ELSEVIER

Artificial Intelligence 107 (1999) 63–98

Artificial Intelligence

Reasoning about nondeterministic and concurrent actions: A process algebra approach [☆]

Xiao Jun Chen [a,1], Giuseppe De Giacomo [b,*]

[a] *School of Computer Science, University of Windsor, 401 Sunset Avenue, Windsor, Ontario, Canada N9B 3S7*

[b] *Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Via Salaria 113, 00198 Roma, Italy*

Received 10 December 1996; received in revised form 12 May 1998

Abstract

We present a framework for reasoning about *processes* (complex actions) that are constituted by several concurrent activities performed by various interacting agents. The framework is based on two distinct formalisms: a representation formalism, which is a CCS-like process algebra associated with an explicit *global store*; and a reasoning formalism, which is an extension of modal mu-calculus, a powerful logic of programs that subsumes dynamic logics such as *PDL* and *ΔPDL*, and branching temporal logics such as *CTL* and *CTL**. The reasoning service of interest in this setting is *model checking* in contrast to logical implication. This framework, although directly applicable only when complete information on the system behavior is available, has several interesting features for reasoning about actions in Artificial Intelligence. Indeed, it inherits formal and practical tools from the area of Concurrency in Computer Science, to deal with complex actions, treating suitably aspects like nonterminating executions, parallelism, communications, and interruptions. © 1999 Published by Elsevier Science B.V. All rights reserved.

Keywords: Process algebra; Modal mu-calculus; Reasoning about actions; Concurrency; Model checking; Logical implication

1. Introduction

In this paper, we present a piece of research that can be regarded as a bridge between the area of Reasoning about Actions in Artificial Intelligence and the area of Concurrency in Computer Science.

[☆] A shorter version of this paper appeared in the Proceeding of the Thirteenth National Conference on Artificial Intelligence (AAAI-96).

[*] Corresponding author. Email: degiacomo@dis.uniroma1.it.

[1] Email: xjchen@cs.uwindsor.ca.

On the one hand, we follow a methodology that is typical of Reasoning about Actions in Artificial Intelligence for specifying and reasoning about dynamic systems (e.g., see the situation calculus in [49]): introducing a set of facts whose value changes as the system evolves (cf. *fluents* in [49]); specifying *effects* of (atomic) actions on such a set of facts (cf. effect axioms in [49]); devising a suitable means to obtain the *successor-state* resulting from executing an action in the current state (cf. successor-state axioms in [49]). We also introduce a specification of *preconditions* for executing actions (cf. precondition axioms in [49]). However, we allow such a specification to change during the evolution of the system (differently from precondition axioms in [49]). In addition to this general picture, we allow for multiple atomic actions to occur together (for reasons that will become clear later on, we call the resulting actions *synchronized actions* instead of concurrent (elementary) actions as, e.g., in [2,5,24,42,50]), and we allow for organizing actions within suitable control structures (sequential composition, parallel composition, iteration, recursion, etc.) by introducing an explicit notion of *process* in describing the system.

On the other hand, we make use of modeling tools that have been developed in the area of Concurrency in Computer Science to formalize concurrent processes, instead of the ones typically used in Reasoning about Action in Artificial Intelligence (i.e., logics). However, in order to make use of such tools, we need to describe the dynamic system on a more concrete level of abstraction than the one typically adopted in Reasoning about Actions.

In general, we may choose among several levels of abstraction when describing a dynamic system, depending on the information we assume available. We distinguish the following three levels:

(1) At a very concrete level, we may characterize the system by its unique *actual evolution*, which can be represented as a sequence of states/actions. At this level, we assume complete information on each state, and we assume knowledge of which action will be performed next.

(2) At a more abstract level, we may characterize the system by all its possible evolutions. In this case, the system is represented as a transition graph, called *transition system*, instead of a single sequence. The single evolution at level 1 is represented as a *path* on such a graph. One of these paths is going to be the actual evolution of the system, yet we do not have the knowledge on which one it is. Each node (representing a state) has several labeled out-arcs which represent the actions that can be performed in that state. Each action causes the transition of the system from the current state to a possible successor-state. We remark that different out-arcs may be labeled by the same action. In this case, the action has several alternative outcomes: the action is *nondeterministic*. At this level, we assume complete information on the possible evolutions of the system: each state is completely known, including which actions enabled to be performed, and each action leads to some completely known state. However, we do not know which action is going to be performed next. Moreover, for nondeterministic actions, it is not known which of the alternative resulting states is going to be the next one.

(3) At the third level, we model the system by selecting a set of transition systems instead of a unique one. Each of such transition systems represents an alternative possible behavior. At this level, we assume partial information on the possible

evolutions of the system: each state is only partially known, and so are the states resulting from performing an action in it.

Generally, level 1 is considered too concrete: it is unrealistic to obtain such complete information in order to single out a unique system run.[2] Levels 2 and 3 instead, have been both used in modeling dynamic systems.

In particular, level 3 is the one usually adopted by research in reasoning about actions [24,37,38,41,42,49,51], where a certain *logic* (situation calculus, dynamic logic, etc.) is used both to represent and to reason about the dynamic systems. The typical reasoning problem of interest in this case is *logical implication* (validity) in the form

$$\Gamma \models S_{init} \Rightarrow \Phi$$

where Γ are axioms used to select the set of transition systems that represent the dynamic system; S_{init} is a formula, which is a (partial) description of the initial state; Φ is a formula, which is the property we want to prove, e.g., the reachability of a state where a certain property (the goal) holds.

In this paper, we adopt the viewpoint of level 2. Following the model checking approach proposed in [27], we use a *representation formalism* to define the *transition system* representing the possible evolutions of the system, and a *reasoning formalism* (a suitable *logic*) for specifying properties we want to check. This framework is the one typically used in process algebras (e.g., CCS [45], CSP [30], ACP [3]) to model concurrent and reactive systems.[3] Process algebras are generally recognized as a convenient tool for describing concurrent and multiprocess systems. They provide us with a clean way to express parallelism, reactivity, communications, interruptions, coordinations, synchronizations/asynchronizations, etc. Moreover, for finite state processes (processes that can be interpreted on finite transition systems), various practical tools have been developed and implemented to verify whether a given modal/temporal logic formula is satisfied by the process (e.g., [6,10,11,44]).

The reasoning problem of interest in this case is *model checking* in the form

$$\mathcal{T}, s_{init} \models \Phi$$

where $\mathcal{T}$ is a transition system representing the possible evolutions of our dynamic system; s_{init} is the initial state of $\mathcal{T}$; Φ is a formula, which is the property we want to prove, e.g., the reachability of a state where a certain property (the goal) holds.

In our work, the state of the system called *configuration*, is composed of an active component called *process* and a passive component called *global store*. The process describes the activities of all the *agents* (e.g., robots, persons, pieces of software, subroutines, environment, etc.) in the system. The global store, which is characterized by a set of primitive propositions, describes the state of the world except for the activities that are going on. The configuration can only be changed by the activities in the process, which in fact make the system evolve.

[2] However, one can describe a dynamic system by specifying its properties using Linear-time Temporal Logics that are interpreted over system runs (see, e.g., [18]).

[3] In Artificial Intelligence, research in search-based planning, including much work on STRIPS (e.g., [8]) can be considered at this level. In contrast, research in deductive planning is typically carried out at level 3.

Making use of a global store associated to a process, we specify the effects of an action in terms of the difference between the current global store and the resulting one. Properties not mentioned among such effects are kept unchanged.[4] Note that this treatment is different from most of the approaches in the literature on logics of programs [34], where all properties of the state resulting from an action must be specified explicitly.

In order to reason about the properties of such modeled dynamic systems, we develop a suitable extension of *modal mu-calculus* [32], a powerful logic of programs which subsumes dynamic logics such as *PDL*, *ΔPDL* [34], and temporal logics such as *CTL*, *CTL** [18]. We show that model checking in our logic can be linearly reduced to model checking in standard modal mu-calculus. By means of this reduction, it is possible to reuse efficiently the existing verification tools mentioned above, for reasoning about actions in our setting.

We also discuss two important additional issues: (1) the relationship between model checking and logical implication in our setting; (2) how to identify two representations of a dynamic system. For the former, we device a suitable notion of *characteristic formula* [53], which is a logical formula that completely characterizes a transition system. For the latter, we introduce a suitable notion of equivalence based on *bisimulation* [45].

The rest of the paper is organized as follows. In Section 2, we present our representation formalism in detail. In Section 3, we discuss the main feature of the representation formalism by illustrating several examples. In Section 4, we present the reasoning formalism and show its ability to express a wide variety of dynamic properties. In Section 5, we show by means of examples, the use of the reasoning formalism for reasoning about actions in the proposed framework. In Section 6, we devise a suitable reasoning technique for model checking in our setting, by reducing it to a standard setting. In Section 7, we discuss the relationship between model checking and logical implication, and the issue of equivalent descriptions. In Section 8, we draw some conclusions and sketch some possible future research directions.

2. Representation formalism: A process algebra with global store

We represent dynamic systems in terms of possible evolutions of the system caused by actions. We call *configuration* the state of the system at a point of its possible evolutions. A configuration is represented as a pair:

$$(p, \sigma)$$

where p is called *process* and σ is called *global store*. Intuitively, the process describes all the activities that are being performed by the *agents* in the system—or to be precise, the status of such activities in the current configuration. The global store describes the properties characterizing the current configuration that do not involve activities being performed. As the activities in the process are performed, both the global store and the process evolve, and hence the configuration of the system changes.

[4] In this way we address the simplified variant of the frame problem that arises in our setting.

We formalize possible global stores simply as propositional interpretations. Let *Prop* be a finite set of propositions of propositional logic (ranged over by $A, B, \ldots$, possibly with a subscript). A global store σ is a propositional interpretation over *Prop*. Given a global store σ and a proposition $A \in Prop$, $\sigma(A) = tt$ if the fact denoted by A is true in σ and $\sigma(A) = ff$ if the fact denoted by A is false in σ.

Before formalizing processes, we need to introduce *elementary actions*. Indeed, a process carries the information on "which elementary actions are *possible next*", and for each of such actions, "what is the *process left to be performed* afterwards".

We consider two kinds of elementary actions:
- *Atomic actions* which are basic uninterruptible actions executed by an agent. We assume the set of all possible atomic actions to be finite.
- *Synchronized actions* which are constituted by any nonempty set of atomic actions performed together by various agents. Intuitively, to execute a synchronized action means to execute a set of atomic actions simultaneously *as a unity*. That is, the execution of each atomic action in a synchronized action, implicitly relies on the feedback of the executions of the others.

Each action has some effects on the global store. The specification of such effects is supplied separately from the process by defining an effect function. The effect function specifies the effects of each atomic action with respect to different conditions on the global store. On the base of such an effect function, a successor-state function is defined which, given a global store and an atomic or synchronized action, returns the set [5] of possible next global stores. [6]

Besides effects, each action has typically some associated preconditions, i.e., conditions under which an action can be performed. In our setting, action preconditions are specified within the process. This treatment provides us with the capability of describing action preconditions which depend not only on the status of the global store but also on the status of the process. So a process can, for instance, dynamically block the possibility of executing an action in some configurations when certain activities are being performed.

2.1. Atomic actions

Let *Act* be the finite set of all possible atomic actions (ranged over by a, $b, \ldots$, possibly with a subscript).

We define an effect function *effct* that associates to each action $a \in Act$, a finite set of pairs of *premise* and *effect*:

$$effct(a) = \{(\psi_1, E_1), \ldots, (\psi_n, E_n)\}$$

where for each pair (ψ_i, E_i):
- The premise ψ_i is a propositional formula over *Prop* describing the properties the global store must satisfy so that the corresponding effect E_i can be applied.

[5] Recall that actions are generally nondeterministic, so we have a set of (rather than a single) possible next global stores.

[6] Such an approach for specifying effects is quite similar to that of the $\mathcal{A}$-family action languages [23,39].

- The effect E_i is a set of literals—atomic propositions or their negations—over *Prop* that describes a possible effect of the execution of action a under premise ψ_i. The literals in E_i are required to be true in the successive global store.

Each pair (ψ_i, E_i) in *effct*(a), can be intuitively interpreted as an assertion that if the premise ψ_i is true in the current global store, then there is a possible execution of a that causes the literals in E_i to be true in the resulting global store. In other words, the action a under the premise ψ_i has E_i as possible effect.

Next, we introduce a simple update operator $\circ$ that, given an interpretation σ and a set of non-contradictory literals $\mathcal{L}$, returns a new interpretation σ'.

Definition 2.1. Let σ be an interpretation over *Prop* and $\mathcal{L}$ a set of non-contradictory literals over *Prop*. We define the update operator $\circ$ (infix) as follows: $\forall A \in Prop$,

$$(\sigma \circ \mathcal{L})(A) = \begin{cases} \mathit{tt} & A \in \mathcal{L}, \\ \mathit{ff} & \neg A \in \mathcal{L}, \\ \sigma(A) & A \notin \mathcal{L} \text{ and } \neg A \notin \mathcal{L}. \end{cases}$$

Intuitively, the operator returns an interpretation that satisfies the literals in $\mathcal{L}$, and retains the value of the original interpretation σ for those literals not occurring in $\mathcal{L}$.

Making use of the above update operator, we define a successor-state function that specifies how an atomic action affects the global store.

Definition 2.2. Let a be an atomic action, and σ a global store (i.e., an interpretation over *Prop*). The set of possible global stores obtained by executing a in σ, denoted by σ/a, is the set of all interpretations:

$$\sigma \circ \mathcal{L}$$

such that

- $\mathcal{L} = \begin{cases} E & \text{if } \exists \psi \text{ s.t. } (\psi, E) \in \mathit{effct}(a) \text{ and } \sigma(\psi) = \mathit{tt}, \\ \emptyset & \text{otherwise.} \end{cases}$
- $\mathcal{L}$ does not contain contradictory literals.

Intuitively, the set σ/a of alternative global stores resulting from executing action a on σ is formed by one alternative updated global store for each effect E of a whose premise ψ is satisfied in σ. For each E, the resulting global store is equal to $\sigma \circ E$, that is, it is identical to σ except that the values of the atomic propositions occurring in E are changed so as to make E true.

Observe that action a is nondeterministic (with respect to the effects on the global store) in σ—i.e., σ/a is not singleton—if more then one premise in *effct*(a) is satisfied. It is deterministic (with respect to the effects on the global store)—i.e., σ/a is singleton[7] —if just one premise is satisfied.

[7] Note that even if σ/a is singleton, there may still be more than one resulting configurations since the current *process* may evolve in several possible ways by performing a.

By definition, if no premise in *effct*(a) is satisfied then $\sigma/a = \sigma$, i.e., the action has no effect (though it may still be performed).

Finally, if an effect E in *effct*(a) is contradictory, it will not generate a possible resulting global store in σ/a. In particular, if every effect E that has its premise satisfied is contradictory, then action a cannot be executed (thus influencing the preconditions for performing a). [8]

2.2. *Synchronized actions*

For synchronized actions (ranged over by α, possibly with a subscript), we use set notations with their obvious meanings. In particular, we denote by $\{a_1, \ldots, a_n\}$ the synchronized action composed by $a_1, \ldots, a_n \in Act$. Observe that as a special case, we have the synchronized action $\{a\}$ which is in fact simply the atomic action a, i.e., every atomic action is vacuously a synchronized action as well.

