| | |
|---|---|
| **Project no.:** | **PIRSES-GA-2011-295261** |
| **Project full title:** | **Mobility between Europe and Argentina applying Logics to Systems** |
| **Project Acronym:** | **MEALS** |
| **Deliverable no.:** | **4.1 / 2** |
| **Title of Deliverable:** | **Towards Fully Observable Non-deterministic Planning as Assumption-based Automatic Synthesis** |

| | |
|---|---|
| **Contractual Date of Delivery to the CEC:** | **30-Sep-2015** |
| **Actual Date of Delivery to the CEC:** | **30-Sep-2015** |
| **Organisation name of lead contractor for this deliverable:** | **ULEIC** |
| **Author(s):** | **Nicolás D'Ippolito, Sebastian Sardina** |
| **Participants(s):** | **UBA** |
| **Work package contributing to the deliverable:** | **WP4** |
| **Nature:** | **R** |
| **Dissemination Level:** | **Public** |
| **Total number of pages:** | **16** |
| **Start date of project:** | 1 Oct. 2011 **Duration:** 48 month |

Abstract:

Whereas previous work on non-deterministic planning has focused on characterizing (and computing) "loopy" but "closed" plans, we look here at the kind of environments that these plans are to be executed in. In particular, we provide a logical characterization of the standard "fairness" assumption used, and show that strong cyclic plans are correct solution concepts for fair environments. We argue then that such logical characterization allows us to recast non-deterministic planning as a reactive synthesis task, and show that for a special case, recent efficient synthesis techniques can be applied.

Note:

This deliverable is based on material that has been published in IJCAI 2015.

*This project has received funding from the European Union Seventh Framework Programme (FP7 2007-2013) under Grant Agreement Nr. 295261.*

# Contents

**1 Introduction** **3**

**2 Preliminaries** **4**
2.1 Fully Observable Non–Deterministic Planning . . . . . . . . . . . . . . . . . . . . 4
2.2 Labelled Transition Systems . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
2.3 Temporal Logics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

**3 Strong-cyclic Plans and Fair Environments** **6**

**4 FOND Planning via Reactive Synthesis** **9**

**5 Conclusions** **13**

**Bibliography** **13**

**MEALS Partner Abbreviations** **16**

# 1 Introduction

In this paper, we relate advanced forms of planning to the general long-standing Computer Science problem of automatic synthesis [20, 1, 25]: *the problem of automatically building an executable piece of code from (high-level) user intent*. Despite planning and synthesis not being areas too far from each other, the connection between the two worlds have not been much explored. We are not in fact the first in noticing that such a relation exists, in the planning community others have pointed it out; see [23]. Still, as no (complete) formalization has been presented, the relation is only informally analyzed. We concentrate in this work on *fully observable non-deterministic* (FOND) planning, a type of planning that has recently attracted much attention and which aims to accommodate events outside the control of the agent [6]. In FOND planning, actions are non-deterministic, in that their execution yields one of a set of possible effects, and this is Nature's choice. Once the effect has ensued, however, the agent is able to observe it.

Strong cyclic plans—those that re-try until success is obtained—have arguably become the de-facto solution concept for FOND planning. [6] provided its first characterization in CTL, as those plans for which "in any of its executions, it is always the case that the goal is always reachable." This was later refined in [23, 3] to disregard what happens after goal achievement. Then, with the advancement of planning and model checking technologies, several promising techniques and systems have emerged for solving FOND planning problems, such as PRP [21], NDP [19], and FIP [12], MBP [3], and GRENDEL [27]. The first three are built on top of classical planners, whereas the last two perform fix-point reasoning via model-checking type techniques. Nonetheless, the point is that they all amount to *specialized—clever though often involved—algorithms* for constructing "loopy" but "closed" plans.

In this work, we revisit what planning under non-deterministic actions is with the aim of framing it as a special case of *controller synthesis* [25]. The study of both novel controller synthesis and planning techniques could lead to interesting developments in the planning community by taking advantages of the expressiveness and guarantees provided by the controller synthesis approaches. Moreover, whereas control synthesis was deemed computationally impossible in the past, recent approaches (e.g., [2, 9]) have shown that for some quite expressive specifications, the task is amenable for computation. In turn, planning problem specifications can inform meaningful specifications—for goals and assumptions—to the reactive synthesis field.

It turns out that to make such link evident, one needs to formalize the assumptions on the environments in which strong cyclic plans are guaranteed to eventually achieve their goals. Surprisingly, this aspect has merely been discussed at an informal level, by calling for "fair" environments, that is, those in which "all actions' outcomes will ensue infinitely often, if tried infinitely often." To settle that formally, we provide a logical characterization of the environment which complements that of "good" plans found in [23]. We show, with an example, that a naive interpretation of the fairness assumption does not yield the expected results, and provide a characterization that accounts for "failure independence." We prove that strong cyclic plans are indeed sound and complete solution concepts under such characterization. Finally, we argue that, by using the characterization developed, reactive synthesis [25] can be directly used to solve FOND planning tasks, and show that, for the special case in which actions have "intended effects," existing effective synthesis techniques can be exploited.

# 2 Preliminaries

## 2.1 Fully Observable Non–Deterministic Planning

We mostly follow the characterisation of non-deterministic planning given in [30], as it provides a more formal framework than others to work on. However, such account is indeed equivalent to the usual "*oneof*" clauses in PDDL based characterisations [14].

A *fully observable non-deterministic (FOND) planning problem* is a tuple $\mathcal{P} = \langle P, O, s_I, \phi_{\text{goal}} \rangle$ consisting of a set of Boolean state propositions $P$ (atoms), an initial state $s_I$, a goal $\phi_{\text{goal}}$ as a conjunction of literals (i.e., atoms or negated atoms), and an operator set $O$ (see below). We use $\bar{l}$ to denote the complement of literal $l$.

A *state* $s$ is a consistent set (or conjunction) of literals such that $|s| = |P|$—every atom is either true or false. We use $S$ to denote the set of all states of task $\mathcal{P}$.

