Bounded reordering in the distributed test architecture

Robert M. Hierons, Mercedes G. Merayo and Manuel Núñez

***Abstract*—In the distributed test architecture, the system under test interacts with its environment at multiple physically distributed ports and the local testers at these ports do not synchronise their actions. This presents many challenges and, in particular, apparently incorrect behaviours can be the consequence of an erroneous assumption about the exact order in which actions were performed at different ports. In previous work, we defined a conformance relation for the distributed test architecture. Essentially, the system under test is faulty if we observe a trace σ such that no admissible reordering of the actions in σ could have been produced by the specification. However, this notion can be weak if the compared traces might be *too* different. This paper introduces conformance relations where, for a given metric, a reordering is only considered if the distance between the two traces is at most a certain bound k. We introduce two different metrics and provide algorithms to construct finite automata accepting these *close*, with respect to each metric, sequences. We also study the computational complexity of the two main problems associated with the new framework: deciding whether a trace is accepted by the new automaton and deciding whether one system conforms to a specification with respect to the new conformance relation.**

I. Introduction

Software testing [2], [37] is the de facto standard technique to validate the correctness of software systems. Testing consists of providing inputs to a system, observing its reactions and deciding whether the resultant behaviour is valid. Traditionally, testing activities have mainly been manual and, as a result, costly and error prone. This situation is changing; it has been shown that it is possible to formalise the classically informal testing techniques [15] and that the combination of formal methods and testing facilitates test automation.

Normally, testing is seen as a process where a single tester interacts with the system under test (SUT). However, sometimes a system interacts with its environment at physically distributed locations (ports) and one places a separate tester at each port. As a result, a local tester only observes the events at its port and it might not be possible to reconstruct the global sequence of events that occurred. For example, if the tester at port 1 observes output $!o_1$ and the tester at port 2 observes $!o_2$ then the global trace might have been $!o_1!o_2$ or $!o_2!o_1$. This complicates testing and leads to the following results when there is a finite model M that acts as the specification: the problem of deciding whether an observation is allowed by M is NP-complete [19] and the problem of deciding whether a finite model N conforms to M (all observations that can be made of N are allowed by M) is undecidable [18]. In contrast, when testing is not distributed the first problem can be decided in low-order polynomial time and the second problem is decidable and can be solved in polynomial time if N and M are deterministic finite automata (or an observable finite state machine or input-output transition system).

As noted above, in distributed testing the separate testers cannot synchronise and so we cannot know the relative order of events at different ports. This leads to a notion of observational equivalence in which traces σ_1 and σ_2 are equivalent if and only if, for every port p we have that σ_1 and σ_2 have the same projections at p. Under this, $!o_1!o_2$ and $!o_2!o_1$ are observationally equivalent as noted above. Indeed, for every natural number m we have that $!o_1^m!o_2$ and $!o_2!o_1^m$ are observationally equivalent. This seems sensible for small values of m but, assuming that processes are non-Zeno, it will cease to make sense if m becomes sufficiently large. For example, if each event takes at least one second and $m = 100,000$ then in the trace $!o_1^m!o_2$ we have that the occurrence of $!o_2$ and the first occurrence of $!o_1$ are separated by more than one day; one might then expect to be able to distinguish between $!o_1^m!o_2$ and $!o_2!o_1^m$. We might therefore want to strengthen our notion of observational equivalence, or similarity, in a way that leads to these traces not being equivalent. This paper explores a method that achieves this based on defining the distance between two traces. As usual, for two traces σ_1 and σ_2 to be observationally equivalent we require them to have the same set of projections at the ports. However, we will also require that the *distance* between σ_1 and σ_2 is at most k for a notion of distance. We consider two ways of defining the distance between traces σ_1 and σ_2 that have the same sets of projections.

1) The distance between σ_1 and σ_2 is the length of the shortest sequence of swaps of adjacent events, that occur at different ports, that can transform σ_1 into σ_2.
2) For every event a, we measure the difference between the location (index) of a in σ_1 and σ_2 and the distance between σ_1 and σ_2 is the maximum such value (over events in σ_1 and σ_2).

We explore these approaches and define the corresponding conformance relations. We also consider two decision problems for these new conformance relations: the problem of deciding whether an observation is allowed by specification M (for finite M) and the problem of deciding whether N is

R. M. Hierons is with Department of Computer Science, Brunel University London, Uxbridge, Middlesex, UB8 3PH UK. e-mail: rob.hierons@brunel.ac.uk

M. G. Merayo and M. Núñez are with Dept. Sistemas Informáticos y Computación, Universidad Complutense de Madrid, Spain. e-mail: mgmerayo@fdi.ucm.es, mn@sip.ucm.es

Research partially supported by the projects DArDOS (TIN2015-65845-C3-1-R (MINECO/FEDER)) and SICOMORo-CM (S2013/ICE-3006).

a correct implementation of M (finite N and M). We restrict attention to finite models since non-trivial decision problems with infinite models are undecidable.

Finally, we have included a glossary (see Figures 1 and 2) where, in particular, all the conformance relations presented in the paper are informally described.

The rest of the paper is organised as follows. In Section II we present related work. In Section III we provide preliminary material. Section IV describes our first approach, in which we bound the number of swaps of adjacent events (at different ports) that can occur. Section V then describes our second approach, in which we bound the distance between corresponding events. Both of these sections provide algorithms and complexity results. Finally, we draw conclusions in Section VI.

II. Related work

In this section we briefly review related work. First, let us remark that formal approaches to testing is an active research area [3], [8], [23]. In addition to having well established theories and methodologies, there are many tools that allow potential users to apply techniques developed in this area [35], [44].

There has been significant recent interest in distributed testing when testing from a formal model written as a finite state machine [20], [21], [29], input-output transition system [25], [26], partial-order automaton [4], [16], or Petri Net [39], [41]. Previous work on distributed testing has considered a notion of observational equivalence in which $\sigma_1 \sim \sigma_2$, for traces σ_1 and σ_2, if σ_1 and σ_2 have the same projections at the ports [4], [7], [12], [13], [16], [20], [21], [25], [26], [29], [31], [32], [34], [38], [39], [40], [41], [42], [43], [45], [46]. An alternative way of describing this is to say that events x and y commute (are independent) if they occur at different ports. As a result, previous conformance relations for distributed testing relate strongly to the notion of a partial-commutation in which certain events commute (see, for example, [36]). However, the conformance relation given in this paper is based on both observational equivalence and a distance metric and cannot be expressed in terms of a partial commutation.

A second line of work aims to use standard conformance relations, devised for the case where testing is not distributed, and to do so by ensuring that the local testers are sufficiently synchronised. Synchronisation is typically achieved through the exchange of messages, called coordination messages. Researchers have explored architectures that can be used to synchronise local testers (see, for example, [7], [42]), approaches that enrich a test generation technique with synchronisation messages (see, for example, [9], [28]), and approaches that minimise the number of synchronisation messages added to a test sequence (see, for example, [5]).

Interestingly, the notion of observational equivalence ($\sigma_1 \sim \sigma_2$) is very similar to sequential consistency, a correctness condition for concurrent objects [33]. A number of other correctness conditions have been defined for concurrent objects and there is a small amount of work that looks at the complexity of decision problems related to those we consider in this paper [1], [11]. There has also been some work that describes quantitative notions of correctness for concurrent objects and this uses a notion of the distance between two traces. In fact, the approach we describe in this paper could be seen as being similar to the notion of k-relaxation [17]. However, there are some important differences between our work and this one. First, we are concerned with testing and so there is a need to differentiate between input and output (they play very different roles in testing). Second, this approach [17] did not explore the complexity of associated decision problems.

III. Preliminaries

In this section we first introduce the main notions that we use throughout this paper. Next, we present the concepts that we use to define the conformance relations that we will study.

A. Basic notation

Given a set A, we let A^* denote the set of finite sequences of elements of A; $\epsilon \in A^*$ denotes the empty sequence. A^k denotes the set of sequences with length $k \geq 1$. Given a sequence $\sigma \in A^*$, we have that $|\sigma|$ denotes its length. Given a sequence $\sigma \in A^*$ and $a \in A$, we have that σa denotes the sequence σ followed by a and $a\sigma$ denotes the sequence σ preceded by a.

Throughout this paper we let I be the set of inputs and O the set of outputs. We assume that the SUT, and its specification, has a set of ports $\mathcal{P}orts = \{1, \ldots, m\}$. A port is typically a location at which the SUT interacts with its environment and in distributed testing we will place a separate local tester at each port. We assume that the sets I and O are partitioned into sets $I_1, \ldots, I_m$ and $O_1, \ldots, O_m$ such that for all $p \in \mathcal{P}orts$, I_p and O_p are the sets of inputs and outputs at port p, respectively. We assume that $I_1, \ldots I_m, O_1, \ldots, O_m$ are pairwise disjoint. In order to distinguish between input and output we usually precede the name of an input by ? and precede the name of an output by !. If we label an action with a subindex, then the value indicates the port where the action is performed. For example, we might use $!o_1$ for an output at port 1 and so we have that $!o_1 \in O_1$. Note that although we will typically use $!o_i$ to denote an output at port i, a set of inputs and outputs (an I_i or O_i) can contain more than one element. If we label an action with a superindex, then the value indicates the position of that action in a sequence of actions.

Since the sets $I_1, \ldots I_m, O_1, \ldots, O_m$ are pairwise disjoint, it might appear that we cannot, for example, model a system that can produce the same output at two or more ports. However, these sets can be seen as sets of events, such as the output of $!a$ at port p. Thus, for example, we can model a system that can produce a particular output $!a$ at ports p and q by introducing two associated events: an event $!a_p \in O_p$ in which $!a$ is output at port p and an event $!a_q \in O_q$ in which a is output at port q.

As is usual in work on testing from an input output transition system, we will allow the observation of *quiescence*: the situation in which the SUT or specification cannot produce output without first receiving input. Typically, quiescence is observed through a sufficiently long timeout. In practice, the time used

Processes, actions and traces	
Concept	*Explanation*
IOTS Def. 1	Input Output Transition System. Labelled transition systems with a distinction between input actions (preceded by ? and grouped in a set I), output actions (preceded by ! and grouped in a set O) and an action to denote quiescence (denoted by δ). The set of actions, denoted by $\mathcal{A}ct$, is equal to $I \cup O \cup \{\delta\}$. Inputs and outputs are usually labelled with a number denoting the port where they are performed.
Trace Def. 3	Sequence of actions that a process can perform.

Relations and distances between traces	
Concept	*Explanation*
$\sim$ Def. 4	Original relation to compare traces. Two traces are related if all their local projections are equal. For example, $!o_1!o_2?i_1?i_2 \sim !o_2?i_2!o_1?i_1$.
$\sim^1$ Def. 8	Two traces are related if they are related by $\sim$ and one can be transformed into the other with one swap. For example, $!o_1!o_2?i_1?i_2 \sim^1 !o_1?i_1!o_2?i_2$ but $!o_1!o_2?i_1?i_2 \sim^1 !o_2?i_2!o_1?i_1$ does not hold (we need at least three swaps).
d^1 Def. 9	First distance between traces. We have $d^1(\sigma, \sigma') = k$ if the minimum number of swaps that we need to transform σ into σ' is equal to k.
d^2 Def. 15	Second distance between traces. We have $d^2(\sigma, \sigma') = k$ if the maximum difference in the position of the same event in each of the traces is equal to k.

Fig. 1: Glossary of concepts (1/2)

in the timeout depends on properties of the SUT and is decided by the tester. We will assume that it is possible for the testers at all ports to observe quiescence; the observation of quiescence is global. In practice the global observation of quiescence might require the use of a longer timeout period, with that period potentially depending on properties of the SUT and also properties of the test case being used. For example, if we have a bound on the time between events (inputs and outputs) for the SUT (similar to the bound normally required to observe quiescence) then the timeout period used might depend on this bound and how many consecutive events can occur without a local tester making an observation (in the current test case). As has previously been noted, there is potential to use different timeouts for the different ports [6].

B. Input output transition systems

An input output transition system is a labelled transition system in which we distinguish between input and output. We use this formalism to define processes. Unusually, we will use the notion of a final state; usually all the states are final and all states of a specification will be final. We will see, in Sections IV and V, how we will take advantage of the distinction between a state that is final and one that is not in defining algorithms since this will allow us to define processes in which the set of corresponding traces is not prefix closed.

Definition 1 *An* input output transition system *(*`IOTS`*) is defined by a tuple $M = (Q, Q_F, I, O, T, q_{in})$ in which Q is a countable set of states, $Q_F \subseteq Q$ is the set of final states, $q_{in} \in Q$ is the initial state (we assume $q_{in} \in Q_F$), I is a countable set of inputs, O is a countable set of outputs, and $T \subseteq Q \times (I \cup O) \times Q$ is the transition relation. A transition $(q, a, q') \in T$ means that from state q it is possible to move to state q' with action $a \in I \cup O$.*

We say that a state $q \in Q$ is quiescent *if from q it is not possible to take a transition whose action is an output without first receiving an input. We extend T to T_δ by adding transition (q, δ, q) for each quiescent state q. We say that M is* input-enabled *if for all $q \in Q_F$ and $?i \in I$ there is some $q' \in Q$ such that $(q, ?i, q') \in T$.*

We let $\texttt{IOTS}(I, O, \mathcal{P}orts)$ denote the set of `IOTS`*s with input set I, output set O and port set $\mathcal{P}orts$.*

We also define some supporting notation that will be used throughout the paper.