We extend the previously defined successor-state function so as to cope with synchronized actions as follows.

Definition 2.3. Let $\alpha = \{a_1, \ldots, a_n\}$, with $n \geqslant 1$, be a synchronized action, and σ a global store (i.e., an interpretation of the propositions in *Prop*). The set of possible global stores obtained by executing α, denoted by σ/α, is the set of all interpretations:

$$\sigma \circ (\mathcal{L}_1 \cup \cdots \cup \mathcal{L}_n)$$

such that, for $i = 1, \ldots, n$,

- $\mathcal{L}_i = \begin{cases} E & \text{if } \exists\psi \text{ s.t. } (\psi, E) \in \mathit{effct}(a_i) \text{ and } \sigma(\psi) = tt, \\ \emptyset & \text{otherwise.} \end{cases}$
- $\mathcal{L}_1 \cup \cdots \cup \mathcal{L}_n$ does not contain contradictory literals.

Observe that $\sigma/\{a\} = \sigma/a$. Intuitively, the effects of a synchronized action are the *sum* of the effects of the participating atomic actions. For example, let a and b be two atomic actions whose applicable effects in a given configuration σ are $\{A\}$, $\{B\}$ for a and $\{C, D\}$ for b. That is, a is nondeterministic and its effect is either to set A to true, or to set B to true in the resulting global store, while b is deterministic and its unique effect is to set both C and D to true. The effect of the synchronized action $\{a, b\}$ is either to set A, C, D to true in the resulting global store, or to set B, C, D to true. In other words, $\{a, b\}$ nondeterministically leads to a global store, where either A, C, D is true and all other atomic propositions, including B, remain unaffected, or B, C, D is true, and all other atomic propositions, including A, remain unaffected.

For actions a, a_1, a_2, with *effct*(a) = *effct*(a_1) = *effct*(a_2), it is easy to check that, if a is deterministic in σ—i.e., σ/a is singleton—then $\sigma/a = \sigma/\{a_1, a_2\}$. However, if a is nondeterministic in σ—i.e., σ/a is not singleton—then executing $\{a_1, a_2\}$ may have different effects with respect to executing a. Generally the nondeterminism of $\{a_1, a_2\}$

[8] In fact, it has often been noticed that state change's laws may influence preconditions of actions, see, for example, [40].

increases with respect to that of a: $\{a_1, a_2\}$ still has all the effects a has, but furthermore it allows to combine such effects in pairs. For example, consider some resources and two consumers each consuming one resource at a time. Their actions have the same effect: to consume one of the resource. If two consumers take the action simultaneously, then two resources will be taken out, while if only one consumer takes the action, there will be only one resource taken out.

Let us now consider action $\{a, b\}$, where the only applicable effect of a is $\{A\}$ and the only applicable effect of b is $\{\neg A\}$. Then the set of alternative global stores resulting from executing $\{a, b\}$ is empty: $\sigma/\{a, b\} = \emptyset$. This means that the synchronized action $\{a, b\}$ cannot be executed, i.e., the atomic actions a and b cannot be synchronized. In general, in our setting, the effects of the atomic actions that constitute a synchronized action must be compatible in order to perform the synchronized action. The intuition behind is this: synchronizing two actions means not only to perform them at the same time, but also to perform each of them taking into account the feedback from the others. Actions with conflict effects cannot be synchronized. For example, pushing and pulling a handle cannot be synchronized.

Observe the difference between performing together actions a and b when they take into account the feedback of each other (as we assume in synchronized action $\{a, b\}$) and performing together a and b when they are fully independent. If a and b are independent, it is reasonable to assume that they can be performed together even though they have contradictory effects.[9] The contradiction can be resolved into nondeterminism. For example, let $\{A\}$ and $\{\neg A\}$ be the only applicable effect of a and b, respectively. Both a and b try to set the proposition A to the desired value independently. Nondeterministically, one of the two actions has "the last word" and succeeds. Hence, two resulting situations are possible: one in which A is true, and another in which A is false. This intuition is formulated in our setting by adopting an interleaving semantics for concurrent processes as in CCS (see below).

Finally, note that we have assumed that the effects of synchronized actions are the sum of those of the component actions. This is sufficient for most of the purposes, especially when we consider the additional modeling power that processes give us. Several alternative proposals for specifying effects of "compound elementary actions" have been considered in the literature, e.g., [2,5,24,42,50]. Many of these proposals are compatible with our framework (especially those based on $\mathcal{A}$-family action languages [2,5]). In general, our framework applies whenever it is possible to provide a successor-state function from global stores to sets of global stores.

2.3. *Processes*

We adopt CCS-style constructs to combine elementary actions into *processes*. CCS, i.e., Calculus of Communicating Systems, is a well-known formalism for expressing concurrent processes, which includes processes constructs for sequence, choice, parallel composition, and restriction [45]. We suitably extend CCS in order to deal with global stores and synchronization of multiple actions.

[9] Discussions on this issue may be found in, e.g., [5].

Due to the appearances of recursions, process equations $P \doteq p$ are used to define processes. Here P is a process name and p is a process expression (or simply process). Each process name is associated with a unique process definition. We will use *Proc* to denote the set of process names.

Processes follow the syntax below:

$$p ::= nil \mid P \mid (\phi \rightarrow a).p \mid p_1 + p_2 \mid p_1 \parallel p_2 \mid p \backslash \gamma$$

where *nil* denotes a predefined atomic process, P is a process name defined in *Proc*, ϕ denotes a propositional formula over *Prop*, a denotes an atomic action in *Act*, and γ denotes a set of expressions of the form $\phi \rightarrow \varrho$ with ϕ a propositional formula over *Prop* and ϱ a propositional formula over *Act*.

Intuitively, process constructs have the following meaning:

(1) *nil* represents the termination of a process.

(2) $(\phi \rightarrow a).p$ is the process which, under the "precondition" ϕ, is capable of performing the action a, and then behaves as the process p. This term can be viewed as an extension of CCS-term $a.p$ where no preconditions are specified.

(3) $p_1 + p_2$ represents the alternative composition of p_1 and p_2.

(4) $p_1 \parallel p_2$, the parallel composition of p_1 and p_2, is the process which can perform any interleaving or synchronizations of the actions of p_1 and p_2.

(5) $p \backslash \gamma$ is the process obtained from p restricting the allowed actions to those satisfying the constraints in γ, i.e.:

$$\{\alpha \mid \forall(\phi \rightarrow \varrho) \in \gamma.(\sigma(\phi) = tt \text{ implies } \alpha(\varrho) = tt)\}$$

where $\sigma(\phi)$ denotes the truth-value of ϕ in σ, and $\alpha(\varrho)$ denotes the truth-value of ϱ in the interpretation over *Act* obtained by assigning the value *tt* to the atomic actions in α and *ff* to those in $Act - \alpha$. The restriction construct can be used to dynamically restrict the possibility of executing synchronized actions, thus specifying "*which* group of actions can be synchronized at *what* time". Note that this construct is an extension of the CCS construct $\cdot\backslash\gamma$, where γ is simply a set of atomic actions that are not allowed.

The semantics of a dynamic system is given in terms of the transition relation $\_ \xrightarrow{-} \_$ defined as the least relation satisfying the set of *structural rules* in Table 1. Such structural rules have the following schema:

$$\frac{\text{ANTECEDENT}}{\text{CONSEQUENT}} \ \text{SIDE-CONDITION}$$

which is to be interpreted logically as:

$$\forall(\text{ANTECEDENT} \wedge \text{SIDE-CONDITION} \Rightarrow \text{CONSEQUENT})$$

where $\forall(\ldots)$ stands for the universal closure of all free variables occurring in $(\ldots)$.[10] In case either the ANTECEDENT or the SIDE-CONDITION is missing, they are interpreted as *true*.

[10] Observe that, typically, ANTECEDENT, SIDE-CONDITION and CONSEQUENT share free variables.

Table 1
Structural rules

Act: $$\frac{}{((\phi \rightarrow a).p, \sigma) \xrightarrow{\{a\}} (p, \sigma')}, \quad \text{where } \sigma(\phi) = tt,\ \sigma' \in \sigma/\{a\}$$

Def: $$\frac{(p, \sigma) \xrightarrow{\alpha} (p', \sigma')}{(P, \sigma) \xrightarrow{\alpha} (p', \sigma')} \quad P \doteq p$$

Sum$_1$: $$\frac{(p_1, \sigma) \xrightarrow{\alpha} (p_1', \sigma')}{(p_1 + p_2, \sigma) \xrightarrow{\alpha} (p_1', \sigma')}$$ Sum$_2$: $$\frac{(p_2, \sigma) \xrightarrow{\alpha} (p_2', \sigma')}{(p_1 + p_2, \sigma) \xrightarrow{\alpha} (p_2', \sigma')}$$

Int$_1$: $$\frac{(p_1, \sigma) \xrightarrow{\alpha} (p_1', \sigma')}{(p_1 \parallel p_2, \sigma) \xrightarrow{\alpha} (p_1' \parallel p_2, \sigma')}$$ Int$_2$: $$\frac{(p_2, \sigma) \xrightarrow{\alpha} (p_2', \sigma')}{(p_1 \parallel p_2, \sigma) \xrightarrow{\alpha} (p_1 \parallel p_2', \sigma')}$$

Syn: $$\frac{(p_1, \sigma) \xrightarrow{\alpha_1} (p_1', \sigma_1') \quad (p_2, \sigma) \xrightarrow{\alpha_2} (p_2', \sigma_2')}{(p_1 \parallel p_2, \sigma) \xrightarrow{\alpha_1 \cup \alpha_2} (p_1' \parallel p_2', \sigma')}, \quad \text{where } \sigma' \in \sigma/\alpha_1 \cup \alpha_2$$

Res: $$\frac{(p, \sigma) \xrightarrow{\alpha} (p', \sigma')}{(p\backslash\gamma, \sigma) \xrightarrow{\alpha} (p'\backslash\gamma, \sigma')}, \quad \text{where } \forall(\phi \rightarrow \varrho) \in \gamma.\ (\sigma(\phi) = tt \text{ implies } \alpha(\varrho) = tt)$$

The rules in Table 1 have the following intuitive meaning:

Act: The configuration $((\phi \rightarrow a).p, \sigma)$ can evolve to the configuration (p, σ') by executing the action $\{a\}$, provided that the precondition ϕ is true in σ, and σ' is a possible global store obtained by executing $\{a\}$ in σ, i.e., $\sigma' \in \sigma/\{a\}$.

Def: The configuration (P, σ), where $P \doteq p$, can evolve to the configuration (p', σ') by executing the (synchronized) action α, provided that (p, σ) can.

Sum: The configuration $(p_1 + p_2, \sigma)$ can evolve to a configuration by executing the (synchronized) action α, provided that either (p_1, σ) or (p_2, σ) can evolve to that configuration.

Int: The configuration $(p_1 \parallel p_2, \sigma)$ can evolve to the configuration $(p_1' \parallel p_2, \sigma')$ by executing the (synchronized) action α, provided that (p_1, σ) can evolve to (p_1', σ') by executing α. Similarly, $(p_1 \parallel p_2, \sigma)$ can evolve to $(p_1 \parallel p_2', \sigma')$ by executing α, provided that (p_2, σ) can evolve to (p_2', σ') by executing α.

Syn: The configuration $(p_1 \parallel p_2, \sigma)$ can evolve to the configuration $(p_1' \parallel p_2', \sigma')$ by executing the (synchronized) action $\alpha_1 \cup \alpha_2$, provided that (p_1, σ) can evolve to (p_1', σ') by executing α_1 and (p_2, σ) can evolve to (p_2', σ') by executing α_2.

Res: The configuration $(p\backslash\gamma, \sigma)$ can evolve to the configuration $(p'\backslash\gamma, \sigma')$ by executing the (synchronized) action α, provided that (p, σ) can evolve to (p', σ') by executing α, and α is allowed by γ, i.e., $\forall(\phi \rightarrow \varrho) \in \gamma.\ (\sigma(\phi) = tt \text{ implies } \alpha(\varrho) = tt)$.

Given an initial configuration $(p_{init}, \sigma_{init})$, the structural rules in Table 1 allow us to associate to a configuration a *transition system* (*Kripke structure*) whose states are the configurations reachable from $(p_{init}, \sigma_{init})$, via the transitions inferred by using the structural rules.

Let us formally define the notion of transition system, and the notion of transition system generated by a configuration.

Definition 2.4. Given a set $\mathcal{P}$ of propositions, and set $\mathcal{A}$ of atomic actions, a *transition system* is a triple $(\mathcal{S}, \{\mathcal{R}_\alpha \mid \alpha \in 2^{\mathcal{A}}\}, \Pi)$, with a set of states $\mathcal{S}$, a family of transition relations $\mathcal{R}_\alpha \in \mathcal{S} \times \mathcal{S}$, and a mapping Π from $\mathcal{P}$ to subsets of $\mathcal{S}$.

Definition 2.5. Given a configuration $(p_{init}, \sigma_{init})$, we call *$(p_{init}, \sigma_{init})$-generated transition system*, the transition system $\mathcal{T} = (\mathcal{S}, \{\mathcal{R}_\alpha \mid \alpha \in 2^{\mathcal{A}}\}, \Pi)$, with $\mathcal{P} = Prop$, $\mathcal{A} = Act$, and

- $\mathcal{S} = \{(p, \sigma) \mid (p_{init}, \sigma_{init})(\longrightarrow)^*(p, \sigma)\}$, where $\_ \longrightarrow \_$ is the least relation satisfying the structural rules in Table 1, and $\_(\longrightarrow)^*\_$ is its reflexive transitive closure;
- $((p, \sigma), (p', \sigma')) \in \mathcal{R}_\alpha$ iff $(p, \sigma) \xrightarrow{\alpha} (p', \sigma')$;
- $\Pi(A) = \{(p, \sigma) \mid \sigma(A) = tt\}$.

The transition system generated by the initial configuration describes all the possible configuration's evolutions, and hence constitutes our model of the dynamic system.

3. Examples of descriptions

The examples in this section illustrate the main aspects of the proposed representation formalism.

3.1. Russian Turkey Shoot

In this first example, we mainly focus on those aspects of the formalism that are common to most formalisms for reasoning about actions, in which the notion of process is not put forward. In particular, we show various instances of effects, premises, preconditions as well as instances of deterministic and nondeterministic actions. Notably, the process described in the example corresponds to the one implicitly assumed in those formalisms where no process is explicitly specified. The transition system obtained from the description is also illustrated.

The scenario of the example is a variant of the well-known Yale Shooting scenario in which we have a turkey that gets killed if it is shot by a loaded gun. The variation consists in adding an extra nondeterministic action `spin` that represents spinning the gun's bullet cylinder. This action has no effects if the gun is unloaded. If the gun is loaded instead, then after the action `spin`, the gun can either still be loaded or be unloaded.[11]

[11] For sake of simplicity, we make the (somewhat unrealistic) hypothesis that if the bullet is not in the shooting chamber, then it is not in the cylinder either. It is easy to modify the representation of the scenario in order to consider the position of the bullet in the cylinder (we invite the reader to try).