An *operator* is a pair $o = \langle Pre_o, \mathit{Eff}_o \rangle$, where $Pre_o$ is a condition describing the *preconditions* of operator $o$, and $\mathit{Eff}_o = e_1 \mid \cdots \mid e_n$ the (non-deterministic) *effects* of $o$, where each $e_i$ is a (deterministic effect) condition and $n \geq 1$. The intended meaning is that *one* of the $e_i$ events ensue non-deterministically, by the environment's choice. Operator $o$ is *executable* on a state $s$ if $s \models Pre_o$. The *successor* states resulting from executing operator $o$ in state $s$ is defined as $next(o, s) = [\![\mathit{Eff}_o]\!]_s$, where:

$$[\![e_1 \mid \cdots \mid e_n]\!]_s = \bigcup_{i=1}^{n} \{(s \setminus \{\bar{l} \mid e_i \models l\}) \cup \{l \mid e_i \models l\}\}.$$

In general, $[\![\mathit{Eff}_o]\!]_s$ yields a *set* of states (one per effect of $o$); if $o$ is deterministic (i.e., $\mathit{Eff}_o = e$), then $|[\![\mathit{Eff}_o]\!]_s| = 1$. A *policy* (or conditional plan) is a function $\pi : S \mapsto 2^O$ mapping states $s \in S$ onto the set of executable operators $\pi(s)$ such that if $o \in \pi(s)$, then $s \models Pre_o$. The *universal policy* for a FOND problem $\mathcal{P}$ is $\hat{\pi}(s) = \{o \mid o \in O, s \models Pre_o\}$. A policy $\pi$ executed from state $s$ defines a set of *possible executions* $\Lambda_\pi(s)$ made up of executions $\lambda = s_0 o_0 s_1 \cdots s_i o_i s_{i+1} \cdots$, where $s_0 = s$, $o_i \in \pi(s_i)$, $s_i \models Pre_{o_i}$, and $s_{i+1} \in next(o_i, s_i)$, for all $i \geq 0$. The set of states *relevant* to a policy $\pi$ from state $s$ is defined as $S_\pi(s) = \bigcup_{\lambda \in \Lambda_\pi(s)} \{s_i \mid s_i \in \lambda\}$ (we abuse notation and write $s \in \lambda$ to say that execution $\lambda$ mentions state $s$). A policy $\pi$ is *closed* iff $\bigcup_{o \in \pi(s_i)} next(o, s_i) \subseteq S_\pi(s_I)$ for all states $s_i \in S_\pi(s_I)$, that is, $\pi$ always returns an action for every non-goal state it reaches. In turn, a state $s$ is *reachable* by a policy $\pi$ from state $s'$ if there is a chance that following $\pi$ leads the agent to $s$; formally, there exists $\lambda \in S_\pi(s')$ such that $s \in \lambda$.

When it comes to FOND planning, the usual solution concept in the literature is that of strong cyclic plans.

**Definition 1** (Strong Cyclic Plan [3])**.** A *strong cyclic plan* is a closed policy such that a goal state is reachable from every reachable state using the policy.

Strong policies are a special case of strong cyclic policies, for which all executions are finite and *acyclic*: they solve the planning problem in a bounded number of steps. We close by noting that non-deterministic planning with full observability is EXPTIME-complete [29].

## 2.2 Labelled Transition Systems

Labelled Transition Systems (LTS) [17] are operational models that describe behaviour in terms of the occurrence of actions and their effect on the system's state. LTSs are widely used in many domains, particularly, as description language for event-based controllers and controller synthesis.

**Definition 2** (Labelled Transition Systems)**.** A *labelled transition system* (LTS) is a tuple $E = (S, A, \Delta, s_0)$, where $S$ is a finite set of states, $A$ is its *communicating alphabet*, $\Delta \subseteq (S \times A \times S)$ is a transition relation, and $s_0 \in S$ is the initial state. We use $\Delta(s) = \{\ell \mid (s, \ell, s') \in \Delta\}$ to denote the set of enabled actions in state $s$. A sequence $t = \ell_0\ell_1 \cdots$ is a *trace* in $E$ if there exists a sequence $s_0\ell_0 s_1\ell_1 \cdots$ such that $(s_i, \ell_i, s_{i+1}) \in \Delta$ for all $i \geq 0$.

The *parallel composition* of two LTS $E_i{=}(S_i, A_i, \Delta_i, e_{i0})$, with $i \in \{1, 2\}$, is defined as the LTS $E_1 \| E_2 = (S_1 \times S_2, A_1 \cup A_2, \Delta, (e_{10}, e_{20}))$ obtained from the synchronous product of $E_1$ and $E_2$, and encodes the concurrent execution of $E_1$ and $E_2$ by interleaving non-shared actions but forcing synchronisation on shared actions [16].

## 2.3 Temporal Logics

Temporal logics are logic formalisms tailored for statements and reasoning which involve the notion of order in time. They allow the formal specification of behavioral properties of systems [26] and are the basis for model checking and synthesis. Some widely known and used such logics are $\mathsf{CTL}^*$ and its fragments $\mathsf{CTL}$ and $\mathsf{LTL}$ [11, 4]. In particular, $\mathsf{CTL}$ has been proved to be a convenient framework for describing the notion of strong cyclic plans in a precise and concise manner. The syntax of $\mathsf{CTL}^*$ is defined with the following grammar:

$$\Psi ::= p \mid \Psi_1 \wedge \Psi_1 \mid \neg\Psi \mid \mathsf{A}\varphi \mid \mathsf{E}\varphi;$$
$$\varphi ::= \Psi \mid \varphi_1 \wedge \varphi_1 \mid \neg\varphi \mid \mathsf{X}\varphi \mid \mathsf{F}\varphi \mid \mathsf{G}\varphi \mid \varphi_1\mathsf{U}\varphi_2 \mid \varphi_1\mathsf{W}\varphi_2,$$

where $p$ ranges over the set of propositions. Formulas of the form $\Psi$ are called state formulas, whereas those of the form $\varphi$ are said to be path formulas. Formula $\mathsf{A}\Psi$ ($\mathsf{E}\Psi$) states that all executions (some execution) from the current state satisfy property $\Psi$. Then, path formulas $\mathsf{X}\varphi$, $\mathsf{F}\varphi$, and $\mathsf{G}\varphi$ state that $\varphi$ is true in the next state of the path, eventually in the path, or always along the path, resp. Finally, $\varphi_1\mathsf{U}\varphi_2$ says that $\varphi_1$ holds along the path until $\varphi_2$ becomes true and $\varphi_2$ is eventually true; $\varphi_1\mathsf{W}\varphi_2$ is its weak version where $\varphi_2$ is not required to eventually come true in the path. Common combinations of path and state quantifiers allow us to say things like *all* or *some* next states satisfy $\varphi$ ($\mathsf{AX}\varphi$ and $\mathsf{EX}\varphi$), and $\varphi$ holds in all or some executions ($\mathsf{AG}\varphi$ or $\mathsf{EG}\varphi$).

