Agents, Actions and Goals in Dynamic Environments

Peter Novák[*]
Department of Cybernetics, FEL
Czech Technical University in Prague
Czech Republic

Wojciech Jamroga[†]
Computer Science and Communications
University of Luxembourg
Luxembourg

Abstract

In agent-oriented programming and planning, agents' actions are typically specified in terms of postconditions, and the model of execution assumes that the environment carries the actions out exactly as specified. That is, it is assumed that the state of the environment after an action has been executed will satisfy its postcondition.

In reality, however, such environments are rare: the actual execution of an action may fail, and the envisaged outcome is not met. We provide a conceptual framework for reasoning about success and failure of agents' behaviours. In particular, we propose a measure that reflects how "good" an environment is with respect to agent's capabilities and a given goal it might pursue. We also discuss which types of goals are worth pursuing, depending on the type of environment the agent is acting in.

1 Motivation

In dynamic environments agent's actions can fail. This might be either directly due to failing effectors of the agent, or indirectly due to incompleteness of information. The latter situation may arise either because precise and complete information is inherently impossible (think of a robot dealing with complex physical phenomena, such as the weather), or technically infeasible, resp. undesirable in the given application domain (e.g., high space complexity of the environment representation, or high rate of environment change in relation to the speed of the agent's deliberation).

Construction of agents in the face of unexpected failures is difficult. As a reaction to difficulties with classical planning in dynamic environments, the paradigm of agent-oriented programming (AOP) based on reactive planning (e.g., [Bordini *et al.*, 2006]) became one of the state-of-the-art techniques for construction of intelligent agents. Even though the motivation behind AOP is rooted in the idea that more reactive style of deliberation is more appropriate for agent's interaction with a dynamic environment, the precise characteristics of the relationship have not yet been deeply studied.

In this paper we precisely investigate *the relationships between capabilities and control mechanisms of an agent, its design objectives, the goal, and characteristics of the environment it is embodied in*. As a conceptual framework supporting the discourse, in Section 2 we introduce a series of agent behaviour performance measures. We start from relating the set of agent's generic capabilities to an environment in which actions can fail. Subsequently, we introduce agent programs constructed from the basic actions and based on their performance in the environment, we provide a taxonomy of their mutual matchings. Finally, in Section 3 we consider the relationship between a program, its design objective, a goal it is aimed to fulfil, and characteristics of the environment. We study the relationships of the triad using *Probabilistic Dynamic CTL** logic (pDCTL*), a novel temporal logic framework facilitating reasoning about agent system specifications and actual programs aimed at realising it. We observe, that imperfect performance of agent programs situated in dynamic environments w.r.t. their goals can in fact be caused by two distinct phenomena. While the environment by its dynamics can make the program fail, it can be also the program itself, which is not implemented perfectly w.r.t. the goal. We conclude the discourse of the paper by discussion in Section 4 of how program construction influences its performance w.r.t. the design specification, and finally, in Section 5 we formally relate the two causes of program imperfection. In particular, we introduce an *impact metric*, a measure indicating how much the imperfections in program implementation influence the chances for reaching the goal.

2 Agents, actions, environments

We focus on actions of an agent acting in an environment. The agent's actions can change the state of the environment, possibly in a probabilistic way. Other agents, if present in the system, are not relevant at this stage and are assumed to be appropriately modelled as a part of the environment. Due to this characteristics, we will model environments as Markov decision processes (MDP's) [Bellman, 1957]. By this, we implicitly assume that agents can always exactly recognise the current state of the environment. We leave analysis of the more general case (partial observability) for future work.

[*]Supported by the Czech Ministry of Education, Youth and Sports, grant MSM6840770038 and the Grant Agency of the Czech Technical University in Prague, grant SGS10/189/OHK3/2T/13.

[†]Supported by the FNR (National Research Fund) Luxembourg under project S-GAMES – C08/IS/03.

2.1 Basic notions

Definition 2.1 (Environment). *Environment* E is modelled as a Markov decision process $(\mathcal{S}, \mathcal{E}, P)$ where $\mathcal{S}$ is a set of states the environment can be in, $\mathcal{E}$ is a set of events which can happen in E and $P : \mathcal{S} \times \mathcal{E} \times \mathcal{S} \to [0, 1]$ is a probabilistic transition function with transitions labelled by events. That is, $P(s, e, s')$ defines the probability that, upon occurrence of the event e in the state s, the next state of the environment will be s'. We will adopt the convention that if e is not enabled in s then $P(s, e, s') = 0$ for all $s' \in \mathcal{S}$.

Furthermore, we assume that some propositional language $\mathcal{L}$ is available to characterise properties of states of E. $\mathcal{L}$ comes with a standard satisfaction relation $\models$, with $E, s \models \phi$ meaning that the formula $\phi \in \mathcal{L}$ holds in the state s of the environment E.