We formalize the scenario by introducing a set of propositions to model the relevant facts, and a set of atomic actions that change the values of these facts:

$$\begin{aligned} Prop &= \{\texttt{Loaded}, \texttt{Alive}\} \\ Act &= \{\texttt{load}, \texttt{shoot}, \texttt{spin}, \texttt{wait}\} \end{aligned}$$

The effects of the actions are:

$$\begin{aligned} \mathit{effct}(\texttt{load}) &= \{(tt, \{\texttt{Loaded}\})\} \\ \mathit{effct}(\texttt{shoot}) &= \{(\texttt{Loaded}, \{\neg\texttt{Alive}, \neg\texttt{Loaded}\})\} \\ \mathit{effct}(\texttt{spin}) &= \{(\texttt{Loaded}, \{\texttt{Loaded}\}), (\texttt{Loaded}, \{\neg\texttt{Loaded}\})\} \\ \mathit{effct}(\texttt{wait}) &= \{\} \end{aligned}$$

which can be read as follows. Performing `load` results in having the gun loaded. Performing `shoot`, under the premise of having the gun loaded, results in having the turkey killed and the gun unloaded. If the gun is unloaded, performing `shoot` has no effect. Performing `spin`, under the premise of having the gun loaded, results in either having the gun still loaded, or having it unloaded. If the gun is unloaded, performing `spin` has no effect. Finally performing `wait` has no effect in any cases.

In this scenario, at any moment, we can (1) load the gun if it is not already loaded; (2) shoot the turkey; (3) spin the cylinder; (4) wait. That is, at any moment, to perform the action `load`, the precondition $\neg$`Loaded` must be satisfied, while for the other actions, no preconditions are required (i.e., their preconditions are vacuously tt). We formulate these requirements by means of the following process:

$$\begin{aligned} \texttt{P} \doteq\ & (\neg\texttt{Loaded} \to \texttt{load}).\texttt{P} + (tt \to \texttt{shoot}).\texttt{P} + (tt \to \texttt{spin}).\texttt{P} \\ & + (tt \to \texttt{wait}).\texttt{P} \end{aligned}$$

Observe that process `P` is very simple: it performs an action, whose precondition is satisfied, and becomes itself. In other words, while the configuration evolves since the effects of the actions change the status of the global store, the process remains always in the same status. Obviously, in this case, the number of possible configurations depends only on the number of possible global stores, which is at most $2^{|Prop|}$.

The form of the above process is typical of those formalisms for reasoning about actions that concentrate only on the specification of elementary actions, specifying preconditions and effects for them, and do not specify explicitly any process. In these cases, the following process is, in fact, implicitly assumed (note that it has exactly the form of the one above):

$$P \doteq \sum_i (\phi_i \to a_i).P$$

where a_i are the actions and ϕ_i are their preconditions.

Once we have specified the process and how actions change the global store, for every initial global store, it is possible to compute all possible evolutions of the system. For example, let the initial configuration be described by $(p_{init}, \sigma_{init})$ with $p_{init} = \texttt{P}$ and $\sigma_{init} = \{\texttt{Alive}, \neg\texttt{Loaded}\}$. From $(p_{init}, \sigma_{init})$, using the structural rules in Table 1, we

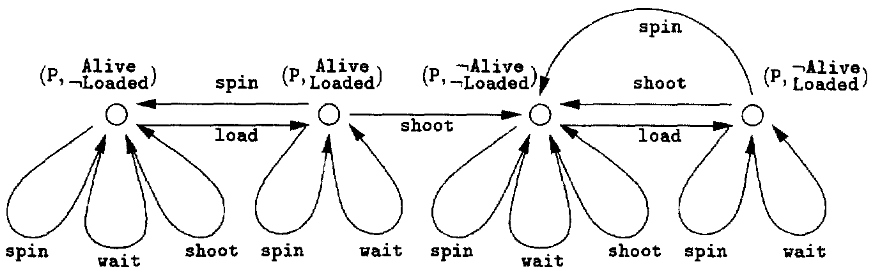

Fig. 1. Transition system for the Russian Turkey Shoot.

generate the transition system $\mathcal{T}$ in Fig. 1, which summarizes in a graph all possible evolutions of the configurations of the system. For example, a possible evolution is:

$$(\mathtt{P}, \{\mathtt{Alive}, \neg\mathtt{Loaded}\}) \xrightarrow{\mathtt{load}} (\mathtt{P}, \{\mathtt{Alive}, \mathtt{Loaded}\}) \xrightarrow{\mathtt{wait}} (\mathtt{P}, \{\mathtt{Alive}, \mathtt{Loaded}\}) \xrightarrow{\mathtt{shoot}} (\mathtt{P}, \{\neg\mathtt{Alive}, \neg\mathtt{Loaded}\})$$

that results in having killed the turkey. Another example is:

$$(\mathtt{P}, \{\mathtt{Alive}, \neg\mathtt{Loaded}\}) \xrightarrow{\mathtt{load}} (\mathtt{P}, \{\mathtt{Alive}, \mathtt{Loaded}\}) \xrightarrow{\mathtt{spin}} (\mathtt{P}, \{\mathtt{Alive}, \neg\mathtt{Loaded}\}) \xrightarrow{\mathtt{shoot}} (\mathtt{P}, \{\mathtt{Alive}, \neg\mathtt{Loaded}\})$$

that results in not having killed the turkey. Another one is:

$$(\mathtt{P}, \{\mathtt{Alive}, \neg\mathtt{Loaded}\}) \xrightarrow{\mathtt{load}} (\mathtt{P}, \{\mathtt{Alive}, \mathtt{Loaded}\}) \xrightarrow{\mathtt{spin}} (\mathtt{P}, \{\mathtt{Alive}, \mathtt{Loaded}\}) \xrightarrow{\mathtt{shoot}} (\mathtt{P}, \{\neg\mathtt{Alive}, \neg\mathtt{Loaded}\})$$

that results in having killed the turkey.

What about if we do not have complete information on the initial situation? A straightforward technique to deal with this is to trade the lack of information with nondeterminism. For example, we may prefix the actual process with a sequence of dummy atomic actions that nondeterministically lead to several global stores according to incomplete information on the initial situation. In the above scenario, we may introduce two dummy actions initAlive and initLoaded. Assuming that initially it is known that the turkey is alive but it is not known whether the gun is loaded:

– we specify

$$\mathit{effct}(\mathtt{initAlive}) = \{(\mathit{tt}, \{\mathtt{Alive}\})\}$$

$$\mathit{effct}(\mathtt{initLoaded}) = \{(\mathit{tt}, \{\mathtt{Loaded}\}), (\mathit{tt}, \{\neg\mathtt{Loaded}\})\}$$

– we prefix the above process P by an initialization sequence (the order of the dummy actions can be arbitrary) getting the new process

$$\mathtt{Q} \doteq (\mathit{tt} \rightarrow \mathtt{initAlive}).(\mathit{tt} \rightarrow \mathtt{initLoaded}).\mathtt{P}$$

– we arbitrarily set the initial global store $\sigma_q = \{\neg\mathtt{Alive}, \neg\mathtt{Loaded}\}$.

Observe that the new initial configuration $(\mathtt{Q}, \sigma_q)$ is forced to evolve by first executing the initialization sequence, and then evolve according to the original process P. After executing the initialization sequence, our partial knowledge on the initial situation will be correctly taken into account.

3.2. Lifting a table

In this example, we illustrate a process denoting the concurrent activities of more agents, showing parallel execution of processes (interpreted by interleaving semantics) and synchronization of atomic actions. Interrupts are also briefly discussed.

The scenario is the following. A vase is on top of a table. If just one side of the table is lifted then the vase falls onto the floor. While if both sides of the table are lifted simultaneously, the vase does not fall.

We formalize the scenario by introducing the following primitive propositions and atomic actions:

$$\begin{aligned} \mathit{Prop} &= \{\mathtt{VaseOnTable}, \mathtt{DownLeftSide}, \mathtt{DownRightSide}\} \\ \mathit{Act} &= \{\mathtt{vaseFalls}, \mathtt{downLeft}, \mathtt{downRight}, \mathtt{upLeft}, \mathtt{upRight}\} \end{aligned}$$

The effects of the atomic actions are the obvious ones:

$$\begin{aligned} &\mathit{effct}(\mathtt{vaseFalls}) = \{(\phi, \{\neg\mathtt{VaseOnTable}\})\} \\ &\mathit{effct}(\mathtt{downLeft}) = \{(\neg\mathtt{DownLeftSide}, \{\mathtt{DownLeftSide}\})\} \\ &\mathit{effct}(\mathtt{downRight}) = \{(\neg\mathtt{DownRightSide}, \{\mathtt{DownRightSide}\})\} \\ &\mathit{effct}(\mathtt{upLeft}) = \{(\mathtt{DownLeftSide}, \{\neg\mathtt{DownLeftSide}\})\} \\ &\mathit{effct}(\mathtt{upRight}) = \{(\mathtt{DownRightSide}, \{\neg\mathtt{DownRightSide}\})\} \end{aligned}$$

where $\phi = ((\mathtt{DownLeftSide} \wedge \neg\mathtt{DownRightSide}) \vee (\neg\mathtt{DownLeftSide} \wedge \mathtt{DownRightSide})) \wedge \mathtt{VaseOnTable}$.

In this scenario, we have three processes going on concurrently: agent $\mathtt{A_l}$ who may either raise or put down the left side of the table; agent $\mathtt{A_r}$ who is in control of right side of the table; the environment Env that makes the vase fall off the table as soon as one of the sides of the table is risen while the other side is not. We model these concurrent activities by a process LT defined as follows:

$$\begin{aligned} \mathtt{A_l} &\doteq (\neg\mathtt{DownLeftSide} \rightarrow \mathtt{downLeft}).\mathtt{A_l} \\ &\quad + (\mathtt{DownLeftSide} \rightarrow \mathtt{upLeft}).\mathtt{A_l} \\ \mathtt{A_r} &\doteq (\neg\mathtt{DownRightSide} \rightarrow \mathtt{downRight}).\mathtt{A_r} \\ &\quad + (\mathtt{DownRightSide} \rightarrow \mathtt{upRight}).\mathtt{A_r} \end{aligned}$$

$$\begin{aligned} \mathtt{Env} &\doteq (\phi \rightarrow \mathtt{vaseFalls}).\mathtt{Env} \\ \mathtt{LT} &\doteq (\mathtt{A_l} \parallel \mathtt{A_r} \parallel \mathtt{Env})\backslash\{\phi \rightarrow \mathtt{vaseFalls} \wedge \neg\mathit{others}\} \end{aligned}$$

where *others* denotes the disjunction of all atomic actions other than vaseFalls.

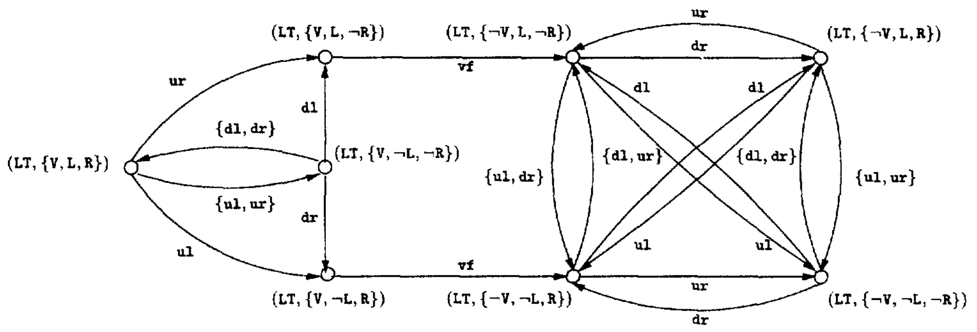

Fig. 2. Transition system for the Lifting a Table.

Observe that the vase can fall only when the precondition ϕ is satisfied. Furthermore, because of the restriction $\cdot\backslash\{\phi \rightarrow$ vaseFalls $\wedge \neg others\}$ in LT, whenever the vase has the possibility to fall, it is forced to do so, while all other atomic actions are blocked.[12]

Let the initial configuration be:[13]

$$(p_{init}, \sigma_{init}) = (\mathtt{LT}, \{\mathtt{VaseOnTable}, \mathtt{DownLeftSide}, \mathtt{DownRightSide}\}).$$

By making use of the structural rules in Table 1, it is possible to build the corresponding transition system, as shown in Fig. 2. The figure uses obvious abbreviations to denote propositions and actions: propositions VaseOnTable, DownLeftSide, DownRight Side are denoted by V, L, R, respectively, and actions vaseFalls, downLeft, downRight, upLeft, upRight are denoted by vf, dl, dr, ul, ur, respectively. The initial configuration $(p_{init}, \sigma_{init})$ is denoted by the node labeled by (LT, {V, L, R}).

It is possible to check that the behavior of the system is the expected one. For instance, in $(p_{init}, \sigma_{init})$, the subprocess Env cannot execute since $\sigma(\phi) = \mathit{ff}$. Instead both $\mathtt{A_l}$ and $\mathtt{A_r}$ can proceed performing upLeft and upRight, respectively. However, unless upLeft and upRight synchronize, the condition ϕ will be true in the successive configuration and hence the vase will be forced to fall. If the synchronized action {upRight, upLeft} is performed, then ϕ will not be true and hence the vase won't fall.

Note that, the process Env in LT can be seen as a process which, when certain conditions are met (ϕ), performs an *interrupt* (the action vaseFalls) and allows for the further execution of the other concurrent processes ($\mathtt{A_l}$ and $\mathtt{A_r}$) only when the interrupt is completed (the action vaseFalls terminates). More generally, an interrupt will set the truth-value of certain flags that are in the preconditions of other actions, thus disallowing their executions. In this way, we can build interrupting processes that block the execution of the other processes, execute without interferences from them, and reset the flags only when the interrupt has been fully handled allowing for the other processes to be resumed and

[12] Note that, if we weaken the restriction in LT to be $\cdot\backslash\{\phi \rightarrow$ vaseFalls$\}$, then we will still force vaseFalls to be executed whenever ϕ is true, but the action to make the vase fall will be allowed to synchronize with other actions.

[13] Note that LT behaves correctly under any other initial global store. And it is easy to verify that it generates the same transition system from every initial global stores (obviously modulo the initial configuration).

continue. Similarly, we can also build processes that block the execution of other processes, without ever allowing them to regain the control. [14]

3.3. Relay race

In this example, we show a more complex process constituted by various subprocesses in which synchronization plays a key role. The example also illustrates a simple but effective technique to deal with actions that are not instantaneous, but have a duration (see [48,50, 59]). Namely, an action that has a duration of "running" is modeled by:

(1) an (instantaneous) atomic action startRun denoting the initiation of the action;
(2) a proposition Running in the global store denoting that the action has started but not yet completed;
(3) an (instantaneous) atomic action endRun denoting the termination of the action.

The scenario is a "relay race" with two competing teams, each composed of two runners. The rules of the race are the following:

(1) when the "go" signal is given, the first runner of each team may start running;
(2) when the first runner reaches the "100 meters line", the second runner may start running;
(3) when the two runners of the same team are both running, the first runner may pass the baton along to the second runner;
(4) the team, whose second runner arrives first to the "finish line" with the baton, wins.

