

Plans in Cooperation Logics: a modular approach

Jelle Gerbrandy and Luigi Sauro

Dipartimento di Informatica, Università di Torino

and

Dipartimento di Scienze Fisiche, Università “Federico II” di Napoli

jelle@gerbrandy.com, sauro@na.infn.it

Abstract

We develop a logical framework to model how agents can cooperate in order to ensure a certain state of affairs. We extend previous work on this issue by adding an explicit planning language to the object language. The resulting logical system combines reasoning about agents and groups of agents, their plans, and the effect of those plans, and becomes quite complex. To reduce this complexity, we show how the model can be defined by two separate modules: one module describes how plans affect the environment, another module describes how a group of agents can cooperate in order to execute a plan. We can combine these models to obtain a new model to reason about the ability of groups of agents to use their plans to obtain certain results. What we gain with respect to earlier systems for reasoning about group ability, such as ATL, apart from handling different problems separately, is the possibility to explicitly describe by means of which plans agents obtain some outcomes.

1 Introduction

In recent years, several logic-based approaches have focused on how groups of agents can cooperate in order to ensure a certain state of affairs [Pauly, 2002; van der Hoek and Wooldridge, 2003; Wooldridge and Jennings, 1994]. Typically, these logics have operators that express *group ability* to construct sentences $\langle\langle Q \rangle\rangle\phi$, which are to be read as “the agents in Q have a way to ensure that ϕ , and the other agents cannot obstruct them.”

One of the best known formalisms in this area is Alternating-Time Temporal Logic [Alur *et al.*, 2002] which extends Computation Tree Logic (CTL) with cooperation operators. In this context, $\langle\langle Q \rangle\rangle\phi$ can be read as: the agents in Q have a way of ensuring that the set of possible runs of the system satisfies the CTL path-formula ϕ .

ATL does not contain the means to explicitly express the plans that the agents in Q have to actually perform in order to obtain the desired results. There is some work on formulating plans in the context of ATL and incorporating these plans in the object language. In [van der Hoek *et al.*, 2005] the

authors provide a variant of ATL in which strategies are represented explicitly – but they consider memory-less strategies only, thereby weakening the semantics of ATL considerably.

In this article, we provide the agents with a more standard imperative-like programming language – we will use an action language which involves also *while - do -* statements. This language can be used to define explicit strategies for enforcing particular states of affairs.

The resulting model is quite complex: it combines reasoning about agents, actions, plans and their effects. To make this complexity somewhat more manageable, we split the definitions, and indeed the whole model, into two parts. Our approach builds directly on the work in [Sauro *et al.*, 2006]: we define two *modules*. First of all, there is an *environment module* that is used to reason about plans and their effects, but that does not contain any references to agents. Independently, we define an *agents module* to describe which plans a group of agents can execute, but which abstracts from the effects of these plans. Finally, we combine the two models in a single model.

The paper is built up as follows. In section 2 we develop the environment module. In particular, we use an imperative-like language with concurrent actions, concatenation, *if - then -* and *while - do -* statements to describe plans, we define a Process Logic [Harel and Singerman, 1999; Henriksen and Thiagarajan, 1999; Chen and De Giacomo, 1999] to explicitly describe which computation traces correspond to a given plan. To reason about the *effects* of executing these plans, we use CTL as a starting point, i.e. we formalize temporal properties of collections of possible computation traces.

In the *agents module* - section 3 - we model agents with their skills and we formalize a logic for reasoning about the ability of groups of agents to execute a plan. Essentially, this logic is based on Coalition Logic [Pauly, 2002], where the states of affairs that groups of agents can enforce by cooperating are interpreted as set of concurrent actions.

By combining these models into a single one, in section 4 we obtain a system for reasoning about agents, their plans, and the kind of results that groups of agents can guarantee when executing these plans.

In section 5 we consider a real scenario which can be formalized in our framework, conclusions end the paper.

2 Environment module

One general way to model actions and their effects is by using a Labeled Transition System (LTS) [Harel *et al.*, 1984]. A LTS is a directed graph in which each vertex corresponds to a state of the environment whereas edges represent possible transitions between states. States are labeled with the set of propositional variables that are true at that state. The label of a transition describes which actions have to be executed to trigger that transition.

Usually a transition label consists of single action a . However, in multiagent systems, and in particular in the kind of models that are the basis of ATL, agents may execute different actions at the same time. A natural way to model such *concurrent actions* is by labeling the transitions not with single actions, but with *sets* of actions. Intuitively, a transition labeled with a set of actions A represents the effect of executing exactly those actions concurrently, and no others. This approach to deal with concurrency is similar to that taken in [Giordano *et al.*, 1998].

Definition 1 (Environment module) *An environment module \mathcal{M}^{env} is a tuple $\langle S, Ac, (\rightarrow)_{A \subseteq Ac}, P, \pi \rangle$ where S is a finite set of states and Ac a finite set of actions. For each $A \subseteq Ac$, \rightarrow_A is a relation over S . P is a set of propositional variables and π is an function that assigns to each state a subset of P .*

We will refer to a set of actions $A \subseteq Ac$ as a *concurrent action*. Given the way the definitions are set up, a concurrent action A can be *non-deterministic*, in the sense that the outcome of performing a set of actions may not always determine a unique result. Executing an action may also be impossible, in the sense that there may not be a transition corresponding to that action.