The meaning of such $\mathsf{CTL}^*$ formulas is given over the states and paths of a transition system, with a branching-time interpretation of time. Concretely, a $\mathsf{CTL}^*$ transition system over a set of propositions $P$, also called Kripke structure, is a tuple $\mathcal{K} = \langle W, R, P\rangle$, where:

- $W \subseteq 2^P$ is the set of states of $\mathcal{K}$; and
- $R \subseteq W \times W$ is the transition relation of $\mathcal{K}$ such that for every $w \in W$, there exists $w' \in W$ such that $R(w, w')$. When $R(w_1, w_2)$ holds, it means that state $w_2$ is a possible successor of state $w_1$.

A run in $\mathcal{K}$ from state $w_0$ is an infinite sequence $\pi = w_0 w_1 w_2 \cdots$ such that $R(w_i, w_{i+1})$, for all $i \geq 0$. Given a state formula $\Psi$ and a state $w \in W$, we can define whether $\Psi$ holds true in structure $\mathcal{K}$ at state $w$, denoted $\mathcal{K}, w \models \Psi$. Finally, $\mathsf{LTL}$ is the fragment obtained from $\mathsf{CTL}^*$ by withholding the $\mathsf{A}$ and $\mathsf{E}$ quantifiers and assuming universal path quantification. $\mathsf{CTL}$, in turn, is the $\mathsf{CTL}^*$ fragment obtained by requiring that each temporal path quantifiers (e.g., $\mathsf{X}$, $\mathsf{F}$, etc.) be under the immediate scope of an $\mathsf{A}$ or $\mathsf{E}$ quantifier.

A variant of $\mathsf{LTL}$ used in the Software Engineering context (from where we will import certain results) is $\mathsf{FLTL}$ (fluent-$\mathsf{LTL}$) [15], which is more tailored towards reasoning about fluents via automata-like structures with emphasis on transitions' labels rather than states' labels. A *fluent* $f = \langle I_f, T_f, init_f \rangle$ is defined by sets of disjoint initiating and terminating actions $I_f$ and $T_f$ from a universal set of actions $Act$, resp., and an initial value $init_f$.

Given a set of fluents, the truth of an $\mathsf{FLTL}$ temporal formula is defined relative to an infinite trace $t = a_0 a_1 \cdots$ over the universal set of actions $Act$. If $t$ is an infinite trace, the satisfaction of an $\mathsf{FLTL}$ formula $\varphi$ at position $i$, denoted $t, i \models \varphi$, is standard and omitted. However, we just point out that since $\mathsf{FLTL}$ traces focus on action/events (i.e., transitions labels in an LTS), the truth of a fluent $f$ at a certain prefix $t^i$ of trace $t = a_0 a_1 \cdots$ is defined as: $t, i \models f$ if and only if one of the following conditions holds: *(i)* $init_f \wedge (\forall j. 0 \leq j \leq i \rightarrow a_j \notin T_f)$; and *(ii)* $\exists j. (j \leq i \wedge a_j \in I_f) \wedge (\forall k \in \mathbb{N}. j < k \leq i \rightarrow a_k \notin T_f)$. We say that $\varphi$ holds in $t$, denoted $t \models \varphi$, if $t, 0 \models \varphi$. Finally, a $\mathsf{FLTL}$ formula $\varphi$ holds in an LTS $E$ (denoted $E \models \varphi$) iff $t, 0 \models \varphi$, for all infinite traces $t$ of $E$.

## 3 Strong-cyclic Plans and Fair Environments

In this section, we revisit the semantics of FOND planning, with special focus on the "environment" where plans are to be executed. In particular, whereas the notion of strong cyclic plans have been vastly studied and formally defined, the type of environments in which such plans will succeed are often discussed at an informal level.

The first precise analysis of what a solution is for FOND planning was given by [5]. There, the authors formally defined strong cyclic plans through a CTL formula on their executions. To capture all executions of a policy, they defined what is basically the projection of the underlying state model for a planning task onto the policy.

**Definition 3** ($(\mathcal{P}, \pi)$-structure)**.** Let $\mathcal{P} = \langle P, O, s_I, \phi_g \rangle$ be a FOND planning problem over the set of propositions $\mathcal{P}$ and $\pi$ a policy over $\mathcal{P}$. The induced structure of $(\mathcal{P}, \pi)$ is a Kripke structure $\mathcal{K}^{\pi}_{\mathcal{P}} = \langle W, R, P \rangle$, where:

- $W = \{\langle s, o \rangle \mid s \in 2^P, o \in O, s \models Pre_o\}$. Intuitively, $\langle s, o \rangle$ represents the execution of operator $o$ in state $s$;
- $R(\langle s, o \rangle, \langle s', o' \rangle)$ iff $o \in \pi(s)$ and $s' \in next(o, s)$.

Structure $\mathcal{K}^{\pi}_{\mathcal{P}}$ represents all evolutions of policy $\pi$ when executed in planning domain $\mathcal{P}$. It is straightforward to define a structure representing all possible executions of the planning domain, by considering the universal policy.

**Definition 4** ($\mathcal{P}$-structure)**.** The structure induced by a FOND planning problem $\mathcal{P}$ is defined as the Kripke structure $\mathcal{K}_{\mathcal{P}} = \mathcal{K}_{\mathcal{P}}^{\hat{\pi}}$, where $\hat{\pi}$ stands for the universal policy.