Definition 2 *Given sets of inputs and outputs, I and O respectively, we let $\mathcal{A}ct$ denote the set of actions, that is, $\mathcal{A}ct = I \cup O \cup \{\delta\}$. Given port $p \in \mathcal{P}orts$, $\mathcal{A}ct_p$ denotes the set of observations that can be made at p, that is, $\mathcal{A}ct_p = I_p \cup O_p \cup \{\delta\}$. We define the function $port : I \cup O \longrightarrow \mathcal{P}orts$ such that $port(a) = p$ if $a \in \mathcal{A}ct_p$; this is a function since the I_i and O_i are pairwise disjoint. Abusing the notation, we will assume that $port(\delta) = p$ holds for all $p \in \mathcal{P}orts$.*

We will not require the notion of a final state when we consider the SUT or the specification; here, all states will be final states. However, later we will construct `IOTS`s in which it makes no sense to observe a particular trace and so we will then require the notion of a final state. Let us suppose, for example, that we consider an `IOTS` M such that M can produce trace $!o_1!o_2$ but it cannot produce trace $!o_2$. Further, let us suppose that we wish to construct a new `IOTS` M' whose set of traces is the union of the following two sets: the set of traces of M; and the set of traces that differ from traces of M by at most swapping two adjacent events that are at different ports. Then M' must have the trace $!o_2!o_1$, since this differs from the trace $!o_1!o_2$ by swapping $!o_1$ and $!o_2$. However, M' should not have the trace $!o_2$ and so we must have that M' has

Conformance relations and related results	
Concept	*Explanation*
N **dioco** M Def. 5	Conformance relation in the distributed architecture [27]. An implementation N should not show behaviours that are not planned in a specification M, with the exception of the order between events performed at different ports.
N $\mathbf{dioco}^k_{\mathbf{sw}}$ M Def. 11	Conformance relation associated with the first distance. The behaviours of the implementation N can be transformed into behaviours of the specification M in a maximum of k swaps.
Proposition 2	Alternative characterisation of $\mathbf{dioco}^k_{\mathbf{sw}}$.
Theorem 1	Automaton ($\mathcal{M}_a(M,k)$) recognising the alternative characterisation ($L^1_k(M)$) of $\mathbf{dioco}^k_{\mathbf{sw}}$.
Theorems 2 & 3	Corresponding correctness problem, and its bounded version, for d^1.
Theorems 4 & 5	Membership problem, and its bounded version, for d^1.
N $\mathbf{dioco}^k_{\mathbf{dis}}$ M Def. 18	Conformance relation associated with the second distance. For each behaviour of the implementation N there exists a behaviour of the specification M such that the maximum difference in the position of the events is equal to k.
Proposition 5	Alternative characterisation of $\mathbf{dioco}^k_{\mathbf{dis}}$.
Theorem 6	Automaton ($\mathcal{M}_b(M,k)$) recognising the alternative characterisation ($L^2_k(M)$) of $\mathbf{dioco}^k_{\mathbf{dis}}$.
Theorems 7 & 8	Corresponding correctness problem, and its bounded version, for d^2.
Theorems 9, 10 & 11	Membership problem, and its bounded versions, for d^2.

Fig. 2: Glossary of concepts (2/2)

trace $!o_2!o_1$ but not $!o_2$. Thus, the set of traces of M' is not prefix closed. We cannot define such an IOTS if we do not have the notion of final state; essentially, we will require that $!o_2$ takes M' to a state q that is not a final state but from q we can reach a final state through a transition with output $!o_1$.

A process can be identified with its initial state and we can define a process corresponding to a state q of M by making q the initial state. Thus, we use states and processes and their notation interchangeably. By default, all the states of an IOTS are final (as we already said, non-final states will be used in the construction of some auxiliary processes). An IOTS can be represented by a diagram. Figure 3 shows an example of the graphical representation of an IOTS. Nodes represent states of the IOTS and transitions are represented by arcs between the nodes. We use a double circle to denote the initial state, q_0, while dotted/non-dotted circles denote non-final/final states. In this case, all the states are final except q_2. In this IOTS, for example, if input $?i_1$ is received when the IOTS is in state q_0 then the IOTS moves to state q_1; a state from which it can produce output $!o_2$ and move to q_3. A sequence of actions labelling a transition denotes different transitions, one per action. For example, we have two transitions departing and reaching q_1 labelled, respectively, by $?i_1$ and $?i_2$.

In this paper, whenever we compare two IOTSs we will assume that they have the same set of ports and the same set of actions $\mathcal{A}ct_p$ for all $p \in \mathcal{P}orts$. Moreover, as usual, we assume that implementations are input-enabled[1]. We also require that specifications are input-enabled since this assumption simplifies the analysis. However, it is possible to remove this restriction in our framework [22], [26]. As usual, for work related to distributed testing from IOTSs, we require that processes are not output-divergent: there cannot be a state from which there is an infinite sequence of consecutive transitions whose labels are all outputs. A *trace* is a sequence of observable actions that can be performed from the initial state of a process and reaches a final state.

[1]If an input cannot be applied in some state of the SUT, then we can assume that there is a response to the input that reports that this input is blocked.

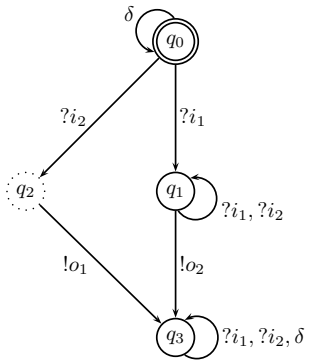


Fig. 3: A diagrammatic representation of an IOTS

Definition 3 *Let $M = (Q, Q_F, I, O, T, q_{in})$ be an IOTS. We use the following notation.*

1) *If $(q, a, q') \in T_\delta$, for $a \in \mathcal{A}ct$, then we write $q \xrightarrow{a} q'$.*
2) *Let $\sigma = a^1 \dots a^k \in \mathcal{A}ct^*$ be a finite sequence of actions. We write $q \stackrel{\sigma}{\Longrightarrow} q'$ if there exist $q_0, \dots, q_k \in Q$ such that $q = q_0$, $q' = q_k$ and for all $1 \leq i \leq k$ we have that $q_{i-1} \xrightarrow{a^i} q_i$.*
3) *We write $q \stackrel{\sigma}{\Longrightarrow}$ if there exists $q' \in Q$ such that $q \stackrel{\sigma}{\Longrightarrow} q'$.*

4) *We write* $M \stackrel{\sigma}{\Longrightarrow}$ *if* $q_{in} \stackrel{\sigma}{\Longrightarrow}$.
5) *Let* $\rho = (q_0, a^1, q_1)(q_1, a^2, q_2)\dots(q_{k-1}, a^k, q_k)$ *be a sequence of consecutive transitions. We say that the label of* ρ*, denoted by* $label(\rho)$*, is the sequence of actions associated with the transitions, that is,* $label(\rho) = a^1 \dots a^k$.

We define the language of M *as* $L(M) = \{\sigma | q_{in} \stackrel{\sigma}{\Longrightarrow} q \wedge q \in Q_F\}$. *We say that* $\sigma \in L(M)$ *is a* trace *of* M. *We let* $L_\delta(M) = \{\sigma | q_{in} \stackrel{\sigma\delta}{\Longrightarrow} q \wedge q \in Q_F\}$*; the set of traces of* M *that can take* M *to a quiescent final state.*

Note that for every state q, we have that $q \stackrel{\epsilon}{\Longrightarrow} q$ holds. Therefore, $\epsilon \in L(M)$ for every process M.

Previous conformance relations, devised for distributed testing, are based on an equivalence relation $\sim$ on traces. Essentially, the relation $\sim$ reflects the fact that in distributed testing each tester observes only the events at its port and this corresponds to a projection of the global trace that occurred.

Definition 4 *Let* $p \in \mathcal{P}orts$ *and* $\sigma \in \mathcal{A}ct^*$ *be a sequence of actions. We let* $\pi_p(\sigma)$ *denote the projection of* σ *onto port* p *and* $\pi_p(\sigma)$ *is called a* local trace. *Formally,*

$$\pi_p(\sigma) = \begin{cases} \epsilon & \text{if } \sigma = \epsilon \\ a\pi_p(\sigma') & \text{if } \sigma = a\sigma' \wedge a \in \mathcal{A}ct_p \\ \pi_p(\sigma') & \text{if } \sigma = a\sigma' \wedge a \in \mathcal{A}ct \setminus \mathcal{A}ct_p \end{cases}$$

Given $\sigma, \sigma' \in \mathcal{A}ct^*$ *we write* $\sigma \sim \sigma'$ *if* σ *and* σ' *cannot be distinguished when making local observations, that is, for all* $p \in \mathcal{P}orts$ *we have that* $\pi_p(\sigma) = \pi_p(\sigma')$. *Obviously,* $\sim$ *is an equivalence relation and we denote by* $[\sigma]$ *the equivalence class of* σ*, that is,* $[\sigma] = \{\sigma' \in \mathcal{A}ct^* | \sigma \sim \sigma'\}$.

Next we define the conformance relation **dioco**. The version that we use in this paper is taken from our previous work [27] and it is slightly different from the original definition [25], [26] because the latter used tuples of outputs, one for each port, instead of single outputs like we consider in this paper. We chose to use this definition in order to simplify the analysis since, for example, there is no need to separate outputs that appear together in a tuple.

Definition 5 *Let* $M, N \in \mathtt{IOTS}(I, O, \mathcal{P}orts)$. *We write* N **dioco** M *if and only if for every quiescent trace* $\sigma\delta \in L(N)$*, there exists a trace* $\sigma' \in L(M)$ *such that* $\sigma' \sim \sigma\delta$.

The main aim of this paper is to define conformance relations following the pattern of the previous one. The difference is that we will place additional restrictions on $\sigma' \sim \sigma\delta$ in the above definition.

It is worth briefly commenting on why the definition of **dioco** only concerns quiescent traces of N. To see this, consider what might happen if we did not include this restriction in Definition 5 and consider processes M and M', with outputs $!o_1$ and $!o_2$ at different ports, such that:

1) M can do $!o_1!o_2$ and then is in a state from which all future events must be inputs or quiescence
2) M' can do $!o_2!o_1$ and then is in a state from which all future events must be inputs or quiescence.

We have that M can do $!o_1$ while M' cannot. Therefore, if we use non-quiescent traces when comparing these processes then we conclude that they are not equivalent. However, in a distributed environment we have that $!o_1!o_2$ and $!o_2!o_1$ are equivalent since in each case the tester at port 1 will observe $!o_1$ and the tester at port 2 will observe output $!o_2$. Thus, we cannot distinguish between these two processes if we do not have additional information. However, sometimes we can partially overcome this problem if we can label actions with the time when they were performed [27], [40] but local clocks, one per port, might not be available and it is very likely that they are not synchronised. The definition of **dioco** solves this problem by using quiescent states to combine the traces observed at each port and reach a verdict. This can be done because the testers can choose to stop testing in a quiescent state and, in addition, we assume that quiescence can be observed by all of the testers. The use of distributed testers also leads to the requirement for us to compare the set of local observations made with the global traces from the specification; if we make observations in non-quiescent states then we cannot know that the observed local traces are all projections of the same global trace of the SUT and we can then appear to be able to distinguish processes that are observationally equivalent. This is the problem we observed above.

In Definition 4, note that $\delta \in \mathcal{A}ct_p$ for all port $p \in \mathcal{P}orts$. This fact will be relevant when we define *valid* swaps because swaps must involve actions in two different ports and δ belongs to all the ports.

Throughout this paper we will need to consider an action appearing in a trace together with its position in the trace. In order to distinguish between different occurrences of an action in one trace we will need to decorate actions with further information.

Definition 6 *Let* $\sigma = a^1 \dots a^n \in \mathcal{A}ct^*$ *be a sequence of actions. We let* $E(\sigma)$ *denote the* set of events of σ*, where* $e = (a^i, k)$ *belongs to* $E(\sigma)$ *if and only if there are exactly* $k-1$ *occurrences of* a^i *in* $a^1 \dots a^{i-1}$.

We introduce notation that allows us to find the position of an event in a trace and map a trace to a sequence of events.