As said in the introduction, our goal is to develop a system to reason about plans. One natural choice is to take Propositional Dynamic Logic (PDL) as our starting point (as in, e.g. [Henriksen and Thiagarajan, 1999]) – a language that contains tests $\phi?$, concatenation $;$ and iteration $*$. However, that language does not serve our purposes well for two reasons. PDL works fine for reasoning about pre- and post-conditions of programs, but it does not contain the means for reasoning about infinite traces – about plans that do not halt. For example, the standard representation of $\text{while } \phi \text{ do } \Gamma$ statements in PDL as $(?\phi; \Gamma)^*$; $\neg\phi$ captures only those execution traces in which the program halts. This is fine for PDL, but as we will want to include a rich temporal language of CTL in which properties of non-halting traces can be expressed as well, we need to have a richer action language. A second point is that PDL does not allow us to express *negative* actions such as *refrain from executing the action a* ([Broersen, 2003]).

Instead, we choose a simple imperative programming language for our planning language

$$\Gamma := \alpha \mid \Gamma_1; \Gamma_2 \mid \text{if } \phi \text{ then } \Gamma \mid \text{while } \phi \text{ do } \Gamma$$

where α is an action expression (a Boolean formula over Ac) and ϕ is a state expression (a Boolean formula over P).

So, in this action language, we can formulate plans such as:

$$\text{while } p \text{ do } a \wedge \neg b$$

which denotes the plan of doing a , but not b , while p is true.

Given an environment module \mathcal{M}^{env} , the rules in definition 2 provide a calculus to derive the possible runs of the system when executing a plan Γ starting from a state s_0 . This kind of definition is similar to the one in [Chen and De Giacomo, 1999; ?] The rules define transitions of the form $(\sigma, s) \xrightarrow{A} (\sigma', s')$, where σ and σ' are (possibly empty) sequences of plan expressions to be executed, s and s' are states in the model, and A is a set of actions. Intuitively, $(\sigma, s) \xrightarrow{A} (\sigma', s')$ means that executing the first plan expression on the sequence σ in state s may bring us, via a A -transition in the model, to a state s' in which the sequence σ' remains to be executed. In the rule set, we write $\Gamma \cdot \sigma$ for the sequence that starts with the plan Γ and has the (possibly empty) sequence σ as its tail.

For example, rule (1) says that when the head of the sequence is an action expression α and there is an A -transition in the model for some A of type α , the system executes this transition and in the new state needs to execute the tail of the sequence. Rule (4), instead, says that in a state where ϕ is true and if ϕ then Γ is the first of the sequence, the first execution step will be the same as the first execution step of Γ .

Definition 2 (Plan execution rules) *Let \mathcal{M}^{env} be the tuple $\langle S, Ac, (\rightarrow)_{A \subseteq Ac}, P, \pi \rangle$. We define:*

1. $(\alpha \cdot \sigma, s) \xrightarrow{A} (\sigma, s')$ if $A \models \alpha$ and $s \rightarrow_A s'$.
2. $(\Gamma_1; \Gamma_2 \cdot \sigma, s) \xrightarrow{A} (\sigma', s')$ if $(\Gamma_1 \cdot \Gamma_2 \cdot \sigma, s) \xrightarrow{A} (\sigma', s')$
3. $(\text{if } \phi \text{ then } \Gamma \cdot \sigma, s) \xrightarrow{A} (\sigma', s')$ if $(\sigma, s) \xrightarrow{A} (\sigma', s')$ and $\pi(s) \not\models \phi$
4. $(\text{if } \phi \text{ then } \Gamma \cdot \sigma, s) \xrightarrow{A} (\sigma', s')$ if $(\Gamma \cdot \sigma, s) \xrightarrow{A} (\sigma', s')$ and $\pi(s) \models \phi$
5. $(\text{while } \phi \text{ do } \Gamma \cdot \sigma, s) \xrightarrow{A} (\sigma', s')$ if $(\sigma, s) \xrightarrow{A} (\sigma', s')$ and $\pi(s) \not\models \phi$
6. $(\text{while } \phi \text{ do } \Gamma \cdot \sigma, s) \xrightarrow{A} (\sigma' \cdot \text{while } \phi \text{ do } \Gamma \cdot \sigma, s')$ if $(\Gamma, s) \xrightarrow{A} (\sigma', s')$ and $\pi(s) \models \phi$

where \models is the standard satisfaction relation from propositional logic.

Given an environment \mathcal{M}^{env} , a state s_0 and a plan Γ , the rules in definition 2 give us a way of generating a new model that represents the possible execution traces of the plan Γ in the model \mathcal{M}^{env} . This works by taking as our new initial state the pair (Γ, s_0) , and using the definitions to generate possible transitions from (Γ, s_0) to new states (σ, s') . Repeating this process gives us a new environment model that represents the all possible execution traces of Γ when it is executed starting in s_0 in the original model.