We formalize the scenario by introducing the following primitive propositions and atomic actions ($i = 1, 2$ denotes the team, $j = 1, 2$ denotes the runner):

$$\begin{aligned}
\mathit{Prop} &= \{\mathtt{Running}_{i,j}, \mathtt{100mLinePassed}_{i,1}, \mathtt{Baton}_{i,j},\\
&\qquad \mathtt{FinishLinePassed}_{i,2}, \mathtt{Won}_i \mid i, j = 1, 2\}\\
\mathit{Act} &= \{\mathtt{startRun}_{i,j}, \mathtt{endRun}_{i,j}, \mathtt{pass100mLine}_{i,1}, \mathtt{passFinishLine}_{i,2},\\
&\qquad \mathtt{giveBaton}_{i,1}, \mathtt{getBaton}_{i,2}, \mathtt{wins}_i, \mathtt{go} \mid i, j = 1, 2\}
\end{aligned}$$

The effects of the atomic actions are the obvious ones:

$$\begin{aligned}
&\mathit{effct}(\mathtt{startRun}_{i,j})) = \{(\neg\mathtt{Running}_{i,j}, \{\mathtt{Running}_{i,j}\})\}\\
&\mathit{effct}(\mathtt{endRun}_{i,j})) = \{(\mathtt{Running}_{i,j}, \{\neg\mathtt{Running}_{i,j}\})\}\\
&\mathit{effct}(\mathtt{pass100mLine}_{i,1}) = \{(\neg\mathtt{100mLinePassed}_{i,1},\\
&\qquad\qquad \{\mathtt{100mLinePassed}_{i,1}\})\}\\
&\mathit{effct}(\mathtt{passFinishLine}_{i,2}) = \{(\neg\mathtt{FinishLinePassed}_{i,2},\\
&\qquad\qquad \{\mathtt{FinishLinePassed}_{i,2}\})\}\\
&\mathit{effct}(\mathtt{giveBaton}_{i,1}) = \{(\mathtt{Baton}_{i,1}, \{\neg\mathtt{Baton}_{i,1}\})\}\\
&\mathit{effct}(\mathtt{getBaton}_{i,2}) = \{(\neg\mathtt{Baton}_{i,2}, \{\mathtt{Baton}_{i,2}\})\}
\end{aligned}$$

[14] *Priorities* among processes can also be easily modeled, by inserting suitable flags in the preconditions of their actions.

$$effct(\mathtt{wins}_i) = \{(\neg(\mathtt{Won}_1 \vee \mathtt{Won}_2), \{\mathtt{Won}_i\})\}$$

$$effct(\mathtt{go}) = \{\}$$

The various activities involved in the scenario are described by the following processes (we abbreviate $(tt \rightarrow a).p$ by $a.p$ and $tt \rightarrow \varrho$ by ϱ):

$$\mathtt{Runner}_{i,1} \doteq \mathtt{startRun}_{i,1}.\mathtt{pass100mLine}_{i,1}.\mathtt{giveBaton}_{i,1}.\mathtt{endRun}_{i,1}.nil$$

$$\mathtt{Runner}_{i,2} \doteq (\mathtt{100mLinePassed}_{i,1} \rightarrow \mathtt{startRun}_{i,2}).\mathtt{getBaton}_{i,2}.$$
$$\mathtt{passFinishLine}_{i,2}.\mathtt{endRun}_{i,2}.nil$$

$$\mathtt{Team}_i \doteq (\mathtt{Runner}_{i,1} \parallel \mathtt{Runner}_{i,2})\backslash\{(\mathtt{giveBaton}_{i,1} \equiv \mathtt{getBaton}_{i,2})\}$$

$$\mathtt{CheckWinner} \doteq \mathtt{wins}_1.nil + \mathtt{wins}_2.nil$$

$$\mathtt{RR} \doteq (\mathit{Ready} \rightarrow \mathtt{go}).(\mathtt{Team}_1 \parallel \mathtt{Team}_2 \parallel \mathtt{CheckWinner})\backslash$$
$$\{(\mathtt{passFinishLine}_{1,2} \equiv \mathtt{wins}_1) \wedge (\mathtt{passFinishLine}_{2,2} \equiv \mathtt{wins}_2)\}$$

where

$$\mathit{Ready} = \wedge_i(\wedge_j \neg\mathtt{Running}_{i,j} \wedge \neg\mathtt{100mLinePassed}_{i,1} \wedge$$
$$\neg\mathtt{FinishLinePassed}_{i,1} \wedge \mathtt{Baton}_{i,1} \wedge \neg\mathtt{Baton}_{i,2} \wedge \neg\mathtt{Won}_i).$$

Let us explain the above processes. The process $\mathtt{Runner}_{i,1}$ describes the activities of the first runner of the team i: the first runner starts running, passes the 100 meters line, passes the baton along to the second runner, and ends running.

The process $\mathtt{Runner}_{i,2}$ describes the activities of the second runner of the team i: the second runner starts running provided that the first process has already reached the 100 meters line, gets the baton from the first runner, passes the Finish line, and ends running.

The process $\mathtt{Team}_i$ describes the activities of the team i. It consists of the concurrent composition of the two processes $\mathtt{Runner}_{i,1}$ and $\mathtt{Runner}_{i,2}$ with the restriction that the actions $\mathtt{giveBaton}_{i,1}$ and $\mathtt{getBaton}_{i,2}$ must be performed synchronously.

The process `CheckWinner` describes the activity that establishes if the first or the second team wins.

The process `RR` describes all the activities of the system. It consists of the concurrent composition of the three processes $\mathtt{Team}_1$, $\mathtt{Team}_2$ and `CheckWinner` prefixed by the action `go` that starts the race under suitable preconditions, and with the restriction that the actions $\mathtt{passFinishLine}_{1,2}$ and $\mathtt{wins}_1$, and, respectively, the actions $\mathtt{passFinishLine}_{2,2}$ and $\mathtt{wins}_2$, must be executed synchronously, thus forcing the activity described by `CheckWinner` to declare the true winner.

As in the previous examples, by using the structural rules in Table 1, it is possible to build the transition system generated by the given initial configuration to make explicit all possible evolutions of the scenario.

3.4. *Translating* `While` *programs*

As the last example, we show that the process description formalism presented here can easily represent traditional programming constructs like "while" and "if-then-else". In particular, we show how programs of a simple sequential programming language, called `While`, can be translated into processes. `While` programs have the following syntax:

$$\delta ::= a \mid \delta_1; \delta_2 \mid \texttt{if}\ \phi\ \texttt{then}\ \delta_1\ \texttt{else}\ \delta_2 \mid \texttt{while}\ \phi\ \texttt{do}\ \delta$$

where a is a generic atomic action whose effects are specified by the effect function as before, and the other constructs are sequential composition, if-then-else, and while, respectively.

To define the translation, we first introduce function tr defined inductively on the structure of `While` programs. For every process p:

$$tr(a, p) = (tt \to a).p$$

$$tr(\delta_1; \delta_2, p) = tr(\delta_1, tr(\delta_2, p))$$

$$tr(\texttt{if}\ \phi\ \texttt{then}\ \delta_1\ \texttt{else}\ \delta_2, p) = (\phi \to \texttt{nop}).tr(\delta_1, p) + (\neg\phi \to \texttt{nop}).tr(\delta_2, p)$$

$tr(\texttt{while}\ \phi\ \texttt{do}\ \delta, p) = Q$ where Q is a new process name defined as:

$$Q \doteq (\phi \to \texttt{nop}).tr(\delta, Q) + (\neg\phi \to \texttt{nop}).p$$

where `nop` is a special atomic action which has no effects on the global store. It is used to reflect the fact that tests are assumed to make a transition. [15] Then, we define the translation of a `While` program δ as the process $tr(\delta, nil)$.

Observe that the intuitive meaning of the constructs is correctly captured by the resulting process. For example, while-loops are translated into processes that behave as follows: first, the entering condition of the while is tested; if such condition is true then the process behaves as the body of the while followed by the whole while-loop again; if the condition is false then it exits the loop. [16]

For example, the following fragment of control code of an elevator:

$$\delta = \texttt{while}\ \neg \mathit{GroundFloor}\ \texttt{do}\ \mathit{goDownOneFloor};\ \mathit{openDoor}$$

is translated into:

$$\begin{aligned} Q \doteq\ & (\neg \mathit{GroundFloor} \to \texttt{nop}).(tt \to \mathit{goDownOneFloor}).Q\ + \\ & (\mathit{GroundFloor} \to \texttt{nop}).(tt \to \mathit{openDoor}).nil \end{aligned}$$

4. Reasoning formalism: the logic $\mathcal{M}_\mu$

Once we have had a representation of a dynamic system, we can use such a representation to infer properties of the system, like the possibility to reach a configuration where a certain property holds (i.e., where a certain "goal" is satisfied), or the invariance of certain statements, etc.

[15] If such assumption is not made, the translation can be modified accordingly.

[16] One could define derived structural rules for the various constructs of `While` on the basis of the associated processes, and verify that such rules correspond to those usually associated to such constructs (see, e.g., [26]). A thorough discussion on this is out of the scope of the paper.

Among the various temporal and modal logics that have been proposed in the process algebra literature for verifying properties of concurrent systems [17,28,43], we focus on one of the most powerful logics of programs which is called *modal mu-calculus* ([19,32, 33,56,57]). Modal mu-calculus is a logic of programs, which is strictly more expressive than logics like *PDL*, *ΔPDL*, *CTL* and *CTL**. It has been proposed as a logic for expressing "temporal" properties of reactive and parallel processes in [9,12,36,54,55,62]. We refer to the excellent tutorial article [55] for a thorough introduction on modal mu-calculus and its use in the context of concurrent processes.

In this paper, we introduce an extension of standard modal mu-calculus, called $\mathcal{M}_\mu$, which allows for boolean combinations of atomic actions in the modalities, and thus, it is suitable to verify properties of systems specified in our representation formalism.

4.1. The logic $\mathcal{M}_\mu$

The logic $\mathcal{M}_\mu$ is basically constituted by three kinds of components:

- *Propositions* to denote properties of the global store in a given configuration.
- *Modalities* to denote the capability of performing certain actions in a given configuration.
- *Least and greatest fixpoint constructs* to denote "temporal" properties of the system, typically defined by *induction* and *coinduction*.

The formulae of $\mathcal{M}_\mu$ are defined on the base of *action formulae* generated by the following abstract syntax:

$$\varrho ::= a \mid \textbf{any} \mid \textbf{none} \mid \neg\varrho \mid \varrho_1 \wedge \varrho_2 \mid \varrho_1 \vee \varrho_2$$

where $a \in \mathcal{A}$ for some fixed set $\mathcal{A}$ of atomic actions, **any** is a special atomic action denoting the union of all actions in $\mathcal{A}$, and **none** is a special atomic action denoting the empty (nonexecutable) action.

The meaning of a generic action formula is given by the satisfaction relation below, where α is a set of atomic actions (denoting a synchronized action in general):

$\alpha \models a$	iff $a \in \alpha$
$\alpha \models \textbf{any}$	(always)
$\alpha \models \textbf{none}$	(never)
$\alpha \models \neg\varrho$	iff not $\alpha \models \varrho$
$\alpha \models \varrho_1 \wedge \varrho_2$	iff $\alpha \models \varrho_1$ and $\alpha \models \varrho_2$
$\alpha \models \varrho_1 \vee \varrho_2$	iff $\alpha \models \varrho_1$ or $\alpha \models \varrho_2$

Note that not all constructs in action formulae are independent. In particular, we have: $\textbf{none} = a \wedge \neg a$, $\textbf{any} = \neg\textbf{none}$, $\varrho_1 \vee \varrho_2 = \neg(\neg\varrho_1 \wedge \neg\varrho_2)$.

Formulae of $\mathcal{M}_\mu$ are formed inductively from action formulae, primitive propositions in some fixed set $\mathcal{P}$, and variable symbols in some fixed set *Var*, according to the following abstract syntax:

$$\Phi ::= A \mid tt \mid ff \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid \langle\varrho\rangle\Phi \mid [\varrho]\Phi \mid \mu X.\Phi \mid \nu X.\Phi \mid X$$

where A is a primitive proposition in $\mathcal{P}$, X is a variable symbol in *Var*, and ϱ is an action formula over $\mathcal{A}$.

The symbols μ and ν can be considered as quantifiers, and in the sequel, we make use of notions of scope, bound and free occurrences of variables, closed formulas, etc. The definitions of these notions are the same as in first-order logic, treating μ and ν as quantifiers.

As usual in mu-calculus, for formulae of the form $\mu X.\Phi$ and $\nu X.\Phi$, we require the *syntactic monotonicity* of Φ with respect to X: every occurrence of the variable X in Φ must be within the scope of an even number of negation signs. This requirement guarantees the existence of the least and the greatest fixpoints associated with Φ (see below).

The semantics of $\mathcal{M}_\mu$ is based on the notions of transition system and valuation. Given a transition system $\mathcal{T}$, a *valuation* $\mathcal{V}$ on $\mathcal{T}$ is a mapping from variables in *Var* to subsets of the states in $\mathcal{T}$.

Given a valuation $\mathcal{V}$, we denote by $\mathcal{V}[X/\mathcal{E}]$, the valuation identical to $\mathcal{V}$ except for $\mathcal{V}[X/\mathcal{E}](X) = \mathcal{E}$, i.e., for every variable Y,

$$\mathcal{V}[X/\mathcal{E}](Y) = \begin{cases} \mathcal{E} & \text{if } Y = X, \\ \mathcal{V}(Y) & \text{if } Y \neq X. \end{cases}$$

Let $\mathcal{T} = (\mathcal{S}, \{\mathcal{R}_\alpha \mid \alpha \in 2^{\mathcal{A}}\}, \Pi)$ be a transition system with Π mapping propositions in $\mathcal{P}$ to subsets of $\mathcal{S}$, and $\mathcal{V}$ a valuation on $\mathcal{T}$. We assign meaning to $\mathcal{M}_\mu$ formulae by associating to $\mathcal{T}$ and $\mathcal{V}$ an *extension function* $(\cdot)^{\mathcal{T}}_{\mathcal{V}}$, which maps $\mathcal{M}_\mu$ formulae to subsets of $\mathcal{S}$. The extension function $(\cdot)^{\mathcal{T}}_{\mathcal{V}}$ is defined inductively as follows:

$$\begin{aligned}
&(A)^{\mathcal{T}}_{\mathcal{V}} = \Pi(A) \subseteq \mathcal{S}\\
&(tt)^{\mathcal{T}}_{\mathcal{V}} = \mathcal{S}\\
&(\mathit{ff})^{\mathcal{T}}_{\mathcal{V}} = \emptyset\\
&(\neg\Phi)^{\mathcal{T}}_{\mathcal{V}} = \mathcal{S} - (\Phi)^{\mathcal{T}}_{\mathcal{V}}\\
&(\Phi_1 \wedge \Phi_2)^{\mathcal{T}}_{\mathcal{V}} = (\Phi_1)^{\mathcal{T}}_{\mathcal{V}} \cap (\Phi_2)^{\mathcal{T}}_{\mathcal{V}}\\
&(\Phi_1 \vee \Phi_2)^{\mathcal{T}}_{\mathcal{V}} = (\Phi_1)^{\mathcal{T}}_{\mathcal{V}} \cup (\Phi_2)^{\mathcal{T}}_{\mathcal{V}}\\
&(\langle\varrho\rangle\Phi)^{\mathcal{T}}_{\mathcal{V}} = \{s \in \mathcal{S} \mid \exists\alpha, s'.\ \alpha \models \varrho \text{ and } (s,s') \in \mathcal{R}_\alpha \text{ and } s' \in (\Phi)^{\mathcal{T}}_{\mathcal{V}}\}\\
&([\varrho]\Phi)^{\mathcal{T}}_{\mathcal{V}} = \{s \in \mathcal{S} \mid \forall\alpha, s'.\ \alpha \models \varrho \text{ and } (s,s') \in \mathcal{R}_\alpha \text{ implies } s' \in (\Phi)^{\mathcal{T}}_{\mathcal{V}}\}\\
&(\mu X.\Phi)^{\mathcal{T}}_{\mathcal{V}} = \bigcap\{\mathcal{E} \subseteq \mathcal{S} \mid (\Phi)^{\mathcal{T}}_{\mathcal{V}[X/\mathcal{E}]} \subseteq \mathcal{E}\}\\
&(\nu X.\Phi)^{\mathcal{T}}_{\mathcal{V}} = \bigcup\{\mathcal{E} \subseteq \mathcal{S} \mid \mathcal{E} \subseteq (\Phi)^{\mathcal{T}}_{\mathcal{V}[X/\mathcal{E}]}\}\\
&(X)^{\mathcal{T}}_{\mathcal{V}} = \mathcal{V}(X) \subseteq \mathcal{S}
\end{aligned}$$

Intuitively, the extension function $(\cdot)^{\mathcal{T}}_{\mathcal{V}}$ assigns to the various constructs of $\mathcal{M}_\mu$ the following meanings:

- The boolean connectives have the expected meaning.

