A Two-Player Asynchronous Game on Fully Observable Petri Nets

Federica Adobbati, Luca Bernardinello(✉), and Lucia Pomello

DISCo, Università degli Studi di Milano - Bicocca, Viale Sarca 336 U14, Milan, Italy
luca.bernardinello@unimib.it

Abstract. A Petri net is distributed if its elements can be assigned to a set of locations so that each element belongs to exactly one location, and each transition belongs to the same location as its input places.

We define an asynchronous game played on the unfolding of a distributed net with two locations, the 'user' and the 'environment'. The user can control the transitions in its location. A play in the game is a run in the unfolding, together with a sequence of cuts in that run. The rules of the game require that the environment satisfies a progress constraint: no transition in its location can be indefinitely postponed. In the general case, the game can be defined so that the user can observe only some transitions. In this paper, we only consider the case in which all transitions are observable, and study a reachability problem, in which the user tries to fire a target transition. We propose an algorithm which decides if the user has a winning strategy and, if so, computes a winning strategy.

1 Introduction

The ideas behind this paper were conceived while studying the problem of *weak observable liveness* [3,6], where we suppose that a Petri net models a system comprising a user and an environment; the user controls a subset of transitions, and observes a subset of transitions. The aim of the user is to force liveness of a special transition (the *target*), whatever the behaviour of the environment. The environment is supposed to guarantee progress of uncontrollable transitions.

The problem can be stated as deciding whether the user has a strategy allowing him to achieve his aim, irrespective of the choices of the environment. The strategy is formalized as a *response function*, mapping observations (sequences of observable transitions) to sets of controllable transitions.

In a first attempt to develop an algorithm for finding a strategy, the problem was translated into an infinite game on a finite graph, where the finite graph is derived from the marking graph of the net [3]. Besides the usual problem of state explosion, this approach hides the potential concurrency in the net, by using an interleaving semantics.

Hence, the authors started to explore the idea of defining an asynchronous game, to be played on the unfoldings of Petri nets, in which to encode the weak observable liveness problem, but also several other problems, formalized by defining a suitable aim for the user. Such a game was proposed in [4], where its application to weak observable liveness was studied.

Other possible applications of such a game could be in the general frame of verification, adaptation and control of distributed systems; so that, in the case of a winning strategy for the user with respect to a specific behavioral property, the system model could be adapted imposing that specific property, for example by adding an interacting component which implements the user behavior by synthesising the winning strategy; a reference for this sort of applications could be for example [12].

In this paper, we consider *distributed* net systems, in which all choices are local to one component, restricted to the case of exactly two components, user and environment, where the user has a sequential behaviour, whereas in the environment, transitions can be concurrent with each other, and with user's transitions.

We propose an asynchronous game played on the unfolding of the system in a general setting, so that by defining proper strategies, we can adapt the same model for the verification of different properties. Here we study a reachability problem, in which the user tries to fire a target transition.

In the general case, in which the user can observe the occurrence of only some environment's transitions, the definition of the game on unfolding may allow to define a winning strategy for the user, whereas this would be not possible by considering a game based on interleaving semantics. This fact has been briefly discussed on the basis of an example in [2] and is motivated by the fact that, in the unfolding, it is possible to distinguish different occurrences of the same transition, occurrences which can be differently related to other occurrences of another transition. In this way, the structure of the unfolding, even with partial observability, may allow to reconstruct the unobservable evolution of the system.

Obviously, the lack of information may even prevent to find a winning strategy; the chances of having a winning strategy for the user increase by observing as much as possible the behaviour of the environment; and if the user has no strategy by observing every transitions, there is no hope in the case of partial observability.

As a first step towards the identification of an algorithm in the general case of partial observability, in this paper we assume full observability, and propose an algorithm on the unfolding which decides if the user has a winning strategy and, if so, it computes such a winning strategy.

The paper, which is an extended, revised version of [1], is structured as follows. In the next section, we recall the needed notions about Petri nets, distributed Petri nets, and unfoldings, In Sect. 3 we define the general game, and the notions of strategy and winning strategy. The problem of controlled reachability is introduced in Sect. 4, together with the algorithm looking for a winning strategy. Several approaches to notions of asynchronous games are briefly discussed in Sect. 5, while prospects for future developments are presented in the final section.

2 Petri Nets

Petri nets model concurrent systems. The basic elements of a net are local states (places) and local transitions. The global state of a net is distributed among its local states. When a transition occurs, it changes the value of local states in its neighbourhood. Several types of nets have been defined and studied. Here, we use *1-safe* net systems.

Definition 1. *A net is a triple $N = (P, T, F)$, where P and T are disjoint sets. The elements of P are called* places *and represented by circles, the elements of T are called* transitions *and represented by squares. F is called* flow relation*, with $F \subseteq (P \times T) \cup (T \times P)$, and is represented by arcs.*

Let $x \in P \cup T$ be an element of the net; the pre-set of x is the set ${}^\bullet x = \{y \in P \cup T \mid (y, x) \in F\}$, the post-set of x is the set $x^\bullet = \{y \in P \cup T \mid (x, y) \in F\}$.

We assume that any transition has non-empty pre-set and post-set: $\forall t \in T$, ${}^\bullet t \neq \emptyset$ and $t^\bullet \neq \emptyset$.

A net is infinite if $P \cup T$ is infinite, finite otherwise.

Two transitions, t_1 and t_2, are *independent* if $({}^\bullet t_1 \cup t_1^\bullet)$ and $({}^\bullet t_2 \cup t_2^\bullet)$ are disjoint.

A net $N' = (P', T', F')$ is a *subnet* of $N = (P, T, F)$ if $P' \subseteq P$, $T', \subseteq T$, and F' is F restricted to the elements in N'.

Definition 2. *A* net system *is a quadruple $\Sigma = (P, T, F, m_0)$ consisting of a finite net $N = (P, T, F)$ and an* initial marking *$m_0 : P \to \mathbf{N}$.*

A transition t is *enabled* at a marking m, denoted $m[t\rangle$, if, for each p in ${}^\bullet t$, $m(p) > 0$. A transition t, enabled at m, can *occur* (or *fire*) producing a new marking

$$m'(p) = \begin{cases} m(p) + 1 & \text{if } p \in t^\bullet \setminus {}^\bullet t \\ m(p) - 1 & \text{if } p \in {}^\bullet t \setminus t^\bullet \\ m(p) & \text{otherwise} \end{cases}$$

A marking m' is reachable from another marking m, if there is a sequence $t_1 t_2 \ldots t_n$ such that $m[t_1\rangle m_1[t_2\rangle \ldots m_{n-1}[t_n\rangle m'$; in this case, we write $m' \in [m\rangle$. The set of reachable markings is the set of markings reachable from the initial marking m_0, denoted $[m_0\rangle$.

A net system is *1-safe* if, for each reachable marking m, and each place p, $m(p) \leq 1$. Markings in 1-safe net systems can, and will be, considered as subsets of places.

In a net system, two transitions, t_1 and t_2, are *concurrent* at a marking m if they are independent and both enabled at m.

The non sequential behaviour of 1-safe net systems can be recorded by occurrence nets, which are used to represent by a single object the set of potential histories of a net system. In the following, by F^* we denote the reflexive and transitive closure of F.

Two elements $x, y \in P \cup T$ are said to be in *conflict*, denoted $x \# y$, iff there exist $t_1, t_2 \in T : t_1 \neq t_2, t_1 F^* x, t_2 F^* y \wedge \exists p \in {}^\bullet t_1 \cap {}^\bullet t_2$.