An agent consists of a template that specifies how it *can* act (i.e., provides a set of basic operations available to the agent) and a program that prescribes how it *will* act (e.g., by defining its deliberation mechanism). An agent must match its environment in the sense that its actions must be events in the environment. Moreover, each action is annotated with a specification of its envisaged effects. In an ideal environment, the annotation should hold after the action has been executed.

Definition 2.2 (Agent template). Let $E = (\mathcal{S}, \mathcal{E}, P)$ be an environment. An *agent template* $(Act, \mathfrak{Ann})$ situated in E specifies the set of basic actions (capabilities) $Act \subseteq \mathcal{E}$ that the agent can execute in E, together with the function $\mathfrak{Ann} : Act \to \mathcal{L}$ that annotates the agent's actions by formal descriptions of their expected effects.

Definition 2.3 (Macro actions and traces). A *macro action* is a possibly infinite sequence of actions $\rho = a_1, \ldots, a_n, \ldots$ with $a_i \in Act$.

An execution trace λ of macro action ρ rooted in a state $s_0 \in \mathcal{S}$ of environment E is a (finite or infinite) sequence of labelled transitions $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n \xrightarrow{a_{n+1}} \cdots$ such that $s_i \in \mathcal{S}$ and $P(s_{i-1}, a_i, s_i) > 0$ for all i. By $\lambda[i] = s_i$, we denote the ith state on λ, and $\lambda[i..j] = s_i \xrightarrow{a_{i+1}} \ldots \xrightarrow{a_j} s_j$ denotes the "cutout" from λ from position i to j. The ith prefix and suffix of λ are defined by $\lambda[0..i]$ and $\lambda[i..\infty]$, respectively. $|\lambda|$ denotes the number of transitions in λ. In the case λ is infinite, we write $|\lambda| = \infty$. For a given trace $\lambda = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n \xrightarrow{a_{n+1}} \cdots$, we denote $\rho(\lambda) = a_1, a_2, \ldots, a_n, \ldots$ the macro action, execution of which resulted in λ.

Definition 2.4 (Situated agent). An *agent* is represented by a tuple $A = (Act, \mathfrak{Ann}, \tau)$ where $(Act, \mathfrak{Ann})$ is an agent template situated in an environment E, and τ is an *agent program* over Act. The set $\mathcal{P}$ of well-formed programs is defined as follows:

- a is a program for every $a \in Act$;
- If τ, τ' are programs, then also $\tau \cup \tau'$, $\tau; \tau'$ and τ^* are programs denoting non-deterministic choice, sequential composition and unbounded iteration respectively;
- Additionally, fixed iteration τ^n is defined recursively as $\tau^1 = \tau$ and $\tau^{n+1} = \tau^n; \tau$.

The semantics of an agent program is defined in terms of the set of traces $\mathcal{T}_E(\tau, s_0)$ it induces in the environment E, rooted in some initial state s_0. Formally, $\mathcal{T}_E(\tau, s_0)$ is defined inductively

- $\mathcal{T}_E(\langle a \rangle, s_0) = \{s_0 \xrightarrow{a} s_1 \mid P(s_0, a, s_1) > 0\}$,
- $\mathcal{T}_E(\tau; \tau', s_0) = \{\lambda \mid$ there exists $i : \lambda[0..i] \in \mathcal{T}_E(\tau, s_0)$ and $\lambda[i..\infty] \in \mathcal{T}_E(\tau', \lambda[i])\}$,
- $\mathcal{T}_E(\tau \cup \tau', s_0) = \mathcal{T}_E(\tau, s_0) \cup \mathcal{T}_E(\tau', s_0)$,
- $\mathcal{T}_E(\tau^*, s_0) = \{\lambda \mid \lambda[0] = s_0,\ k_0 = 0$ and there exist $k_1, k_2, \ldots$ s.t., $\lambda[k_i, k_{i+1}] \in \mathcal{T}_E(\tau, \lambda[k_i])\}$.

Additionally, $\mathcal{T}_E(\tau)$ denotes the set of all traces induced by τ in states of E, i.e., $\mathcal{T}_E(\tau) = \bigcup_{s \in \mathcal{S}} \mathcal{T}_E(\tau, s)$. Moreover, given a set of sequences X, we will use $Fin(X)$ to denote the set of finite prefixes of the sequences from X. For example, $Fin(\mathcal{T}_E(\tau))$ is the set of finite histories that can occur during execution of τ in E.

Note, that the generic form of agent programs we define in Definition 2.4, serves only for exposition of ideas in this paper. In fact, any succinct way of encoding of agent's behaviour in terms of enabled execution traces (intended system evolutions) would serve equally well. This broader understanding of agent behaviours naturally includes various planning mechanisms, as well as most state-of-the-art AOP languages.