Definition 3 (Plan execution) *Given an environment module $\mathcal{M}^{\text{env}} = \langle S, Ac, (\rightarrow)_{A \subseteq Ac}, P, \pi \rangle$, a state $s_0 \in S$ and a plan Γ , $\mathcal{M}^{\text{env}}[\Gamma, s_0]$ is an environment module $\langle S', Ac, (\rightarrow)_{A \subseteq Ac}, P, \pi' \rangle$ such that*

1. S' is the smallest set that contains (Γ, s_0) for which it holds that if $(\sigma, s) \in S'$ and $(\sigma, s) \xrightarrow{A} (\sigma', s')$ is derivable, then $(\sigma', s') \in S'$.

2. $(\sigma, s) \rightarrow_A (\sigma', s')$ iff $(\sigma, s) \xrightarrow{A} (\sigma', s')$ is derivable.
3. $\pi'((\sigma, s)) = \pi(s)$

Given a finite environment, definition 3 generates a new environment (which is finite as well) that describes the possible evolutions of the environment when, starting from a state s , a plan Γ is executed. Even if $\mathcal{M}^{\text{env}}[\Gamma, s]$ may be much larger than the original model \mathcal{M}^{env} , the set of paths in $\mathcal{M}^{\text{env}}[\Gamma, s]$ is always a subset of the paths in \mathcal{M}^{env} – there is a precise sense in which a plan *restricts* the way that the environment may evolve.

To reason about properties of plans (such as liveness or certain reachability conditions) we borrow the syntax of CTL. This results in the following extension of that language: The syntax of the environment logic is:

$$\phi, \psi := p \mid \phi \wedge \psi \mid \neg\phi \mid AX\phi \mid A\phi U\psi \mid E\phi U\psi \mid [\Gamma]\phi$$

where Γ is a plan.

Informally, the symbol A means *in all possible evolutions of the system*, the symbol E means *in at least a possible evolution of the system*. The symbols X and U denote properties of paths of states: $X\phi$ means *in the next state ϕ holds* and $\phi U\psi$ means *ϕ holds until ψ will hold*. The new formulas of the type $[\Gamma]\phi$ means that *during the execution of the plan Γ , ϕ holds*.

In the definition below, given a model \mathcal{M}^{env} , we denote with \rightarrow the union of all \rightarrow_A and $[s_0]$ is the set of all maximal paths starting in s_0 ; more precisely, $[s_0]$ is the set of finite and infinite sequences $\mathbf{s} = (s_0, s_1 \dots s_n \dots)$ such $s_i \rightarrow_A s_{i+1}$ for each i , and if \mathbf{s} is finite with s_n as its last element, then there is no s' and A such that $s_n \rightarrow_A s'$.

Definition 4 (Semantics of environment logic)

1. $\mathcal{M}^{\text{env}}, s \models^e p$ iff $p \in \pi(s)$.
2. $\mathcal{M}^{\text{env}}, s \models^e \phi \wedge \psi$ iff $\mathcal{M}^{\text{env}}, s \models^e \phi$ and $\mathcal{M}^{\text{env}}, s \models^e \psi$.
3. $\mathcal{M}^{\text{env}}, s \models^e \neg\phi$ iff $\mathcal{M}^{\text{env}}, s \not\models^e \phi$.
4. $\mathcal{M}^{\text{env}}, s \models^e AX\phi$ iff for all s' such that $s \rightarrow s'$, $\mathcal{M}^{\text{env}}, s' \models^e \phi$.
5. $\mathcal{M}^{\text{env}}, s \models^e A\phi U\psi$ iff for all sequences $\mathbf{s} = s_0 \dots$ in $[s]$ there exists a state s_i in \mathbf{s} such that for all $0 \leq j < i$, $\mathcal{M}^{\text{env}}, s_j \models^e \phi$ and $\mathcal{M}^{\text{env}}, s_i \models^e \psi$.
6. $\mathcal{M}^{\text{env}}, s \models^e E\phi U\psi$ iff there exists a sequence $\mathbf{s} = s_0 \dots$ in $[s]$ and a state s_i in \mathbf{s} such that for all $0 \leq j < i$, $\mathcal{M}^{\text{env}}, s_j \models^e \phi$ and $\mathcal{M}^{\text{env}}, s_i \models^e \psi$.
7. $\mathcal{M}^{\text{env}}, s \models^e [\Gamma]\phi$ iff $\mathcal{M}^{\text{env}}[s, \Gamma], (\Gamma, s) \models^e \phi$.

The following abbreviations are often used: $EX\phi := \neg AX\neg\phi$, $EF\phi := E\top U\phi$, $AF\phi := A\top U\phi$, $EG\phi := \neg AF\neg\phi$, $AG\phi := \neg EF\neg\phi$.

A formula of the type $[\Gamma]\phi$ checks the truth of ϕ in the model of the execution of Γ . In some cases we want to verify that Γ makes a formula true not only during its execution but also in all the possible evolutions after Γ has halted. This can be expressed by following up Γ with the ‘trivial plan’ while $(p \vee \neg p)$ do $(a \vee \neg a)$ (keep on executing an arbitrary action).