Definition 3. *A net $N = (B, E, F)$ is an* occurrence net *if*

- *for all $b \in B$, $|{}^\bullet b| \leq 1$*
- *F^* is a partial order on $B \cup E$*
- *for all $x \in B \cup E$, the set $\{y \in B \cup E \mid yF^*x\}$ is finite*
- *for all $x \in B \cup E$, $x\#x$ does not hold*

We will say that two elements x and y, $x \neq y$, of N are *concurrent*, and write x **co** y, if they are not ordered by F^*, and they are not in conflict.

By $\min(N)$ we will denote the set of minimal elements with respect to the partial order induced by F^*.

A *B-cut* of N is a maximal set of pairwise concurrent elements of B. B-cuts represent potential global states through which a process can go in a history of the system. By analogy with net systems, we will sometimes say that an event e of an occurrence net is *enabled* at a B-cut γ, denoted $\gamma[e\rangle$, if ${}^\bullet e \subseteq \gamma$. We will denote by $\gamma + e$ the B-cut $(\gamma \backslash {}^\bullet e) \cup e^\bullet$. A B-cut is a *deadlock cut* if no event is enabled at it.

Let Γ be the set of B-cuts of N. A partial order on Γ can be defined as follows: let γ_1, γ_2 be two B-cuts. We say $\gamma_1 < \gamma_2$ iff

1. $\forall y \in \gamma_2 \exists x \in \gamma_1 : xF^*y$
2. $\forall x \in \gamma_1 \exists y \in \gamma_2 : xF^*y$
3. $\exists x \in \gamma_1, \exists y \in \gamma_2 : xF^+y$

In words, $\gamma_1 < \gamma_2$ if any condition in the second B-cut is or follows a condition of the first B-cut and any condition in the first B-cut is or comes before a condition of the second B-cut (and there exists at least one condition coming before).

A sequence of B-cuts, $\gamma_0\gamma_1 \ldots \gamma_i \ldots$ is *increasing* if $\gamma_i < \gamma_{i+1}$ for all $i \geq 0$.

We will say that an event $e \in E$ precedes a B-cut γ, and write $e < \gamma$, iff there is $y \in \gamma$ such that eF^+y. In this case, each element of γ either follows e or is concurrent with e in the partial order induced by the occurrence net.

Definition 4. *A* branching process *of $\Sigma = (P, T, F, m_0)$ is an occurrence net $N = (B, E, F)$, together with a labelling function $\mu : B \cup E \rightarrow P \cup T$, such that*

- *$\mu(B) \subseteq P$ and $\mu(E) \subseteq T$*
- *for all $e \in E$, the restriction of μ to ${}^\bullet e$ is a bijection between ${}^\bullet e$ and ${}^\bullet \mu(e)$; the same holds for $e^\bullet$*
- *the restriction of μ to* $\min(N)$ *is a bijection between* $\min(N)$ *and m_0*
- *for all $e_1, e_2 \in E$, if ${}^\bullet e_1 = {}^\bullet e_2$ and $\mu(e_1) = \mu(e_2)$, then $e_1 = e_2$*

A run *of Σ is a branching process (N, μ) such that the conflict relation of the underlying occurrence net is empty.*

For γ a B-cut of N, the set $\{\mu(b) \mid b \in \gamma\}$ is a reachable marking of Σ, and we refer to it as the marking corresponding to γ.

Let (N_1, μ_1) and (N_2, μ_2) be two branching processes of Σ. We say that (N_1, μ_1) is a *prefix* of (N_2, μ_2) if N_1 is a subnet of N_2, and

– $\min(N_1) = \min(N_2)$
– if $b \in B_1$ and $(e, b) \in F_2$, then $e \in E_1$
– if $e \in E_1$, and b is either a precondition or a postcondition of e in N_2, then $b \in B_1$

For any 1-safe net system Σ, there exists a unique, up to isomorphism, maximal branching process of Σ. We will call it the *unfolding* of Σ, and denote it by $\textsc{unf}(\Sigma)$ (see [7]).

A *run* of Σ describes a particular history of Σ, in which conflicts have been solved. Any run of Σ is a prefix of the unfolding $\textsc{unf}(\Sigma)$; we will also say that it is a run on $\textsc{unf}(\Sigma)$.

In this paper we are interested in Petri nets modelling systems in which a *User* controls a subset of transitions, while interacting with an *Environment*. Intuitively, this means that the User can decide whether to fire such a transition when it is enabled.

We also assume that choices among transitions are local either to the Environment or to the User, and that transitions controlled by the User are never concurrent with each other, while they can be concurrent with transitions in the Environment.

As a formal setting, we refer to the so-called *distributed net systems*, as introduced and studied in [5] and in [10].

Definition 5. *A* distributed net system *over a set L of locations is a 1-safe net system $\Sigma = (P, T, F, m_0)$ together with a map*

$$\lambda : (P \cup T) \to L$$

such that for every $p \in P$, $t \in T$, if $p \in {}^\bullet t$, then $\lambda(p) = \lambda(t)$.

In this paper, we consider the special case of distributed net systems $\langle \Sigma, \lambda \rangle$ such that $L = \{\text{Environment}, \text{User}\}$, i.e. of distributed net systems with only two components, representing the Environment and the User, respectively; we assume that the User controls all transitions in its location, and these transitions are never concurrent with each other. From now on, by distributed net system we will mean a net system satisfying these constraints.

In distributed net systems, when a transition is enabled, it can never be disabled by the occurrence of transitions belonging to different components. In the case of a cycle this observation justifies the following lemma.

Lemma 1. *Let $\langle \Sigma, \lambda \rangle$ be a distributed net system with two locations, A and G. Let m be a marking, and*

$$m_1[t_1\rangle m_2[t_2\rangle m_3[...\rangle m_1$$

be a firing sequence with $\lambda(t_i) = A$ for each i. Then, if $\lambda(t) = G$, and t is enabled at m_i for some i between the two repetitions of m_1, then t is enabled at m_j for each m_j in the cycle.

The notions of unfolding and run apply in the obvious ways to distributed net systems. We will use E_c to denote the set of controllable events in the unfolding (occurrences of controllable transitions, performed by the *User*), and $E_{nc} = E \setminus E_c$ to denote uncontrollable events. Uncontrollable transitions are meant to represent actions performed by the *Environment*.

In the graphical representation, controllable transitions and events will be represented by black squares.

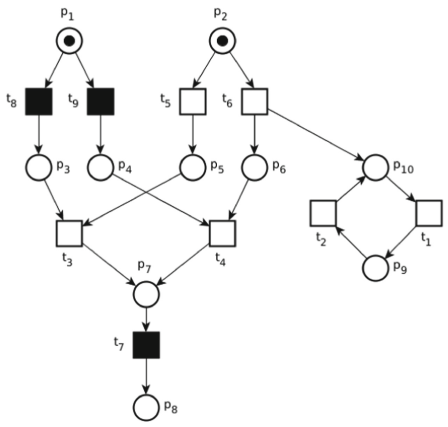

Fig. 1. A distributed net system with two locations

Example 1. Figure 1 shows a distributed net system with two locations. Places are not explicitly divided into the two components, because their partition can be inferred by their post-transitions. A prefix of the unfolding of the system is shown in Fig. 2. Each element of the unfolding is decorated with the label of an element in the net, with an exponent which distinguishes different occurrences of the same element. The dotted line suggests that the unfolding goes on by repeating occurrences of transitions t_1 and t_2, and of their neighbouring places.

3 An Asynchronous Game on the Unfolding

Let Σ be a distributed net system with two locations, Environment and User. We assume that the Environment is subject to a progress (or weak fairness) property: if an uncontrollable transition is enabled, then it will eventually either fire or become disabled.

We define a game on $\textsc{unf}(\Sigma)$. A play in the game is a run, weakly fair with respect to uncontrollable transitions, together with an increasing sequence of B-cuts, which can be seen as a potential record of the play as observed by an external entity. Several transitions can occur between two contiguous cuts in the sequence.