Hereafter, unless specifically stated otherwise, we will assume an agent $A = (Act, \mathfrak{Ann}, \tau)$ situated in an environment $E = (\mathcal{S}, \mathcal{E}, P)$, s.t. the annotation function $\mathfrak{Ann}$ is expressed in some propositional language $\mathcal{L}$ with a satisfaction relation $\models$ that interprets formulae of $\mathcal{L}$ in states of E.

2.2 Probabilistic execution of actions and programs

Annotations play dual role in the specification of agents' capabilities. On one hand, they put forward the envisaged outcome of an action in an ideal environment. On the other, they allow to define the notion of successful execution of the action (and, dually, the notion of execution failure).

Definition 2.5 (Success and failure of actions). A transition $s \xrightarrow{a} s'$ is a *successful execution* of a if $s' \models \mathfrak{Ann}(a)$, otherwise it is a *failure*.

Given a state $s \in \mathcal{S}$, the probability of successful execution of action $a \in Act$ in s is defined as

$$P_{ok}(a, s) = \sum_{s' \models \mathfrak{Ann}(a)} P(s, a, s')$$

Straightforwardly, the probability of failure of a in s is $P_{fail}(a, s) = 1 - P_{ok}(a, s)$.

A macro action is successful if and only if all its components succeed along the trace.

Definition 2.6 (Success of macro actions). Let $\rho = a_1, \ldots, a_n$ be a macro action. The probability of successful execution is extended to macro actions as follows:

$$P_{ok}(\langle a_1, \ldots, a_n \rangle, s_0) = \sum_{\substack{\lambda = s_o \xrightarrow{a_1} \cdots \xrightarrow{a_n} s_n \\ s_i \models \mathfrak{Ann}(a_i)}} \prod_{i=1}^{n} P(s_{i-1}, a_i, s_i).$$

Note that P_{ok} is a probability distribution determining execution success of sequences of actions in states of E.

Now we are ready to introduce probability-based measures of successful execution of actions and programs in an environment.

Definition 2.7 (Execution success measures: actions). Let ρ be a (macro) action. The minimal certainty of successful execution of ρ in the environment E is defined as:

$$P^{-}_{ok}(\rho, E) = \min_{s \in \mathcal{S}} P_{ok}(\rho, s).$$

Similarly, we define the maximal certainty of successful execution of ρ in E:

$$P^{+}_{ok}(\rho, E) = \max_{s \in \mathcal{S}} P_{ok}(\rho, s).$$

Definition 2.8 (Execution success measures: programs). Let $\tau \in \mathcal{P}$ be an agent program over the actions of some agent A. The minimal certainty of successful execution of τ w.r.t. a state s in the environment E is the minimal probability of execution success among the individual traces induced by the program rooted in s:

$$P^{-}_{ok}(\tau, s) = \min_{\lambda \in \mathcal{T}_E(\tau, s)} P^{-}_{ok}(\rho(\lambda), s)$$

Moreover, the minimal certainty of execution success for program τ in the environment E is defined as

$$P^{-}_{ok}(\tau, E) = \min_{s \in \mathcal{S}} P^{-}_{ok}(\tau, s).$$

The notions of maximal certainty of successful execution $P^{+}_{ok}(\tau, s)$ and $P^{+}_{ok}(\tau, E)$ for an agent program are defined analogically.

2.3 Classification of environments w.r.t agent templates

In the first approach, we gauge how well the agent's basic capabilities (e.g., effectors of a robot) match the environment in which the agent is to be situated.

Definition 2.9 (Taxonomy of environments w.r.t. success of actions). Given an agent template $(Act, \mathfrak{Ann})$ in an environment E, we can distinguish several relations between the two, depending on how well the template specification meets the dynamics of the environment. Formally, we say that the matching between $(Act, \mathfrak{Ann})$ and E is:

ideal iff $P^{-}_{ok}(a, E) = 1$ for all $a \in Act$,

consistent iff $P^{-}_{ok}(a, E) > 0$ for all $a \in Act$,

strictly consistent iff it is consistent and $P^{+}_{ok}(a, E) < 1$ for all $a \in Act$,

inconsistent otherwise.

We argue, that the most interesting cases of relationship between an agent and an environment is when from the agent's perspective the environment is *strictly consistent*, or at least contains a significant *strictly consistent fragment*. For a mobile robot, a controlled indoor environment is usually ideal. Examples of (strictly) consistent agent-environment systems include e.g., outdoor robots operating in rain, snow, on icy, or sandy surfaces, or on gravel roads. From a robot's perspective, an environment is also strictly consistent when it features only imprecise sensors and/or unreliable effectors. For the agent's controller, such situation is indistinguishable from the case when the failures are truly exogenous.