To illustrate the logic, we quickly observe some facts. First of all, plans can precisely describe which concurrent actions should be performed in which order, but they can also be highly non-deterministic. At its most extreme, the ‘plan’ $a \vee \neg a$ denotes the set of all transitions – it is the plan of non-deterministically doing anything at all. Such expressions may not be very useful as a plan, but it is useful to have them in the logic. For example, consider α ; while \top do $(a \vee \neg a)$. This describes the plan of first doing ‘ α ’, then continue to repeat any combination of actions. The execution model of this plan in a state s_0 of an environment \mathcal{M}^{env} consists of a model in which only α -actions can be executing, and each possible result of such an action brings us in an isomorphic copy of the state we would have reached in the original model. This means that the ‘classical’ modality $\langle \alpha \rangle \phi$ – there exists an α -successor in which ϕ is true – can be expressed as in our language as well, by the sentence $[\alpha$; while $p \vee \neg p$ do $a \vee \neg a$] $EX\phi$.

The non-determinism of plans is essential for the use we make of our planning language when we talk about agents in the next sections. For example, we can provide an agent that, say, can only do either a or b , with the plan $a \wedge \neg b$. Even if this plan describes a precise course of action for our agent, in the wider environment the effect of executing this plan may depend on the actions of the other agents – and therefore the plan cannot denote a single execution trace.

3 Agents module

The environment module enables us to describe how a system evolves when a plan is executed. Now we want to formalize which groups of agents are able to execute a given plan. We do this independently from the environment model – i.e. in this module, we ignore the question of what happens with the states in an environment when a plan is executed as much as possible, and we concentrate exclusively on the ability of agents to execute a plan.

Definition 5 (Agents module) *Given a finite set of agents Ag and a finite set of actions Ac , an agents module is a tuple $\langle Ag, Ac, \text{act} \rangle$, where act is a function $\mathcal{P}(Ag) \mapsto \mathcal{P}(\mathcal{P}(Ac))$ that assigns to each set of agents Q a set $\{A_1, \dots, A_n\}$ of sets of actions. The function act satisfies the following properties:*

1. $\text{act}(\emptyset) = \{\emptyset\}$.
2. for all disjoint sets of agents Q_1 and Q_2 , if $A_1 \in \text{act}(Q_1)$ and $A_2 \in \text{act}(Q_2)$, then $A_1 \cup A_2 \in \text{act}(Q_1 \cup Q_2)$.

So, an agents module assigns to each set of agents the set of concurrent actions that they can execute.

Following [Peleg, 1998] we define an effectivity function for each group Q . Such a function gives us, for each set of agents, the sets of actions that it is capable of executing concurrently.

Definition 6 (Effectivity function for actions) *Let be \mathbf{A} be a subset of $\mathcal{P}(Ac)$, say that \mathbf{A} is attainable by the set of agents Q if and only if there exists a set of actions $A \in \text{act}(Q)$ such that for all $B \in \text{act}(Ag \setminus Q)$, $A \cup B \in \mathbf{A}$.*

The effectivity function E assigns to each group of agents Q the set of \mathbf{A} attainable by Q .

Pauly [Pauly, 2002] focuses on a particular class of effectivity functions: those he calls ‘playable’ (because they correspond to the effectivity functions of a strategic game). Playable effectivity functions satisfy the following properties:

outcome monotonicity: if $\mathbf{A} \in E(Q)$ and $\mathbf{A}' \supseteq \mathbf{A}$, then $\mathbf{A}' \in E(Q)$.

superadditivity: for all disjoint sets of agents Q_1 and Q_2 , if $\mathbf{A}_1 \in E(Q_1)$ and $\mathbf{A}_2 \in E(Q_2)$, then $\mathbf{A}_1 \cap \mathbf{A}_2 \in E(Q_1 \cup Q_2)$.

Ag-maximality: for all \mathbf{A} if $(\mathcal{P}(Ac) \setminus \mathbf{A}) \notin E(\emptyset)$, then $\mathbf{A} \in E(Ag)$.

accessibility: for all set of agents Q , $\mathcal{P}(Ac) \in E(Q)$ and $\emptyset \notin E(Q)$.

The following theorem shows that Definition 6 defines playable effectivity functions.

Theorem 1 *Given an agents module $\langle Ag, Ac, act \rangle$, the corresponding effectivity function E , as defined in definition 6, is playable.*

A sketch of the proof for this theorem is in the appendix.

Given a concurrent action α , we say that a set of agents is effective for α meaning that $\{A \mid A \models \alpha\} \in E(Q)$.

The language of agents logic consists of boolean combinations of sentences of the form $\langle\langle Q \rangle\rangle \Gamma$ – intuitively *the set of agents Q is capable to enforce the plan Γ* . Formally, agent logic formulas have the syntactic form:

$$\chi := \langle\langle Q \rangle\rangle \Gamma \mid \neg \chi \mid \chi_1 \wedge \chi_2$$

where Q denotes a set of agents and Γ denotes a plan. We follow Pauly’s Coalition Logic in order to describe the capabilities of agents in the case of concurrent actions α : a set of agents Q is capable to execute a concurrent action α if Q is effective for α . However, we need to take into account also programming constructs which are not considered in Coalition Logic. The assumption that the capabilities of agents do not depend on the state of the environment enables us to handle concatenation straightforwardly: a set of agents is capable to execute a plan $\Gamma_1; \Gamma_2$ just in the case it is capable to execute Γ_1 and Γ_2 . It remains to consider the if _ then _ and while _ do _ statements, also here we consider the simplest way to extend Coalition Logic semantics by assuming total observability for the agents, i.e. each agent is capable to verify if a formula ϕ is true in a certain state of the environment. This way, a set of agents is capable to execute a plan if ϕ then Γ or while ϕ do Γ just in the case it is capable to execute Γ .