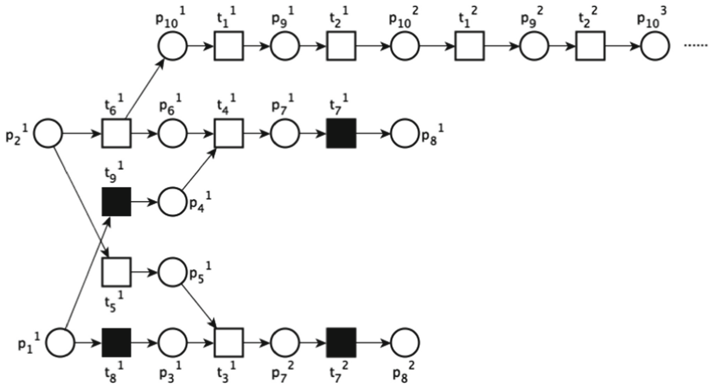

Fig. 2. The unfolding of the distributed net system in Fig. 1

Definition 6. *Let $\rho = (B_\rho, E_\rho, F_\rho, \mu_\rho)$ be a run on* UNF(Σ) *and* $\pi = \gamma_0, \gamma_1, \cdots, \gamma_i, \cdots$ *an increasing sequence of B-cuts. The pair (ρ, π) is said to be a* play *if:*

- *$\forall e \in E_{nc} \backslash E_\rho$, the net obtained by adding e and its postconditions to ρ is not a run of* UNF(Σ)*;*
- *$\forall e \in E_\rho$ there is a B-cut $\gamma_i \in \pi$ such that $e < \gamma_i$.*

In general, the winning condition for the User is defined by a set of plays. The significant cases to analyse are those in which the winning set is determined by a property that we are interested in investigating on the model.

For example, let us suppose that we are interested in knowing if a user is able to force the occurrence of a target transition once. We can model this problem as a game in which the User wins a play if the corresponding run contains an occurrence of the target transition.

Another possible goal of a play, as analysed for example in [9], is to verify if it is always possible to avoid a certain marking in a controllable system. In this case the User wins those plays in which there are no cuts associated with that marking. Whatever the goal of the game is, a strategy is a function formalizing the behaviour of the User during a play.

In general, one might suppose that the User cannot observe everything in the system. For instance, it might not directly observe firings of some transitions in the Environment. In this paper, we suppose that the User can see all occurrences of transitions. This implies that the User can determine the current cut in the unfolding on the basis of the transition occurrences observed so far; hence, a strategy can be defined as a map from B-cuts to sets of controllable events.

Definition 7. *Let Γ be the set of B-cuts in* UNF(Σ)*. A* strategy *is a function $\alpha : \Gamma \to 2^{E_c}$ such that for every $\gamma \in \Gamma$ and for every $e \in E_c$, if $e \in \alpha(\gamma)$, then e is enabled in γ.*

Definition 8. *Let (ρ, π) be a play. An event $e \in E_c$ is* finally postponed *in (ρ, π) iff there is a cut $\gamma_i \in \pi$ in which e is enabled and such that $\forall k \geq i$, $\gamma_k[e\rangle$.*

Definition 9. *Let (ρ, π) be a play and α be a strategy. An event $e \in E_c$ is* finally eligible *in (ρ, π) by α iff there is a cut $\gamma_i \in \pi$ such that $e \in \alpha(\gamma_i)$ and $\forall k \geq i$, $e \in \alpha(\gamma_k)$.*

A play complies with a strategy if all controllable events in the play have been chosen according to the strategy, and no controllable event is finally postponed and eligible.

Definition 10. *Let $\rho = (B_\rho, E_\rho, F_\rho, \mu_\rho)$ be a run in $\text{UNF}(\Sigma)$, $\pi = \gamma_1\gamma_2, ...$ be an increasing infinite sequence of B-cuts and α be a strategy. The pair (ρ, π) is an* $\alpha-$play *iff:*

1. *(ρ, π) is a play;*
2. *For every controllable event e belonging to E_ρ, there must be a B-cut $\gamma_i \in \pi$ such that $e \in \alpha(\gamma_i)$ and $\gamma_{i+1} = (\gamma_i \backslash {}^\bullet e) \cup e^\bullet$.*
3. *If $|E_\rho \cap E_c| < \infty$, there is no event $e \in E_c \cap E_\rho$ finally eligible by α and finally postponed in the play.*

A strategy $\alpha : \Gamma \rightarrow 2^{E_c}$ is winning iff the User wins all the α-plays. In general, if there is a winning strategy, it is not unique.

Example 2. The net system shown in Fig. 1 is distributed, with two locations. Define a game on its unfolding, shown in Fig. 2, so that the User wins a play if the play contains an occurrence of t_7.

By inspecting the net, it is clear that a winning strategy for the User consists in waiting for the Environment to choose between t_5 and t_6, and then fire, respectively, either t_8 or t_9. Since the Environment cannot postpone its choice forever, and will be forced to eventually fire either t_3 or t_4, the User will be able to fire t_7, and win the game. Formally, the winning strategy can be defined as follows: $\alpha(\{p_1^1, p_6^1, p\}) = \{t_9^1\}$, where p is any occurrence of either p_9 or p_{10}, $\alpha(\{p_1^1, p_5^1\}) = \{t_8^1\}$, $\alpha(\{p_7^2\}) = \{t_7^2\}$, $\alpha(\{p_7^1, p\}) = \{t_7^1\}$, where p is any occurrence of either p_9 or p_{10}, $\alpha(\gamma) = \emptyset$ for any other B-cut γ. In particular, $\alpha(\{p_1^1, p_2^1\}) = \emptyset$, to encode the decision to wait, in the initial cut, for the Environment to choose its first move. Figure 3 shows an α-play.

4 Controlled Reachability

In this section we apply the general idea of asynchronous game to a specific reachability problem, and propose an algorithm to determine if the User has a winning strategy.

Let $\langle \Sigma, \lambda \rangle$, where $\Sigma = (P, T, F, m_0)$, be a distributed net system. The problem of *controlled reachability* consists in determining if the User is able to lead the system to fire a certain transition once, despite the Environment behaviour, starting from the initial marking. This can be analysed through a game on the unfolding. Let t be the target transition; we define as winning condition for the User the set of plays (ρ, π) in which there is an event $e \in E_\rho$ labelled with t.

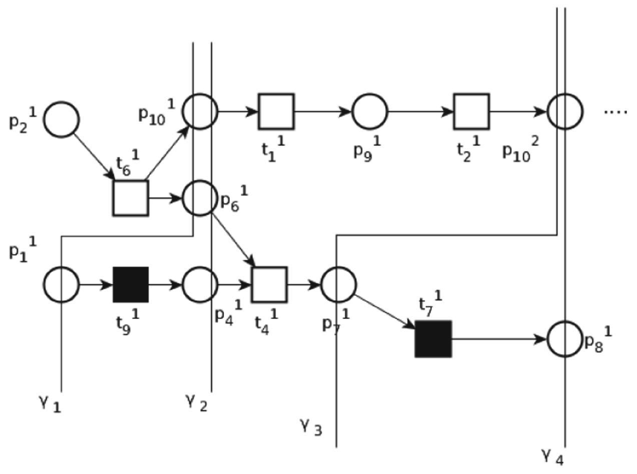

Fig. 3. An α-play on the unfolding in Fig. 2

A target transition t is controllably reachable in Σ if, and only if, there is a strategy α on $\text{UNF}(\Sigma)$ such that the User wins every α-play. Example 2 above can be seen as a game of controlled reachability. The strategy discussed in the example is a winning strategy for this game.