Definition 7 *Let* $\sigma = a^1 \dots a^n \in \mathcal{A}ct^*$ *be a sequence of actions. We define the function* $pos_\sigma : E(\sigma) \longrightarrow \boldsymbol{N}$ *such that for all* $e = (a, k) \in E(\sigma)$ *we have that* $pos_\sigma(e) = i$ *if* $a = a^i$ *and there are exactly* $k-1$ *occurrences of* a *in* $a^1 \dots a^{i-1}$. *We define the annotated sequence of* σ*, denoted by* $\widetilde{\sigma}$*, as the sequence* $(a^1, k_1) \dots (a^n, k_n)$.

Usually, we will decorate an event with its position. For example, e^i denotes that $pos_\sigma(e) = i$.

Example 1 *Consider the trace* $\sigma = ?i_1!o_1?i_2!o_1!o_2$*, where the index denotes the port at which the action is performed. The corresponding set of events is*

$$E(\sigma) = \{(?i_1, 1), (!o_1, 1), (?i_2, 1), (!o_1, 2), (!o_2, 1)\}$$

where, for example, $pos_\sigma((?i_1, 1)) = 1$ *and* $pos_\sigma((!o_1, 1)) = 2$*. We will refer to event* $(?i_2, 1)$ *by* e^3.

It is trivial to prove that observationally equivalent traces produce the same sets of events. We will use this result when we define the distance between traces.

Lemma 1 *Let* $\sigma, \sigma' \in \mathcal{A}ct^*$ *be sequences of actions such that* $\sigma \sim \sigma'$*. We have* $E(\sigma) = E(\sigma')$.

IV. Placing a bound on the number of swaps required

In this section we provide our first notion of distance and the corresponding conformance relation that is based on this metric. The essential idea is that the local testers might have local clocks that are not exactly synchronised but that are close to agreeing. In such situations, the local testers might timestamp observations and we can then order the observations based on these timestamps to form a trace σ_1. Since local clocks provide the timestamps, σ_1 need not be the same as the trace σ_2 that actually occurred. However, since the local clocks are close to being synchronised, we should expect σ_1 and σ_2 to be 'similar'. We capture this notion of similarity through the use of a metric on traces without explicitly representing time. Since one can always use local projections to distinguish between traces that are not related under $\sim$, it will only be necessary to define the distance between traces σ_1 and σ_2 if $\sigma_1 \sim \sigma_2$.

The metric we define will say that the distance between traces σ_1 and σ_2, with $\sigma_1 \sim \sigma_2$, is the number of swaps of adjacent events required to transform σ_1 into σ_2. First, we introduce an equivalence relation $\sim^1$ that only allows one pair of events to be swapped and requires that these events are at different ports.

Definition 8 *Let* $\sigma, \sigma' \in \mathcal{A}ct^*$ *be sequences of actions. We write* $\sigma \sim^1 \sigma'$ *if there exist* $p, q \in \mathcal{P}orts$*, with* $p \neq q$*,* $\sigma_1, \sigma_2 \in \mathcal{A}ct^*$*,* $a \in \mathcal{A}ct_p \backslash \{\delta\}$ *and* $b \in \mathcal{A}ct_q \backslash \{\delta\}$ *such that* $\sigma = \sigma_1 ab \sigma_2$ *and* $\sigma' = \sigma_1 ba \sigma_2$.

Note that in the previous definition, we have that a and b must be different from δ because we require that they are performed at different ports. The previous relation induces our distance in the expected way: we simply count the number of swaps.

Definition 9 *Let* $\sigma, \sigma' \in \mathcal{A}ct^*$ *be sequences of actions such that* $\sigma \sim \sigma'$*. We define the distance between these two traces, denoted by* $d^1(\sigma, \sigma')$*, to be the smallest integer* k *such that there exist* $\sigma_0, \sigma_1, \ldots, \sigma_k$ *where* $\sigma = \sigma_0$*,* $\sigma' = \sigma_k$*, and for all* $0 \leq i < k$ *we have that* $\sigma_i \sim^1 \sigma_{i+1}$.

In a minor abuse of notation, we let d^1 be defined over each equivalence class $[\sigma]$, that is, it induces a function $d^1_\sigma : [\sigma] \times [\sigma] \longrightarrow \mathbb{N}$ in the obvious way. We will usually remove the σ index.

Example 2 *Consider the traces* $\sigma_1 = ?i_1!o_1?i_2!o_1!o_2$ *and* $\sigma_2 = ?i_2!o_2?i_1!o_1!o_1$*, where the index denotes the port at which the action is performed. Both traces have the same associated set of events (see Example 1). The following is one of the shortest sequences of swaps to transform* σ_1 *into* σ_2

$$\begin{aligned}\sigma_1 = ?i_1!o_1?i_2!o_1!o_2 &\sim^1 ?i_1?i_2!o_1!o_1!o_2 \sim^1 ?i_2?i_1!o_1!o_1!o_2\\ &\sim^1 ?i_2?i_1!o_1!o_2!o_1 \sim^1 ?i_2?i_1!o_2!o_1!o_1\\ &\sim^1 ?i_2!o_2?i_1!o_1!o_1 = \sigma_2\end{aligned}$$

Therefore, we have $d^1(\sigma_1, \sigma_2) = 5$.

Note that $\sim^1$ is simply an auxiliary notion: we still need to work on classes induced by $\sim$. Next we show that the previously defined notion induces a metric space on each equivalence class with respect to $\sim$. A metric space is a set in which a notion of *distance*, between the elements of the set, has been defined. A distance must fulfil certain properties: distances must be positive (equal to zero only if we compute the distance from one element to itself), symmetric (that is, the distance from element a to element b is equal to the distance from element b to element a) and satisfy the *triangle inequality*. More formally, if X is the space then the function $f : X \times X \longrightarrow \mathbf{R}$ is a metric if the following properties hold:

1) For all $x, y \in X$ we have that $f(x, y) \geq 0$;
2) For all $x, y \in X$ we have that $f(x, y) = 0$ if and only if $x = y$;
3) For all $x, y \in X$ we have that $f(x, y) = f(y, x)$; and
4) For all $x, y, z \in X$, $f(x, z) \leq f(x, y) + f(y, z)$.

In our case, given a sequence of actions, our set includes all the allowed permutations of this sequence. In this section, the distance between two sequences is given, as we have already explained, by the minimum number of swaps needed to transform one sequence into the other. The proof of the following result can be found in the appendix.

Proposition 1 *Let* $\sigma \in \mathcal{A}ct^*$ *be a sequence of actions. The pair* $([\sigma], d^1)$ *is a metric space.*

Based on the metric d^1, we can define what it means for a trace to be allowed if the specification is M and we allow at most distance k.

Definition 10 *Let* $M \in \texttt{IOTS}(I, O, \mathcal{P}orts)$*,* $k \geq 0$*, and* $\sigma \in \mathcal{A}ct^*$*. We say that* σ *is allowed by* M *given* d^1 *and* k *if there is some trace* $\sigma' \in L(M)$ *such that* $\sigma' \sim \sigma$ *and* $d^1(\sigma, \sigma') \leq k$.

We can extend the previous notion of *allowed* to define a new conformance relation for distributed testing.

Definition 11 *Let* $M, N \in \texttt{IOTS}(I, O, \mathcal{P}orts)$ *and* $k \geq 0$*. We write* N $\mathbf{dioco}^k_{\mathbf{sw}}$ M *if every trace* $\sigma \in L_\delta(N)$ *is allowed by* M *given* d^1 *and* k.

Given an IOTS M we can define the set of sequences that are within a certain distance of traces of M.

Definition 12 *Let* $M \in \texttt{IOTS}(I, O, \mathcal{P}orts)$ *and* $k \geq 0$*. We define the set* $L^1_k(M) \subseteq \mathcal{A}ct^*$ *as*

$$L^1_k(M) = \{\sigma | \exists \sigma' \in L(M) : \sigma' \sim \sigma \wedge d^1(\sigma, \sigma') \leq k\}$$

We have that $L^1_k(M)$ denotes the set of traces that are at distance at most k from traces of M. Note that if $k = 0$ then we obtain the original set of traces, that is, $L^1_0(M) = L(M)$.

The following is immediate if we take into account that a sequence σ is allowed by M given d^1 and k if and only if there is some trace $\sigma' \in L(M)$ with $d^1(\sigma', \sigma) \leq k$.

Proposition 2 *Let $M, N \in \texttt{IOTS}(I, O, \mathcal{P}orts)$ and $k \geq 0$. We have N $\mathbf{dioco}^k_{\mathbf{sw}}$ M if and only if $L_\delta(N) \subseteq L^1_k(M)$.*

A. An IOTS that recognises $L^1_k(M)$

It has been shown that if M is a finite model then the union of the equivalence classes of traces in $L(M)$, given equivalence relation $\sim$, need not be a regular language. To see this it is sufficient to let M be an IOTS with outputs $!o_1$ and $!o_2$ at ports 1 and 2 such that $L(M)$ is the set of prefixes of sequences in $(!o_1!o_2)^*$; the union of equivalence classes of traces in $L(M)$ contains all sequences in $\{!o_1, !o_2\}^*$ where the number of occurrences of $!o_1$ is the same as, or one more than, the number of occurrences of $!o_2$ and this clearly does not define a regular language. This helps explain the negative decidability and complexity results regarding distributed testing: in effect, we are not reasoning about regular languages. Fortunately, our distances, for each k, induce sets that are regular.

We will now prove that the sets $L^1_k(M)$ are regular. We will achieve this by, for M and k, defining a finite automaton $\mathcal{M}_a(M, k)$ that recognises $L^1_k(M)$.

Definition 13 *Let $M \in \texttt{IOTS}(I, O, \mathcal{P}orts)$ and $k \geq 0$. We inductively define the IOTS $\mathcal{M}_a(M, k)$ as follows. If $k = 0$ then $\mathcal{M}_a(M, k) = M$. Otherwise, if $k > 0$ then let us suppose that $\mathcal{M}_a(M, k-1) = (Q, Q_F, I, O, T, q_{in})$. Then $\mathcal{M}_a(M, k) = (Q', Q'_F, I, O, T', q'_{in})$ in which $q'_{in} = q^1_{in}$ and Q', Q'_F and T' are given in Figure 4.*

Essentially, the inductive step duplicates all the states of the previous IOTS so that we have two copies of a state q: q^1 corresponds to the situation in which the state is q and no events have been swapped, while q^2 corresponds to the situation in which the state is q and two events (at different ports) have been swapped. Algorithm 1 implements the construction of $\mathcal{M}_a(M, k)$. The generation of $\mathcal{M}_a(M, k)$ proceeds through k steps. The algorithm begins with the construction of $\mathcal{M}_a(M, 1)$ and this process is successively applied k times over the returned machine. Starting from the duplication of the states and transitions of M the algorithm adds new transitions to form $\mathcal{M}_a(M, 1)$. For every pair $t = (q_1, a, q_2)$ and $t' = (q_2, b, q_3)$ of consecutive transitions of M, with events a and b at different ports, a new state $q_{tt'}$ and two transitions are included in $\mathcal{M}_a(M, 1)$ in order to capture the swap of these events. Naturally, the auxiliary states $q_{tt'}$ are not final. This process is successively applied to the machine returned k times.

Example 3 *Consider the IOTS M depicted in Figure 5 (a) in which indexes denote the port where the actions are produced. Figure 5 (b) shows $\mathcal{M}_a(M, 1)$, where we have considered two pairs of consecutive transitions:*

- *$t = (q_0, a_1, q_1)$ and $t' = (q_1, b_2, q_2)$.*
- *$t' = (q_1, b_2, q_2)$ and $t'' = (q_2, c_3, q_3)$.*

The following result, whose proof is given in the appendix, states that the language accepted by the automaton constructed by using Algorithm 1 coincides with the one given by Definition 12.

Theorem 1 *Let $M \in \texttt{IOTS}(I, O, \mathcal{P}orts)$ and $k \geq 0$. We have that $L^1_k(M) = L(\mathcal{M}_a(M, k))$.*

Note that the above result shows that if M is finite then we can construct a finite automaton that recognises $L^1_k(M)$ and so a number of important problems will be decidable.

B. Decision problems and their complexity

We now explore two complexity problems associated with a conformance relation. As previously noted, when we reason about complexity we restrict attention to the case where the models are finite since almost all decision problems are trivially undecidable when we allow models to be infinite. First, we formally define the notions of *membership* and *correctness* in our framework.

Definition 14 *Let $M \in \texttt{IOTS}(I, O, \mathcal{P}orts)$ be a finite IOTS, $\sigma \in \mathcal{A}ct^*$ be a sequence of actions, d be a metric, and $k \geq 0$. The corresponding* membership problem *is to decide whether there is some $\sigma' \in L(M)$ such that $\sigma' \sim \sigma$ and $d(\sigma, \sigma') \leq k$. If there is such a σ' then we say that σ is allowed by M given d and k.*

Let $M, N \in \texttt{IOTS}(I, O, \mathcal{P}orts)$ be two finite IOTSs, d be a metric, and $k \geq 0$. The corresponding correctness problem *is to decide whether all $\sigma \in L(N)$ are allowed by M given d and k.*

We now explore the correctness and membership problems and their complexity. Since this only makes sense in the context of finite models, throughout the rest of this section we assume that M has finite sets of states, inputs, and outputs. We first give a result, whose proof is given in the appendix, regarding the size of $\mathcal{M}_a(M, k)$ and how it relates to the size of M.