The view promoted in the Definition 2.9 is rather pessimistic. Especially for agent templates based on large libraries of actions, even if one action does not match the environment, the whole agent template is seen as a mismatch. In a more complex agent program, designer might want to take into account also some failures of actions e.g., by encoding various contingencies, or refining the conditions under which actions and plans can be executed. The following refinement of the environment vs. agent classification takes the agent program as a basis for the matching.

Definition 2.10 (Environments vs. agents). Given an agent $A = (Act, \mathfrak{Ann}, \tau)$ in an environment E, we say that the matching between A and E is:

ideal iff $P^{-}_{ok}(\tau, E) = 1$,

consistent iff $P^{-}_{ok}(\tau, E) > 0$,

strictly consistent iff it is consistent and $P^{+}_{ok}(\tau, E) < 1$,

inconsistent otherwise.

Still, this notion of matching between agents and environments is not perfect. In particular, if τ includes unbounded iteration (an infinite deliberation cycle in an event-driven agent is a good example) then it is easy to see that all the environments are either ideal or inconsistent w.r.t. such an agent. This is because the notion of success relates executions to the annotations of all the actions that are going to be performed, regardless of their relevance to a larger context. In many scenarios, such context is provided by an objective, a goal, that the agent is supposed to pursue. We will formalise the concept and discuss the consequences in the next section.

3 Reasoning about temporal goals

In the previous section, we showed how "perfectness" of an environment can be classified with respect to the agent's repository of actions and/or its main algorithm. However, the classification is quite rough in the sense that it depends on *all* the actions behaving as expected (according to the provided annotations). An alternative is to consider a particular objective, and to measure how the environment reacts in the context of the objective.

Objectives that refer to execution patterns (like *achievement* of a property sometime in the future, or *maintenance* of a safety condition in all future states) can be conveniently specified in *linear time logic*. LTL [Pnueli, 1977] enables reasoning about properties of execution traces by means of temporal operators $\bigcirc$ (in the next moment) and $\mathcal{U}$ (strong until). To facilitate reasoning about finite sequences of actions and compositions thereof, we will use a version of LTL that includes the "chop" operator $\mathcal{C}$ [Rosner and Pnueli, 1986].

3.1 Temporal goals: LTL

Formally, the version of LTL used in this paper is defined as follows.

Definition 3.1 (LTL). The syntax of LTL is given by the following grammar:

$$\phi ::= \mathsf{p} \mid \neg\phi \mid \phi \wedge \phi \mid \bigcirc \phi \mid \phi \mathcal{U} \phi \mid \phi \mathcal{C} \phi.$$

Other Boolean operators (disjunction $\vee$, material implication $\rightarrow$, etc.) are defined in the usual way. The semantics is defined through the clauses below (where E is an environment and λ is an execution trace of some macro action of an agent A situated in E):

$E, \lambda \models \mathsf{p}$ iff $E, \lambda[0] \models \mathsf{p}$,

$E, \lambda \models \neg\phi$ iff $E, \lambda \not\models \phi$,

$E, \lambda \models \phi \wedge \phi'$ iff $E, \lambda \models \phi$ and $E, \lambda \models \phi'$,

$E, \lambda \models \bigcirc \phi$ iff $E, \lambda[1..\infty] \models \phi$,

$E, \lambda \models \phi \mathcal{U} \phi'$ iff there exists $i \geq 0$, such that $E, \lambda[i..\infty] \models \phi'$, and $E, \lambda[j..\infty] \models \phi$ for every $0 \leq j < i$,

$E, \lambda \models \phi \mathcal{C} \phi'$ iff there exists $i \geq 0$, such that $E, \lambda[0..i] \models \phi$ and $E, \lambda[i..\infty] \models \phi'$.

Additional operators $\Diamond$ (sometime in the future) and $\Box$ (always in the future) are defined as $\Diamond\phi \equiv \top \mathcal{U} \phi$ and $\Box\phi \equiv \neg\Diamond\neg\phi$. Note that *achievement goals* can be naturally specified with formulae of type $\Diamond\phi$, whereas a goal to maintain ϕ corresponds to the formula $\Box\phi$. Finally, we say that LTL formula ϕ is *valid in E w.r.t. a program τ* (written $E, \tau \models \phi$) iff for all $s \in \mathcal{S}$, ϕ holds on every trace $\lambda \in \mathcal{T}(\tau, s)$.

For an agent situated in an environment, we can easily define how likely the agent is to bring about a given goal.