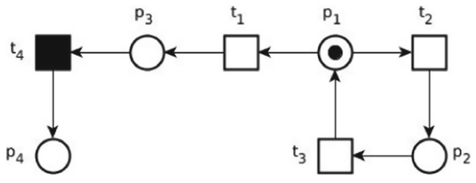

Fig. 4. A distributed net system

Example 3. The net shown in Fig. 4 is distributed, with two locations. Consider the game of controlled reachability played on its unfolding, shown in Fig. 5, where the target transition is t_4. If the Environment cooperates with the User by eventually choosing t_1, then the target is reached. However, the Environment can choose t_2 at every cut consisting in an occurrence of p_1. The Environment is subject to a weak fairness constraint, but not to a strong fairness constraint. Hence, irrespective of the strategy chosen by the User, an infinite play made of repeated occurrences of the cycle p_1, t_2, p_2, t_3, p_1 is admissible.

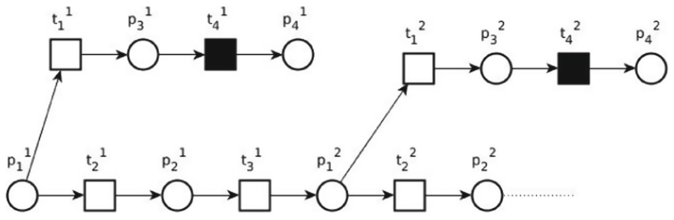

Fig. 5. The (prefix of the) unfolding of the net system shown in Fig. 4

In a general case, given a strategy α, there are infinitely many $\alpha-$plays in UNF(Σ), and some plays could be infinite, hence the exhaustive exploration of them would take infinite time. We propose an algorithm that, given a distributed net system and a target transition, establishes if there is a winning strategy for the controlled reachability of the target and, if so, computes a winning strategy.

4.1 Algorithm for a Winning Strategy

In this section we present the algorithm looking for a winning strategy for the reachability of a target transition on fully observable systems, and we illustrate it on the system in Fig. 1, already discussed in Example 2. The algorithm we present generates a prefix of the unfolding of a given net system, deciding whether there exists a winning strategy for the User. In the positive case, it gives as output a strategy as a function on reachable markings; the strategy is initially associated to B-cuts of the unfolding, but the algorithm works so that, for distinct cuts corresponding to the same marking, the strategy gives the same answer.

The input data are the following:

- A net in which the transitions are enumerated so that all the uncontrollable transitions precede all the controllable ones. If the target is a controllable transition, it must be the first of the controllable transitions.
- The position of the first controllable transition.
- The initial marking m_0 of the system.
- The target transition.

The value of these variables is available for all the functions of the algorithm and does not change during its execution.

The core of the algorithm is the recursive function UNF_EXPLORATION (see Algorithm 1), which unfolds the net by exploring reachable cuts, and constructs at the same time a prefix of the unfolding and a strategy.

The function takes five input arguments:

1. γ: the cut that must be analysed;
2. M: the list of markings associated to the cuts already analysed in the current run;
3. E_l: the list of events that fired in the current run;

4. e: the last event added to the current run, leading to γ;
5. sz: the set of events enabled in γ that are part of a cycle or that are in conflict with events that are in a cycle.

It returns a Boolean variable, that is equal to True if there is a winning strategy, for all the plays passing from the input cut γ consistent with the strategy, False otherwise. In addition, it possibly modifies the prefix and the strategy, initially empty, filling them with events, cuts and choices already explored.

The first time that the function is called, the input consists always in the initial cut γ_0 in the unfolding, empty lists for the list of visited markings, the list of analysed events and the list of events that are part of cycles or in conflict with them (those events will be discovered during the execution of the algorithm), a fictitious event i. The function UNF_EXPLORATION uses some auxiliary functions:

– ENAB_N is a function that has an input cut and returns the list of uncontrollable events which are enabled in that specific cut;
– similarly, ENAB_C returns the controllable enabled events.
– EXTRACT returns the first element of an input list, and the list deprived of this element.
– STABLE_ZONE returns the set of events that can be part of a cycle, and those in conflict with them.

Let us recall that we denote with $\gamma + e$ the cut obtained by firing the event e in the cut γ.

The function constructs every run by adding uncontrollable events until one of the following cases occurs: (1) the target occurs; (2) a deadlock cut is reached or a cut is reached in which only transitions that are part of a cycle or that are in conflict with events in a cycle are enabled; (3) a cut that has been previously analysed is reached (two subcases are considered); (4) a cut is reached in which no uncontrollable event is enabled, and some controllable events are enabled; (5) a cut is reached corresponding to a marking which has already been visited in the current run, and there are not uncontrollable enabled events that are concurrent with all the ones that occurred between the two equivalent markings.

In case (1), the current run corresponds to a play won by the User; hence the function tries to backtrack along choices among uncontrollable events, if possible. Symmetrically, in case (2), the current run corresponds to a play won by the Environment; hence, the function tries to backtrack along choices among controllable events, if possible. In cases (3), the current run is the prefix of a set of runs that have been already analysed. The User wins or loses according to the analysis previously done. In case (4), a controllable event is added, and the exploration restarts from the new cut. Finally, in case (5), if possible, a controllable event is added, and the exploration restarts from the new cut; if this is not possible, the run corresponds to a play won by the Environment and the function tries to backtrack and change the previous controllable choices.

Example 4. Consider the net system shown in Fig. 1, and its unfolding (Fig. 2), where the ordering on the set of transitions is given by their indices. Starting from

Algorithm 1. Unfolding exploration

```
function UNF_EXPLORATION($\gamma$, $M$, $E_l$, $e$, $sz$)
    if $e == target$ then return true
    else if $\gamma$ is a deadlock or enables only transitions in $sz$ then return false
    else if $\gamma \in \Gamma_{bad}$ then return false
    else if $\gamma \in \Gamma_{good}$ then return true
    else if $\mu(\gamma) \in M$ then return EXPLORE_CUT_C($\gamma, M, E_l$)
    else if ENAB_N($\gamma$) $\neq \emptyset$ then
        $E$ =ENAB_N($\gamma$)
        repeat
            $e_0, E$ = EXTRACT($E$)
            $v$ = UNF_EXPLORATION($\gamma + e_0, M.append(\mu(\gamma)), E.append(e), e_0$)
            if $v ==$ true then
                unf = unf $\cup[\gamma, e_0, \gamma + e_0]$
            end if
        until $E == \emptyset \vee v ==$ false
        if $v == true$ then
            if $\gamma \in ver$ then
                $sz$ =STABLE_ZONE($E$)
                v = UNF_EXPLORATION($\gamma, M, E_l, e$)
            else
                $\Gamma_{good}.append(\gamma)$
            end if
        else
            $\Gamma_{bad}.append(\gamma)$
        end if
        return $v$
    else
        $E$ =ENAB_C($\gamma$)
        repeat
            $e_0, E$ = EXTRACT($E$)
            v = UNF_EXPLORATION($\gamma + e_0, M.append(\mu(\gamma)), E_l.append(e), e_0$)
            if $v ==$ true then
                unf = unf $\cup[\gamma, e_0, \gamma + e_0]$
                str = str $\cup$ $[\gamma, e_0]$
            end if
        until $E == \emptyset \vee v ==$ true
        if $v == true$ then
            if $\gamma \in ver$ then
                $sz$ =STABLE_ZONE($E$)
                v = UNF_EXPLORATION($\gamma, M, E_l, e$)
            else
                $\Gamma_{good}.append(\gamma)$
            end if
        else
            $\Gamma_{bad}.append(\gamma)$
        end if
        return $v$
    end if
end function
```

the initial B-cut, the algorithm adds the event t_5^1, reaching a cut in which only controllable transitions are enabled. It then adds t_8^1, reaching the cut $\{p_3^1, p_5^1\}$, and starts again adding uncontrollable transitions. This run will lead to the target event t_7^2, hence it is not necessary to backtrack on controllable events.