- The extension of $\langle \varrho \rangle \Phi$ includes the states $s \in \mathcal{S}$ such that starting from s, there is an execution of some action satisfying ϱ that leads to a successive state s' included in the extension of Φ.
- The extension of $[\varrho]\Phi$ includes the states s such that starting from s, each execution of an action satisfying ϱ leads to some successive state s' included in the extension of Φ.
- The extension of $\mu X.\Phi$ is the *smallest subset* $\mathcal{E}_\mu$ of $\mathcal{S}$ such that, assigning to X the extension $\mathcal{E}_\mu$, the resulting extension of Φ is contained in $\mathcal{E}_\mu$. That is, the extension of $\mu X.\Phi$ is the *least fixpoint* of the operator $\lambda\mathcal{E}.(\Phi)^{\mathcal{T}}_{\mathcal{V}[X/\mathcal{E}]}$. The syntactic monotonicity of Φ with respect to X guarantees the monotonicity of such operator and hence, by the Tarski–Knaster Theorem [58], the unique existence of the least fixpoint.
- Similarly, the extension of $\nu X.\Phi$ is the *greatest subset* $\mathcal{E}_\nu$ of $\mathcal{S}$ such that, assigning to X the extension $\mathcal{E}_\nu$, the resulting extension of Φ contains $\mathcal{E}_\nu$. That is, the extension of $\nu X.\Phi$ is the *greatest fixpoint* of the operator $\lambda\mathcal{E}.(\Phi)^{\mathcal{T}}_{\mathcal{V}[X/\mathcal{E}]}$. The syntactic monotonicity of Φ with respect to X guarantees the monotonicity of such operator and hence, by the Tarski–Knaster Theorem [58], the unique existence of the greatest fixpoint.

Note that not all the $\mathcal{M}_\mu$ constructs are independent. In particular, we have: $f\!f = A \wedge \neg A$; $tt = \neg f\!f$; $\Phi_1 \vee \Phi_2 = \neg(\neg\Phi_1 \wedge \neg\Phi_2)$; $[\varrho]\Phi = \neg\langle\varrho\rangle\neg\Phi$; and $\nu X.\Phi = \neg\mu X.\neg\Phi[X/\neg X]$, where $\Phi[X/\neg X]$ is the formula obtained by substituting all free occurrences of X by the formula $\neg X$. We also use $\Phi_1 \Rightarrow \Phi_2$ as an abbreviation for $\neg\Phi_1 \vee \Phi_2$.

Let us consider some interesting examples of $\mathcal{M}_\mu$ formulae (we assume that the scope of μ and ν extends to the right as much as possible):

(1) $\langle\varrho\rangle tt$ expresses the capability of executing some action satisfying ϱ.

(2) $[\varrho]f\!f$ states the inability of executing any action satisfying ϱ.

(3) $\langle \mathbf{any}\rangle tt \wedge [\neg\varrho]f\!f$ indicates the inevitability/necessity of executing some action satisfying ϱ.

(4) $\mu X.\Phi \vee \langle\mathbf{any}\rangle X$ expresses that there *exists an evolution* of the system such that Φ *eventually* holds. Indeed, its extension $\mathcal{E}_\mu$ is the smallest set that includes (1) the states in the extension of Φ; and (2) the states that can execute an action leading to a successive state that is in $\mathcal{E}_\mu$. In other words, the extension $\mathcal{E}_\mu$ includes each state s such that there exists a run from s leading eventually (i.e., in a finite number of steps) to a state in the extension of Φ. Note the inductive nature of this property which is typical of properties defined by least fixpoint.

(5) $\nu X.\Phi \wedge [\mathbf{any}]X$—i.e., $\neg(\mu X.\neg\Phi \vee \langle\mathbf{any}\rangle X)$—expresses the *invariance* of Φ under all of the evolutions of the system. Indeed, its extension $\mathcal{E}_\nu$ is the largest set of states in the extension of Φ from which every executable action leads to a successive state which is still in $\mathcal{E}_\nu$. In other words, the extension $\mathcal{E}_\nu$ includes each state s such that every state along every run from s is in the extension of Φ. Note the coinductive nature of this property which is typical of properties defined by greatest fixpoint.

(6) $\mu X.\Phi \vee (\langle\mathbf{any}\rangle tt \wedge [\mathbf{any}]X)$ expresses that for *all evolutions* of the system, Φ *eventually* holds. Indeed, its extension $\mathcal{E}_\mu$ is the smallest set that includes (1) the states in the extension of Φ; and (2) the states that can execute an action and such that every executable action leads to a state in $\mathcal{E}_\mu$. In other words, the extension $\mathcal{E}_\mu$

includes each state s such that every run from s leads eventually (i.e., in a finite number of steps) to a state in the extension of Φ.

In general, $\mathcal{M}_\mu$ allows for expressing very sophisticated properties of dynamic systems, such as forms of *liveness*, *safety*, and also *fairness* [55]. For example, the formula $\nu X.\mu Y.[a]((\langle b\rangle tt \wedge X) \vee Y)$ expresses a fairness constraint: b is possible infinitely often throughout any infinite length run consisting wholly of a actions.

Finally, note that if Φ is closed (no free variables are present in Φ), as in the examples above, then the extension of $(\Phi)^{\mathcal{T}}_{\mathcal{V}}$ is in fact independent of the valuation $\mathcal{V}$. It is usual to say that *a closed Φ is true in a state s of the transition system $\mathcal{T}$* iff $s \in (\Phi)^{\mathcal{T}}_{\mathcal{V}}$ for any valuation $\mathcal{V}$ (the extension of Φ is in fact independent of $\mathcal{V}$ with Φ closed).

4.2. Model checking

In the setting proposed in this paper, the reasoning problem of interest is model checking: given a transition system and one of its states, verify whether a certain closed formula is true in such a state. The formal definition of model checking in our setting is then the following one.

Definition 4.1. Let $\mathcal{T} = (\mathcal{S}, \{\mathcal{R}_\alpha \mid \alpha \in 2^{\mathcal{A}}\}, \Pi)$ be a transition system with Π mapping propositions in $\mathcal{P}$ to subsets of $\mathcal{S}$, let $s \in \mathcal{S}$ be one of its states, and let Φ be a closed (no free variables are present) $\mathcal{M}_\mu$ formula. The related *model checking* problem is to verify whether

$$s \in (\Phi)^{\mathcal{T}}_{\mathcal{V}}$$

where $\mathcal{V}$ is any valuation, since Φ is closed.

In the following we abbreviate $s \in (\Phi)^{\mathcal{T}}_{\mathcal{V}}$ by $\mathcal{T}, s \models \Phi$ or simply by $s \models \Phi$ referring to $\mathcal{T}$ only implicitly.

5. Reasoning about actions

Having presented both the representation formalism and the reasoning formalism, we can discuss how reasoning about actions is done in this setting. The basic idea is to use model checking.

Specifically, given an initial configuration $(p_{init}, \sigma_{init})$ and a $\mathcal{M}_\mu$ formulae Φ, we verify:

$$(p_{init}, \sigma_{init}) \models \Phi$$

where the transition system we are implicitly referring to is the $(p_{init}, \sigma_{init})$-generated transition system.

Let us consider some examples. First, consider the Russian Turkey scenario in Section 3 with $p_{init} = \texttt{P}$ and $\sigma_{init} = \{\texttt{Alive}, \neg\texttt{Loaded}\}$. We can verify that:

$$(p_{init}, \sigma_{init}) \models (\langle\texttt{load}\rangle\langle\texttt{wait}\rangle\langle\texttt{shoot}\rangle tt) \wedge ([\texttt{load}][\texttt{wait}][\texttt{shoot}]\neg\texttt{Alive})$$

that is, the sequence of actions `load`, `wait`, `shoot` can be performed and (necessarily) results in having killed the turkey. Observe that, this is a typical instance of the so called *projection problem*: given an initial configuration and a sequence of actions, determine the truth-value of a certain fact in the resulting configuration.

The nondeterminism of the action spin is reflected in the following property:

$$(p_{init}, \sigma_{init}) \models (\langle \texttt{load}\rangle\langle \texttt{spin}\rangle\langle \texttt{shoot}\rangle \texttt{Alive}) \wedge (\langle \texttt{load}\rangle\langle \texttt{spin}\rangle\langle \texttt{shoot}\rangle \neg \texttt{Alive})$$

that is, the sequence of actions load, spin, shoot may result either in having killed the turkey or not. However it (necessarily) results in having unloaded the gun, since

$$(p_{init}, \sigma_{init}) \models (\langle \texttt{load}\rangle\langle \texttt{spin}\rangle\langle \texttt{shoot}\rangle tt) \wedge ([\texttt{load}][\texttt{spin}][\texttt{shoot}]\neg \texttt{Loaded}).$$

Consider now the following instance of model checking:

$$(p_{init}, \sigma_{init}) \models \mu X.\phi_g \vee \langle \mathbf{any}\rangle X.$$

It expresses the existence of a (not yet determined) sequence of actions that, starting from the initial configuration, can reach a configuration where ϕ_g is true.

If only deterministic atomic actions are allowed, then the one above is a formalization of the *planning problem*: it asks for a sequence of actions—a *plan*—to reach the *goal* ϕ_g starting from the initial configuration. Thus, we may do planning by using model checking techniques.

If nondeterministic atomic actions are allowed, the above formalization of planning is too weak since it expresses only the possibility that a certain sequence of actions achieves the goal. For example, in the Russian Turkey scenario

$$(p_{init}, \sigma_{init}) \models \mu X.\neg \texttt{Alive} \vee \langle \mathbf{any}\rangle X$$

is verified by the sequence of actions load, spin, shoot, yet the execution of load, spin, shoot does not necessarily achieves the goal of having killed the turkey, as shown in Section 3.1.

However, we can still formalize the planning problem as follows:

$$(p_{init}, \sigma_{init}) \models \mu X.\phi_g \vee \bigvee_{a \in Act} \langle a\rangle tt \wedge [a]X$$

which expresses the existence of a (not yet determined) sequence of actions that, starting from the initial configuration, *necessarily* reaches a configuration where ϕ_g is true. For example, in the Russian Turkey scenario

$$(p_{init}, \sigma_{init}) \models \mu X.\neg \texttt{Alive} \vee \bigvee_{a \in Act} \langle a\rangle tt \wedge [a]X$$

is verified by the sequence of actions load, wait, shoot, but not by load, spin, shoot.[17]

[17] If concurrency is taken into account, the planning problem become more involved, since issues such as which agent is in control of a given atomic action, which agent is supposed to execute a given plan (or part of a plan), etc., become relevant. Moreover, other forms of planning, which are closer to the synthesis of a control process than to the generation of a sequence of actions, may be more appropriate in this context. Although some of these issues can be tackled within the proposed setting, we do not discuss them further here.

Next, consider the case discussed at the end of Section 3.1 where we have incomplete information on the initial situation (in particular, we do not know whether the gun is loaded). Checking whether a property Φ holds in the initial situation is reduced to checking whether

$$(\mathtt{Q}, s_q) \models [\mathtt{initAlive}][\mathtt{initLoaded}]\Phi$$

that is, checking whether Φ is true in every configuration right after the initialization sequence.[18] For example, even if we do not know whether the gun is loaded, we can verify that there exists a plan to kill the turkey:

$$(\mathtt{Q}, s_q) \models [\mathtt{initAlive}][\mathtt{initLoaded}]\left(\mu X.\neg\mathtt{Alive} \vee \bigvee_{a\in Act} \langle a\rangle tt \wedge [a]X\right)$$

It is easy to see that a possible plan is: `load` followed by `shoot`.

Let us now consider the Lifting a Table scenario as formalized in Section 3. Let the initial configuration be $(p_{init}, \sigma_{init})$ with:

$$p_{init} = \mathtt{LT} \text{ and } \sigma_{init} = \{\mathtt{VaseOnTable}, \mathtt{DownLeftSide}, \mathtt{DownRightSide}\}.$$

We can verify that if agents $\mathtt{A_l}$ and $\mathtt{A_r}$ raise the table synchronously, then the vase won't fall on the floor.

$$(p_{init}, \sigma_{init}) \models [\mathtt{upLeft} \wedge \mathtt{upRight}][\mathtt{vaseFalls}]ff.$$

Instead, if they do not synchronize, the vase falls off the table:

$$(p_{init}, \sigma_{init}) \models [(\mathtt{upLeft} \wedge \neg\mathtt{upRight}) \vee (\neg\mathtt{upLeft} \wedge \mathtt{upRight})]$$
$$[\mathbf{any}]\neg\mathtt{VaseOnTable}.$$

We can also prove that whenever the vase can fall, it does fall:[19]

$$(p_{init}, \sigma_{init}) \models \nu X.((\langle\mathtt{vaseFalls}\rangle tt \Rightarrow \langle\mathbf{any}\rangle tt \wedge [\neg\mathtt{vaseFalls}]ff) \wedge [\mathbf{any}]X.$$

Finally, consider the Relay Race scenario as formalized in Section 3. Let the initial configuration be $(p_{init}, \sigma_{init})$ with $p_{init} = \mathtt{RR}$ and σ_{init} such that *Ready* is true. We can verify that at the beginning, the action `go` *must* be executed:

$$(p_{init}, \sigma_{init}) \models \langle\mathtt{go}\rangle tt \wedge [\neg\mathtt{go}]ff$$

That is, in the initial configuration `go` is executable, and (synchronized) actions not including `go` are not executable. In fact, it is easy to verify that no other atomic action is executable.