Proposition 3 *Let $M \in \texttt{IOTS}(I, O, \mathcal{P}orts)$ be a finite IOTS and $k \geq 0$. We have that $\mathcal{M}_a(M, k)$ has $O(|T|^{2^k})$ transitions, where T is the finite set of transitions of M.*

Since we have a bound on the number of transitions of the systems generated from M and k, we can now explore complexity results, starting with the correctness problem. The proofs of the following four results are given in the appendix.

Theorem 2 *Let $M, N \in \texttt{IOTS}(I, O, \mathcal{P}orts)$ be finite IOTSs and $k \geq 0$. The corresponding correctness problem, given d^1 and k, is in 2-EXPSPACE and is PSPACE-hard.*

$$Q' = \{q^1 | q \in Q\} \cup \{q^2 | q \in Q\} \cup \{ q_{t_1 t_2} \mid \exists t_1 = (q_1, a, q_2) \in T, t_2 = (q_2, b, q_3) \in T : port(a) \neq port(b) \}$$

$$Q'_F = \{q^1 | q \in Q_F\} \cup \{q^2 | q \in Q_F\}$$

$$T' = \{(q_1^1, a, q_2^1) | (q_1, a, q_2) \in T\} \cup \{(q_1^2, a, q_2^2) | (q_1, a, q_2) \in T\} \cup \{(q_1^1, b, q_{t_1 t_2}), (q_{t_1 t_2}, a, q_3^2) | \exists q_2 \in Q, t_1 = (q_1, a, q_2) \in T, t_2 = (q_2, b, q_3) \in T : port(a) \neq port(b)\}$$

Fig. 4: Description of Q', Q'_F and T' in Definition 13

Algorithm `Produce_`$\mathcal{M}_a(M, k)$
/*$M = (Q, Q_F, I, O, T, q_{in})$, $M' = (Q', Q'_F, I, O, T', q'_{in})$*/
if $k = 1$ **then**
  $Q' := \emptyset$; $Q'_F := \emptyset$; $T' := \emptyset$;
  foreach *state* $q \in Q$ **do**
    $Q' := Q' \cup \{q^1, q^2\}$;
    /*q^1, q^2 are fresh states*/
    if $q \in Q_F$ **then** $Q'_F := Q'_F \cup \{q^1, q^2\}$;
  end
  $q'_{in} := q^1_{in}$;
  foreach *transition* $(q_1, a, q_2) \in T$ **do**
    $T' := T' \cup \{(q_1^1, a, q_2^1), (q_1^2, a, q_2^2)\}$
  end
  foreach *pair of transitions* $t = (q_1, a, q_2), t' = (q_2, b, q_3) \in T$ *with* $port(a) \neq port(b)$ **do**
    $Q' := Q' \cup \{q_{tt'}\}$;
    /*$q_{tt'}$ is a fresh state*/
    $T' := T' \cup \{(q_1^1, b, q_{tt'}), (q_{tt'}, a, q_3^2)\}$;
  end
  return(M');
else
  return(`Produce_`$\mathcal{M}_{\mathtt{a}}$(`Produce_`$\mathcal{M}_{\mathtt{a}}(M, k-1), 1)$);
end

Algorithm 1: Producing $\mathcal{M}_a(M, k)$ for $k > 0$.

Note that, in contrast, the corresponding problem for **dioco** is undecidable [18]. Also note that there is a gap between the bounds in the above case; reducing this gap is a problem for future work. The gap between the upper and lower bounds in the above result comes from the fact that our representation of $L^1_k(M)$ has double exponential size; the (2-EXPSPACE) upper bound uses this but the lower bound does not assume that this is the most compact representation of $L^1_k(M)$. It should be possible to remove this gap between the upper and lower bounds if we can prove that we cannot represent $L^1_k(M)$ more compactly.

In practice, we might have a relatively small value of k. This would be the case, for example, if the local testers are close to being synchronised; we might know that there can only be a very small number of differences. As a result, we are interested in how solutions to the membership problem scale as M and σ grow but k is bounded above (or, equivalently, if k is fixed). We now therefore explore the complexity of the correctness problem for bounded k. The main observation is that for bounded k we have that $\mathcal{M}_a(M, k)$ has size that is polynomial in the size of M since we can fix k to be the upper bound.

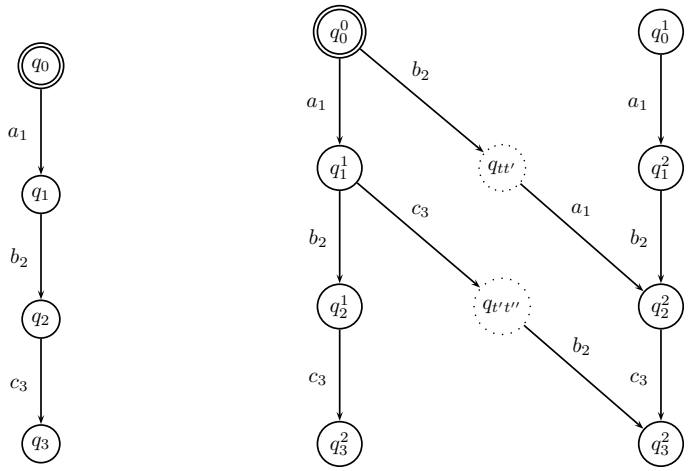
Theorem 3 *Let $M, N \in$ `IOTS`$(I, O, \mathcal{P}orts)$ be finite* `IOTS`*s and $k \geq 0$. The corresponding bounded correctness problem, given d^1 and k, is PSPACE-complete.*

Note that this is the same complexity as regular language inclusion. Next, we explore the complexity of the membership problem. Note that the membership problem is known to be NP-hard for **dioco** [19], a result we use in the proof of the following result.

Theorem 4 *Let $M \in$ `IOTS`$(I, O, \mathcal{P}orts)$ be a finite* `IOTS`*, $\sigma \in \mathcal{A}ct^*$ be a sequence of actions and $k \geq 0$. The problem of deciding whether M has a trace $\sigma' \sim \sigma$ such that $d^1(\sigma, \sigma') \leq k$ is NP-complete.*

Again, the problem becomes simpler if we bound k.

Theorem 5 *Let $M \in$ `IOTS`$(I, O, \mathcal{P}orts)$ be a finite* `IOTS`*, $\sigma \in \mathcal{A}ct^*$ be a sequence of actions and $k \geq 0$. If we have an upper bound on k then the problem of deciding whether M has a trace $\sigma' \sim \sigma$ such that $d^1(\sigma, \sigma') \leq k$ can be solved in polynomial time.*



(a) M (b) $\mathcal{M}_a(M,1)$

Fig. 5: Generation of $\mathcal{M}_a(M,1)$ by Algorithm 1

V. Placing a bound on the distance between events

In this section we introduce our second notion of distance and the associated conformance relation. In the new setting, the distance between two traces is equal to the maximum difference between the positions of each action in both traces.

Definition 15 *Let $\sigma_1, \sigma_2 \in \mathcal{A}ct^*$ be sequences of actions such that $\sigma_1 \sim \sigma_2$. We define the distance d^2 between these two sequences as follows:*

$$d^2(\sigma_1,\sigma_2) = max_{e\in E(\sigma_1)}\{|pos_{\sigma_1}(e) - pos_{\sigma_2}(e)|\}$$

*Given $1 \leq j \leq d^2(\sigma_1,\sigma_2)$, if $pos_{\sigma_1}(e) - pos_{\sigma_2}(e) = j$ then we say that $e \in E(\sigma_1)$ is j-*early *in σ_2 with respect to σ_1 and we also say that $e \in E(\sigma_1)$ is j-*late *in σ_1 with respect to σ_2.*

Abusing the notation, we have that d^2 is defined over each equivalence class $[\sigma]$, that is, it induces a function $d^2_\sigma : [\sigma] \times [\sigma] \longrightarrow \mathbf{N}$ in the obvious way and we will usually remove the σ index.

Example 4 *Consider the traces $\sigma_1 = ?i_1!o_1?i_2!o_1!o_2$ and $\sigma_2 = ?i_2!o_2?i_1!o_1!o_1$, where the index denotes the port at which the action is performed. We know that both traces have the same associated set of events (see Example 1). However, the positions of the events in each trace are different. For example, $pos_{\sigma_1}((?i_1,1)) = 1$ while $pos_{\sigma_2}((?i_1,1)) = 3$. We have that the biggest difference between the positions of events is equal to 3 (given by event $(!o_2,1)$). Therefore, $d^2(\sigma_1,\sigma_2) = 3$. In addition, we have that $(?i_1,1)$ is 2-late in σ_2 with respect to σ_1 and $(!o_2,1)$ is 3-early in σ_2 with respect to σ_1.*

Next we state that the previously defined notion induces a metric space on each equivalence class with respect to $\sim$ (the proof is in the appendix).

Proposition 4 *Let $\sigma \in \mathcal{A}ct^*$ be a sequence of actions. The pair $([\sigma], d^2)$ is a metric space.*

Given an IOTS M we can define the set of sequences that are within a certain distance of M.

Definition 16 *Let $M \in \mathtt{IOTS}(I,O,\mathcal{P}orts)$ and $k \geq 0$. We define the set $L^2_k(M) \subseteq \mathcal{A}ct^*$ as*

$$L^2_k(M) = \{\sigma \in \mathcal{A}ct^* | \exists \sigma' \in L(M) : \sigma' \sim \sigma \wedge d^2(\sigma,\sigma') \leq k\}$$

We have that $L^2_k(M)$ denotes the set of traces that are at distance at most k from a trace of M. Note that $L^2_0(M) = L(M)$.

Similar to before, we can now define the notion of a trace being allowed by the specification (for a given bound) and so a new conformance relation.

Definition 17 *Let $M \in \mathtt{IOTS}(I,O,\mathcal{P}orts)$, $k \geq 0$, and $\sigma \in \mathcal{A}ct^*$. We say that σ is allowed by M, given d^2 and k, if there is some trace $\sigma' \in L(M)$ such that $\sigma' \sim \sigma$ and $d^2(\sigma,\sigma') \leq k$.*

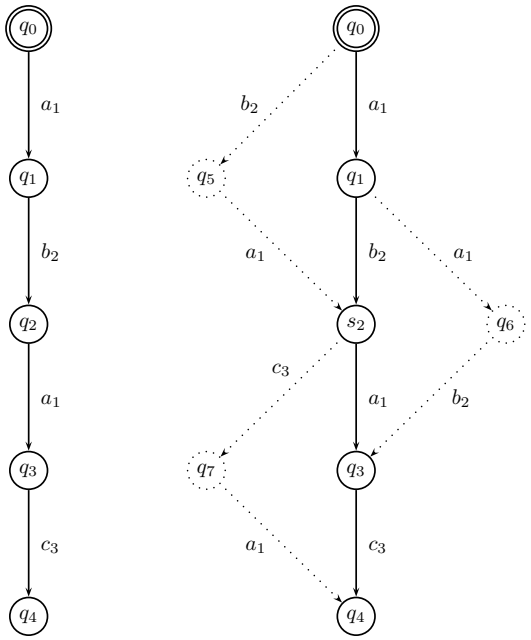
Definition 18 *Let $M, N \in \mathtt{IOTS}(I,O,\mathcal{P}orts)$ and $k \geq 0$. We write N $\mathbf{dioco}^k_{\mathbf{dis}}$ M if every trace $\sigma \in L_\delta(N)$ is allowed by M given d^2 and k.*

The following is immediate from the fact that a sequence σ is allowed by M given k if and only if there is some trace $\sigma' \in L(M)$ with $d^2(\sigma',\sigma) \leq k$.