So, [5] defined a policy $\pi$ as a strong cyclic solution for a planning problem $\mathcal{P}$ iff $\mathcal{K}_{\mathcal{P}}^{\pi}, \langle s_I, o_I \rangle \models \mathsf{AGEF}\phi_{\text{goal}}$, for all $o_I \in \pi(s_I)$. In words, starting from the initial state, whatever actions we choose to execute (from the policty) and whatever their outcomes are, we always ($\mathsf{AG}$) have a way of reaching the goal ($\mathsf{EF}\phi_{\text{goal}}$).

Because goals are *achievement* goals in planning, what happens after the goal is achieved is irrelevant. Hence, [5]'s definition was then further refined as follows:

**Definition 5** ([23])**.** A policy $\pi$ is a *strong cyclic plan solution* for a FOND planning problem $\mathcal{P}$ iff $\mathcal{K}_{\mathcal{P}}^{\pi}, \langle s_I, o_I \rangle \models \mathsf{A}(\mathsf{EF}\phi_{\text{goal}} \mathsf{W} \phi_{\text{goal}})$, for all $o_I \in \pi(s_I)$.

In words, in all possible executions of the policy, the goal is always eventually reachable, at least until the goal is indeed reached. Observe that this definition precisely captures the notion of closeness and goal-reachable (as in Definition 1).

The availability of a precise notion for strong-cyclic plans has facilitated the development of various techniques capable of synthesising such type of plans. Planning systems like MBP [3], PRP [21], FIP [12], and GRENDEL [27] all search for policies that are closed and goal-reachable.

Nevertheless, besides understanding what a strong-cyclic plan amounts to, we argue here that it is also important to understand and formalize *the contexts under which these type of plans will indeed achieve the objectives*, namely, bringing about the goal. This has not received much attention and has mostly been discussed informally.

Clearly, under non-determinism, there is in principle the possibility of never achieving the goal, as "strong cyclic solutions can produce executions that loop forever" (without ever reaching the goal) [3]. However, the common understanding in the literature is that strong cyclic plans are adequate solutions under the assumption that the underlying environment described by the planning domain is '*'fair.*" Unlike strong cyclic plans, which have been formally characterized (see above), this fairness assumption is always stated at an informal level. We shall provide a precise characterization of such fairness in LTL and show that strong cyclic plans are sound and complete plan types for fair domains.

A usual (informal) understanding is that fair domains are those for which *"if an action is executed infinitely many times, every non-deterministic outcome will occur infinitely often."* Formally, this can be stated as the following strong-fairness LTL constraint (on the induced structure $\mathcal{K}_{\mathcal{P}}$):

$$\Phi_{\mathcal{P}}^{fair} \overset{\text{def}}{=} \bigwedge_{o \in O, e \in \mathit{Eff}_o} (\mathsf{GF}o \rightarrow \mathsf{GF}e).$$

Note that while strong cylic plans have been characterized in CTL (Definition 5), $\Phi_{\mathcal{P}}^{fair}$ above is an LTL formula not expressible in CTL. While simple, [9] argued, in the context of reactive synthesis for Software Engineering, that such strong-fairness assumptions are not enough to guarantee success of controllers that are meant to "re-try." Consider the following counter example.

*Example* 1. Imagine the problem of opening a door that has some operational defects, by inserting the key and then turning the knob. However, both actions may fail non-deterministically and not achieve their objectives. When that happens, the agent has to take the key out (which can only be done with the knob turned) and start all over again. Thus, consider planning problem $\mathcal{P}_{safe} = \langle P, O, s_I, \phi_{\text{goal}} \rangle$, where:

- $P = \{open, kIn, kStuck, turned\}$.
- $O$ includes the operators:
  - INSERT $= \langle \neg kIn, kIn \mid kIn \wedge kStuck \rangle$.
  - TURN$=\langle kIn \wedge \neg turned, open \wedge turned \mid turned \rangle$.
  - REMOVE$=\langle turned, \neg kIn \wedge \neg kStuck \wedge \neg open \wedge \neg turned \rangle$.
- $s_I = \{\neg open, \neg kIn, \neg kStuck, \neg turned\}$.
- $\phi_{\text{goal}} = open \wedge \neg kStuck$.

Take next the following (partial) policy $\pi$: *(i)* if $s \models \neg kIn$, then perform INSERT; *(ii)* if $s \models \neg turned \wedge kIn$, then perform TURN; and *(iii)* if $s \models turned$, then perform REMOVE.

It is not hard to see that $\pi$ is a strong cyclic solution: it continuously tries to first insert and use the key, and then turn the knob and remove the key, until success is achieved.

Now, the formula $\Phi^{fair}_{\mathcal{P}_{safe}}$, as defined above, implies that if the agent tries to use the key infinitely often, then she will succeed infinitely often. Similarly, if the agent keeps turning the knob, then she will keep succeeding getting the door open. It turns out, though, that:

$$\mathcal{K}^{\pi}_{\mathcal{P}_{safe}}, \langle s_I, \text{INSERT} \rangle \not\models \mathsf{A}[\Phi^{fair}_{\mathcal{P}_{safe}} \rightarrow \mathsf{F}(open \wedge \neg kStuck)].$$

That is, there is an execution of $\pi$ in $\mathcal{P}$ in which all outcomes of INSERT and TURN arise infinitely often, but it is always the case that either the door is closed or the key stuck. In fact, consider the following states of $\mathcal{K}^{\pi}_{\mathcal{P}_{safe}}$: *(i)* $w_0 = \langle s_I, \text{INSERT} \rangle$; *(ii)* $w_1 = \langle \{kIn, kStuck\}, \text{TURN} \rangle$; *(iii)* $w_2 = \langle \{kIn, turned\}, \text{REMOVE} \rangle$; and *(iv)* $w_3 = \langle \{kIn\}, \text{TURN} \rangle$; *(v)* $w_4 = \langle \{kIn, open, kStuck, turned\}, \text{REMOVE} \rangle$. Then, run $\lambda = (w_0 w_1 w_4 w_0 w_3 w_2)^{\omega}$ does satisfy the assumption formula $\Phi^{fair}_{\mathcal{P}_{safe}}$, but it never reaches the goal. Intuitively, the failure of the action TURN is *not* independent of failure of previous action INSERT—their failures are "synchronized."