Definition 3.2 (Probabilistic fulfilment of goals). Given an agent program τ situated in an environment $E = (\mathcal{S}, \mathcal{E}, P)$ and an LTL formula ϕ, we first define the probability space $(\mathcal{T}_E(\tau, s), \mathit{Fin}(\mathcal{T}_E(\tau, s)), pr)$ induced by the next-state transition probabilities P. In this space, elementary outcomes are runs from $\mathcal{T}_E(\tau, s)$, events are sets of runs that share the same finite prefix (i.e., ones from $\mathit{Fin}(\mathcal{T}_E(\tau, s))$), and the probability measure $pr : \mathit{Fin}(\mathcal{T}_E(\tau, s)) \rightarrow [0, 1]$ is defined as

$$pr(s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s_n) = P(s_0, a_1, s_1) \cdot \ldots \cdot P(s_{n-1}, a_n, s_n).$$

Then, the probability of *fulfilling goal ϕ from state s on* is defined through the following Lebesgue integral:

$$P_{ok}(\tau, s, \phi) \;=\; \lim_{k \to \infty} \sum_{\lambda \in T^k_\phi(\tau, s)} pr(\lambda), \text{ where}$$

$$T^k_\phi(\tau, s) \;=\; \{\lambda \in \mathit{Fin}(\mathcal{T}_E(\tau, s)) \mid E, \lambda \models \phi \text{ and } |\lambda| = k\}.$$

The interested reader is referred to [Kemeny *et al.*, 1966] for details of the construction and a proof of correctness.

Analogously to Section 2.2, the basic measure of fulfilment for a whole environment is based on the worst case analysis.

Definition 3.3 (Fulfilment of goals: P^-_{ok}, P^+_{ok}). Let $\tau \in \mathcal{P}$ be an agent program over the actions of some agent A. The minimal certainty of fulfilment of goal ϕ by τ in an environment E is the probability of fulfilment from the "worst" state in E:

$$P^-_{ok}(\tau, \phi) = \min_{s \in \mathcal{S}} P_{ok}(\tau, s, \phi).$$

The maximal certainty of fulfilment is defined analogously.

3.2 Reasoning about goals in pDCTL*

In order to reason about the expected fulfilment of goals, we propose a refinement of the branching-time logic CTL* [Emerson, 1990] with explicit quantification over program executions and probability thresholds. In the extension, $[\tau]_\zeta \phi$ reads as "agent program τ fulfils goal ϕ with probability at least ζ". The logic is called "*probabilistic Dynamic CTL**" (pDCTL*). It is a straightforward extension of "dynamic CTL*" from [Novák and Jamroga, 2009] (which in turn can be seen as a variant of Harel's process logic [Harel and Kozen, 1982]) along the lines of probabilistic temporal logics [Aziz *et al.*, 1995; Hansson and Jonsson, 1994].

Definition 3.4 (pDCTL*). The syntax of pDCTL* is defined as an extension of LTL by the following grammar:

$$\begin{aligned} \theta &::= \mathsf{p} \mid \neg\theta \mid \theta \wedge \theta \mid [\tau]_\zeta \phi \\ \phi &::= \theta \mid \neg\phi \mid \phi \wedge \phi \mid \bigcirc \phi \mid \phi \mathcal{U} \phi \mid \phi \mathcal{C} \phi \end{aligned}$$

where p is a propositional formula from $\mathcal{L}$, and τ is an agent program.

The semantics of pDCTL* extends that of LTL by the clauses below:

$E, \lambda \models \theta$ iff $E, \lambda[0] \models \theta$,

$E, s \models \mathsf{p}$ iff $s \models \mathsf{p}$,

$E, s \models \neg\theta$ iff $E, s \not\models \theta$,

$E, s \models \theta_1 \wedge \theta_2$ iff $E, s \models \theta_1$ and $E, s \models \theta_2$,

$E, s \models [\tau]_\zeta \phi$ iff the probability of fulfilling ϕ by τ from s on is at least ζ, i.e., $P_{ok}(\tau, s, \phi) \geq \zeta$.

pDCTL* formula θ is *valid in E* (written $E \models \theta$) iff $E, s \models \theta$ for every state s of E. Finally, ψ is a semantic consequence of ϕ (written: $\phi \Rightarrow \psi$) iff for every environment E, $E \models \phi$ implies $E \models \psi$.

The following proposition is straightforward and shows a strong relationship between formulae of pDCTL* and the measures of goal fulfilment introduced in Section 3.1.

Proposition 3.5. *$E \models [\tau]_\zeta \phi$ iff $P^-_{ok}(\tau, \phi) \geq \zeta$.*

Additionally, we define $[\tau]\phi$ as $[\tau]_1 \phi$. It is easy to see that the semantics of $[\tau]\phi$ in pDCTL* and DCTL* coincide. Moreover, pDCTL* validity corresponds to LTL validity w.r.t. a program (the proofs are straightforward).