The next backtracking step goes back to the initial cut, and starts exploring a new run by adding t_6^1; from $\{p_1^1, p_6^1, p_{10}^1\}$, the events t_1^1 and t_2^1 fire. This produces the cut $\{p_1^1, p_6^1, p_{10}^2\}$, that corresponds to a marking that has already been visited. Hence, the controllable event t_8^1 is added, leading to a cut in which only the cycle formed by occurrences of t_1 and t_2 can occur, thus repeating the same marking. The algorithm backtracks and tries t_9^1. The events t_1^2 and t_4^1 are enabled in $\{p_4^1, p_6^1, p_{10}^2\}$. Due to the order of the transitions of the net, t_1^2 and t_2^2 occur, reproducing the same marking. In order to guarantee the progress of the system, the algorithm adds only events that have been enabled since the first repetition of the marking associated with the current cut and have never been disabled from that moment on.

In Example 4, the only event that satisfies these requirements is t_4^1. By proceeding in this way, the algorithm continues until the target is reached.

In the following, we explain in detail how UNF_EXPLORATION works in a general step of execution of the algorithm. If γ is a cut of a play on the unfolding, one of these situations is verified:

1. γ is not a deadlock, it enables events that are not part of cycles or in conflict with them, has not been previously analysed, it is the first time that the associated marking is visited in the play, the target has not occurred yet and there are k uncontrollable enabled transitions to analyse in $\mu(\gamma)$. In this case, the prefix of the play currently ending with γ is extended in k different plays, each of them obtained by adding a different uncontrollable event after γ. The output for this step is True only if the values returned by all recursive calls on the cuts that immediately follow γ is True.
   Considering the system in Fig. 1 and its unfolding (Fig. 2), we find the described situation in the initial cut of the unfolding: in $\{p_1^1, p_2^1\}$, both the events t_6^1 and t_5^1 are enabled. Therefore, the algorithm extends the current prefix considering the two plays obtained by adding the two events and the cuts that follow their occurrence.
2. γ is not a deadlock, $\mu(\gamma)$ has never been analysed in the play, the target did not fire in the previous part of the play and the only enabled events that are not part of cycles or in conflict with them are controllable. In this case, the algorithm analyses the controllable events in the order induced by enumeration of the transitions in the net, until it finds an extension that returns True as output or it ends the analysis of all the controllable events enabled in γ.
   Referring to Fig. 2, the cut $\{p_1^1, p_5^1\}$ enables t_8^1 and t_9^1. The algorithm starts constructing the play with t_8^1. After verifying that the User wins all the α-plays passing from the cut $\{p_3^1, p_5^1\}$, the function does not continue with the analysis of t_9^1, and returns the Boolean value True.

3. Either γ is a deadlock, or all the enabled events are part of a cycle or in conflict with events in a cycle, or γ follows the target transition. These are base cases for the recursive algorithm. Their occurrence stops the exploration for that play. If the target fired, the algorithm returns True, in all the other cases of this situation, it returns False.
   In the considered example, all the plays ending with a cut in which there is an occurrence of p_8 are winning for the User (because an occurrence of t_7 has necessarily fired).
4. γ has already been considered in a previous step. In this case, the analysis stops and the function returns True, if the first analysis of the cut returned True, and False otherwise. This case is verified in case of concurrency in the Environment component.
5. $\mu(\gamma)$ was already visited in the play. In this case, the algorithm checks if any controllable event fired between the two repetitions. If this happens it returns False. (This is justified by the fact that the victory of the user cannot depend on the choice of a controllable transition that contributes to a cycle without the target.) Otherwise, it analyses only the events that are enabled and concurrent with all the ones fired in the cycle. If there are uncontrollable events among them, then it behaves as in 1; if there are only controllable events, it behaves as in 2; if there is no event satisfying the requirements, it behaves like in a deadlock situation.
   During the execution of the algorithm on the system in Fig. 1, the cut $\{p_3^1, p_6^1, p_{10}^3\}$ is analysed. The only enabled event is t_1^3, but it is not added to the play, because it depends on the repeated occurrences of transitions t_1 and t_2, that create a cycle in the system. Hence, the algorithm returns False for this particular play. Later, changing the controllable choice, it analyses the cut $\{p_4^1, p_6^1, p_{10}^3\}$. In this cut, t_4^1 is enabled and concurrent with all the occurrences of t_1 and t_2, hence, the algorithm extends the play with it.

The functions EXPLORE_CUT_C (Algorithm 2) and F (Algorithm 3) deal with concurrency. Specifically, EXPLORE_CUT_C is called by UNF_EXPLORATION when a cut associated with a marking repeated in the run is detected. The function F is called by EXPLORE_CUT_C; it takes the current cut, the list of the previously visited markings, and the list of events that have been fired. It checks whether a controllable event fired in the cycle; if not, it returns the list E of events concurrent with all the events occurred after the first cut in the run associated to the same marking as the current one. The events in E are the only ones considered by EXPLORE_CUT_C to extend the prefix of the run.

In Algorithm 2, there are two more auxiliary functions:

- E_{NC} takes a list of events as input, and returns only the uncontrollable ones.
- Symmetrically, E_C takes a list as input, and returns the controllable events in it.

Both UNF_EXPLORATION and EXPLORE_CUT_C are responsible for the construction of the prefix and the strategy. The prefix is updated every time that UNF_EXPLORATION returns the value True (with the exception of the very first call). When this happens, the receiving function appends to the prefix a triple consisting of its input cut γ, the following cut $\gamma + e$ that was in input to the call to the function that just returned True, and the event e. If the added event e is controllable, then the strategy is also updated. In particular, the algorithm appends the input cut γ coupled with the controllable transition $\mu(e)$ to the current strategy.

At the end of the execution of UNF_EXPLORATION, if there is a winning strategy, it is defined on the cuts of the prefix. To complete it, we have to define it on the markings, detect the parts of the plays corresponding to a cyclic behaviour on the system and, if the strategy chooses a transition immediately after them, the algorithm has to fill the strategy, attributing the same choice to all the markings in the cycle.

4.2 Discussion

In this section, we discuss the correctness of the proposed algorithm.

Lemma 2. *Every play exploration ends due to one of the following ending criteria:*

1. *The target fires. In this case the User wins all the $\alpha-$plays with the constructed prefix.*
2. *The play reaches a deadlock cut γ before reaching the target. In this case the User loses the play.*
3. *The play reaches a cut in which the target has not fired, and the only enabled transitions can be part of cycles or in conflict with transitions that can be part of a cycle. In this case the user loses the play.*
4. *The play reaches a cut γ that was previously analysed.*
5. *The play reaches a cut γ' such that there is another cut $\gamma : \gamma < \gamma'$ for which $\mu(\gamma) = \mu(\gamma')$, γ corresponds to the first occurrence of $\mu(\gamma)$ in the play, and*
   - *either γ' does not enable any event that is concurrent with all the events occurred between γ and γ',*
   - *or there is a controllable event e such that $\gamma < e < \gamma'$.*

   If the prefix is consistent with the strategy, the User loses at least an $\alpha-$play.

Moreover, if α is a strategy defined on the markings, then, for every prefix of an $\alpha-$play determined with one of these criteria, we can decide if the User wins all the $\alpha-$plays starting with such a prefix.