The example above shows that the failures and successes of two non-deterministic actions can be "coordinated" in a way that will preclude goal achievability, despite the fact that all outcomes of both actions ensue infinitely often.

To rule out coordinated outcomes [9] provided the stronger notion of t-strong fair environments, but their framework assumes that it is known which action outcomes are "good" (and which ones are considered "failures"). Still, the intuition there is to have action-outcome fairness in every state, as captured by the following definition.

**Definition 6** (State Strong Fairness; t-strong fair)**.** Let $\mathcal{P}$ be a planning problem and $\mathcal{K}_{\mathcal{P}} = \langle W, R, P \rangle$ its corresponding Kripke structure. The *state strong, or simply t-strong, fairness* constraint is defined as the following LTL formula:

$$\gamma^{sfair}_{\mathcal{P}} \overset{\text{def}}{=} \bigwedge_{\{\langle s,o,e \rangle \mid \langle s,o \rangle \in W, e \in Eff_o\}} (\mathsf{GF}(s \wedge o) \rightarrow \mathsf{GF}[\![e]\!]_s).$$

A run of $\lambda$ of $\mathcal{K}_{\mathcal{P}}$ is *t-strong fair* if $\lambda \models \gamma_{\mathcal{P}}^{sfair}$.

In words, $\gamma_{\mathcal{P}}^{sfair}$ states that if an operator is executed infinitely often *in the same state*, then all its effects ought to arise infinitely often. This LTL constraint formalizes [3]'s claim that unfair executions are those where "some actions are executed infinitely often *in given states*, but some of its outcomes never occur," and corresponds to the *semantically expressed* requirement used in various FOND planning works (e.g., [22, 28, 13]).

Returning to our example above, the run $\lambda$ does *not* satisfy $\gamma_{\mathcal{P}_{safe}}^{sfair}$, and $\mathcal{K}_{\mathcal{P}}^{\pi} \models \mathsf{A}[\gamma_{\mathcal{P}_{safe}}^{sfair} \rightarrow \mathsf{F}\phi_{\text{goal}}]$ holds.

We now have all the machinery to state the main contribution of this section, namely, linking state strong fairness with [23]'s account of strong cyclic plans (see Definition 5). First we show that all such plans do achieve the goal in state strong fair environments.

**Theorem 1.** *Let $\pi$ be a strong cyclic plan for a FOND planning problem $\mathcal{P}$. Then, $\mathcal{K}_{\mathcal{P}}^{\pi}, \langle s_I, o_I \rangle \models$ $\mathsf{A}[\gamma_{\mathcal{P}}^{sfair} \rightarrow \mathsf{F}(\phi_{goal})]$, for all (initial) operators $o_I \in \pi(s_I)$.*

This theorem provides a "soundness" result for state strong fair environments. More importantly and less obvious, if a plan guarantees the goal in a state strong fair environment, then it has to be strong cyclic.

**Theorem 2.** *Let $\pi$ be a policy and $\mathcal{P}$ a FOND problem such that $\mathcal{K}_{\mathcal{P}}^{\pi}, \langle s_I, o_I \rangle \models \mathsf{A}[\gamma_{\mathcal{P}}^{sfair} \rightarrow$ $\mathsf{F}(\phi_{goal})]$, for all (initial) operators $o_I \in \pi(s_I)$. Then, $\pi$ is a strong cyclic plan.*

The above two theorems say that state strong fairness is a *complete* characterization of the type of environment for which strong cyclic plans are successful. We note that, together, the above theorems are closely related to Theorem 2 in [22], though ours crystallize the assumption-solution relationship directly within Traverso *et al.*'s FOND planning foundational framework. Having a precise characterization of what adequate environments are opens the door for applying (reactive) synthesis techniques to solve FOND problems. That is the topic of the next section.

## 4 FOND Planning via Reactive Synthesis

Reactive synthesis or controller synthesis studies techniques that given a model of the assumed behaviour of the environment ($E$) and a system goal ($G$), produces an operational behaviour model for a component $M$ that when, executed in an environment consistent with the assumptions, results in a system that is guaranteed to satisfy the goal, namely, $E \| M \models G$.

Arguably, the application of controller synthesis as means for solving planning problems has been discouraged due to the high complexity of the general approach. Indeed, controller synthesis is, in general, computationally expensive: synthesising for $\mathsf{LTL}$ goals is 2EXPTIME complete and known techniques have resisted practical implementations due to complementation [25, 18]. However, recent advances (e.g., [2, 9]) may, under appropriate assumptions, provide efficient solutions to planning problems. The fact is that restricting the form of the specification system goal allows for more effective solutions and implementable systems. For instance, goal specifications consisting uniquely of safety requirements can be solved in polynomial time, and so can particular, yet expressive, subsets of liveness properties such as GR(1) [24].

As stated in Section 3, FOND strong cyclic plans are guaranteed to achieve the goals in t-strong fair environments. Following [2] and relying on Theorems 3 and 4, one can solve a FOND planning task $\mathcal{P}$ with goal $\phi_{\text{goal}}$ by checking LTL realizability of specification formula

$$\varphi_{\mathcal{P}} \overset{\text{def}}{=} \varphi_{\mathcal{P}}^{1} \wedge [\varphi_{\mathcal{P}}^{2} \rightarrow (\gamma_{\mathcal{P}}^{sfair} \rightarrow \mathsf{F}\phi_{\text{goal}})],$$

where $\varphi_{\mathcal{P}}^{1}$ and $\varphi_{\mathcal{P}}^{2}$ are simple (safety) LTL formulas encoding structure $\mathcal{K}_{\mathcal{P}}$.[1] Unfortunately, $\varphi_{\mathcal{P}}$ results in an expression that does not allow for efficient synthesis techniques, as it does not fall into known efficient LTL fragments like GR(1). Hence, in general, there is still an efficiency challenge when solving FOND planning via controller synthesis.

In some cases, however, it is possible to reduce a planning problem to an equivalent *efficient* controller synthesis one. For example, [7] does so for *standard* conditional planning (i.e., planning for strong solutions). Here, we discuss a special such case for FOND planning, namely, when every action of a non-deterministic planning problem can be annotated with its "expected" or "intended" effect—that is, actions have a clear fail/success semantics. For instance, in the example in Section 3, the expected effect of turning the knob is to successfully open the door. All other effects can be considered failures. Arguably, many meaningful FOND planning problems come with a clear success/failure semantics of actions.