Definition 7 (Semantics of Agent logics)

1. $\mathcal{M}^{\text{agt}} \models^a \langle\langle Q \rangle\rangle \alpha$ iff $\alpha^{\mathcal{M}^{\text{agt}}} \in E(Q)$, where $\alpha^{\mathcal{M}^{\text{agt}}} = \{A \mid A \models \alpha\}$ and E derives from Definition 6.
2. $\mathcal{M}^{\text{agt}} \models^a \langle\langle Q \rangle\rangle \Gamma_1; \Gamma_2$ iff $\mathcal{M}^{\text{agt}} \models^a \langle\langle Q \rangle\rangle \Gamma_1$ and $\mathcal{M}^{\text{agt}} \models^a \langle\langle Q \rangle\rangle \Gamma_2$.
3. $\mathcal{M}^{\text{agt}} \models^a \langle\langle Q \rangle\rangle \text{if } \phi \text{ then } \Gamma$ iff $\mathcal{M}^{\text{agt}} \models^a \langle\langle Q \rangle\rangle \Gamma$.
4. $\mathcal{M}^{\text{agt}} \models^a \langle\langle Q \rangle\rangle \text{while } \phi \text{ do } \Gamma$ iff $\mathcal{M}^{\text{agt}} \models^a \langle\langle Q \rangle\rangle \Gamma$.
5. $\mathcal{M}^{\text{agt}} \models^a \neg \chi$ iff $\mathcal{M}^{\text{agt}} \not\models^a \chi$.

6. $\mathcal{M}^{\text{agt}} \models^a \chi_1 \wedge \chi_2$ iff $\mathcal{M}^{\text{agt}} \models^a \chi_1$ and $\mathcal{M}^{\text{agt}} \models^a \chi_2$.

Due to Theorem 1, the relevant axioms of Coalition Logic - regarding superadditivity, outcome monotonicity, Ag-maximality, etc. - are also valid in our logic. Formally, valid formulas of agent logic are:

1. All axioms propositional logic.
2. $\langle\langle Q \rangle\rangle \top$.
3. $\neg \langle\langle Q \rangle\rangle \perp$.
4. $\neg \langle\langle \emptyset \rangle\rangle \neg \alpha \rightarrow \langle\langle Ag \rangle\rangle \alpha$.
5. $\langle\langle Q \rangle\rangle (\alpha \wedge \alpha') \rightarrow \langle\langle Q \rangle\rangle \alpha$ (outcome monotonicity).
6. $\langle\langle Q_1 \rangle\rangle \alpha_1 \wedge \langle\langle Q_2 \rangle\rangle \alpha_2 \rightarrow \langle\langle Q_1 \cup Q_2 \rangle\rangle (\alpha_1 \wedge \alpha_2)$ if $Q_1 \cap Q_2 = \emptyset$ (superadditivity).
7. $\langle\langle Q \rangle\rangle \Gamma_1; \Gamma_2 \leftrightarrow \langle\langle Q \rangle\rangle \Gamma_1 \wedge \langle\langle Q \rangle\rangle \Gamma_2$.
8. $\langle\langle Q \rangle\rangle \text{if } \phi \text{ then } \Gamma \leftrightarrow \langle\langle Q \rangle\rangle \Gamma$.
9. $\langle\langle Q \rangle\rangle \text{while } \phi \text{ do } \Gamma \leftrightarrow \langle\langle Q \rangle\rangle \Gamma$.

Even if the agents module allows for the modeling of sophisticated interrelationships between the sets of actions that agents can execute concurrently, the model is also limited in the sense that the capabilities of agents is given in a context-independent way. In the present model, we have no direct way of describing how the capabilities of agents may depend on their own states. For example, we cannot model agents that have agents that have bounded resources, and can execute an action only a limited number of times, say. This assumption ‘flattens’ the notion of capability, which becomes clear from the way that the last three axioms ‘reduce’ complex plans to simpler ones.

This does not mean that we cannot model the fact that actions have effects that depend on the state in which they are executed. We can do that in the environment module, and is therefore independent of the agents performing the actions.

So, even if our agents module is limited in this respect, it still is a natural one for modeling a wide range of scenarios. For example, any type of scenarios in which agents manipulate their environment via some kind of control panel (e.g. a keyboard) has a natural model in which the fact that an agent can press a button does not depend on the state of the environment, while the actual effects of pressing this button may depend on that state.

A clarifying distinction here may be that between *capability* and *executability*. The first notion refers to what an agent can do ‘in theory’ – for example, a robot may either go left or right, or you are capable of turning off your computer. The notion of *executability* refers to what can be done in an particular situation – for example, the robot can go left, but not right, when is is at a certain point in the maze; or you cannot turn off your computer because you have already done so.