We can also verify that both teams may win:

$$(p_{init}, \sigma_{init}) \models (\mu X.\mathtt{Won_1} \vee \langle\mathbf{any}\rangle X) \wedge (\mu X.\mathtt{Won_2} \vee \langle\mathbf{any}\rangle X)$$

That is, from the initial configuration, there exists an execution where team 1 wins and there exists an execution where team 2 wins. Moreover, for all executions, either team 1 or team 2 wins:

$$(p_{init}, \sigma_{init}) \models \mu X.(\mathtt{Won_1} \vee \mathtt{Won_2}) \vee [\mathbf{any}]X$$

[18] Note that, it is always possible to execute the initialization sequence.

[19] Even when we weaken the restriction in LT to be $\cdot\backslash\{\phi \rightarrow \mathtt{vaseFalls}\}$.

Furthermore, it is impossible that both teams win. Indeed, we can verify that as soon as one of the team wins, the other cannot win anymore.

$$(p_{init}, \sigma_{init}) \models \nu X.((\mathtt{Won}_1 \Rightarrow \nu Y.\neg\mathtt{Won}_2 \wedge [\mathbf{any}]Y) \wedge (\mathtt{Won}_2 \Rightarrow \nu Y.\neg\mathtt{Won}_1 \wedge [\mathbf{any}]Y)) \wedge [\mathbf{any}]X$$

That is, in every configuration, if $\mathtt{Won}_1$ ($\mathtt{Won}_2$) is true, then $\mathtt{Won}_2$ ($\mathtt{Won}_1$) is false from then on along all possible configuration's evolutions.

6. Reasoning techniques

In this section, we derive a technique to perform model checking within the proposed setting. We do so, by devising two transformation functions:

- A transformation F from transition systems whose arcs represent sets of atomic actions (synchronized actions) to transition systems whose arcs represent single atomic actions.
- A transformation H from $\mathcal{M}_\mu$ formulae, which allow for boolean combinations of atomic actions in the modalities, to standard modal mu-calculus formulae, which allow only for single atomic actions in the modalities.

The setting resulting from applying such transformations is a standard one for which various model checking techniques have been developed (see, e.g., [19,55]). Hence, by means of the transformations F and H, we can make use of such model checking techniques.

The idea at the base of the transformations F and H is to reify transitions, i.e., to introduce a new state for each transition, so that the action formula is transformed into a formula on the new state. Fig. 3 illustrates the reification:

- Fig. 3(a) illustrates the original transition from the state s to the state t. We have that $(s,t) \in \mathcal{R}_\alpha$ and σ_s (σ_t) is the propositional interpretation associated with s (t). It assigns to each primitive proposition $A \in \mathcal{P}$ the truth-value tt iff $s \in \Pi(A)$ ($t \in \Pi(A)$).
- Fig. 3(b) illustrates the resulting reified transition constituted by the transition from the state s to a newly introduced state, denoted by (s,α,t), and the transition from (s,α,t) to the state t. We require that (1) $(s,(s,\alpha,t)) \in \mathcal{R}_{w_1}$ and $((s,\alpha,t),t) \in \mathcal{R}_{w_2}$; (2) σ_s and σ_t be the propositional interpretations associated with s and t, respectively (the same as in the original transition); (3) σ_α be the propositional interpretation associated with (s,α,t), which assigns to each atomic action $a \in \mathcal{A}$ the truth-value tt iff $a \in \alpha$.

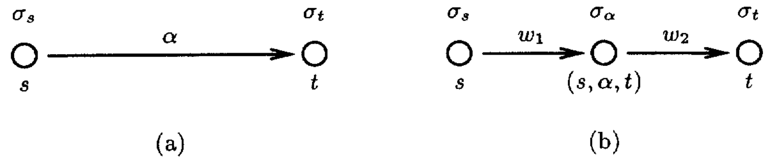

Fig. 3. Transition reification: (a) the original one; (b) the transformed one.

Definition 6.1. Let $\mathcal{A}$ be a set of actions and $\mathcal{P}$ a set of propositions. Given transition system $\mathcal{T} = (\mathcal{S}, \{\mathcal{R}_\alpha \mid \alpha \in 2^{\mathcal{A}}\}, \Pi)$ with Π mapping propositions in $\mathcal{P}$ to subsets of S, we define:

$$F(\mathcal{T}) = \bigl(\mathcal{S}^F, \{\mathcal{R}^F_{w_1}, \mathcal{R}^F_{w_2}\}, \Pi^F\bigr)$$

where

$$\begin{aligned}
&\mathcal{S}^F = \mathcal{S} \cup \{(s,\alpha,t) \mid \alpha \subseteq \mathcal{A}, (s,t) \in \mathcal{R}_\alpha\}\\
&\mathcal{R}^F_{w_1} = \{(s,(s,\alpha,t)) \mid \alpha \subseteq \mathcal{A}, (s,t) \in \mathcal{R}_\alpha\}\\
&\mathcal{R}^F_{w_2} = \{((s,\alpha,t),t) \mid \alpha \subseteq \mathcal{A}, (s,t) \in \mathcal{R}_\alpha\}\\
&\Pi^F(p) = \{s \mid s \in \Pi(p)\} \quad \text{for each } p \in \mathcal{P}\\
&\Pi^F(a) = \{(s,\alpha,t) \mid (s,s') \in \mathcal{R}_\alpha,\ a \in \alpha\} \quad \text{for each } a \in \mathcal{A}.
\end{aligned}$$

In addition, given a valuation $\mathcal{V}$ on $\mathcal{T}$, we define:

$$F(\mathcal{V})(X) = \{s \mid s \in \mathcal{V}(X)\} \quad \text{for each } X \in \mathit{Var}.$$

Observe that, arcs in the transition system $F(\mathcal{T})$ can only be labeled by either w_1 or w_2.

Definition 6.2. Given a $\mathcal{M}_\mu$ formula Φ, we define $H(\Phi)$ inductively as follows:

$$\begin{aligned}
&H(A) = A && H(\langle\varrho\rangle\Phi) = \langle w_1\rangle(\varrho \wedge \langle w_2\rangle H(\Phi))\\
&H(tt) = tt && H([\varrho]\Phi) = [w_1](\varrho \Rightarrow [w_2]H(\Phi))\\
&H(ff) = ff && H(\mu X.\Phi) = \mu X.H(\Phi)\\
&H(\neg\Phi) = \neg H(\Phi) && H(\nu X.\Phi) = \nu X.H(\Phi)\\
&H(\Phi_1 \wedge \Phi_2) = H(\Phi_1) \wedge H(\Phi_2) && H(X) = X\\
&H(\Phi_1 \vee \Phi_2) = H(\Phi_1) \vee H(\Phi_2) &&
\end{aligned}$$

Observe that, $H(\Phi)$ is a standard modal mu-calculus formula, since only single actions (w_1 or w_2) may appear in the modalities.

Theorem 6.3. *Let transition system $\mathcal{T} = (\mathcal{S}, \{\mathcal{R}_\alpha \mid \alpha \in 2^{\mathcal{A}}\}, \Pi)$. The size of $F(\mathcal{T})$ is linearly bounded by the size of $\mathcal{T}$.*

Proof. For any transition system $\mathcal{T}$, let *state_no.*$(\mathcal{T})$ and *edge_no.*$(\mathcal{T})$ denote the number of the states in $\mathcal{T}$ and the number of the edges in $\mathcal{T}$, respectively. By the definition of F, we have that *state_no.*$(F(\mathcal{T}))$ = *state_no.*$(\mathcal{T})$ + *edge_no.*$(\mathcal{T})$, and *edge_no.*$(F(\mathcal{T}))$ = $2 \times$ *edge_no.*$(\mathcal{T})$. □

Theorem 6.4. *Let Φ be a $\mathcal{M}_\mu$ formula. The size of $H(\Phi)$ is linearly bounded by the size of Φ.*

Proof. Let $size(\Phi)$ denote the size of formula Φ. It is easy to verify by induction on the structure of the formula Φ, that $size(H(\Phi)) < c \cdot size(\Phi)$. $\square$

Theorem 6.5. *Let transition system $\mathcal{T} = (\mathcal{S}, \{\mathcal{R}_\alpha \mid \alpha \in 2^{\mathcal{A}}\}, \Pi)$ and Φ a $\mathcal{M}_\mu$ formula. Then for every valuation $\mathcal{V}$ on $\mathcal{T}$ and for every $s \in \mathcal{S}$, we have:*

$$s \in (\Phi)^{\mathcal{T}}_{\mathcal{V}} \quad \textit{iff} \quad s \in (H(\Phi))^{F(\mathcal{T})}_{F(\mathcal{V})}.$$

Proof. First, we show by induction on the structure of ϱ that:

$$\big((s, s') \in \mathcal{R}_\alpha \text{ and } \alpha \models \varrho\big) \quad \text{iff} \quad \big(s, \alpha, s'\big) \in (\varrho)^{F(\mathcal{T})}_{F(\mathcal{V})}.$$

Indeed, for $\varrho = a$: if $(s, s') \in \mathcal{R}_\alpha$ and $\alpha \models a$—i.e., $a \in \alpha$—then, by definition of $F(\mathcal{T})$, we have $(s, \alpha, s') \in \Pi^F(a)$; on the converse, if $(s, \alpha, s') \in \Pi^F(a)$ then, again by definition of $F(\mathcal{T})$, we have $(s, s') \in \mathcal{R}_\alpha$ and $a \in \alpha$, i.e., $\alpha \models a$. For the other cases, the result follows immediately by the induction hypothesis.

Now, we are ready to prove the thesis of the theorem. Without loss of generality, we restrict our attention to formulae Φ of the form:

$$\Phi ::= A \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \langle\varrho\rangle\Phi \mid \mu X.\Phi \mid X.$$

The proof is given by the induction on the number of nested fixpoints constructs $\mu X.\Phi$.

Base case. If no fixpoint constructs are present in Φ, then we can verify that the thesis holds by the induction of the structure of Φ. All cases are immediate except the case:

$$s \in (\langle\varrho\rangle\Psi)^{\mathcal{T}}_{\mathcal{V}} \quad \text{iff} \quad s \in \big(\langle w_1\rangle(\varrho \wedge \langle w_2\rangle H(\Psi))\big)^{F(\mathcal{T})}_{F(\mathcal{V})}$$

that we prove below.

($\Longrightarrow$) By definition of $\cdot^{\mathcal{T}}_{\mathcal{V}}$, $s \in (\langle\varrho\rangle\Psi)^{\mathcal{T}}_{\mathcal{V}}$ implies that there exist α and s' such that $(s, s') \in \mathcal{R}_\alpha$, $\alpha \models \varrho$, and $s' \in \Psi^{\mathcal{T}}_{\mathcal{V}}$. By induction hypothesis, we have $s' \in \Psi^{\mathcal{T}}_{\mathcal{V}}$ iff $s' \in (H(\Psi))^{F(\mathcal{T})}_{F(\mathcal{V})}$. By the definition of $F(\mathcal{T})$, we have $((s, \alpha, s'), s') \in \mathcal{R}^F_{w_2}$, so we have $(s, \alpha, s') \in (\langle w_2\rangle H(\Psi))^{F(\mathcal{T})}_{F(\mathcal{V})}$. Moreover since $(s, s') \in \mathcal{R}_\alpha$ and $\alpha \models \varrho$, we have $(s, \alpha, s') \models (\varrho)^{F(\mathcal{T})}_{F(\mathcal{V})}$. Finally, by definition of $F(\mathcal{T})$, we have $(s, \ (s, \alpha, s')) \in \mathcal{R}^F_{w_1}$. Hence, we can conclude $s \in (\langle w_1\rangle(\varrho \wedge \langle w_2\rangle H(\Psi)))^{F(\mathcal{T})}_{F(\mathcal{V})}$.

($\Longleftarrow$) By definition of $\cdot^{F(\mathcal{T})}_{F(\mathcal{V})}$, $s \in (\langle w_1\rangle(\varrho \wedge \langle w_2\rangle H(\Psi)))^{F(\mathcal{T})}_{F(\mathcal{V})}$ implies that there exist α and s' such that $(s, (s, \alpha, s')) \in \mathcal{R}^F_{w_1}$, $((s, \alpha, s'), s') \in \mathcal{R}^F_{w_2}$, $(s, \alpha, s') \in (\varrho)^{F(\mathcal{T})}_{F(\mathcal{V})}$ and $s' \in (H(\Psi))^{F(\mathcal{T})}_{F(\mathcal{V})}$. By induction hypothesis, we have $s' \in \Psi^{\mathcal{T}}_{\mathcal{V}}$ iff $s' \in (H(\Psi))^{F(\mathcal{T})}_{F(\mathcal{V})}$. Moreover, since $(s, \alpha, s') \models (\varrho)^{F(\mathcal{T})}_{F(\mathcal{V})}$, we have $(s, s') \in \mathcal{R}_\alpha$ and $\alpha \models \varrho$. Hence, we can conclude $s \in (\langle\varrho\rangle\Psi)^{F(\mathcal{T})}_{F(\mathcal{V})}$.

Inductive case. Let us assume that the thesis holds for the formula Ψ with k nested fixpoint constructs. We prove it for $\Phi = \mu X.\Psi$ with $k + 1$. We recall that, by the Tarski–Knaster Theorem on fixpoints [58], $s \in (\mu X.\Psi)^{\mathcal{T}}_{\mathcal{V}}$ iff there exists an ordinal ξ such that $s \in (\mu_\xi X.\Psi)^{\mathcal{T}}_{\mathcal{V}}$, where $(\mu_\xi X.\Psi)^{\mathcal{T}}_{\mathcal{V}}$ is defined by transfinite induction as:

- $(\mu_0 X.\Psi)^{\mathcal{T}}_{\mathcal{V}} = \emptyset$.
- $(\mu_{\xi+1} X.\Psi)^{\mathcal{T}}_{\mathcal{V}} = \Psi^{\mathcal{T}}_{\mathcal{V}[X/(\mu_\xi X.\Psi)^{\mathcal{T}}_{\mathcal{V}}]}$.
- $(\mu_\lambda X.\Psi)^{\mathcal{T}}_{\mathcal{V}} = \bigcup_{\xi<\lambda} (\mu_\xi X.\Psi)^{\mathcal{T}}_{\mathcal{V}}$, if λ is a limit ordinal.

We proceed by transfinite induction on ordinals ξ.

Base case of the transfinite induction. $\mu_0 X.\Psi$ is defined as *ff*, thus trivially we have $s \in (\mu_0 X.\Psi)^{\mathcal{T}}_{\mathcal{V}}$ iff $s \in (\mu_0 X.H(\Psi))^{F(\mathcal{T})}_{F(\mathcal{V})}$.