Proposition 5 *Let $M, N \in \mathtt{IOTS}(I,O,\mathcal{P}orts)$ and $k \geq 0$. We have that N $\mathbf{dioco}^k_{\mathbf{dis}}$ M if and only if $L_\delta(N) \subseteq L^2_k(M)$.*

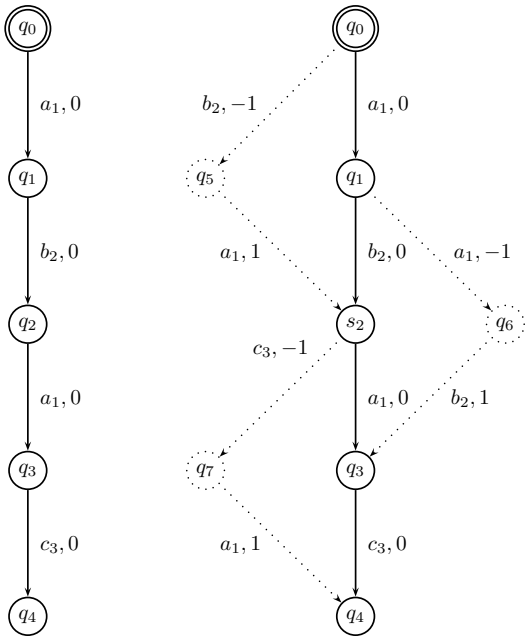
A. An IOTS that recognises $L^2_k(M)$

Given M, it is straightforward to construct an IOTS that accepts all traces within distance 1 of $L(M)$; for a pair $((q_1,a,q_2),(q_2,b,q_3))$ of consecutive transitions whose events are at different ports, we create a new state q_4 and transitions $(q_1,b,q_4),(q_4,a,q_3)$ to represent the possibility of the two



(a) M (b) $\mathcal{M}_b(M,1)$

Fig. 6: An IOTS and one that includes all traces of distance at most 1



(a) M (b) $\mathcal{M}_b(M,1)$

Fig. 7: Generation of $\mathcal{M}_b(M,1)$ by Algorithm 3

events being swapped. Naturally, state q_4 is not a final state since otherwise we would include b as a possible trace. Figure 6 gives an example of the application of such a procedure and Algorithm 2 shows how one can implement this procedure.

Having shown how one can take an IOTS and construct an IOTS that accepts all traces within distance 1, it might seem that we can simply apply this procedure k times. However, the following shows that this need not be sufficient.

Example 5 *Consider M whose language is the trace $\sigma_1 = a_1a_2a_3$, and all prefixes of this, and consider also the trace $\sigma_2 = a_3a_2a_1$. It is clear that the distance between σ_1 and σ_2 is 2. If we apply Algorithm 2 to M, to form $\mathcal{M}_b(M,1)$, then we add the traces $a_1a_3a_2$ and $a_2a_1a_3$. However, if we apply Algorithm 2 to $\mathcal{M}_b(M,1)$ then we add the traces $a_3a_1a_2$ and $a_2a_3a_1$, and so some traces at distance 2 from σ_1, but we do not obtain σ_2.*

It is not too hard to see that, in the example, three applications of Algorithm 2 would have allowed us to obtain the trace σ_2. If we call the application of Algorithm 2 a *step* then the following result, whose proof is given in the appendix, gives an upper bound on the number of steps required.

Proposition 6 *Let σ_1, σ_2 be such that $\sigma_1 \sim \sigma_2$ and $d^2(\sigma_1, \sigma_2) = k$ for some $k \geq 0$. Then it is possible to rewrite σ_1 to σ_2 through at most $k \cdot (k+1)$ steps.*

It might seem that we can just apply Algorithm 2 a total of $k \cdot (k+1)$ times. However, this might allow some events to be moved too far. In order to avoid this, in effect we will repeatedly apply Algorithm 2 but whenever we swap an event we update a label that tells us how far it has been swapped and we do not allow swaps that exceed the upper bound k. This idea is implemented in Algorithm 3.

Definition 19 *Let $M \in \mathtt{IOTS}(I, O, \mathcal{P}orts)$ and $k \geq 1$. We say that $\mathcal{M}_b(M,k)$ is the IOTS returned by* Produce_$\mathcal{M}_b(M,k)$.

In order to prove that $L^2_k(M)$ is regular, it is enough to show that this set of traces coincides with the set of sequences recognised by the finite automaton produced from the original system after applying Algorithm 3. This is given by the following result, whose proof is a direct consequence of Proposition 6 and Algorithm 2.

Theorem 6 *Let $M \in \mathtt{IOTS}(I, O, \mathcal{P}orts)$. We have that $L^2_k(M) = L(\mathcal{M}_b(M,k))$.*

In Figures 7 and 8 we show applications of the algorithm, where redundant transitions, that is, transitions labelled by the same action and outgoing/reaching the same states, have been removed for clarity.

B. Decision problems and their complexity

As before, we explore two complexity problems.

1) Deciding whether a sequence of actions σ is allowed by M given k (a membership problem); deciding whether σ is a member of the regular language $L^2_k(M)$.
2) Deciding whether N is a correct implementation of M given k; deciding whether $L_\delta(N)$ is a subset of the regular language $L^2_k(M)$.

A single step adds two transitions for every pair of consecutive transitions at different ports. Since we require at most

Algorithm Produce_$\mathcal{M}_b(M,1)$ /*$M=(Q,Q_F,I,O,T,q_{in})$, $\mathcal{M}_b(M,1)=(Q^1,Q^1_F,I,O,T^1,q_{in})$ */
$\mathcal{M}_b(M,1) \leftarrow M$;
foreach $(q,a,q'),(q',b,q'') \in T$ *with* $port(a) \neq port(b)$ **do**
  $Q^1 \leftarrow Q^1 \cup \{q_w\}$;
  /*q_w is a fresh state */
  $T^1 \leftarrow T^1 \cup \{(q,b,q_w),(q_w,a,q'')\}$;
end
return$(\mathcal{M}_b(M,1))$;

Algorithm 2: Producing $\mathcal{M}_b(M,1)$.

Algorithm Produce_$\mathcal{M}_b(M,k)$ /*$M=(Q,Q_F,I,O,T,q_{in})$, $\mathcal{M}_b(M,k)=(Q^k,Q^k_F,I,O,T^k,q_{in})$ */
$Q^k \leftarrow Q$;
$Q^k_F \leftarrow Q_F$;
$T^k \leftarrow \{(q,(a,0),q')|(q,a,q') \in T\}$;
/*each transition has a different value of i */
for $n=1$ *to* $k\cdot(k+1)$ **do**
  foreach $(q,(a,d_a),q'),(q',(b,d_b),q'') \in T^k$ *with* $port(a) \neq port(b)$ **do**
    if $|d_b-1| \leq k \wedge |d_a+1| \leq k$ **then**
      $Q^k \leftarrow Q^k \cup \{q_w\}$;
      /*q_w is a fresh state */
      $T^k \leftarrow T^k \cup \{(q,(b,d_b-1),q_w),(q_w,(a,d_a+1),q'')\}$;
    end
  end
end
$T^k \leftarrow \{(q,a,q')|(q,(a,d_a),q') \in T^k\}$;
/*by construction, $d_a \leq k$ */
return$(\mathcal{M}_b(M,k))$;

Algorithm 3: Producing $\mathcal{M}_b(M,k)$

$k\cdot(k+1)$ steps, we obtain the following result that is similar to Proposition 3. The only difference between the proofs is that there are now $k\cdot(k+1)$ steps rather than k steps.

Proposition 7 *Let $M \in \texttt{IOTS}(I,O,\mathcal{P}orts)$ be a finite* IOTS *and $k \geq 0$. We have that $\mathcal{M}_b(M,k)$ has $O(|T|^{2^{k\cdot(k+1)}})$ transitions, where T is the finite set of transitions of M.*

We start by examining the correctness problem. The proof is similar to that of Theorem 2 and we omit it.

Theorem 7 *Let $M,N \in \texttt{IOTS}(I,O,\mathcal{P}orts)$ be finite* IOTSs *and $k \geq 0$. The corresponding correctness problem, given d^2 and k, is in 2-EXPSPACE and is PSPACE-hard.*

In practice, we might have a relatively small value of k and are interested in how solutions to the decision problems scale as M and σ grow. We now therefore explore the complexity of the correctness problem for bounded k. The main observation is that for bounded k we have that $\mathcal{M}_b(M,k)$ has polynomial size. The proof is similar to that of Theorem 3 and we omit it.

Theorem 8 *Let $M,N \in \texttt{IOTS}(I,O,\mathcal{P}orts)$ be finite* IOTSs *and $k \geq 0$. The corresponding bounded correctness problem, given d^2 and k, is PSPACE-complete.*

We now explore the complexity of the membership problem: that of checking whether an observation made is allowed (whether it is a trace of $\mathcal{M}_b(M,k)$). The proof is similar to that of Theorem 4 and we omit it.

Theorem 9 *Let $M \in \texttt{IOTS}(I,O,\mathcal{P}orts)$ be a finite* IOTS*, $\sigma \in \mathcal{A}ct^*$ be a sequence of actions and $k \geq 0$. The problem of deciding whether M has a trace $\sigma' \sim \sigma$ such that $d^2(\sigma,\sigma') \leq k$ is NP-complete.*

We now explore the complexity of the membership problem for bounded k and show that this can be solved in polynomial time. Given a sequence of actions σ we can place a partial order $\prec_k$ on the events in $E(\sigma)$, see Definition 6, such that $e^i \prec_k e^j$ if for all $\sigma' \sim \sigma$ we have that if $d^2(\sigma,\sigma') \leq k$ then a^i must appear before a^j in σ', where as usual a^x is the action associated with the event e^x. This partial order induces the notions of *ideal* and *antichain* that we will use in the proof of our result. Note that we use events, instead of actions, to deal with different occurrences of the same action in a sequence.

Definition 20 *Let $\sigma = a^1 \ldots a^n \in \mathcal{A}ct^*$ be a sequence of actions and $k \geq 0$. We define the relation $\prec_k$ over $E(\sigma)$ so that $e^i \prec_k e^j$ if and only if one of the following holds.*

1) *a^i and a^j are at the same port and $i<j$;*

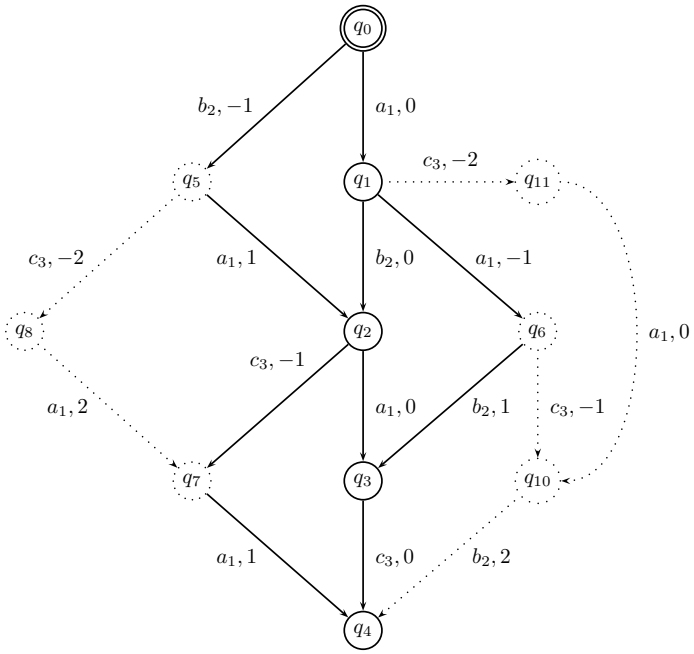


Fig. 8: Generation of part of $\mathcal{M}_b(M, 2)$ by Algorithm 3

2) a^i *and* a^j *are at different ports and* $j - i \geq 2 \cdot k$.

Let $E \subseteq E(\sigma)$ *be a subset of events. We say that* E *is* downwardly closed *if whenever we have that* $e \in E$ *and* $e' \prec_k e$ *we also have that* $e' \in E$. *In this case we say that* E *is an* ideal.

Let $E \subseteq E(\sigma)$ *be a subset of events. We say that* E *is an* anti-chain *if there do not exist* $e, e' \in E$ *such that* $e \prec_k e'$.

Let $E \subseteq E(\sigma)$ *be a subset of events. We say that an event* $e \in E$ *is* maximal *in* E *if there is no* $e' \in E$ *such that* $e \prec_k e'$.

The first condition in the definition of $\prec_k$ corresponds to the requirement that we only consider traces with the same projections; if two actions are at the same port then their relative order must be preserved. The second condition corresponds to the constraint introduced by k: at most a_i can be delayed by k and a_j brought forward by k and so if the differences in position is $2 \cdot k$ or above then they cannot be swapped in forming σ' with $d^2(\sigma, \sigma') \leq k$. Note that σ is an implicit parameter of $\prec_k$ but we omit it in order to avoid overloading the notation.

We now show how we can construct an IOTS $\mathcal{M}_b(\sigma, k)$ that recognises all sequences within distance k of σ (according to d^2). The key point is that if $d^2(\sigma, \sigma') \leq k$ then the partial order $\prec_k$ must be a subset of the order in σ' and so all prefixes of σ' must also be consistent with $\prec_k$. Thus, if σ'' is a prefix of σ' and $d^2(\sigma, \sigma') \leq k$ then the elements of $E(\sigma'')$ must be downwardly closed under $\prec_k$.

Definition 21 *Let* $\sigma \in \mathcal{A}ct^*$ *be a sequence of actions and* $k \geq 0$. *We denote by* $\mathcal{M}_b(\sigma, k)$ *the* IOTS $(Q, q_0, E(\sigma), T, F)$ *such that the state set* Q *is the set of downwardly closed (under* $\prec_k$*) subsets of* $E(\sigma)$, $q_0 = \{\}$ *is the initial state, the only final state is* $E(\sigma)$, *and* (q, a, q') *is a transition if and only if* $e \in E(\sigma) \setminus q$, $q' = q \cup \{e\}$ *and* $e = (a, j)$ *for some* $j \in \mathbb{N}$.

