT FSE*, pages 273–282. ACM, 2005.