Proposition 3.6. *For every environment E, program τ and DCTL* formula ϕ, we have $E, s \models_{\mathit{DCTL}^*} [\tau]\phi$ iff $E, s \models_{\mathit{pDCTL}^*} [\tau]\phi$.*

Proposition 3.7. *For every environment E, program τ and LTL formula ϕ, we have: $E, \lambda \models_{\mathit{LTL}} \phi$ for every $\lambda \in \mathcal{T}_E(\tau)$ iff $E \models_{\mathit{pDCTL}^*} [\tau]\phi$.*

We note that the operator dual to $[\tau]_\zeta$ has almost the same meaning, except of being underpinned by strict, instead of weak inequality.

Proposition 3.8. *Let $\langle\tau\rangle_\zeta \phi \equiv \neg[\tau]_{1-\zeta}\neg\phi$.*
Then $E, s \models \langle\tau\rangle_\zeta \phi$ iff $P_{ok}(\tau, s, \phi) > \zeta$.

Proof. $E, s \models \langle\tau\rangle_\zeta \phi$ iff $E, s \not\models [\tau]_{1-\zeta}\neg\phi$ iff $P_{ok}(\tau, s, \neg\phi) < 1 - \zeta$ iff $1 - P_{ok}(\tau, s, \phi) < 1 - \zeta$ iff $P_{ok}(\tau, s, \phi) > \zeta$. □

Thus, pDCTL* allows also to refer to the exact probability of fulfilment by $[\tau]_{=\zeta}\phi \equiv [\tau]_\zeta \phi \wedge \neg\langle\tau\rangle_\zeta \phi$.

3.3 Classifying environments w.r.t. goals

Assuming a particular temporal goal allows for a finer-grained taxonomy of environments than we proposed in Subsection 2.3.

Definition 3.9 (Taxonomy of environments w.r.t. goal fulfilment). Given an agent program τ and a goal ϕ, we can distinguish between several types of environments according to the probability of fulfilment of ϕ by τ. We say that E is:

ideal iff $P^{-}_{ok}(\tau, \phi) = 1$ (alternative formulation: $E \models [\tau]\phi$),

manageable iff $P^{-}_{ok}(\tau, \phi) \geq \zeta$ for some $\zeta > 0$ (equivalently: $E \models [\tau]_{\zeta}\phi$ for some $\zeta > 0$). Note that, for finite environments, it is equivalent to $P^{-}_{ok}(\tau, \phi) > 0$ (and $E \models \langle\tau\rangle_0\phi$);

strictly manageable iff the environment is manageable and $P^{+}_{ok}(\tau, \phi) < 1$,

hopeless $P^{-}_{ok}(\tau, \phi) = 0$ (or equivalently: $E \models [\tau]_{=0}\phi$).

Consider the mobile robot from Subsection 2.3. Aiming to move along a pre-defined path, an outdoor environment is manageable for the robot, but a crowded place could even turn hopeless due to continuously moving people interfering with the robot. In effect, such a classification can give a good formal hint on limitations of usability of the robot.

For agent programs inducing only finite traces, we can distinguish environments w.r.t. the degree of iteration needed for the program to achieve a goal.

Definition 3.10 (Taxonomy of environments cont.). Given a program τ such that $|\lambda| < \infty$ for every $\lambda \in \mathcal{T}_E(\tau)$, we can distinguish between several additional types of environments according to the certainty of fulfilment of ϕ by τ in E:

iteratively manageable w.r.t. some $\zeta > 0$ iff there exists $k \in \mathbb{N}$ s.t. $E \models [\tau^k]_{\zeta}\phi$,

completely hopeless iff $E \models [\tau^*]_{=0}\phi$.

Additionally, we define the following limit cases:

- $E \models [\tau^*]\phi$, but there is no $k \in \mathbb{N} : E \models [\tau^k]\phi$,
- there exists $\zeta > 0$, s.t. $E \models [\tau^*]_{\zeta}\phi$, but there is no $k \in \mathbb{N} : E \models [\tau^k]_{\zeta}\phi$, and
- for all $k \in \mathbb{N}$, there exists $\zeta > 0$, s.t. $[\tau^k]_{\zeta}\phi$, however $[\tau^*]_{=0}\phi$.

This line of thought can be elaborated upon further by considering special types of programs. E.g., $\tau = a_1 \cup \cdots \cup a_n$, where $Act = \{a_1, \ldots, a_n\}$. Now $\mathcal{T}_E(\tau)$ include all the possible plans which can be constructed out of the actions in Act.

4 Program composition vs. goals

In this section, we turn our interest to the the following question: *how program composition affects the certainty level of goal fulfilment by agents in dynamic environments?* In particular, we are interested in how likelihood of fulfilling general maintenance and achievement goals (involving the $\Box$ and $\Diamond$ modalities respectively) relates to the way how the corresponding programs are constructed. Throughout this section, we implicitly assume that environments are strictly manageable w.r.t. programs and goals in consideration.