4 Multi-agent system

In the previous sections we defined a module describing an environment, and a separate module describing the actions that agents can choose to perform, either by themselves or together. These two modules can be related by identifying the

set of actions in the two respective modules. Such a combination provides us with a semantics for reasoning about agents that act in an environment.

Definition 8 (Multi-agent system) A multi-agent system \mathcal{M}^{mas} is a tuple

$$\langle S, Ac, (\rightarrow)_{A \subseteq Ac}, P, \pi, Ag, \text{act} \rangle$$

where $\langle S, Ac, (\rightarrow_{A'})_{A \subseteq Ac}, P, \pi \rangle$ is an environment module (in the sense of definition 1) and $\langle Ac, Ag, \text{act} \rangle$ is an agents module (in the sense of definition 5).

An important thing to note is that this approach is *modular*, in the sense that the two components of the multi-agent system can be defined independently of each other. For example, we could have chosen for a more simple agents module by associating one single set of actions to each agent, as in [Sauro et al., 2006] (and adapting the definitions accordingly).

Of course, everything that we could express with the languages we used to talk about the environment and about the capabilities of the agents remains interesting in the combined system as well. However, in the new, more complex, system there are some other notions of interest that did not make sense in the separate modules.

In particular, we want to reason not only about the ability to enforce complex *actions*, but about the ability to enforce certain *results* as well. For this purpose we add expressions of the form $\langle\langle Q \rangle\rangle\phi$ to the language. Intuitively, $\langle\langle Q \rangle\rangle\phi$ means that the agents in Q have the power enforce a set of execution traces for which ϕ is true.

This leads to the following syntax for multi-agent systems:

$$\phi, \psi := p \mid \phi \wedge \psi \mid \neg\phi \mid AX\phi \mid A\phi U\psi \mid E\phi U\psi \mid [\Gamma]\phi \mid \langle\langle Q \rangle\rangle\Gamma \mid \langle\langle Q \rangle\rangle\phi$$

where Γ is a plan expression and Q denotes a set of agents.

We use concepts from ATL to define the semantics of sentences $\langle\langle Q \rangle\rangle\phi$. Intuitively, $\langle\langle Q \rangle\rangle\phi$ is true whenever the agents in Q can act in a way that, no matter what the other agents do, ϕ is true. The crucial observation is that this situation can be seen as a zero-sum *game* in which the group Q plays against $Ag \setminus Q$ and tries to obtain a result that satisfies ϕ .

To make this precise, we need some formal definitions. A *history* in a model is a finite sequence of states in that model. We say that two histories $s_0 \dots s_n$ and $s'_0 \dots s'_m$ are *indistinguishable* iff $m = n$ and for each $1 \leq i \leq n$, s_i and s'_i satisfy the same propositional variables.

A *strategy* f_Q for the agents Q is a, possibly partial, function from histories to sets of concurrent actions from $\text{act}(Q)$, with the property that if two histories s and s' are indistinguishable, then $f_Q(s) = f_Q(s')$ – intuitively, if a strategy assigns a set of concurrent actions to a path, this means that the strategy consists of executing any one of these actions whenever the history of the game played so far is given by that path. If a strategy is undefined at a certain history, the execution is taken to halt.

Given a multi-agent system \mathcal{M}^{mas} , a state s_0 and strategy f_Q for Q , we define a new model $\mathcal{M}^{\text{mas}}[Q, f_Q, s_0]$ that represents all possible traces that may result if the agents in Q execute the actions in the way prescribed by f_Q . The other agents are at liberty to choose any combination of actions

from their own action sets. Define relations \rightarrow'_A between histories by setting $(s_0, \dots, s_n) \rightarrow'_A (s_0, \dots, s_n, s_{n+1})$ iff $s_n \rightarrow_A s_{n+1}$ in \mathcal{M}^{mas} , and there is a $B \in f_Q((s_0 \dots s_n))$ and a $C \in \text{act}(Ag \setminus Q)$ such that $A = B \cup C$. The model $\mathcal{M}^{\text{mas}}[Q, f_Q, s_0]$ has the same agent module as the original model, its set of states consists of all finite histories reachable from (s_0) by \rightarrow'_A , and we set $\pi(s) = \pi(s_n)$, where s_n is the last element of s .

Definition 9 (Semantics of Multi-Agent Logic) Let \mathcal{M}^{mas} be a multi-agent system and $Q \subseteq Ag$ a group of agents. \models^m is inductively defined on the grammar of the multi-agent system logic. For the operators from the environment module and the agent module, we can simply copy the previous definitions. We only need to define the new operator $\langle\langle Q \rangle\rangle\phi$.

1. $\mathcal{M}^{\text{mas}}, s \models^m \langle\langle Q \rangle\rangle\phi$ iff there is a strategy f_Q for Q such that $\mathcal{M}^{\text{mas}}[Q, f_Q, s], s \models \phi$.