Note that given two sequences such that $\sigma \sim \sigma'$, we have $d^2(\sigma, \sigma') \leq k$ if and only if $\sigma' \in L(\mathcal{M}_b(\sigma, k))$. The following example shows the construction of finite automata, for different values of k, for a given trace.

Example 6 *Consider the trace* $\sigma = ?i_1!o_1?i_2!o_1!o_2$ *and its set of events* $E(\sigma) = \{(?i_1, 1), (!o_1, 1), (?i_2, 1), (!o_1, 2), (!o_2, 1)\}$ *given in Example 1. The ideals of* $E(\sigma)$ *for* $\prec_0$ *are:*

$$\begin{aligned}
\mathcal{I}_0 &= \emptyset \\
\mathcal{I}_1 &= \{(?i_1, 1)\} \\
\mathcal{I}_2 &= \{(?i_1, 1), (!o_1, 1)\} \\
\mathcal{I}_3 &= \{(?i_1, 1), (!o_1, 1), (?i_2, 1)\} \\
\mathcal{I}_4 &= \{(?i_1, 1), (!o_1, 1), (?i_2, 1), (!o_1, 2)\} \\
\mathcal{I}_5 &= \{(?i_1, 1), (!o_1, 1), (?i_2, 1), (!o_1, 2), (!o_2, 1)\}
\end{aligned}$$

and $\mathcal{M}_b(\sigma, 0)$ *can be found in Figure 9 (left). The ideals of* $E(\sigma)$ *for* $\prec_1$ *are the previous ones plus the next ones:*

$$\begin{aligned}
\mathcal{I}_6 &= \{(?i_1, 1), (?i_2, 1)\} \\
\mathcal{I}_7 &= \{(?i_1, 1), (!o_1, 1), (!o_1, 2)\} \\
\mathcal{I}_8 &= \{(?i_1, 1), (!o_1, 1), (?i_2, 1), (!o_2, 1)\}
\end{aligned}$$

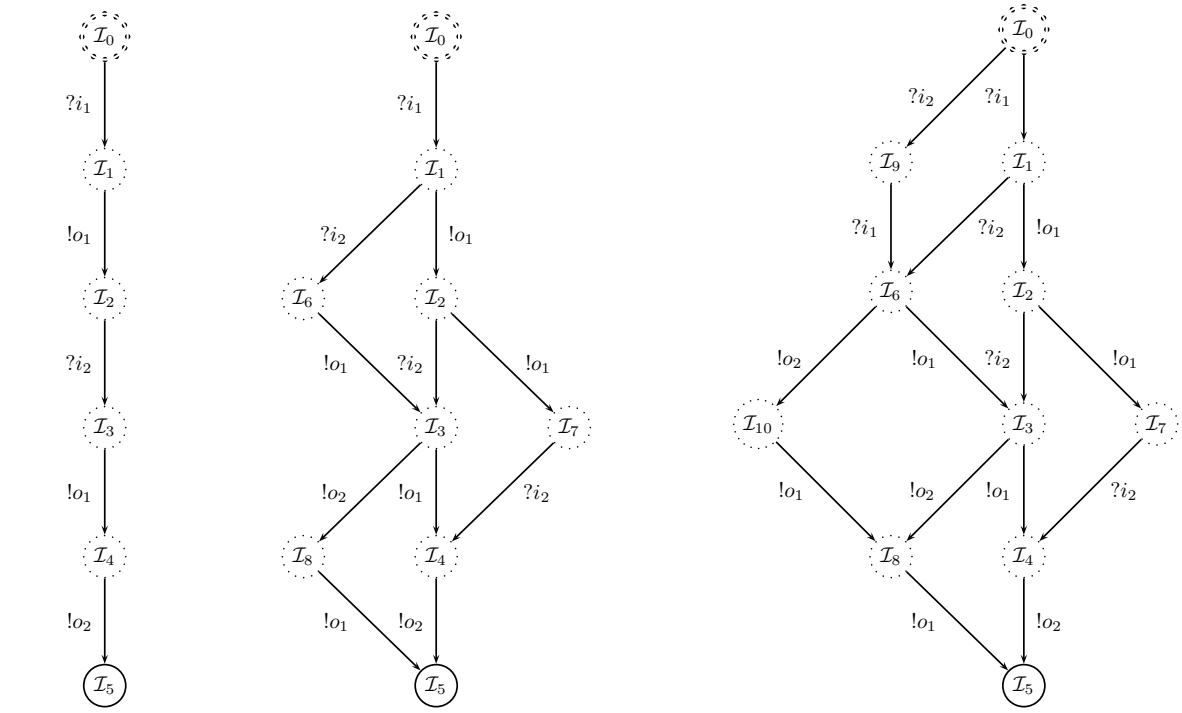
and $\mathcal{M}_b(\sigma, 1)$ *can be found in Figure 9 (center). Finally, the ideals of* $E(\sigma)$ *for* $\prec_2$ *are the previous ones plus the next ones:*

$$\mathcal{I}_9 = \{(?i_2, 1)\} \qquad \mathcal{I}_{10} = \{(?i_1, 1), (?i_2, 1), (!o_2, 1)\}$$

and $\mathcal{M}_b(\sigma, 2)$ *can be found in Figure 9 (right).*

We now consider the complexity of the membership problem if we bound k. The proof of the following two results are given in the appendix.

Theorem 10 *Let* $M \in$ IOTS$(I, O, \mathcal{P}orts)$ *be a finite* IOTS, $\sigma \in \mathcal{A}ct^*$ *be a sequence of actions and* $k \geq 0$. *If we have an*



(a) $\mathcal{M}_b(\sigma, 0)$ (b) $\mathcal{M}_b(\sigma, 1)$ (c) $\mathcal{M}_b(\sigma, 2)$

Fig. 9: $\mathcal{M}_b(\sigma, 0)$, $\mathcal{M}_b(\sigma, 1)$, and $\mathcal{M}_b(\sigma, 2)$ for $\sigma = ?i_1!o_1?i_2!o_1!o_2$

upper bound on k then the problem of deciding whether M has a trace $\sigma' \sim \sigma$ such that $d^2(\sigma, \sigma') \leq k$ can be solved in polynomial time.

Interestingly, the problem can also be solved in polynomial time if we have an upper bound on the number of ports.

Theorem 11 *Let $M \in \texttt{IOTS}(I, O, \mathcal{P}orts)$ be a finite* IOTS*, $\sigma \in \mathcal{A}ct^*$ be a sequence of actions and $k \geq 0$. If we have an upper bound on the number of ports then the problem of deciding whether M has a trace $\sigma' \sim \sigma$ such that $d^2(\sigma, \sigma') \leq k$ can be solved in polynomial time.*

VI. Conclusions and future work

The study of conformance relations in the distributed testing architecture carries additional complications that must be solved. A liberal interpretation is that as long as an observed trace is a permutation of a trace of the specification, that has the same projections at the ports, then we cannot claim that we found an error. The underlying assumption is that since the local testers do not synchronise during testing, it is impossible to determine the relative order of events observed at different ports. However, there are situations in which this assumption does not hold: there can be events e^1 and e^2 at different ports where, after testing, one can determine that e^1 occurred before e^2. In such situations, classical conformance relations for distributed testing are too weak: they can claim that an IOTS N is correct with respect to specification IOTS M in situations in which N is not a correct implementation of M. To see this, let us suppose that the SUT produces trace $!o_3!o_2!o_1$, the specified trace is $!o_1!o_2!o_3$, and the $!o_i$ are at different ports. This is acceptable under the standard conformance relation for distributed testing; these two traces are observationally equivalent since we cannot determine the relative order in which the $!o_i$ were observed. However, in this case we observe a failure if, for example, the trace of the implementation is only allowed to differ from a trace of the specification through at most one swap of adjacent events.

In this paper, we addressed this problem by introducing conformance relations that use notions of distance. The essential idea is that even if trace σ_1 of the SUT and a trace σ_2 of the specification are observationally equivalent ($\sigma_1 \sim \sigma_2$), they can be distinguished during testing if the distance between σ_1 and σ_2 is greater than some given value k. The motivation is that if the two traces are too different then it will be possible to distinguish them in distributed testing. For example, we should be able to distinguish between one process that outputs $!o_1$ repeatedly for two days before output $!o_2$ at a different port and another process that first outputs $!o_2$ before outputting $!o_1$ for two days. This is despite these traces being observationally equivalent under $\sim$ due to them having the same local projections.

We considered two possible metrics and for each we defined new conformance relations and studied the computational complexity of the main decision problems: determining whether a trace is allowed by the specification (a membership/oracle problem) and deciding whether an IOTS N is a correct implementation of a specification IOTS M (a correctness problem). In classical distributed testing the membership problem is NP-complete and the correctness

problem is undecidable. The two conformance relations given in this paper had similar complexity results and, similar to the classical case, we found that the membership problem is NP-complete. However, unlike the classical case the correctness problem is decidable: it is PSPACE-hard and in 2-EXPSPACE. It transpired that if we fix, or bound above, the value k used then the membership problem is polynomial time solvable and the correctness problem is PSPACE-complete. For the bounded case, these results are equivalent to the corresponding problems for centralised testing.

There is the interesting question as to when an conformance relation might be used and this is likely to depend on properties of the SUT and test infrastructure. For example, we might know that the local testers have clocks that are sufficiently close to being synchronised that the order derived from timestamps will only very rarely be wrong. In such cases, we would have reason to bound the number of swaps of adjacent events, and so we might use our first conformance relation. A bound might be derived using a probabilistic argument and is likely to depend on the test sequence length. Alternatively, if events during testing occur at roughly regular intervals and we have a bound on the differences between the local clocks then we could derive a bound for use with our second conformance relation. Let us suppose, for example, that the gap between adjacent events is approximately one millisecond and that the local clocks are known to differ by at most two milliseconds[2]. We might then conclude, for example, that the difference between the actual position of an event and the estimated position is at most three milliseconds. As noted, the bounds used are likely to depend on properties of the SUT but might also depend on the test case used and how this is applied. If the local testers can communicate then it may well be possible to design an approach in which the exchange of synchronisation messages is used to ensure that a relatively low value for a bound can be used.

We contemplate three main lines of future work in addition to that described above. First, we would like to study other metrics and compare their properties concerning which systems are conforming to a given specification. Second, we would like to complete our study on the computational complexity of certain problems so that we can close the gap in the bounds presented in this paper. Finally, we would like to study test derivation to provide sound and complete test suites. The starting point is our previous test derivation algorithms in the distributed architecture for *controllable* test cases [24], [26]. However, our preliminary exploration shows that the extension to remove the controllability restriction is not easy. We are also exploring the possibility of reducing test generation for *bounded* versions of **dioco** to test generation and application in the non-distributed setting. Intuitively, we should consider suitable permutations of the sequences that the specification can perform.

[2]If the local testers are connected to the internet then the Network Time Protocol could be used to derive a bound between the differences between local clocks.

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for the careful reading and for their constructive comments that have helped to further strengthen the paper.

REFERENCES

[1] R. Alur, K. L. McMillan, and D. Peled. Model-checking of correctness conditions for concurrent objects. *Information and Computation*, 160(1-2):167–188, 2000.

[2] P. Ammann and J. Offutt. *Introduction to Software Testing*. Cambridge University Press, 2nd edition, 2017.

[3] R. V. Binder, B. Legeard, and A. Kramer. Model-based testing: where does it stand? *Communications of the ACM*, 58(2):52–56, 2015.

[4] G. von Bochmann, S. Haar, C. Jard, and G.-V. Jourdan. Testing systems specified as partial order input/output automata. In *Joint 20th IFIP TC6/WG6.1 Int. Conf. on Testing of Software and Communicating Systems, TestCom'08, and 8th Int. Workshop on Formal Approaches to Software Testing, FATES'08, LNCS 5047*, pages 169–183. Springer, 2008.

[5] S.C. Boyd and H. Ural. The synchronization problem in protocol testing and its complexity. *Information Processing Letters*, 40(3):131–136, 1991.

[6] L. Brandán Briones and E. Brinksma. Testing real-time multi input-output systems. In *7th Int. Conf. on Formal Engineering Methods, ICFEM'05, LNCS 3785*, pages 264–279. Springer, 2005.

[7] L. Cacciari and O. Rafiq. Controllability and observability in distributed testing. *Information and Software Technology*, 41(11–12):767–780, 1999.

[8] A. R. Cavalli, T. Higashino, and M. Núñez. A survey on formal active and passive testing with applications to the cloud. *Annales of Telecommunications*, 70(3-4):85–93, 2015.

[9] W. Chen and H. Ural. Synchronizable checking sequences based on multiple UIO sequences. *IEEE/ACM Transactions on Networking*, 3:152–157, 1995.