The following theorem articulates the intuition, that concatenation of programs leads to a strict decrease of the certainty of goal fulfilment by the joined program.

Theorem 4.1. *If τ_1, τ_2 are programs, s.t., $E \models [\tau_1]_{=\zeta}\phi_1 \wedge E \models [\tau_2]_{\zeta}\phi_2$ and at the same time $E \models \neg[\tau_1]_{\zeta}\phi_2$, then $E \models \neg[\tau_1; \tau_2]_{\zeta}\phi_1\mathcal{C}\phi_2$.*

Proof sketch. The idea behind the proof is that due to the independence of τ_1 and τ_2 w.r.t. ϕ_2 ($E \models \neg[\tau_1]_{\zeta}\phi_2$), to establish $P_{ok}((\tau_1; \tau_2), s_o, \phi_1\mathcal{C}\phi_2)$, we must consider traces induced by τ_1 *prolonged* by traces induced by τ_2. By approximating the sum of probabilities for sets of such prolongations rooted in terminal states of traces of τ_1 by $\xi = P^{+}_{ok}(\tau_2, E, \phi_2)$, we arrive to the inequality $P_{ok}((\tau_1; \tau_2), s_o, \phi_1\mathcal{C}\phi_2) \leq \xi \cdot \zeta$. Since the environment is only *strictly manageable* w.r.t. τ_2 and ϕ_2, by necessity $\xi < 1$, hence $\xi \cdot \zeta < \zeta$, i.e., the minimal certainty of execution success of $\tau_1; \tau_2$ w.r.t. the goal $\phi_1\mathcal{C}\phi_2$ and the environment E is strictly less than the original ζ.

□

In Theorem 4.1 we used the $\mathcal{C}$ operator for joining the goal formulae. Concatenation and further iteration of the same program leads to the following property of maintenance goals involving $\Box$ modality on ever longer traces.

Corollary 4.2. *Given τ is a program, s.t., $E \models [\tau]_{=\zeta}\phi$, then $E \models \neg[\tau^k]_{\zeta}\Box\phi$.*

Let's assume that the behaviour of an agent conforms to some performance quality measure when continuously operating at a particular level of probability of fulfilment of its goal by the agent program in a particular environment. We can formulate the following informal consequence of Theorem 4.1 and Corollary 4.2: *in dynamic environments, behaviour specifications involving temporal maintenance goals of the form $\Box\phi$ are undesirable*. The main reason behind this conjecture is, that to ensure satisfaction of strict maintenance goals, such as $\Box$p, necessarily programs have to be joined by sequential composition, what leads to decrease of the level of certainty of execution success w.r.t. the envisaged goal. In fact, by subsequently prolonging the induced traces by sequential program composition, the level of certainty eventually eventually drops below the minimal required performance threshold.

Let's turn our attention to achievement goals, i.e., those which involve the $\Diamond$ modality.

Theorem 4.3. *Given τ is a program, s.t., $E \models [\tau]_{\zeta}\phi$, then $E \models [\tau^k]_{\zeta}\Diamond\phi$ for every $k > 0$.*

Proof sketch. The idea behind the proof is that by inductively iterating τ^k, the traces which satisfy $\Diamond\phi$ are i) those on which the goal was already satisfied for lower k's and prolonged by any trace regardless whether it satisfies $\Diamond\phi$, or not; plus ii) all the traces which did not satisfy $\Diamond\phi$ for lower k's prolonged by the traces induced by τ which satisfy $\Diamond\phi$. In result, for every k, the number of traces which satisfy the formula $\Diamond\phi$ is not decreasing, nor is the ratio to those which do not satisfy the goal. □

In fact, we hypothesise that if $E \models [\tau]_{=\zeta}\phi$, then we should have $E \models [\tau^k]_{=\xi}\Diamond\phi$, where $\xi > \zeta$ and $k > 0$.

5 Program perfection vs. fulfilment

Agent programs in dynamic environments can perform in an imperfect manner w.r.t. their goals due to two reasons. Firstly, it can be the environment by its dynamics which can cause the program not to fulfil its goal. Secondly, it can be the implementation of the program itself, which results in execution traces along which the goal is not fulfilled. In the following, we look at the probability of goal fulfilment *had the implementation of the agent's capabilities been ideal*. This allows us to measure the impact of the imperfection in the implementation on the fulfilment likelihood.