Successor case of the transfinite induction. We want to show that $s \in (\mu_{\xi+1} X.\Psi)^{\mathcal{T}}_{\mathcal{V}}$ iff $s \in (\mu_{\xi+1} X.H(\Psi))^{F(\mathcal{T})}_{F(\mathcal{V})}$, which, by definition, reduces to:

$$s \in (\Psi)^{\mathcal{T}}_{\mathcal{V}[X/(\mu_\xi X.\Psi)^{\mathcal{T}}_{\mathcal{V}}]} \quad \text{iff} \quad s \in (\Psi)^{F(\mathcal{T})}_{F(\mathcal{V})[X/(\mu_\xi X.H(\Psi))^{F(\mathcal{T})}_{F(\mathcal{V})}]}. \tag{1}$$

Since Ψ contains k fixpoint constructs, by inductive hypothesis on k, we have:

$$s \in (\Psi)^{\mathcal{T}}_{\mathcal{V}[X/(\mu_\xi X.\Psi)^{\mathcal{T}}_{\mathcal{V}}]} \quad \text{iff} \quad s \in \big(H(\Psi)\big)^{F(\mathcal{T})}_{F(\mathcal{V}[X/(\mu_\xi X.\Psi))^{\mathcal{T}}_{\mathcal{V}}])}.$$

So it remains to prove that:

$$s \in \big(H(\Psi)\big)^{F(\mathcal{T})}_{F(\mathcal{V}[X/(\mu_\xi X.\Psi)^{\mathcal{T}}_{\mathcal{V}}])} \quad \text{iff} \quad s \in \big(H(\Psi)\big)^{F(\mathcal{T})}_{[X/(\mu_\xi X.H(\Psi))^{F(\mathcal{T})}_{F(\mathcal{V})}]}. \tag{2}$$

Notice that the two valuations above may differ only on the value of X. If it holds that:

$$s \in (X)^{F(\mathcal{T})}_{F([X/(\mu_\xi X.\Psi)^{\mathcal{T}}_{\mathcal{V}}])} \quad \text{iff} \quad s \in (X)^{F(\mathcal{T})}_{F(\mathcal{V})[X/(\mu_\xi X.H(\Psi))^{F(\mathcal{T})}_{F(\mathcal{V})}]} \tag{3}$$

then by straightforward induction on the formation of $H(\Psi)$, we have that (2) holds as well. Let us prove (3), which can be written as:

$$s \in F\big(\mathcal{V}\big[X/(\mu_\xi X.\Psi)^{\mathcal{T}}_{\mathcal{V}}\big]\big)(X) \quad \text{iff} \quad s \in F(\mathcal{V})\big[X/(\mu_\xi X.H(\Psi))^{F(\mathcal{T})}_{F(\mathcal{V})}\big](X).$$

By definition of F on valuations, this reduces to:

$$s \in (\mu_\xi X.\Psi)^{\mathcal{T}}_{\mathcal{V}} \quad \text{iff} \quad s \in \big(\mu_\xi X.H(\Psi)\big)^{F(\mathcal{T})}_{F(\mathcal{V})}$$

which indeed holds by transfinite inductive hypothesis.

Hence, considering (1) and (2), we can conclude that $s \in (\mu_{\xi+1} X.\Psi)^{\mathcal{T}}_{\mathcal{V}}$ iff $s \in (\mu_{\xi+1} X.H(\Psi))^{F(\mathcal{T})}_{F(\mathcal{V})}$.

Limit case of the transfinite induction. Let λ be a limit ordinal, then $s \in (\mu_\lambda X.\Psi)^{\mathcal{T}}_{\mathcal{V}}$ iff there exists an ordinal $\xi < \lambda$ such that $s \in (\mu_\xi X.\Psi)^{\mathcal{T}}_{\mathcal{V}}$. By transfinite induction hypothesis, it holds that $s \in (\mu_\xi X.\Psi)^{\mathcal{T}}_{\mathcal{V}}$ iff $s \in (\mu_\xi X.H(\Psi))^{F(\mathcal{T})}_{F(\mathcal{V})}$, and thus

$$s \in (\mu_\lambda X.\Psi)^{\mathcal{T}}_{\mathcal{V}} \quad \text{iff} \quad s \in \big(\mu_\lambda X.H(\Psi)\big)^{F(\mathcal{T})}_{F(\mathcal{V})}.$$

This completes the transfinite induction. So for all ordinals ξ it holds that

$$s \in (\mu_\xi X.\Psi)^{\mathcal{T}}_{\mathcal{V}} \quad \text{iff} \quad s \in \big(\mu_\xi X.H(\Psi)\big)^{F(\mathcal{T})}_{F(\mathcal{V})}.$$

The induction on the nesting of fixpoint constructs is completed as well, hence we have proven the theorem. □

Theorem 6.5 gives us a sound and complete technique to do model checking in the setting proposed in this paper. To check if

$$\mathcal{T}, s \models \Phi$$

check if

$$F(\mathcal{T}), s \models H(\Phi).$$

Theorems 6.3 and 6.4 which state that $F(\mathcal{T})$ and $H(\Phi)$ are linearly related to $\mathcal{T}$ and Φ, respectively, allow us to conclude that such a technique is, in fact, quite efficient.

The problem of model checking a standard mu-calculus formula in a finite transition system is known to be in NP $\cap$ coNP [21]. Model checking algorithms are known that run in $(|\mathcal{T}| \cdot |\Phi|)^{\mathrm{o}(k)}$ [22], where $|\mathcal{T}|$ is the size of the transition system $\mathcal{T}$, $|\Phi|$ is the size of the formula Φ, and k is the number of "alternating" least and greatest fixpoints whose variables are one within the scope of the other (see [19]). Moreover, the properties that are typically of interest can be expressed with a very small number of alternating fixpoints (one or two), and hence typically model checking can be performed within a low order polynomial time. By Theorems 6.3 and 6.4, such results can be applied immediately to our setting. [20]

We conclude the section by observing that, from a practical point of view, the above transformations allow us to use the software tools such as the Edinburgh Concurrency Workbench [10] [21] or the Concurrency Workbench of North Carolina [11], [22] that have been implemented for the automated model checking of standard modal mu-calculus formulae.

7. Further issues on reasoning about dynamic systems

In this section, we discuss two important issues related to the representation of and reasoning on dynamic systems.

First, we relate model checking in our setting to logical implication, showing that the former is a special case of the latter. We get this special case when we have enough information to isolate a single model, hence reducing logical implication to the simple verification of the truth-value of a formula.

Second, we discuss the issue of equivalent descriptions. In the context of process algebras, the equivalence of two descriptions of the same system has been well investigated and various tools have been implemented for verifying the equivalences. Here we show that the equivalence relation deduced by $\mathcal{M}_\mu$ coincides with a natural extension of the well-known bisimulation equivalence which has been proved of an experimental impact.

[20] Note that the transformation H does not change the number of alternating fixpoints.

[21] Available at `http://www.cs.bris.ac.uk/~neil/ExternalLinks/comms94/cwb/cwb.html`.

[22] Available at `http://www.csc.ncsu.edu/eos/users/r/rance/WWW/cwb-nc.html`.

7.1. Relating model checking to logical implication

In this subsection, we relate model checking to logical implication, following the line of reasoning in [25,27,52,53]. Given a finite transition system $\mathcal{T}$, it is not difficult to build a set of formulae $D_{\mathcal{T}}$ that encode $\mathcal{T}$.

Let $\mathcal{T} = (\mathcal{S}, \{\mathcal{R}_\alpha \mid \alpha \in 2^{\mathcal{A}}\}, \Pi)$ be a transition system, the set of formulae $D_{\mathcal{T}}$ is obtained by including, for each $s \in \mathcal{S}$, a formula of the form:

$$\begin{aligned}\theta_s \equiv\ & \bigwedge_{s \in \Pi(A)} A \wedge \\ & \bigwedge_{s \notin \Pi(A)} \neg A \wedge \\ & \bigwedge_{(s,s') \in \mathcal{R}_\alpha} \langle\alpha\rangle \theta_{s'} \wedge \\ & \bigwedge_{\exists s'.(s,s') \in \mathcal{R}_\alpha} [\alpha] \bigvee_{(s,s') \in \mathcal{R}_\alpha} \theta_{s'} \wedge \\ & \bigwedge_{\neg\exists s'.(s,s') \in \mathcal{R}_\alpha} [\alpha] \mathit{ff}\end{aligned}$$

The set of formulae $D_{\mathcal{T}}$ encodes the transition system $\mathcal{T}$, in the sense given by the following proposition.

Proposition 7.1. *Given finite transition system $\mathcal{T} = (\mathcal{S}, \{\mathcal{R}_\alpha \mid \alpha \in 2^{\mathcal{A}}\}, \Pi)$, for every $s \in \mathcal{S}$ and every $\mathcal{M}_\mu$ formulae Φ, we have:*

$$\mathcal{T}, s \models \Phi \quad \textit{iff} \quad D_{\mathcal{T}} \models \theta_s \Rightarrow \Phi.$$

This proposition follows directly from the work on *characteristic formulae* in [25,52, 53], by applying the transformations F and H defined in the previous section.

The above result shows that model checking can be seen as a *special case* of logical implication. The set of formulae $D_{\mathcal{T}}$ can be seen as providing enough information to essentially single out a unique model, and thus logical implication is reduced to the verification of the truth-value of a formula in such a model, i.e., it is reduced to model checking.

Note that, from the practical point of view, using a generic theorem prover for $\mathcal{M}_\mu$ to do logical implication in $D_{\mathcal{T}}$ instead of model checking in $\mathcal{T}$, although possible, is highly inefficient. Indeed $D_{\mathcal{T}}$ has essentially the same size as $\mathcal{T}$, and logical implication for $\mathcal{M}_\mu$ is EXPTIME-complete, according to the linearity of transformation H and the EXPTIME-completeness of standard mu-calculus [19,20].

7.2. Equivalent descriptions

A question that naturally arises is when two descriptions of a dynamic system can be considered equivalent. Observe that in this case we are asking about a property of the descriptions which is not necessarily related to properties of the modeled dynamic system.

Adopting an algebraic approach for such descriptions allows us to benefit from the study of equivalence classes on processes, to identify equivalent descriptions.[23] Two main notions of equivalences have been proposed in the process algebra literature: *trace equivalence* [30], and *bisimulation equivalence* [45]. Trace equivalence identifies systems that have the same set of possible runs (traces). Bisimulation equivalence, on the other hand, identifies two systems if during every run whenever one system *can* perform a certain action, then the other system can perform the same action matching such a move.

Here we focus on bisimulation equivalence, since we are interested in identifying two systems not only on the base of their "traces" (trace equivalence) but also on the base of their "branching behaviors".

We introduce a natural extension of the bisimulation equivalence studied in [45] to our representation formalism. We say state s_1 of the transition system $\mathcal{T}_1$ is equivalent to s_2 of $\mathcal{T}_2$, if the truth-value of each primitive proposition is the same in both s_1 and s_2, and whenever s_1 can evolve into s_1' by a (synchronized) action α, s_2 can also do the action α arriving at a state s_2' which is equivalent to s_1', and vice versa. Formally:

Definition 7.2. Let $\mathcal{T} = (\mathcal{S}, \{\mathcal{R}_\alpha \mid \alpha \in 2^{\mathcal{A}}\}, \Pi)$ and $\mathcal{T}' = (\mathcal{S}', \{\mathcal{R}'_\alpha \mid \alpha \in 2^{\mathcal{A}}\}, \Pi')$ be two transition systems.

(1) $\mathfrak{R} \subseteq \mathcal{S} \times \mathcal{S}'$ is a *bisimulation* if for all $(r, r') \in \mathfrak{R}$, $\alpha \in \mathcal{A}$,
   (a) $\{A \mid r \in \Pi(A)\} = \{A \mid r' \in \Pi'(A)\}$;
   (b) $(r, t) \in \mathcal{R}_\alpha$ implies $\exists t' : (r', t') \in \mathcal{R}'_\alpha$ and $(t, t') \in \mathfrak{R}$; and
   (c) $(r', t') \in \mathcal{R}'_\alpha$ implies $\exists t : (r, t) \in \mathcal{R}_\alpha$ and $(t, t') \in \mathfrak{R}$.

(2) for $s \in \mathcal{S}$, $s' \in \mathcal{S}'$, s and s' are equivalent, written $s \approx_b s'$, if there exists a *bisimulation* $\mathfrak{R}$ such that $(s, s') \in \mathfrak{R}$.

The bisimulation equivalence $\approx_b$ can be expressed as a simple formula of first-order logic with fixpoints (first-order mu-calculus) [46,47]. As a consequence, the verification of bisimulation equivalence $\approx_b$ on finite transition systems can be performed in polynomial time with respect to the size of the systems (see, e.g., [1]).

We also remark that some investigations have been done to check bisimulation equivalence directly on process descriptions. In particular, algorithms that run in polynomial time with respect to the size of process descriptions have been devised for some typical forms of processes [29].

An alternative way to define equivalence of descriptions is to make use of logic: two systems are considered as the same iff no logical formula can distinguish them (see [28]).

Definition 7.3. Let $\mathcal{T} = (\mathcal{S}, \{\mathcal{R}_\alpha \mid \alpha \in 2^{\mathcal{A}}\}, \Pi)$ and $\mathcal{T}' = (\mathcal{S}', \{\mathcal{R}'_\alpha \mid \alpha \in 2^{\mathcal{A}}\}, \Pi')$ be two transition systems. For $s \in \mathcal{S}$, $s' \in \mathcal{S}'$, s and s' are equivalent with respect to $\mathcal{M}_\mu$, written $s \approx_{\mathcal{M}_\mu} s'$, iff

$$\{\Phi \in \mathcal{M}_\mu \mid \mathcal{T}, s \models \Phi\} = \{\Phi \in \mathcal{M}_\mu \mid \mathcal{T}', s' \models \Phi\}.$$

In fact, the two forms of equivalence $\approx_b$ and $\approx_{\mathcal{M}_\mu}$ coincide under the very loose condition of *image finiteness*. A configuration (p, σ) is *image finite*, if $\forall \alpha \in \mathcal{A}$. $\{(p', \sigma') \mid$

[23] Good comparisons of various notions of systems equivalences can be found in [16,60,61].

$((p,\sigma),(p',\sigma')) \in \mathcal{R}_\alpha\}$ is finite. A system is *image finite* if all its reachable configurations are image finite.

Proposition 7.4. *Let $\mathcal{T} = (\mathcal{S}, \{\mathcal{R}_\alpha \mid \alpha \in 2^{\mathcal{A}}\}, \Pi)$ and $\mathcal{T}' = (\mathcal{S}', \{\mathcal{R}'_\alpha \mid \alpha \in 2^{\mathcal{A}}\}, \Pi')$ be two image finite transition systems, and $s \in \mathcal{S}$, $s' \in \mathcal{S}'$.*

$$s \approx_b s' \quad \text{iff} \quad s \approx_{\mathcal{M}_\mu} s'.$$

This proposition follows from the analogous theorem [24] (see, e.g., [55]) on bisimulation equivalence [45] and standard mu-calculus, by applying the transformations F and H defined in the previous section.

We remark that Proposition 7.4 implies that $\mathcal{M}_\mu$ is well dimensioned for verifying properties of our descriptions, in the sense that, $\mathcal{M}_\mu$ distinguishes two transition systems if and only if they are not equivalent according to the bisimulation equivalence $\approx_b$. Or, in other words, it implies that the bisimulation equivalence $\approx_b$ captures exactly the notion of distinguishability with respect to the logic $\mathcal{M}_\mu$.

8. Conclusion

The research presented in this paper can be regarded as a bridge between the area of Reasoning about Actions in Artificial Intelligence and the area of Concurrency in Computer Science.