Proof. 1. If the target fired in the prefix, then every play with such a prefix is winning for the User, because it includes the target.
2. If the target does not fire and the play is in a deadlock, the prefix coincides with the whole play. Since it does not have the target, it is losing for the User.
3. If the target does not fire and the only enabled transitions can be part of a cycle or in conflict with transitions in cycles, then the user cannot prevent the environment to remain in the cycles forever (the transitions in a cycle are uncontrollable by construction). Since the target is not part of this cycle, the user cannot be sure to reach it.

Algorithm 2. Cuts associated with markings already visited in the prefix

Input: the cut γ that must be analysed, the ordered list M and E of the markings and events that occurred in the run before γ.

```
function EXPLORE_CUT_C($\gamma, M, E_l$)
    $E, E_{l_{reap}} = \mathrm{F}(\gamma, M, E_l)$
    if $E = \emptyset \vee E_C(E_{l_{reap}}) \neq \emptyset$ then return false
    else
        $E = \mathrm{F}(\gamma, M)$
        $E_{nc} = E_{NC}(E)$
        $E_c = E_C(E)$
        if $E_{nc} \neq \emptyset$ then
            $v$ = true
            repeat
                $e_0, E_{nc}$ = EXTRACT($E_{nc}$)
                $v$ = UNF_EXPLORATION($\gamma + e_0, M, e_0$)
                if $v ==$ true then
                    unf = unf $\cup[\gamma, e_0, \gamma + e_0]$
                end if
            until $E_{nc} == \emptyset \vee v ==$ false
            if $v == true$ then
                $\mathit{\Gamma}_{good}.append(\gamma)$
            else
                $\mathit{\Gamma}_{bad}.append(\gamma)$
            end if
            $ver.append(\mu(\gamma), E_{l_{reap}})$
            return $v$
        else
            $v$ = false
            repeat
                $e_0, E_c$ = EXTRACT($E_c$)
                $v$ = UNF_EXPLORATION($\gamma + e_0, M, e_0$)
                if $v ==$ true then
                    unf = unf $\cup[\gamma, e_0, \gamma + e_0]$
                    str = str $\cup$ $[\gamma, e_0]$
                end if
            until $E_c == \emptyset \vee v ==$ true
            if $v == true$ then
                $\mathit{\Gamma}_{good}.append(\gamma)$
            else
                $\mathit{\Gamma}_{bad}.append(\gamma)$
            end if
            $ver.append(\mu(\gamma), E_{l_{reap}})$
            return $v$
        end if
    end if
end function
```

Algorithm 3. Events that are concurrent with the ones that already fired in the run

Input: the cut γ that must be analysed and the ordered lists M, E_l of the markings and the events that occurred in the run before γ.
Output: list of events that have been enabled from the cut associated with the first occurrence of the marking $\mu(\gamma)$ to the current cut γ, list of events that occurred in the run between the two repetitions on $\mu(\gamma)$.

```
function F($\gamma, M, E_l$)
    i = 0
    while $M[i] \neq \mu(\gamma)$ do
        i = i+1
    end while
    $E_{l_{reap}} = E_l[i : len(E_l)]$
    $E = []$
    for all $e \in$ENAB_C($\gamma$) do
        if $\mu(e)$ enabled in $m$ $\forall m \in M[i : len(M)]$ then
            $E.append(e)$
        end if
    end for
return $E, E_{l_{reap}}$
end function
```

Algorithm 4. Full strategy

```
$v$ = UNF_EXPLORATION($\gamma_0$, [], i)
if v == True then
    str = CUTS_TO_MARKINGS()
    str = COMPLETE_STRATEGY()
end if
```

4. If two prefixes end with the same cut γ, it means that they differ only for the order in which the concurrent events occurred, and their possible elongations are the same. The winning condition for the User does not depend on the order in which events occurred, but only from the presence of the target in the run. Hence, if the algorithm is requested to analyse a cut for which it has already determined if α is winning, it can immediately stop and return the same answer.
5. First, we have to show that if the play does not reach the target, does not end with a deadlock, and does not reach a cut previously analysed, then this last criterion is verified. The number of reachable markings in the system is finite, hence after a number of steps equal at most to the number of reachable markings, the algorithm analyses a cut γ', such that $\mu(\gamma) = \mu(\gamma')$, where γ is a cut preceding γ' and belonging to the same play. Let us suppose that k events are enabled in γ' and concurrent with all the ones fired between γ and γ'. The algorithm adds one of these to the play and continues as before. If the play reaches a cut γ'' such that $\mu(\gamma) = \mu(\gamma'')$, then the events that the algorithm analyses are necessarily strictly less then k, because they should be concurrent

both with the events occurred between γ and γ' and with those fired between γ' and γ''. Since for every repetition, the number of events satisfying the requirements to be added decreases, after at most k cuts corresponding to the same marking $\mu(\gamma)$, the third criterion is satisfied. Notice that this does not depend on the specific cut: the same reasoning applies to all markings.
The next step is showing that if there is an $\alpha-$play with such a prefix, then there is at least an $\alpha-$play in which the User loses. We first consider the case in which there are not enabled events concurrent with all the ones in the cycle. If the prefix follows the strategy α, then the play repeating infinitely many times the behaviour of the prefix is an $\alpha-$play and the target never occurs. We cannot guarantee that the User will lose all the α-plays with such a prefix, but the fact that there is at least one is enough to state that α is not a winning strategy for the User. Secondly, we consider the case where a controllable transition fired between two occurrences of the same marking. By construction, the algorithm analyses controllable events only when all the significant uncontrollable events have been fired; hence, there cannot be any uncontrollable event that is concurrent with the cycle and that leads to the target, otherwise it would have been analysed before in the prefix. Again, if the prefix is consistent with the strategy, the play that repeats infinitely often the cycle is an $\alpha-$play and does not contain the target.

□

A consequence of Lemma 2 is the termination of the algorithm. We proved that every prefix constructed by the algorithm is finite. The number of considered $\alpha-$plays is also finite, because at every step there is only a finite number of enabled events to extend the prefix.

By construction, if the algorithm finds a winning strategy, all the runs in the prefix: (1) are consistent with the strategy, and (2) contain the target.

(1) All the plays in the list are consistent with the strategy. Every time that the algorithm analyses a cut γ and chooses to extend the prefix with a controllable event, it explores all the plays including γ and, one by one, each of the controllable enabled events. It stops when it finds a controllable enabled event such that, from the cut of the unfolding following this event, the User has a winning strategy. When this happens, the prefix is updated, adding the event and the cuts preceding and following it. Also the strategy is updated, choosing the associated controllable transition in γ. In this way, at every step, all the parts of runs in the prefix constructed until that moment are consistent with the strategy updated until that moment. If in γ there is no controllable enabled event such that, after it, the User has a winning strategy, then the part of the prefix already generated is not connected to the initial cut in the unfolding, since the event connecting this part to γ is not added to the prefix. At the same way, if there is a winning strategy, it cannot depend on the strategy calculated on the disconnected parts of the unfolding. If the algorithm finds a winning strategy and a disconnected part was found, since the algorithm chooses a controllable event in γ only when it is necessary to win, then there must be another cut in the prefix, that

precedes γ in the partial order, in which the algorithm adds a controllable transition that allows the User to avoid γ.
(2) All maximal runs in the prefix contain the target. If a run ends without the target, then the strategy allowing that run is not winning and must be changed. If it cannot be changed, then the algorithm will not state that there is a winning strategy, hence there must be a controllable node in which the decision previously taken can be changed. When another possible choice is analysed, all parts of runs depending on the previous one are deleted. Hence all the remained runs contain the target.

If the algorithm finds a winning strategy, every play in the unfolding consistent with this strategy is equivalent to an extension of a play in the prefix. This is shown in two steps.