[10] R. P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950.

[11] B. Dongol and R. M. Hierons. Decidability and complexity for quiescent consistency. In *31st Annual ACM/IEEE Symposium on Logic in Computer Science,LICS'16*, pages 116–125. ACM Press, 2016.

[12] R. Dssouli and G. von Bochmann. Error detection with multiple observers. In *5th WG6.1 Int. Conf. on Protocol Specification, Testing and Verification, PSTV'85*, pages 483–494. North-Holland, 1985.

[13] R. Dssouli and G. von Bochmann. Conformance testing with multiple observers. In *6th WG6.1 Int. Conf. on Protocol Specification, Testing and Verification, PSTV'86*, pages 217–229. North-Holland, 1986.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman and Company, 1979.

[15] M.-C. Gaudel. Testing can be formal, too! In *6th Int. Joint Conf. CAAP/FASE, Theory and Practice of Software Development, TAPSOFT'95, LNCS 915*, pages 82–96. Springer, 1995.

[16] S. Haar, C. Jard, and G.-V. Jourdan. Testing input/output partial order automata. In *Joint 19th IFIP TC6/WG6.1 Int. Conf. on Testing of Software and Communicating Systems, TestCom'07, and 7th Int. Workshop on Formal Approaches to Software Testing, FATES'07, LNCS 4581*, pages 171–185. Springer, 2007.

[17] T. A. Henzinger, C. M. Kirsch, H. Payer, A. Sezgin, and A. Sokolova. Quantitative relaxation of concurrent data structures. In *40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'13*, pages 317–328. ACM Press, 2013.

[18] R. M. Hierons. Reaching and distinguishing states of distributed systems. *SIAM Journal on Computing*, 39(8):3480–3500, 2010.

[19] R. M. Hierons. Oracles for distributed testing. *IEEE Transactions on Software Engineering*, 38(3):629–641, 2012.

[20] R. M. Hierons. Verifying and comparing finite state machines for systems that have distributed interfaces. *IEEE Transactions on Computers*, 62(8):1673–1683, 2013.

[21] R. M. Hierons. Generating complete controllable test suites for distributed testing. *IEEE Transactions on Software Engineering*, 41(3):279–293, 2015.

[22] R. M. Hierons. A more precise implementation relation for distributed testing. *Computer Journal*, 59(1):33–46, 2016.

[23] R. M. Hierons, K. Bogdanov, J.P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Luettgen, A.J.H Simons, S. Vilkomir, M.R. Woodward, and H. Zedan. Using formal specifications to support testing. *ACM Computing Surveys*, 41(2), 2009.

[24] R. M. Hierons, M. G. Merayo, and M. Núñez. Controllable test cases for the distributed test architecture. In *6th Int. Symposium on Automated Technology for Verification and Analysis, ATVA'08, LNCS 5311*, pages 201–215. Springer, 2008.

[25] R. M. Hierons, M. G. Merayo, and M. Núñez. Implementation relations for the distributed test architecture. In *Joint 20th IFIP TC6/WG6.1 Int. Conf. on Testing of Software and Communicating Systems, TestCom'08, and 8th Int. Workshop on Formal Approaches to Software Testing, FATES'08, LNCS 5047*, pages 200–215. Springer, 2008.

[26] R. M. Hierons, M. G. Merayo, and M. Núñez. Implementation relations and test generation for systems with distributed interfaces. *Distributed Computing*, 25(1):35–62, 2012.

[27] R. M. Hierons, M. G. Merayo, and M. Núñez. Timed implementation relations for the distributed test architecture. *Distributed Computing*, 27(3):181–201, 2014.

[28] R. M. Hierons and H. Ural. UIO sequence based checking sequences for distributed test architectures. *Information and Software Technology*, 45(12):793–803, 2003.

[29] R. M. Hierons and H. Ural. The effect of the distributed test architecture on the power of testing. *The Computer Journal*, 51(4):497–510, 2008.

[30] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 3rd edition, 2006.

[31] C. Jard, T. Jéron, H. Kahlouche, and C. Viho. Towards automatic distribution of testers for distributed conformance testing. In *TC6 WG6.1 Joint Int. Conf. on Formal Description Techniques and Protocol Specification, Testing and Verification, FORTE'98*, pages 353–368. Kluwer Academic Publishers, 1998.

[32] A. Khoumsi. A temporal approach for testing distributed systems. *IEEE Transactions on Software Engineering*, 28(11):1085–1103, 2002.

[33] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, 28(9):690–691, 1979.

[34] G. Luo, R. Dssouli, and G. von Bochmann. Generating synchronizable test sequences based on finite state machine with distributed ports. In *6th IFIP Workshop on Protocol Test Systems, IWPTS'93*, pages 139–153. North-Holland, 1993.

[35] R. Marinescu, C. Seceleanu, H. Le Guen, and P. Pettersson. *A Research Overview of Tool-Supported Model-based Testing of Requirements-based Designs*, volume 98 of *Advances in Computers*, chapter 3, pages 89–140. Elsevier, 2015.

[36] A. W. Mazurkiewicz. Traces, histories, graphs: Instances of a process monoid. In *11th Symposium on Mathematical Foundations of Computer Science, MFCS'84, LNCS 176*, pages 115–133. Springer, 1984.

[37] G. J. Myers, C. Sandler, and T. Badgett. *The Art of Software Testing*. John Wiley & Sons, 3rd edition, 2011.

[38] H. N. Nguyen, F. Zaïdi, and A. R. Cavalli. A framework for distributed testing of timed composite systems. In *21st Asia-Pacific Software Engineering Conference, APSEC'14*, pages 47–54. IEEE Computer Society, 2014.

[39] H. Ponce de León, S. Haar, and D. Longuet. Unfolding-based test selection for concurrent conformance. In *25th IFIP WG 6.1 Int. Conf. on Testing Software and Systems, ICTSS'13, LNCS 8254*, pages 98–113. Springer, 2013.

[40] H. Ponce de León, S. Haar, and D. Longuet. Distributed testing of concurrent systems: Vector clocks to the rescue. In *11th Int. Colloquium on Theoretical Aspects of Computing, ICTAC'14, LNCS 8687*, pages 369–387. Springer, 2014.

[41] H. Ponce de León, S. Haar, and D. Longuet. Model-based testing for concurrent systems: unfolding-based test selection. *International Journal on Software Tools for Technology Transfer*, 18(3):305–318, 2016.

[42] O. Rafiq and L. Cacciari. Coordination algorithm for distributed testing. *The Journal of Supercomputing*, 24(2):203–211, 2003.

[43] B. Sarikaya and G. von Bochmann. Synchronization and specification issues in protocol testing. *IEEE Transactions on Communications*, 32:389–395, 1984.

[44] M. Shafique and Y. Labiche. A systematic review of state-based test tools. *International Journal on Software Tools for Technology Transfer*, 17(1):59–76, 2015.

[45] H. Ural and C. Williams. Constructing checking sequences for distributed testing. *Formal Aspects of Computing*, 18(1):84–101, 2006.

[46] T. Walter, I. Schiferdecker, and J. Grabowski. Test architectures for distributed systems: State of the art and beyond. In *11th IFIP Workshop on Testing of Communicating Systems, IWTCS'98*, pages 149–174. Kluwer Academic Publishers, 1998.

APPENDIX
PROOFS OF MAIN RESULTS

Proposition 1 Let $\sigma \in \mathcal{A}ct^*$ be a sequence of actions. The pair $([\sigma], d^1)$ is a metric space.

Proof: Given $\sigma_1, \sigma_2, \sigma_3 \in [\sigma]$ we have to prove the following conditions.

1) $d^1(\sigma_1, \sigma_2) \geq 0$. Trivial.
2) $d^1(\sigma_1, \sigma_2) = 0$ if and only if $\sigma_1 = \sigma_2$. Also trivial.
3) $d^1(\sigma_1, \sigma_2) = d^1(\sigma_2, \sigma_1)$. It is clear that the minimum number of swaps between events to transform σ_2 into σ_1 is equal to the minimum number required to transform σ_1 into σ_2.
4) $d^1(\sigma_1, \sigma_2) \leq d^1(\sigma_1, \sigma_3) + d^1(\sigma_3, \sigma_2)$. We will prove this by contradiction, assuming that $d^1(\sigma_1, \sigma_2) > d^1(\sigma_1, \sigma_3) + d^1(\sigma_3, \sigma_2)$. In this case, we can transform σ_1 into σ_3 and then σ_3 into σ_2 and this can be done with $d^1(\sigma_1, \sigma_3) + d^1(\sigma_3, \sigma_2)$ swaps. This number is smaller than $d^1(\sigma_1, \sigma_2)$ and, by definition, $d^1(\sigma_1, \sigma_2)$ is the smallest number of swaps we must perform to obtain σ_2 from σ_1. We thus obtain a contradiction as required.

■

Theorem 1 Let $M \in \texttt{IOTS}(I, O, \mathcal{P}orts)$ and $k \geq 0$. We have that $L^1_k(M) = L(\mathcal{M}_a(M, k))$.

Proof: The proof will be by induction on k. The result holds immediately for the base case $k = 0$ since, by definition, $\mathcal{M}_a(M, 0) = M$ and $L^1_0(M) = L(M)$.

We now consider the inductive case and assume that the result holds for all values less than k, $k > 0$, and we are required to prove that the result holds for k.

We first prove the left to right inclusion. Let σ be a trace of $L^1_k(M)$. If $\sigma \in L^1_{k-1}(M)$ then the result follows immediately from the inductive hypothesis. Now assume that $\sigma \notin L^1_{k-1}(M)$. By definition, we have that there exists a trace $\sigma' \in L^1_{k-1}(M)$ such that $\sigma' \sim \sigma$ and $d^1(\sigma, \sigma') \leq 1$. By the inductive hypothesis, $\sigma' \in L(\mathcal{M}_a(M, k-1))$. Thus, there exists a path in $\mathcal{M}_a(M, k-1)$ from the initial state that reaches a final state and has label σ'. By the construction of $\mathcal{M}_a(M, k)$ there also exists a path labelled by σ' in $\mathcal{M}_a(M, k)$. In this path we only have two consecutive transitions labelled by two actions that are exchanged with respect to σ. We replace these transitions by the new transitions generated in the construction of $\mathcal{M}_a(M, k)$ that represent the events being swapped; this is allowed since the events must have been at different ports. In this way, we obtain a path in $\mathcal{M}_a(M, k)$ labelled by σ that reaches a final state. Therefore, we have that $\sigma \in L(\mathcal{M}_a(M, k))$ as required.

Now we prove the right to left inclusion. Let σ be a trace of $L(\mathcal{M}_a(M, k))$. Therefore, there exists a trace of $\mathcal{M}_a(M, k)$ labelled by σ that reaches a final state. If σ is a trace of $\mathcal{M}_a(M, k-1)$ then the result follows immediately from the inductive hypothesis. Otherwise, by the definition of $\mathcal{M}_a(M, k)$ we have that $\mathcal{M}_a(M, k-1)$ has transitions $t = (q_1, a, q_2)$ and $t' = (q_2, b, q_3)$ with $port(a) \neq port(b)$ such that

$\mathcal{M}_a(M,k)$ has a path of the form $\rho_1(q_1^1, b, q_{tt'})(q_{tt'}, a, q_3^2)\rho_2$ whose label is σ. Further, ρ_1 is equivalent to a path of $\mathcal{M}_a(M,k-1)$ from the initial state (once states are relabelled) and ρ_2 is equivalent to a path of $\mathcal{M}_a(M,k-1)$ from state q_3 (once states are relabelled). Thus, we have that $label(\rho_1)\ a\ b\ label(\rho_2) \in L(\mathcal{M}_a(M,k-1))$. By the inductive hypothesis, $label(\rho_1)\ a\ b\ label(\rho_2) \in L^1_{k-1}(M)$. The result now follows from a and b being at different ports, the definition of $L^1_k(M)$ and by taking into account that $L^1_k(M)$ is equal to

$$L^1_{k-1}(M) \cup \{\sigma_1 ba\sigma_2 | \sigma_1 ab\sigma_2 \in L^1_{k-1}(M) \wedge port(a) \neq port(b)\}$$

■

Proposition 3 Let $M \in \texttt{IOTS}(I,O,\mathcal{P}orts)$ be a finite IOTS and $k \geq 0$. We have that $\mathcal{M}_a(M,k)$ has $O(|T|^{2^k})$ transitions, where T is the finite set of transitions of M.