In what follows, we show how efficient reactive synthesis techniques developed in the area of Software Engineering can be imported to solve this type of planning problems. Specifically, we rely on [9]'s polynomial approach to building LTS controllers for control problems *with explicit description of failures* against GR(1)-like specifications. Failures are defined by introducing *try-response* sets that relate the attempt to perform a controlled action and the corresponding environment's reaction. Technically, a try-response triple is a tuple $t = \langle a, suc_a, fail_a \rangle$ where $a$ and $suc_a$ are actions in the environment model representing an action and its intended result, respectively, and $fail_a$ is a set of actions representing failures of $a$'s execution. In planning terms, a try would be an operator, while the responses would its effects. A key aspect of the approach is the *Strong Independent Fairness* (SIF) condition, which provides restrictions on how failures behave [9, Def. 4.3]. Intuitively, SIF requires a windows of opportunity ("gaps") in which all expected effects are executed without *any* failures. It turns out that the SIF assumption allows to reduce the problem of having gaps with no failures to one in which failures are assumed to occur finitely.

When interpreting failures as operator's effects, most of the required restrictions for try-response triples are inherently guaranteed (e.g., re-tries cannot occur before a response), except for one restriction that needs to be lifted. While [9] assumes single action representation for failures and successes, here we need failures to be a set of actions—the set of non-intended effects. However, since their approach reduces to solving a control problem where failures occur only finitely often, having multiple failures for the same try action does not affect key fact that

[1]It is possible to, almost directly, transform $\mathcal{K}_{\mathcal{P}}$ into a game structure $G$ as used in [2], and then extract $\varphi_{\mathcal{P}}^{1}$ and $\varphi_{\mathcal{P}}^{2}$ from $G$. For lack of space, we omit this here.

effective controllers (re)try actions knowing that success will eventually occur. For lack of space we omit the technical details of such generalization to *set* of failures.

Next, we formally define try-response sets for a given LTS restricting attention to the requirements described above.

**Definition 7** (Try-Response)**.** Let $E = (S, A, \Delta, s_0)$ be an LTS and $A_c \subseteq A$ (the set of controllable actions). Set $T$ is a *try-response* set for $E$ if:

1. $(a, suc_a, fail_a) \in T$ iff $a \in A_c$, $suc_a \in A \setminus A_c$, and $fail_a = \{fail_a^1, \ldots, fail_a^k\} \subseteq A \setminus A_c$ for some $k \geq 1$.
2. For all $t_a = (a, suc_a, fail_a), t_b = (b, suc_b, fail_b) \in T$, if $a = b$ then $t_a = t_b$.
3. $E \models \mathsf{G}[\mathsf{X}(\bigvee_{1 \leq i \leq |fail_a|} fail_a^i \vee suc_a) \leftrightarrow a]$, for all $a \in A_c$
4. For all $s \in S$ and $try_a \in A_c$, $fail_a$ is enabled from $s$ iff $suc_a$ is enabled from $s$.

We note that this definition is a special case of that in [9], tailored to the fact that outcomes ensue immediately after operators' executions. In their work, they also need to deal with delayed failures/successes.

Now, the specific control problem we are interested in is defined with an LTS environment model, a GR(1)-like specification, a set of controllable actions, and a try-response set.

**Definition 8** (Recurrent Success LTS Control)**.** A *recurrent success LTS control* (RSGR) problem is a tuple $\mathcal{E} = \langle E, A_c, \varphi, T \rangle$ where $E$ is an LTS, $A_c$ is a set of controllable actions, $\varphi$ is a GR(1) FLTL formula, and $T$ is a try-response set for $E$.

A *solution for* $\mathcal{E}$ is an LTS $M$ with controlled actions $A_c$ such that $E$ only disables controllable actions, $E \| M$ is deadlock free, and every trace $\pi$ in $M \| E$ is such that $\pi \models \varphi$.

That is, a solution to an RSGR control problem is an LTS controller that, in parallel composition with the environment model, meets the system specification goal. Thanks to the assumed failure semantics, such controller can be computed in polynomial time via a reduction to a GR(1) synthesis task [9].

So, let us next show how to reduce a FOND planning problem with intended effects to an RSGR control problem.

**Definition 9** (FOND to RSGR)**.** Let $\mathcal{P} = \langle P, O, s_I, \phi_{\text{goal}} \rangle$ be a a FOND planning problem, such that each operator $o \in O$ is defined as $o = \langle Pre_o, \mathit{Eff}_o, Int_o \rangle$ where $Int_o \in \mathit{Eff}_o$ stands for the intended effect of operator $o$. The corresponding RSGR problem $\mathcal{E}_{\mathcal{P}} = \langle E, A_c, \varphi, T \rangle$ is defined as follows (here $\mathit{Eff}(O) = \{e \mid o \in O, e \in \mathit{Eff}_o\}$ is all the effects of $O$):

- $A_c = O$;
- $T = \{(o, Int_o, E_o) \mid o \in O \textit{ and } E_o = \mathit{Eff}_o \setminus Int_o\}$; and
- $\varphi = \mathsf{F}(\bigwedge_{l \in \phi_{\text{goal}}} f_l)$, where:

$$f_l = \langle \{e \in \mathit{Eff}(O) \mid e \models l\}, \{e \in \mathit{Eff}(O) \mid e \models \bar{l}\}, \bot \rangle;$$

- $E = (2^P \cup (2^P \times O), \mathit{Eff}(O) \cup O, \Delta, s_I)$ is an LTS with:

$$\Delta = \{(s, o, (s, o)) \mid o \in O, s \models \mathit{Pre}_o\} \cup \\ \{((s, o), e_o, s') \mid e_o \in \mathit{Eff}_o, s' \in [\![e_o]\!]_s\}.$$

A trace of $E$ is an interleaving sequence of operators and effects (from where one can reconstruct the fluents that hold true at any point). Observe that a controller for an RSGR control problem only restricts controllable actions (in our case, planning operators). Also, as we deal with reachability goals, it does not require additional memory: effectively, it is a subgraph of the environment model. This last property enables us to produce a strong cyclic policy from it.