The ability to execute a plan Γ , and with that plan obtain a certain result, is closely related to the ability to obtain that result with a certain strategy. This is a central result of this paper:

Theorem 2 For any \mathcal{M}^{mas} , if there exists a plan expression Γ such that $\mathcal{M}^{\text{mas}}, s \models^m \langle\langle Q \rangle\rangle\Gamma \wedge [\Gamma]\phi$, then $\mathcal{M}^{\text{mas}}, s \models^m \langle\langle Q \rangle\rangle\phi$.

A proof of this theorem is in the appendix. The proof consists of a construction that gives us for each plan Γ such that $\langle\langle Q \rangle\rangle\Gamma$, a strategy f_Q for Q that has the same execution traces of Γ .

Moreover, we conjecture that if a multi-agent system \mathcal{M}^{mas} is finite, then this holds also in the other direction.¹

Theorem 2 is important in this paper, because it shows that our technique of composing a model of ability by combining different models for agents and for the environment actually works. The theorem demonstrates that the relatively abstract notion of ability of a group of agents G to obtain a result corresponds to the existence of a concrete plan that must have two independent properties: the agents must be capable of executing that plan, and the plan must guarantee the desired result.

In the introduction we announced that the present logic is a generalization of ATL. The idea here is that the ATL model is just a special kind of multi-agent system in the sense of definition 8. ATL models are “concurrent game structures” in which each agent gets assigned a set of actions, and at each state, each of the agents chooses an action from his set. A choice by each of the agents leads to a unique result.

Each concurrent game structure from ATL can be represented as a multi-agent system with the following properties. For each agent i there is a set of actions Ac_i – the sets Ac_i are disjoint. At each state, each agent can execute exactly one of

¹This conjecture might be proven along the following lines. Because the game that is defined by a formula $\langle\langle Q \rangle\rangle\phi$ is played on a finite model, we know that if there is a strategy to win such a game, there must exist a *finite* automaton that represents this strategy [Buchi and Landweber, 1969]. Using the fact that our strategies assign the same strategy sets to indistinguishable histories, we can then find a plan Γ with the wanted properties that has the same execution traces as this finite automaton.

his possible actions. So we can define the function act of the agents module by setting $\text{act}(\{i_0 \dots i_n\}) = \{\{a_{i_0} \dots a_{i_n}\} \mid a_{i_j} \in Ac_j\}$.

In the environment, the relations \rightarrow_A for $A = \{a_1 \dots a_n\}$ a set that contains exactly one action for each of the agents must be deterministic and serial. If a set of actions A is not of this form, \rightarrow_A is empty.

We can now define a translation from formulas into our language by setting:

1. $(\langle\langle Q \rangle\rangle \circ \phi)^* = \langle\langle Q \rangle\rangle (AX\phi \wedge EX\top)$
2. $(\langle\langle Q \rangle\rangle \square \phi)^* = \langle\langle Q \rangle\rangle (AG\phi \wedge AGEX\top)$
3. $(\langle\langle Q \rangle\rangle \phi_1 U \phi_2)^* = \langle\langle Q \rangle\rangle (A\phi U \psi \wedge AGEX\top)$

Theorem 3 *Let M be an ATL model, and $\mathcal{M}^{\text{mas}}_M$ the corresponding mas model. Let ϕ be a formula of ATL, and ϕ^* be the translation in our language. Then it holds that:*

$$M, s \models_{\text{atL}} \phi \text{ iff } \mathcal{M}^{\text{mas}}_M, s \models_m \phi^*$$

A proof of this theorem is in the appendix.

5 A final example

In this section we test our formalism with a concrete scenario where two guards have to control the doors of a prison.

A prison is composed of two interiors, a dormitory and a courtyard. One door opens to the outside, and it is important that it is not opened by accident. It can therefore be opened or closed only if both guards act together. The other door separates the cell block from the courtyard, and each guard can open that door by himself. The fact that the inner gate is open is expressed by a proposition in_open ; if the outer gate is open, out_open is true. The state of the gates is ruled by four buttons: i_a, i_b, o_a and o_b . The gate in_open toggles its state if and only if at least one of i_a and i_b is pressed. The outer gate out_open toggles its state if and only if both o_a and o_b are pressed.

This description can be captured by an environment model \mathcal{M}^{env} with four states determined by the fact whether the gates are open or not. For example, we will write 01 for the state in which the inner door is closed (in_open is false) and the outer door is open (out_open is true). Each of the buttons corresponds to an atomic action in the model, and the transitions are as described. So, for example, it will hold that

$$\begin{aligned} 01 &\rightarrow_{\{o_a, o_b\}} 00 \\ 01 &\rightarrow_{\{i_a, o_b\}} 11 \\ 00 &\rightarrow_{\{i_a, o_a, o_b\}} 11 \end{aligned}$$

We want to verify that in the state 00, i.e. when all the doors are closed, there exists a way for the two guards to enable a prisoner to exit safely, i.e. in such a manner in each instant not both the doors are opened. We also want to know if both the agents are necessary for this scope or only one of them is self-sufficient. Intuitively, starting from the state 00 we want that (1) to open the inner door while the outer door is still closed, then (2) to close the inner door before opening the outer door (or, at most, to switch both of them at the same time), finally (3) we want to close the outer door. Let's see step by step which concurrent actions force the system to make these transitions. Starting from the state 00, to transit

to the state 10 (in_open true and out_open false) one of the buttons i_a or i_b have to be pressed and at least one of o_a or o_b have to be not pressed in order to keep the outer door closed. To transit from the state 10 to the state 01 (corresponding to a concurrent switch of the two doors) one of the buttons i_a or i_b have to be pressed and both the buttons o_a and o_b has to be pressed. Then, to close the outer door, both the buttons o_a and o_b have to be pressed.