1. Let us first consider the case without uncontrollable cyclic behaviours of the system.
   The strategy α constructed by the algorithm chooses a controllable transition only if there are not uncontrollable enabled ones. Let $\{t_1, ..., t_n\}$ be a set of uncontrollable transitions in a play, so that after their occurrence, there are not other uncontrollable enabled transitions. In whatever order the transitions are considered, the cut in the unfolding after their occurrence is the same, and the strategy will choose the same transition, because the following part of the unfolding is every time visited in the same way. Considering an $\alpha-$play, there must be a prefix of its run in the unfolding, because all the uncontrollable transitions are analysed in all the uncontrollable cuts of the prefix and the strategy chooses only a transition for every cut, hence the controllable choices must be the same of the ones considered in the prefix. This is enough to state that the play is won by the User, because in the common prefix of the run there is the target transition.
2. If there are uncontrollable cyclic behaviours, such that there is a concurrent enabled transition leading to the target, then there is more variety in the possible $\alpha-$plays, because the strategy is defined on markings in which uncontrollable events are enabled. Anyway, if an α-play has a prefix with the same events of one of the prefixes produced by the algorithm, then it is won by the User, regardless of the order of the cuts in the play. Some of the $\alpha-$plays have a longer uncontrollable part, because if an uncontrollable transition would be finally enabled, or a controllable transition would be finally enabled and eligible, there must be a certain point in which it will fire, but the precise point is unknown. However, since we complete every cycle at least once and from every cut that is not a repetition all the possible uncontrollable extensions are explored, and since the part of the unfolding starting from a given cut is isomorphic to the part of the unfolding starting from every cut corresponding to the same marking, the uncontrollable sequence of the $\alpha-$play can be divided in parts such that an isomorphic one has been considered by the algorithm.

Based on the previous observations, if the algorithm finds a winning strategy, the proposed strategy is winning in the unfolding.

Finally, we wish to show that if the algorithm states the existence of a winning strategy and proposes one on the cuts of the prefix, to complete it adding the same choice to all the markings that are part of a cycle is necessary and does not change the correctness.

- Let us suppose that a cycle with only uncontrollable transitions is in the net, and there is a controllable enabled transition that is concurrent with all the transitions in the cycle and which is necessary for the victory of the User. The strategy constructed together with the prefix adds the choice of the controllable transition only in the cut associated to the last repeated marking. This strategy is incomplete, because the chosen transition is not finally eligible, since every time that the system is in a marking of the cycle that is not the one that has been repeated in the prefix, the strategy does not choose it. To overcome this problem we fill the strategy by adding the choice of the controllable transition in every marking along the cycle.
- This preserves the correctness of the strategy. Actually, if γ is a cut that in a certain play is between γ_1 and γ_2 with $\gamma_1 < \gamma_2$ and $\mu(\gamma_1) = \mu(\gamma_2)$ and α' is the strategy computed by Algorithm 1 and translated on markings, then necessarily $\alpha'(\mu(\gamma)) = \alpha'(\mu(\gamma_1))$ or $\alpha'(\mu(\gamma)) = \emptyset$.
  Specifically, if $\alpha'(\mu(\gamma)) \neq \emptyset$, then it has to be $\alpha'(\mu(\gamma)) = \alpha'(\mu(\gamma_1))$. Let us suppose that $\{t_i\} = \alpha'(\mu(\gamma))$, $\{t_j\} = \alpha'(\mu(\gamma_2))$ and t_i precedes t_j in the enumeration defined by the input net. Then, t_i is not a winning choice in $\mu(\gamma_2)$, but there is a play that leads from γ to γ_2 and between these two cuts only uncontrollable events fire (because the controllable component is sequential). The algorithm updates the strategy in a cut only if, starting from that cut, the User is able to win every play. This cannot be the case, because the play can arrive in γ_2 and the User loses the play. Reasoning in the same way, it cannot be that t_i follows t_j in the enumeration. Hence it must be $\{t_i\} = \alpha'(\mu(\gamma)) = \alpha'(\mu(\gamma_1))$.
  If $\mu(\gamma)$ is never visited more than once in any run, then $\alpha'(\mu(\gamma)) = \emptyset$. We construct a final strategy α such that $\alpha = \alpha'$ for every marking m in which $\alpha'(m) \neq \emptyset$ and $\alpha(m') = \alpha'(m_1)$ for all m' such that there is a play in the unfolding with two cuts $\gamma_1, \gamma_2 : \mu(\gamma_1) = \mu(\gamma_2) = m_1$ and a cut $\gamma : \mu(\gamma) = m'$ and $\gamma_1 < \gamma < \gamma_2$.
  The marking m' could be reached in more than one run, and if it is part of two different uncontrollable cycles, with different repeated markings, there could be the doubt that $\alpha(m')$ is not well defined, but this is not possible. Let us suppose that there is another play in the unfolding with two cuts $\gamma_1', \gamma_2' : \mu(\gamma_1') = \mu(\gamma_2') = m_2 \neq m_1$ and a cut $\gamma' : \mu(\gamma') = \mu(\gamma) = m'$ and $\gamma_1' < \gamma' < \gamma_2'$. We have to show that if there is a winning strategy, then $\alpha'(\mu(\gamma_1)) = \alpha'(\mu(\gamma_1'))$. By contradiction, let us assume $\{t_i\} = \alpha'(\mu(\gamma_1))$, $\{t_j\} = \alpha'(\mu(\gamma_1'))$ and t_i precedes t_j in the enumeration (the opposite case is equivalent due to the symmetry of definitions). Then, t_i is not a winning choice for γ_1', otherwise it would have been chosen before analysing t_j. If t_i is not winning for γ_1', then it cannot be winning from γ_1, because, starting from γ_1 the play can arrive in γ firing only uncontrollable transitions, and from γ

there is a path made only by uncontrollable transitions to a cut γ_2'' such that $\mu(\gamma_2'') = \mu(\gamma_1')$. Since the unfolding starting from γ_2'' is isomorphic to the one starting from γ_1', if the strategy is not winning from γ_1' it cannot be winning from γ_2'' and therefore from γ_1.

4.3 Experiments

This work is mainly theoretical, and a full experimental evaluation of the algorithm is beyond the aim of this paper. However, we tested the algorithm on some preliminary examples, and we plan to extend experimentation in future works. The set of the examples that we considered and a Python implementation of the algorithm are available at https://github.com/MC3-lab/PNstrunf.

The parameters of the net that we think are important to consider are: (1) the number of elements in the net; (2) the number of controllable transitions; (3) the level of concurrency, i.e. the maximum number of concurrent transitions that are enabled in a reachable marking; (4) the presence of cycles. We evaluate the performance of the algorithm by showing the total number of calls to the functions UNF_EXPLORATION and EXPLORE_CUT_C, and the number of cuts in the prefix at the end of the execution. The results of the experiments are shown in Table 1. In all these cases, the User has a winning strategy. From the results, we see that the level of concurrency and the cycles increase the computational cost of the algorithm. In some cases, cycles raise a lot the number of necessary steps to arrive at the solutions, without contributing in the research of the strategy (this is the case in the comparison between the nets *bc* and *bc2*). We are currently working to develop a preprocessing of the net, in order to identify these inactive part that may not be considered in the research of the strategy.

Table 1. Results of the experiments

Net	$\|P \cup T\|$	$\|K\|$	Conc	Cycles	#calls	g_dim
Heart	15	2	2	no	10	8
Double_heart	26	2	3	No	31	24
Big_heart	141	30	2	No	355	126
HeartC	19	2	2	Yes	20	14
bc	23	2	3	No	19	16
bc2	27	2	4	Yes	1882	1162
10conc0	32	0	10	No	5122	1024
10conc1	32	1	10	No	2307	1025
10conc2	32	2	10	No	1028	258
conc	12	2	3	Yes	255	143

5 Other Approaches to Asynchronous Games

The general notion of asynchronous game presented in this paper was defined in [4], where it was applied to a problem of controlled liveness, under the hypothesis of full observability.