**Definition 10** (LTS to Policy)**.** Given a planning problem $\mathcal{P} = \langle P, O, s_I, \phi_{\text{goal}}\rangle$, and an LTS $M = (S, A, \Delta, s_0)$ solution for the RSGR control problem $\mathcal{E}_\mathcal{P}$ obtained from $\mathcal{P}$ as described above, we construct $M$'s associated policy $\pi_M$ as follows: for all $s \in 2^P$: $o \in \pi_M(s)$ iff $o \in \Delta(s)$.

So, the next result states that the induced policy from a solution to the corresponding RSGR control problem is indeed a strong cyclic solution for the planning task of concern.

**Theorem 3** (Soundness)**.** *Let $\mathcal{P}$ be a planning problem with goal $\phi_{goal}$, $M$ be an LTS solution to the RSGR control problem $\mathcal{E}_\mathcal{P}$, and $\pi_M$ the policy obtained from $M$. Then, $\mathcal{K}_\mathcal{P}^{\pi_M}, \langle s_I, o_I\rangle \models \mathsf{A}(\mathsf{EF}\phi_{goal}\mathsf{W}\phi_{goal})$, for all $o_I \in \pi_M(s_I)$.*

*Proof.* Suppose $\mathcal{K}_\mathcal{P}^{\pi_M}, \langle s_I, o_I\rangle \not\models \mathsf{A}(\mathsf{EF}\phi_{\text{goal}}\mathsf{W}\phi_{\text{goal}})$. Hence, there is a state $s$ in $\mathcal{K}_\mathcal{P}^{\pi_M}$ such that every execution $\lambda$ in $\mathcal{K}_\mathcal{P}^{\pi_M}$ that visits $s$ (i.e., $s \in \lambda$) is such that $\lambda \not\models \mathsf{F}\phi_{\text{goal}}$. By construction, $s$ also belongs to $M$. Further, all traces in $M$ that visit $s$ must also avoid the goal—i.e., never satisfy the FLTL formula $(\bigwedge_{l\in\mathcal{L}(\phi_{\text{goal}})} f_l)$. In particular, every trace $t$ satisfying SIF condition that visits $s$ will not satisfy the goal (i.e. $t \not\models \mathsf{F}(\bigwedge_{l\in\mathcal{L}(\phi_{\text{goal}})} f_l)$), which contradicts our hypothesis. □

A direct consequence of Theorems 1 and 3 is that $\mathcal{K}_\mathcal{P}^{\pi} \models \mathsf{A}[\gamma_\mathcal{P}^{sfair} \rightarrow \mathsf{F}\phi_{\text{goal}}]$. Secondly, if the FOND problem has a solution that *is based on the operators' intended effects*, then it is accounted in the solution to the corresponding RSGR problem. Intuitively, a policy is intention-based if it can reach the goal relying (eventually) only on the intended effects of operators—i.e., it achieves the goal "under no failures."

**Definition 11** (Intention-Based Policy)**.** Let $\mathcal{P}$ be a FOND planning problem with goal $\phi_{\text{goal}}$ and operators of the form $o = \langle \mathit{Pre}_o, \mathit{Eff}_o, \mathit{Int}_o\rangle$, where $\mathit{Int}_o \in \mathit{Eff}_o$ stands for the intended effect of $o$. A strong cyclic policy $\pi$ for $\mathcal{P}$ is *intention-based* iff for all $w_I \in \{s_I\} \times \pi(s_I)$:

$$\mathcal{K}_\mathcal{P}^{\pi}, w_I \models \mathsf{A}[\mathsf{F}(\bigwedge_{s\in S, o\in O} \mathsf{G}(s \wedge o) \rightarrow [\![\mathit{Int}_o]\!]_s) \rightarrow \mathsf{F}\phi_{\text{goal}}].$$

**Theorem 4** (Completeness)**.** *Let $\mathcal{P}$ be a problem, and $\mathcal{E}_\mathcal{P}$ its corresponding RSGR control problem. If an intention-based strong cyclic policy $\pi$ for $\mathcal{P}$ exists, then there exists a solution $M$ for $\mathcal{E}_\mathcal{P}$ such that $\pi \subseteq \pi_M$.*

*Proof.* We construct an LTS $M$ from policy $\pi$ and show that $M$ is a solution to $\mathcal{E}_{\mathcal{P}}$. Let $M = (S, A, \Delta_M, s_0)$ be an LTS defined as follows: $s_0 = s_I$; $S = 2^P \cup (2^P \times O)$; $A = \{e \mid O, e \in \mathit{Eff}_o\} \cup O$; and $\Delta = \{(s, o, (s, o)) \mid o = \pi(s)\} \cup \{((s, o), e_o, s') \mid e_o \in \mathit{Eff}_o, s' \in [\![e_o]\!]_s\}$. Now, the proof amounts to proving that for all traces $t$ of $M \| E$, if $t$ is $\mathsf{SIF}$, then $t \models \mathsf{F}(\bigwedge_{l \in \phi_{\text{goal}}} f_l)$. Since $t$ is $\mathsf{SIF}$, we know that $t$ exhibits long enough gaps showing all intended effects of operators and no failures. Consider the prefix $t^i$ of $t$ that includes the first such a gap. There exists an execution $\lambda_{t^i}$ of policy $\pi$ corresponding to prefix $t^i$, which is "intended base" and hence achieves the goal. Therefore, prefix $t^i$ also achieves the goal specification and $t \models \mathsf{F}(\bigwedge_{l \in \phi_{\text{goal}}} f_l)$ applies. □

Now, observe that [9] proved that solving a RSGR control problem can be done polynomially, by reducing it to a GR(1) synthesis task [2]. Since the encoding of a FOND problem into a RSGR control one (Definition 9) is exponential, we get that we can use reactive synthesis to obtain intended-based strong cyclic solution in exponential time (which is the complexity of conditional planning). In addition, unlike standard solutions via specialized FOND algorithms, the policy is "maximal," in that it accounts for every initial state for which a solution exists.