Specifically, we have presented a model checking based framework for reasoning about complex actions (processes) that are constituted by several concurrent activities performed by various interacting agents.

We have shown that this framework, arisen originally in the area of Concurrency in Computer Science, is well-suited for reasoning about complex actions in Artificial Intelligence, in the simplified but significant case of having complete information on the state of the world.

The strong connection with the area of Concurrency in Computer Science has allowed us to make use of the body of results devised in that area in the last decade, and to address issues like nonterminating executions, synchronizations, communications and interrupts, which have been hardly tacked so far in Artificial Intelligence.

Besides the technical results, this work gives some conceptual tools for better understanding of the issues involved in integrating concurrent processes within Reasoning about Actions in Artificial Intelligence. In particular, we are referring to:
- The separation of the specification of how atomic actions affect the state of the world from the specification of the process, as noticed in general for complex actions in [37].
- The possibility of specifying preconditions for actions (i.e., establishing *when* a given action can be executed) within the process in order to have them under the control of the process. This is the choice we have made in our proposal.
- The need to maintain, together with the information about the current state of the world (the global store, in our case), the information about the current state

[24] Such a theorem is in turn an extension of the Hennessy–Milner Theorem [28].

of the activities that are going on (the part of the process that remains to be executed).

- Related to the above point, the use of a "single-step" transitions, i.e., transitions that return together with the new state of the world what remains to be executed of the process. In general, this allows for a simple and elegant treatment of both concurrency and nonterminating behaviors (see [14,15] for a use of "single-step" transitions within the situation calculus).

Further technical extensions of the present work are possible along several directions. We outline some of them below.

The first extension concerns the form of the *update* on the global store. In Section 2, we have introduced a very simple form of the update to compute the set $\sigma/\{a_1, \ldots, a_n\}$ of the possible global store resulting by performing the action $\{a_1, \ldots, a_n\}$ on σ (see Definitions 2.1 and 2.2). However, the only essential point to retain precisely the proposed setting is to have some function returning the set $\sigma/\{a_1, \ldots, a_n\}$ from the inputs $\{a_1, \ldots, a_n\}$ and σ. It follows that we may adopt a more complex form of update, based, for example, on some notion of distance among global stores, and specify effects of actions as general formulae over *Prop* instead of literals. Moreover in this case, we can also address indirect effects by specifying domain constrains that must hold in each global store. Observe that the update we are interested in applies to interpretations, and thus is much simpler than update of theories discussed, e.g., in [31]. Research on $\mathcal{A}$-family action languages, whose semantics is based on defining a transition function (which is essentially a successor-state function in our terms), e.g., [2,5,23,39], is relevant.

Another possible extension concerns the form of the global store. In the present work, we describe the state of the world at a given point by a set of atomic propositions in the global store. That is, the global store can be thought of as a set of boolean variables, one for each atomic proposition. A possible extension is to consider the global store as a set of multi-valued variables, or even as a first-order interpretation over some fixed domain. Such an extension can be easily accommodated in our setting. Indeed, the way transition systems are built remains essentially the same, while the logic used for verification needs to be extended in order to take into account the new kind of properties expressed in the global store. Research in Databases on query languages based on first-order logic plus fixpoints (see, e.g., [1]) and that on complex transactions e.g., [4], are relevant.

Finally, let us consider again the levels of abstractions introduced in Section 1. We believe that it is of great interest mixing representations at level 2 and at level 3, by mixing the process algebra approach presented here with the usual logical approach. This would allow us to introduce incomplete information in a better controlled way. For example, we could specify agents whose behavior is completely known by means of process description presented here, and agents whose behavior is only partially known (as happens typically for the environment) by logical axioms. To this end, the research on "loose specification" in process algebras [7,35], as well as research in knowledge representation on description logics that include assertions about "individuals" (which can be interpreted as a partial description of a transition system) [13], is relevant.

Acknowledgements

We are deeply indebted to Rocco De Nicola for invaluable discussions on process algebra and modal mu-calculus. Also we would like to thank the anonymous referees for their remarks which have improved both the style and the content of the paper.

References

[1] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison Wesley, Reading, MA, 1995.

[2] C. Baral, M. Gelfond, Representing concurrent actions in extended logic programming, in: Proc. 13th International Joint Conference on Artificial Intelligence (IJCAI-93), Chambéry, France, 1993, pp. 866–871.

[3] J. Bergstra, J. Klop, Process algebra for synchronous communication, Inform. and Control 60 (1984) 109–137.

[4] A.J. Bonner, M. Kifer, Concurrency and communication in transaction logic, in: Proc. 5th International Conference on Database Theory (ICDT-95), 1995.

[5] S.-E. Bornscheuer, M. Thielscher, Representing concurrent actions and solving conflicts, Journal of the Interest Group in Pure and Applied Logics (IGPL) (Special issue of the ESPRIT project MEDLAR) 4 (3) (1996) 355–368.

[6] G. Boudol, R. de Simone, V. Roy, D. Vergamini, Process calculi, from theory to practice: Verification tools, in: Proc. Workshop on Automatic Verification Methods for Finite State Systems, Lecture Notes in Computer Science, Vol. 407, Springer, Berlin, 1990.

[7] G. Boudol, K. Larsen, Graphical versus logical specifications, Theoret. Comput. Sci. 106 (1992) 3–20.

[8] T. Bylander, Complexity results for planning, in: Proc. 12th International Joint Conference on Artificial Intelligence (IJCAI-91), Sydney, Australia, 1991, pp. 274–279.

[9] R. Cleaveland, Tableaux-based model checking in the propositional mu-calculus, Acta Informatica 27 (1990) 725–747.

[10] R. Cleaveland, J. Parrow, B. Steffen, The concurrency workbench: A semantics-based tool for the verification of concurrent systems, ACM Trans. Programming Languages Syst. 15 (1993) 36–72.

[11] R. Cleaveland, S. Sims, The NCSU concurrency workbench, in: Computer-Aided Verification (CAV-96), Lecture Notes in Computer Science, Vol. 1102, Springer, Berlin, 1996, pp. 394–397.

[12] M. Dam, CTL* and ECTL* as fragments of the modal mu-calculus, in: Proc. Colloquium on Trees and Algebra in Programming, Lecture Notes in Computer Science, Vol. 581, Springer, Berlin, 1992, pp. 145–164.

[13] G. De Giacomo, M. Lenzerini, Boosting the correspondence between description logics and propositional dynamic logics, in: Proc. 12th National Conference on Artificial Intelligence (AAAI-94), Seattle, WA, 1994, pp. 205–212.

[14] G. De Giacomo, H.J. Levesque, Y. Lesperance, Reasoning about concurrent executions, prioritized interrupts, and exogenous action in the situation calculus, in: Proc. 15th International Joint Conference on Artificial Intelligence (IJCAI-97), Nagoya, Japan, 1997.

[15] G. De Giacomo, R. Reiter, M. Soutchanski, Execution monitoring of high-level robot programs, in: Proc. 6th International Conference on Principles of Knowledge Representation and Reasoning (KR'98), 1998.

[16] R. De Nicola, Extentional equivalences for transition system, Acta Informatica 24 (1987) 211–237.

[17] E. Emerson, J. Halpern, "Sometimes" and "not never" revisited: On branching time versus linear time temporal logic, J. ACM 33 (1) (1986) 151–178.

[18] E.A. Emerson, Handbook of Theoretical Computer Science, Vol. B, Elsevier, Amsterdam, 1990, Chapter 16.

[19] E.A. Emerson, Automated temporal reasoning about reactive systems, in: Logics for Concurrency: Structure versus Automata, Lecture Notes in Computer Science, Vol. 1043, Springer, Berlin, 1996, pp. 41–101.

[20] E.A. Emerson, C.S. Jutla, The complexity of tree automata and logics of programs, in: Proc. 20th Annual Symposium on the Foundations of Computer Science, 1988, pp. 328–337.

[21] E.A. Emerson, C.S. Jutla, A.P. Sistla, On model checking for fragments of the mu-calculus, in: Proc. 5th International Conference of Computer-Aided Verification, Lecture Notes in Computer Science, Vol. 697, Springer, Berlin, 1993, pp. 385–396.

[22] E.A. Emerson, C.-L. Lei, Efficient model checking in fragments of the mu-calculus, in: Proc. 1st IEEE Symposium on Logics in Computer Science (LICS'86), 1986, pp. 267–278.

[23] M. Gelfond, V. Lifschitz, Representing action and change by logic programs, J. Logic Programming 17 (1993) 301–322.

[24] M. Gelfond, V. Lifschitz, A. Rabinov, What are the limitations of the situation calculus? in: R. Boyer (Ed.), Automated Reasoning: Essays in Honor of Woody Bledsoe, Kluwer, Dordrecht, 1991, pp. 167–179.

[25] S. Graf, J. Sifakis, A model characterization of observational congruence on finite terms of CCS, Inform. and Control 68 (1986) 125–145.

[26] F.N.H.R. Nielson, Semantics with Applications, Wiley, New York, 1992.

[27] J.Y. Halpern, M.Y. Vardi, Model checking vs. theorem proving: A manifesto, in: V. Lifschitz (Ed.), Artificial Intelligence and Mathematical Theory of Computation—Papers in Honor of John McCarthy, Academic Press, New York, 1991, pp. 151–176.

[28] M. Hennessy, R. Milner, Algebraic laws for nondeterminism and concurrency, J. ACM 32 (1) (1985) 137–161.

[29] Y. Hirishfeld, F. Moller, Decidability results in automata and process theory, in: Logics for Concurrency: Structure versus Automata, Lecture Notes in Computer Science, Vol. 1043, Springer, Berlin, 1996, pp. 102–148.

[30] C. Hoare, Communicating Sequential Processes, Prentice Hall, London, 1985.

[31] H. Katsuno, A. Mendelzon, On the difference between updating a knowledge base and revising it, in: Proc. 2nd International Conference on the Principles of Knowledge Representation and Reasoning (KR-91), Cambridge, MA, 1991, pp. 387–394.

[32] D. Kozen, Results on the propositional mu-calculus, Theoret. Comput. Sci. 27 (1983) 333–355.

[33] D. Kozen, R. Parikh, A decision procedure for the propositional mu-calculus, in: Proc. 2nd Workshop on Logic of Programs, Lecture Notes in Computer Science, Vol. 164, Springer, Berlin, 1983, pp. 313–325.

[34] D. Kozen, J. Tiuryn, Logics of programs, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Elsevier, Amsterdam, 1990, pp. 790–840.

[35] K.G. Larsen, L. XinXin, Compositionality through an operational semantics of contexts, J. Logic Comput. 1 (6) (1991) 761–795.

[36] K.J. Larsen, Proof systems for satisfiability in Hennessy–Milner logic with recursion, Theoret. Comput. Sci. 72 (1990) 265–288.

[37] H. Levesque, R. Reiter, Y. Lesperance, F. Lin, R. Scherl, GOLOG: A logic programming language for dynamic domains, J. Logic Programming 31 (1997) 59–84.

[38] V. Lifschitz, Frames in the space of situations, Artificial Intelligence 46 (1990) 365–376.

[39] V. Lifshitz, G. Karta, Actions with indirect effects (preliminary report), in: Proc. 4th International Conference on the Principles of Knowledge Representation and Reasoning (KR-94), Bonn, Germany, 1994, pp. 341–350.

[40] F. Lin, R. Reiter, State constraints revisited, J. Logic Comput. (Special Issue on Action and Processes) 4 (5) (1994) 655–678.

[41] F. Lin, Y. Shoham, Provably correct theories of action (preliminary report), in: Proc. 9th National Conference on Artificial Intelligence (AAAI-91), Anaheim, CA, 1991, pp. 349–354.

[42] F. Lin, Y. Shoham, Concurrent actions in the situation calculus, in: Proc. 10th National Conference on Artificial Intelligence (AAAI-92), San Jose, CA, 1992, pp. 590–595.

[43] Z. Manna, A. Pnueli, The anchored version of the temporal framework, in: J. de Bakker, P. de Roever, G. Rozenberg (Eds.), Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, Lecture Notes in Computer Science, Vol. 354, Springer, Berlin, 1989, pp. 201–284.

[44] K.L. McMillan, Symbolic Model Checking, Kluwer, Dordrecht, 1993.

[45] R. Milner, Communication and Concurrency, Prentice Hall, London, 1989.

[46] Y.N. Moschovakis, Elementary Induction on Abstract Structures, North-Holland, Amsterdam, 1974.

[47] D. Park, Fixpoint induction and proofs of program properties, in: B. Meltzer and D. Michie (Eds.), Machine Intelligence, Vol. 5, Edinburgh University Press, 1970, pp. 59–78.

[48] J.A. Pinto, Temporal reasoning in situation calculus, Ph.D. Thesis, Department of Computer Science, University of Toronto, 1994.

[49] R. Reiter, The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression, in: V. Lifschitz (Ed.), Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy, Academic Press, New York, 1991, pp. 359–380.

[50] R. Reiter, Natural actions, concurrency and continuous time in the situation calculus, in: Proc. 5th International Conference on Principles of Knowledge Representation and Reasoning (KR'96), 1996, pp. 2–13.
[51] S. Rosenschein, Plan synthesis: A logical approach, in: Proc. 8th International Joint Conference on Artificial Intelligence (IJCAI-81), Vancouver, BC, 1981.
[52] B. Steffen, Characteristic formulae, Proc. ICALP, Lecture Notes in Computer Science, Vol. 372, Springer, Berlin, 1989, pp. 723–732.
[53] B. Steffen, A. Ingolfsdottir, Characteristic formulae for processes with divergence, Inform. and Comput. 110 (1) (1994) 149–163.
[54] C. Stirling, Modal and temporal logic, in: S. Abramsky, D.M. Gabbay, T.S.E. Maibaum (Eds.), Handbook of Logic in Computer Science, Clarendon Press, Oxford, 1992, pp. 477–563.
[55] C. Stirling, Modal and temporal logics for processes, in: Logics for Concurrency: Structure versus Automata, Lecture Notes in Computer Science, Vol. 1043, Springer, Berlin, 1996, pp. 149–237.
[56] R.S. Streett, E.A. Emerson, The propositional mu-calculus is elementary, in: Proc. 6th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science, Vol. 172, Springer, Berlin, 1984, pp. 465–472.
[57] R.S. Streett, E.A. Emerson, An automata theoretic decision procedure for the propositional mu-calculus, Inform. and Control 81 (1989) 249–264.
[58] A. Tarski, A lattice-theoretical fixpoint theorem and its applications, Pacific J. Math. 5 (1955) 285–309.
[59] E. Ternovskaia, Interval situation calculus, in: Proc. ECAI'94 Workshop on Logic and Change, 1994, pp. 153–164.
[60] J. van Benthem, J. van Eijck, V. Stebletsova, Modal logic, transition systems and processes, J. Logic Comput. 4 (5) (1994) 811–855.
[61] R. van Glabbeek, The linear time—branching time spectrum, CONCUR'90, Lecture Notes in Computer Science, Vol. 458, Springer, Berlin, 1990, pp. 278–297.
[62] G. Winskel, A note on model checking the modal ν-calculus, in: Proc. 11th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science, Vol. 372, Springer, Berlin, 1989, pp. 761–772.