Definition 5.1 (Idealised environment). The idealised variant of an environment $E = (\mathcal{S}, \mathcal{E}, P)$ with respect to an agent template $(Act, \mathfrak{Ann})$ is defined as the environment $E^{\mathfrak{Ann}} = (\mathcal{S}, \mathcal{E}, P^{\mathfrak{Ann}})$ where the new probabilistic transition relation $P^{\mathfrak{Ann}}$ is as follows:

$$P^{\mathfrak{Ann}}(s,a,s') = \begin{cases} 0 & \text{iff } s' \not\models \mathfrak{Ann}(a) \\ \frac{P(s,a,s')}{\sum_{s'' \models \mathfrak{Ann}(a)} P(s,a,s'')} & \text{otherwise} \end{cases}$$

That is, we take the transition relation P in E and remove all the transitions that do not conform with $\mathfrak{Ann}$ (normalising P afterwards). Note that this scheme can be seen as a probabilistic version of *model update* similar to the one in *Public Announcement Logic* [Baltag, 2002].

The idealised probability of fulfilling a goal ϕ is the probability of fulfilment of ϕ under the assumption that the agent's actions will behave as specified.

Definition 5.2 (Idealised fulfilment of ϕ). $P^{\mathfrak{Ann}}_{ok}(\tau, s, \phi)$ (resp. $P^{\mathfrak{Ann}}_{ok}(\tau, \phi)$) in an environment E is simply defined as $P_{ok}(\tau, s, \phi)$ (resp. $P^{-}_{ok}(\tau, \phi)$) in the idealised environment $E^{\mathfrak{Ann}}$.

Now we can measure the impact of imperfect implementation as the difference in certainty of fulfilment between the ideal and the real case:

Definition 5.3 (Impact metric Imp). Given environment $E = (\mathcal{S}, \mathcal{E}, P)$, agent $A = (Act, \mathfrak{Ann}, \tau)$, and goal ϕ, we define

$$Imp(\phi) = P^{\mathfrak{Ann}}_{ok}(\tau, \phi) - P^{-}_{ok}(\tau, \phi).$$

$Imp(\phi)$ indicates how much the imperfections in implementation of the agent's capabilities influence the chances for reaching the goal ϕ. In this context, it can be interpreted in two ways:

- Imp indicates *how inaccurate the agent implementation is with respect to the given goal*. In contrast, $P^{-}_{ok}(\tau, \phi)$ shows only the general inaccuracy of the agent implementation.
- Assuming that the designer has some control over the deployment of the agent in the environment, Imp can be understood as a measure of *how much they should improve the implementation of the agent's capabilities in order to get the objective met*.

6 Final remarks

The discussion in the last two sections only scratched the surface of what the introduced conceptual framework allows to rigorously investigate. One of interesting examples is the notion of a *planning horizon*. I.e., given a set of agent's capabilities, a goal and a specific level of certainty with which it should be satisfied, we can ask: *what is the maximal plan length beyond which the probability of fulfilling the goal decreases below the required quality threshold?* An estimate of the planning horizon could have an impact on parametrisation of planning algorithms used in dynamic environments, or can lead to constraints on lengths of plans in a library of a reactive planner. Similarly, deeper insights into how program composition influences the impact metric of the resulting programs have a potential to influence formulation of useful code patterns for agent programming, such as those introduced in [Novák and Jamroga, 2009]. We leave these interesting issues for further work along this line of research.

References

[Aziz *et al.*, 1995] A. Aziz, V. Singhal, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. It usually works: The temporal logic of stochastic systems. In *Proceedings of CAV*, volume 939 of *LNCS*, pages 155–165, 1995.

[Baltag, 2002] A. Baltag. A logic for suspicious players. *Bulletin of Economic Research*, 54(1):1–46, 2002.

[Bellman, 1957] R. Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, 6:679–684, 1957.

[Bordini *et al.*, 2006] Rafael H. Bordini, Lars Braubach, Mehdi Dastani, Amal El Fallah Seghrouchni, Jorge J. Gomez-Sanz, João Leite, Gregory O'Hare, Alexander Pokahr, and Alessandro Ricci. A survey of programming languages and platforms for multi-agent systems. *Informatica*, 30:33–44, 2006.

[Emerson, 1990] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier Science Publishers, 1990.

[Hansson and Jonsson, 1994] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[Harel and Kozen, 1982] D. Harel and D. Kozen. Process logic: Expressiveness, decidability, completeness. *Journal of Computer and System Sciences*, 25(2):144–170, 1982.

[Kemeny *et al.*, 1966] J. G. Kemeny, L. J. Snell, and A. W. Knapp. *Denumerable Markov Chains*. Van Nostrand, 1966.

[Novák and Jamroga, 2009] P. Novák and W. Jamroga. Code patterns for agent oriented programming. In *Proceedings of AAMAS'09*, pages 105–112, 2009.

[Pnueli, 1977] A. Pnueli. The temporal logic of programs. In *Proceedings of FOCS*, pages 46–57, 1977.

[Rosner and Pnueli, 1986] R. Rosner and A. Pnueli. A choppy logic. In *Proceedings of LICS*, pages 306–313, 1986.