An asynchronous game on Petri nets was also defined by Finkbeiner and Olderog in [9]. This game is developed for Place/Transition nets, and is played on their unfoldings. The players are represented by tokens, moving on the places of the unfolding, divided into two teams: system and environment. System players have an equivalent function as the User in the game defined by [4] and used in this paper. Their objective is to guarantee a safety property. For example, the aim might be to avoid reaching a certain place. The places are divided into system places, where system players can move, and environment places, reserved to environment players. The strategy is defined on each place and states which is the next place where a token has to move. Places are the central elements in this game, in contrast to the game in [4] where the focus is on transitions.

The information available to the players is another difference. In [4], and in our approach, this information consists in observed transitions. If a transition is observable, then the User knows whether the transition occurred or not. If a transition is unobservable, then there is no way for the User to know whether it occurred, unless he can infer this from observations. In the game described by Finkbeiner and Olderog, the players communicate by means of synchronizations. Participating in the same transition, they acquire the knowledge of the past of the players that take part to the synchronization. One or the other approach may be more convenient for the User/System depending on the structure of the system and on the property that has to be verified.

In [9] a strategy for the System is defined on the unfolding of the net system, and must be fair, i.e. if a System player can move, then it must do it. This requirement avoids the trivial case in which safety is verified just because the players refuse to move. In the game in [4] for a similar reason, progress is granted by the environment. In that case the User wishes to force a transition to occur infinitely often. In almost every case this goal would be impossible to reach if the environment does not fire any of its transitions.

Under the restricted hypothesis of just one environment player (and an arbitrary number of system players), and complete information, Bernd Finkbeiner, Manuel Gieseking and Ernst-Rüdiger Olderog developed a tool, presented in [8], finding a strategy for the game as defined in [9]. The tool translates the game to a standard two-players game over finite graphs.

A different approach for the verification of properties through asynchronous games was developed by several authors, among which Glynn Winskel in [13] and Julian Gutierrez in [11]. The game is defined on event structures. An event structure is a set of events in which a partial order and a conflict relation are defined. Event structures are in relation with Petri nets used in this paper: given an occurrence net, there is always an event structure with the same partial order and the same conflict relations of the events in the occurrence net. The opposite is also true: constructing an occurrence net in which the partial order between events is the same as in an event structure is always possible. However, this occurrence net is not always equivalent to the unfolding of a Petri net. As in the game in [9], the two players have limited knowledge of what happens in the system. When two or more events cause the occurrence of another one, there is an exchange of information that can be used by the strategy. Gutierrez shows that the game can be applied to the bisimulation problem and model-checking.

6 Conclusions

In this work we have presented an algorithm for the computation of a strategy for a reachability problem in a distributed net system with full observability. The algorithm has been implemented and tested on different nets. The next step consists in studying its complexity.

We plan to apply the general idea of the game to different problems and to define proper algorithms to find winning strategies in each case.

On the theoretical side, we will consider the case of partial observability. In this extended case the definition of a strategy needs to be redefined, because in general, if only some transitions are observable, the current marking of the system, and the current cut on the unfolding, are unknown. Moreover, while with full observability the information given by the observations on the system or on the unfolding is the same, with partial observability a strategy on the unfolding may be able to distinguish two different evolutions of the system, even if the observed transitions are the same. This happens because in the structure of the unfolding there is a track of the different stories of the system, hence being able to distinguish two events corresponding to the same transition would mean being able to reconstruct also the unobservable story of the system up to every observed event.

In addition, we will study the possibility of implementing the strategy, by adding causal dependencies between controllable and uncontrollable transitions, formalizing them with the insertion of new places in the net. If it is possible for a winning strategy to be implemented in such a way, then the goal of the User will be reached in every execution of the obtained net system.

Another future generalization is increasing the number of players. It would be interesting to analyse a game in which more than two players try to reach a goal, eventually in a cooperative or in a competitive way, and considering a game in which concurrency is allowed also in the User component.

Acknowledgments. This work has been partially supported by MIUR. The authors thank the anonymous referees for their useful comments.

References

1. Adobbati, F., Bernardinello, L., Pomello, L.: An asynchronous game on distributed Petri nets. In: Moldt, D., Kindler, E., Wimmer, M. (eds.) Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE 2019), co-located with the 40th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2019 and the 19th International Conference on Application of Concurrency to System Design ACSD 2019 and the 1st IEEE International Conference on Process Mining Process Mining 2019, Aachen, Germany, June 23–28, 2019. CEUR Workshop Proceedings, vol. 2424, pp. 17–36. CEUR-WS.org (2019). http://ceur-ws.org/Vol-2424/paper2.pdf
2. Adobbati, F., Bernardinello, L., Pomello, L.: Asynchronous games on Petri nets and partial order. In: Cherubini, A., Sabadini, N., Tini, S. (eds.) Proceedings of the 20th Italian Conference on Theoretical Computer Science, ICTCS 2019, Como, Italy, September 9–11, 2019. CEUR Workshop Proceedings, vol. 2504, pp. 139–144. CEUR-WS.org (2019). http://ceur-ws.org/Vol-2504/paper17.pdf
3. Bernardinello, L., Kılınç, G., Pomello, L.: Weak observable liveness and infinite games on finite graphs. In: van der Aalst, W., Best, E. (eds.) PETRI NETS 2017. LNCS, vol. 10258, pp. 181–199. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57861-3_12
4. Bernardinello, L., Pomello, L., Puerto Aubel, A., Villa, A.: Checking weak observable liveness on unfoldings through asynchronous games. In: Moldt, D., Kindler, E., Rölke, H. (eds.) Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE2018), Bratislava, Slovakia, June 24–29, 2018. CEUR Workshop Proceedings, vol. 2138, pp. 15–34. CEUR-WS.org (2018). http://ceur-ws.org/Vol-2138/paper1.pdf
5. Best, E., Darondeau, P.: Petri net distributability. In: Clarke, E., Virbitskaite, I., Voronkov, A. (eds.) PSI 2011. LNCS, vol. 7162, pp. 1–18. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29709-0_1
6. Desel, J., Kılınç, G.: Observable liveness of Petri nets. Acta Inf. **52**(2), 153–174 (2015). https://doi.org/10.1007/s00236-015-0218-1
7. Engelfriet, J.: Branching processes of Petri nets. Acta Inf. **28**(6), 575–591 (1991). https://doi.org/10.1007/BF01463946
8. Finkbeiner, B., Gieseking, M., Olderog, E.: Adam: causality-based synthesis of distributed systems. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 433–439. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_25
9. Finkbeiner, B., Olderog, E.: Petri games: synthesis of distributed systems with causal memory. Inf. Comput. **253**, 181–203 (2017). https://doi.org/10.1016/j.ic.2016.07.006
10. van Glabbeek, R.J., Goltz, U., Schicke-Uffmann, J.: On characterising distributability. Logical Methods in Comput. Sci. **9**(3) (2013). https://doi.org/10.2168/LMCS-9(3:17)2013
11. Gutierrez, J.: Concurrent logic games on partial orders. In: Beklemishev, L.D., de Queiroz, R. (eds.) WoLLIC 2011. LNCS (LNAI), vol. 6642, pp. 146–160. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20920-8_17
12. Ramadge, P., Wonham, W.: The control of discrete event systems. Proc. IEEE **77**(1), 81 (1989)
13. Winskel, G.: Distributed games and strategies. arXiv preprint arXiv:1607.03760 (2016)