## 5 Conclusions

In this paper we formally studied the kind of environments in which strong cyclic plans are adequate, thus complementing the existing formal characterization of such plans in CTL and contributing to bridging the gap between automated planning and reactive synthesis. We showed that an adequate fairness assumption should encode the independence of different non-deterministic steps, and provided a characterization of such environments in CTL*/LTL. We argued that doing so allows for direct specification of the FOND planning task into a reactive synthesis one. While the general synthesis task is not amenable for efficient computation, we showed a special case of FOND problems—those in which actions have intended effects—-for which low complexity techniques are available.

We believe that making the connection between planning and controller synthesis more evident can benefit both areas. Automated planning can exploit recent powerful techniques—like synthesis of GR(1) or safety specifications—and open for more general planning settings. As pointed out in [8], one could be interested in planning under "selective" and/or "conditional" fairness assumptions (e.g., the dice is fair unless an action has "loaded" it), something that specialized planners cannot handle. Indeed, the special case of actions with intended effects is one such case. For reactive synthesis, planning settings can provide concrete applications and inform what types of system goal specifications are meaningful and worth studying.

In future work, we plan to investigate other solution concepts for non-deterministic planning. In particular, we are interested in fault-tolerant planning [10].

## Bibliography

[1] Martín Abadi, Leslie Lamport, and Pierre Wolper. Realizable and unrealizable specifications of reactive systems. In *Proc. of the International Colloquium on Automata, Languages and*

*Programming (ICALP)*, pages 1–17, 1989.

[2] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. *Journal of Computer and System Sciences*, 78(3):911–938, 2012.

[3] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1-2):35–84, 2003.

[4] Edmund Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In Dexter Kozen, editor, *Logics of Programs*, volume 131 of *Lecture Notes in Computer Science (LNCS)*, chapter 5, pages 52–71. Springer, Berlin/Heidelberg, 1982.

[5] Marco Daniele, Paolo Traverso, and Moshe Y. Vardi. Strong cyclic planning revisited. In *Proc. of the European Conference on Planning (ECP)*, pages 35–48, 1999.

[6] M. Daniele, P. Traverso, and M. Vardi. Strong cyclic planning revisited. *Recent Advances in AI Planning*, pages 35–48, 2000.

[7] Giuseppe De Giacomo, Fabio Patrizi, Paolo Felli, and Sebastian Sardina. Two-player game structures for generalized planning and agent composition. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, pages 297–302, 2010.

[8] Giuseppe De Giacomo, Fabio Patrizi, and Sebastian Sardina. Generalized planning with loops under strong fairness constraints. In *Proc. of the Int. Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 351–361, 2010.

[9] Nicolás D'Ippolito, Víctor A. Braberman, Nir Piterman, and Sebastián Uchitel. Synthesis of live behaviour models for fallible domains. In *Proceedings of the International Conference on Software Engineering*, pages 211–220, 2011.

[10] Carmel Domshlak. Fault tolerant planning: Complexity and compilation. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2013.

[11] E. Allen Emerson and Joseph Y. Halpern. *Journal of the ACM*, 33(1):151–178, January 1986.

[12] Jicheng Fu, Vincent Ng, Farokh Bastani, and I-Ling Yen. Simple and fast strong cyclic planning for fully-observable non-deterministic planning problems. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1949–1954, 2011.

[13] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers, 2013.

[14] Alfonso Gerevini, Blai Bonet, and Bob Givan, editors. *Booklet of 4th International Planning Competition*, Lake District, UK, 2006.

[15] Dimitra Giannakopoulou and Jeff Magee. Fluent model checking for event-based systems. *SIGSOFT Softw. Eng. Notes*, 28(5):257–266, September 2003.

[16] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[17] Robert M. Keller. Formal verification of parallel programs. *Communications of the ACM*, 19:371–384, 1976.

[18] Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. Safraless compositional synthesis. In *Proc. of the International Conference on Computer Aided Verification (CAV)*, pages 31–44, 2006.

[19] U. Kuter, S. Nau, D., E. Reisner, and P. Goldman, R. Using classical planners to solve nondeterministic planning problems. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 190–197, 2008.

[20] Zohar Manna and R. Waldinger. How to clear a block: A theory of plans. *Journal of Automed Reasoning*, 4(3):343–377, 1987.

[21] Christian Muise, Sheila A. McIlraith, and J. Christopher Beck. Improved non-deterministic planning by exploiting state relevance. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 172–180, 2012.

[22] Fabio Patrizi, Nir Lipovetzky, and Hector Geffner. Fair LTL synthesis for non-deterministic systems using strong cyclic planners. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[23] Marco Pistore and Paolo Traverso. Planning as model checking for extended goals in non-deterministic domains. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 479–486, 2001.

[24] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. In *Proc. of the International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 3855 of *Lecture Notes in Computer Science (LNCS)*, pages 364–380. Springer, 2006.

[25] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Proc. of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 179–190, 1989.

[26] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (SFCS)*, pages 46–57, 1977.

[27] Miguel Ramirez and Sebastian Sardina. Directed fixed-point regression-based planning for non-deterministic domains. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 235–243, 2014.

[28] Miguel Ramirez, Nitin Yadav, and Sebastian Sardina. Behavior composition as fully observable non-deterministic planning. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 180–188, 2013.

[29] Jussi Rintanen. Complexity of planning with partial observability. In *Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 345–354, 2004.

[30] Jussi Rintanen. Regression for classical and nondeterministic planning. In *Proc. of the European Conference in Artificial Intelligence (ECAI)*, pages 568–572, 2008.

## MEALS Partner Abbreviations

**SAU:** Saarland University, D

**RWT:** RWTH Aachen University, D

**TUD:** Technische Universität Dresden, D

**INR:** Institut National de Recherche en Informatique et en Automatique, FR

**IMP:** Imperial College of Science, Technology and Medicine, UK

**ULEIC:** University of Leicester, UK

**TUE:** Technische Universiteit Eindhoven, NL

**UNC:** Universidad Nacional de Córdoba, AR

**UBA:** Universidad de Buenos Aires, AR

**UNR:** Universidad Nacional de Río Cuarto, AR

**ITBA:** Instituto Técnológico Buenos Aires, AR