Proof: Let T_k be the transition set of $\mathcal{M}_a(M,k)$ and T_{k-1} be the transition set of $\mathcal{M}_a(M,k-1)$. The reasoning is essentially inductive: we observe that in building $\mathcal{M}_a(M,k)$ we include two copies of each transition of $\mathcal{M}_a(M,k-1)$ and also every pair of consecutive transitions t and t' has the potential to be replicated (if the events are at different ports). Since $|T_{k-1}|^2$ provides an upper bound on the number of pairs of successive transitions in $\mathcal{M}_a(M,k-1)$, and at most two transitions are added for each such pair, we have that

$$|T_k| \leq 2 \cdot |T_{k-1}| + 2 \cdot |T_{k-1}|^2$$

Since the quadratic term dominates we have that

$$|T_k| = O(|T_{k-1}|^2)$$

and so the result follows by a simple inductive argument. ■

Theorem 2 Let $M, N \in \texttt{IOTS}(I,O,\mathcal{P}orts)$ be finite IOTSs and $k \geq 0$. The corresponding correctness problem, given d^1 and k, is in 2-EXPSPACE and is PSPACE-hard.

Proof: The correctness problem reduces to deciding whether $L_\delta(N) \subseteq L^1_k(M)$ and so $L_\delta(N) \subseteq L(\mathcal{M}_a(M,k))$. This is an instance of the PSPACE-complete regular language inclusion problem in which $\mathcal{M}_a(M,k)$ has size that is double-exponential. Since regular language inclusion is in PSPACE [14], this shows that the correctness problem is in 2-EXPSPACE.

The problem being PSPACE-hard follows immediately from the fact that if we set $k = 0$ then we can capture all instances of the PSPACE-complete regular language inclusion problem. ■

Theorem 3 Let $M, N \in \texttt{IOTS}(I,O,\mathcal{P}orts)$ be finite IOTSs and $k \geq 0$. The corresponding bounded correctness problem, given d^1 and k, is PSPACE-complete.

Proof: Again, the correctness problem reduces to deciding whether $L_\delta(N) \subseteq L(\mathcal{M}_a(M,k))$. This is an instance of the PSPACE-complete regular language inclusion problem and the size of $\mathcal{M}_a(M,k)$ is bounded above by a polynomial in terms of the size of M. The problem is thus in PSPACE.

The problem being PSPACE-hard again follows immediately from the case $k = 0$ and the regular language inclusion problem being PSPACE-complete. ■

Theorem 4 Let $M \in \texttt{IOTS}(I,O,\mathcal{P}orts)$ be a finite IOTS, $\sigma \in \mathcal{A}ct^*$ be a sequence of actions and $k \geq 0$. The problem of deciding whether M has a trace $\sigma' \sim \sigma$ such that $d^1(\sigma,\sigma') \leq k$ is NP-complete.

Proof: We first prove that the problem is in NP. A non-deterministic Turing machine can guess a sequence of at most k swaps to form a trace σ'. It then simply checks whether $\sigma' = \sigma$. This process can be performed in polynomial time and so the problem is in NP.

We now prove that the problem is NP-hard. Given trace σ of length ℓ, it is straightforward to see that ℓ^ℓ provides an upper bound on the size of $[\sigma]$ and thus on $d^1(\sigma,\sigma')$ for $\sigma' \sim \sigma$. Thus, if we set $k = \ell^\ell$ then the membership problem for M, k and σ is exactly that of deciding whether there is some $\sigma' \sim \sigma$ such that $\sigma' \in L(M)$. However, the problem of deciding whether there is some $\sigma' \sim \sigma$ such that $\sigma' \in L(M)$ is NP-hard [19]. Since ℓ^ℓ can be stored in polynomial space, taking into account that $O(log_2(\ell^\ell)) = O(\ell \cdot log_2(\ell))$, if we can solve the membership problem for bounded k in polynomial time then we have a polynomial time solution to the NP-hard problem of deciding whether there is some $\sigma' \sim \sigma$ such that $\sigma' \in L(M)$. The result therefore follows. ■

Theorem 5 Let $M \in \texttt{IOTS}(I,O,\mathcal{P}orts)$ be a finite IOTS, $\sigma \in \mathcal{A}ct^*$ be a sequence of actions and $k \geq 0$. If we have an upper bound on k then the problem of deciding whether M has a trace $\sigma' \sim \sigma$ such that $d^1(\sigma,\sigma') \leq k$ can be solved in polynomial time.

Proof: As before, since we have an upper bound on k, the size of $\mathcal{M}_a(M,k)$ is polynomial in terms of the size of M. Thus, the membership problem reduces to that of deciding whether $\sigma \in L(\mathcal{M}_a(M,k))$ for a finite automaton $\mathcal{M}_a(M,k)$ of polynomial size. The result now follows from there being polynomial time solutions to the problem of deciding whether a word is recognised by a finite automaton [30]. ■

Proposition 4 Let $\sigma \in \mathcal{A}ct^*$ be a sequence of actions. The pair $([\sigma], d^2)$ is a metric space.

Proof: Given $\sigma_1, \sigma_2, \sigma_3 \in [\sigma]$ we must prove that the following four conditions hold in order to ensure that d^2 is indeed a distance.

1) $d^2(\sigma_1,\sigma_2) \geq 0$. Trivial because we compute the maximum of non-negative numbers.
2) $d^2(\sigma_1,\sigma_2) = 0$ if and only if $\sigma_1 = \sigma_2$. We have that $d^2(\sigma_1,\sigma_2) = 0$ if and only if $\forall\ e \in E(\sigma_1) : |pos_{\sigma_1}(e) - pos_{\sigma_2}(e)| = 0$. The previous statement is equivalent to having that $\forall\ e \in E(\sigma_1) : pos_{\sigma_1}(e) = pos_{\sigma_2}(e)$ and this is trivially equivalent to having $\sigma_1 = \sigma_2$.
3) $d^2(\sigma_1,\sigma_2) = d^2(\sigma_2,\sigma_1)$. Trivial because $\forall\ e \in E(\sigma_1) : |pos_{\sigma_1}(e) - pos_{\sigma_2}(e)| = |pos_{\sigma_2}(e) - pos_{\sigma_1}(e)|$.
4) $d^2(\sigma_1,\sigma_2) \leq d^2(\sigma_1,\sigma_3) + d^2(\sigma_3,\sigma_2)$. Let $e^m \in E(\sigma_1)$ be an event such that $d^2(\sigma_1,\sigma_2) = |pos_{\sigma_1}(e^m) - pos_{\sigma_2}(e^m)|$. Without loss of generality, let us suppose that $pos_{\sigma_1}(e^m) \leq pos_{\sigma_2}(e^m)$ (the other case is symmetric). Then, one of the following holds:
   - $pos_{\sigma_3}(e^m) \leq pos_{\sigma_1}(e^m)$ and $pos_{\sigma_3}(e^m) \leq$

$pos_{\sigma_2}(e^m)$. In this case we have:

$$\begin{aligned} d^2(\sigma_1,\sigma_2) &\leq |pos_{\sigma_3}(e^m) - pos_{\sigma_2}(e^m)| \\ &\leq d^2(\sigma_3,\sigma_2) \\ &\leq d^2(\sigma_1,\sigma_3) + d^2(\sigma_3,\sigma_2) \end{aligned}$$

- $pos_{\sigma_3}(e^m) \geq pos_{\sigma_1}(e^m)$ and $pos_{\sigma_3}(e^m) \geq pos_{\sigma_2}(e^m)$. In this case we have:

$$\begin{aligned} d^2(\sigma_1,\sigma_2) &\leq |pos_{\sigma_3}(e^m) - pos_{\sigma_1}(e^m)| \\ &\leq d^2(\sigma_1,\sigma_3) \\ &\leq d^2(\sigma_1,\sigma_3) + d^2(\sigma_3,\sigma_2) \end{aligned}$$

- $pos_{\sigma_1}(e^m) \geq pos_{\sigma_3}(e^m) \geq pos_{\sigma_2}(e^m)$. In this case we have:

$$\begin{aligned} d^2(\sigma_1,\sigma_2) &= |pos_{\sigma_1}(e^m) - pos_{\sigma_3}(e^m)| \\ &\quad + \\ &\quad |pos_{\sigma_3}(e^m) - pos_{\sigma_2}(e^m)| \\ &\leq d^2(\sigma_1,\sigma_3) + d^2(\sigma_3,\sigma_2) \end{aligned}$$

■

Proposition 6 Let σ_1, σ_2 be such that $\sigma_1 \sim \sigma_2$ and $d^2(\sigma_1,\sigma_2) = k$ for some $k \geq 0$. Then it is possible to rewrite σ_1 to σ_2 through at most $k \cdot (k+1)$ steps.

Proof: We will use proof by induction on k. The result clearly holds for the base case $k = 0$ since we then have that $\sigma_1 = \sigma_2$.

Inductive hypothesis: $k > 0$ and the result holds for all distances less than k. Now consider σ_1, σ_2 such that $\sigma_1 \sim \sigma_2$ and $d^2(\sigma_1,\sigma_2) = k$. Let $\sigma_1 = a^1 \ldots a^n$.

Let us suppose that one or more elements of σ_1 are k-early. We find all a^i such that: a^i is k-early and a^{i+1} is not k-early. Clearly, if one or more elements of σ_1 are k-early then there must be some such a^i. Given such an a^i, a^{i+1} cannot be $(k-1)$-early since otherwise we would have that a^i and a^{i+1} correspond to the same element of σ_2, which is not possible. Thus, we have that a^{i+1} is not k-early and also is not $k-1$-early. Thus, if we apply the swap $a^i a^{i+1} \rightarrow a^{i+1}a^i$ in σ_1 then we obtain a sequence in which the number of letters that are k-early is reduced and the number of letters that are k-late does not change. Further, we can apply this process to all such a^i (a^i is k-early and a^{i+1} is not k-early) at the same time in one step. Since continuous blocks of elements in σ_1 that are k-early cannot have length greater than k, we can repeat this process at most k times (at most k steps) to obtain a word σ_3 in which no letters are k-early.

Now consider any letters of σ_3 that are k-late. Similar to before, we can find all a^i such that: a^i is k-late and a^{i-1} is not k-late. Using similar reasoning to the above case, a^{i-1} cannot be $(k-1)$-late. Thus, if we apply the swap $a^{i-1}a^i \rightarrow a^i a^{i-1}$ in σ_3 then we obtain a sequence in which the number of letters that are k-late is reduced and the number of letters that are k-early does not change. Further, we can apply this process to all such a^i (a^i is k-late and a^{i-1} is not k-late) at the same time in one step. Since continuous blocks of elements in σ_1 that are k-late cannot have length greater than k, we can repeat this process at most k times to obtain a word σ_4 in which no letters are k-late or k-early.

We now observe that $\sigma_4 \sim \sigma_2$ and $d^2(\sigma_4,\sigma_2) < k$. We can now apply the inductive hypothesis, to σ_4 and σ_2, and this tells us that it is possible to rewrite σ_4 to σ_2 through at most $(k-1)\cdot(k-1+1) = k\cdot(k-1)$ steps. Thus, since it is possible to rewrite σ_1 to σ_4 in at most $2 \cdot k$ steps, the total number of steps required to rewrite σ_1 to σ_2 is at most $k \cdot (k-1) + 2 \cdot k$ and this is equal to $k \cdot (k+1)$. The result thus follows. ■

Theorem 10 Let $M \in \texttt{IOTS}(I, O, \mathcal{P}orts)$ be a finite IOTS, $\sigma \in \mathcal{A}ct^*$ be a sequence of actions and $k \geq 0$. If we have an upper bound on k then the problem of deciding whether M has a trace $\sigma' \sim \sigma$ such that $d^2(\sigma,\sigma') \leq k$ can be solved in polynomial time.

Proof: First observe that the number of states of $\mathcal{M}_b(\sigma, k)$ is equal to the number of subsets of $E(\sigma)$ that are downwardly closed under $\prec_k$. In addition, a downwardly closed set is uniquely defined by the anti-chain formed from the maximal elements of that set [10]. However, for two elements e^i and e^j to be in an anti-chain we require that $|j-i| < 2 \cdot k$. Thus, the number of anti-chains in $E(\sigma)$ is bounded above by the number of ways of choosing at most $2 \cdot k$ elements from $E(\sigma)$. In turn, this is bounded above by $|\sigma|^{2 \cdot k}$. For bounded k this is a polynomial. The problem now reduces to deciding whether there is a trace accepted by both M and $\mathcal{M}_b(\sigma, k)$, a problem that can be solved in polynomial time. ■

Theorem 11 Let $M \in \texttt{IOTS}(I, O, \mathcal{P}orts)$ be a finite IOTS, $\sigma \in \mathcal{A}ct^*$ be a sequence of actions and $k \geq 0$. If we have an upper bound on the number of ports then the problem of deciding whether M has a trace $\sigma' \sim \sigma$ such that $d^2(\sigma,\sigma') \leq k$ can be solved in polynomial time.

Proof: We again consider the number of states. We now observe that an anti-chain of $E(\sigma)$ can contain at most one element for each port. Thus, the number of anti-chains, and so states, is bounded above by $|\sigma|^m$, which is a polynomial since m is bounded above. Again, we observe that the problem reduces to deciding whether there is a trace accepted by both M and $\mathcal{M}_b(\sigma, k)$, a problem that can be solved in polynomial time. ■