More formally, given $\Gamma = i_a \wedge \neg o_a; o_a \wedge o_b \wedge i_a; o_a \wedge o_b$, we can verify that

$$\mathcal{M}^{\text{mas}}, 00 \models^m [\Gamma] AG \neg (\text{in_open} \wedge \text{out_open})$$

i.e. Γ is safe. We want also to verify that Γ gets a prisoner to exit and hence that the inner and the outer doors are opened sequentially. Formally, it can be verified that

$$\mathcal{M}^{\text{mas}}, 00 \models^m [\Gamma] EF (\text{in_open} \wedge EF \text{out_open})$$

Now consider the case that one of the guards, gd_A , controls the buttons i_a and o_a , whereas the other guard, gd_B , controls the buttons i_b and o_b .

Since together the two guards have a full access to four buttons i_a, i_b, o_a, o_b , it is easy to foresee that they can actually execute the plan Γ . Formally, $\mathcal{M}^{\text{mas}} \models^m [\{gd_A, gd_B\}] \Gamma$. However, individually they cannot press both o_a and o_b - $\mathcal{M}^{\text{mas}} \models^m \neg [\{gd_B\}] (o_a \wedge o_b)$ and $\mathcal{M}^{\text{mas}} \models^m \neg [\{gd_A\}] (o_a \wedge o_b)$ - and since $o_a \wedge o_b$ is necessary in the state 00 to make the outer door open

$$\mathcal{M}^{\text{mas}}, 00 \models^m [\text{while } \top \text{ do } \neg(o_a \wedge o_b)] AG \neg \text{out_open}$$

Then they are both necessary to escape the prison.

6 Conclusions

We have defined a logic-based framework to reason about cooperation among agents. Our framework builds on concepts from cooperation logic, such as Coalition Logic [Pauly, 2002] and ATL [Alur *et al.*, 2002; van der Hoek and Wooldridge, 2003] and CTL [McMillan, 1992]. We have provided these cooperation logics with an explicit formalism for reasoning about plans. Our framework generalizes earlier approaches to the problem of providing ATL with an action language — e.g. [Sauro *et al.*, 2006] consider only one-step strategies, while [van der Hoek *et al.*, 2005] consider only memory-less strategies.

We have addressed this problem in a modular way, in the sense that we have tried to separate the part of the model that concerns reasoning about actions with the part that concerns reasoning about cooperation ([Sauro *et al.*, 2006]). An *environment module* can be seen as a description of the environment in which agents live and its causal laws; an *agents module* describes in what way agents can combine forces to execute a certain plans. The modules share a planning language that contains constructions from a simple programming language. In combining the two models, we obtain a model that is, in a precise sense, a generalization of the ATL model.

In this paper we have given formal definitions of these modules in terms of Kripke structures, defined languages and provided them with a semantics. Most importantly, with theorem 2, we showed how the basic logical of the combined

model, that of the ability of a group to obtain certain results, can be seen as combining features from the two modules: the ability to execute a plan (in the agents module) and the fact that this plan gives a certain result (in the environment module). Furthermore, we showed in theorem 3 how, in particular, our model gives us a precise way of providing ATL with an action language. Finally, we gave an example to show how our framework works and that suggests its possible applications in real and more complex scenarios.

There are some clear directions for future work. Apart from the conjecture below theorem 2, there are open questions regarding model checking and axiomatization. With respect to the latter, we hope to be able to use ideas from [Sauro *et al.*, 2006], that provides a complete axiomatization of a similar, but simpler, system. In particular they have shown how, given an axiomatization for the environment module and for the agents module, finding complete axioms for the multi-agent system module is straightforward.

References

- [Alur *et al.*, 2002] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of ACM*, 49(5):672–713, 2002.
- [Broersen, 2003] Jan Broersen. *Modal Action Logics for Reasoning about Reactive Systems*. PhD thesis, Vrije Universiteit Amsterdam, 2003.
- [Buchi and Landweber, 1969] J. Richard Buchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.
- [Chen and De Giacomo, 1999] X. J. Chen and G. De Giacomo. Reasoning about nondeterministic and concurrent actions: A process algebra approach. *Artificial Intelligence*, 107:63–98, 1999.
- [Giordano *et al.*, 1998] Laura Giordano, Alberto Martelli, and Camilla Schwind. Dealing with concurrent actions in modal action logic. In Henri Prade, editor, *ECAI 98. 13th European Conference on Artificial Intelligence*, 1998.
- [Harel and Singerman, 1999] D. Harel and E. Singerman. Computation paths logic: An expressive, yet elementary, process logic. *Annals of Pure and Applied Logic*, 96:167–186, 1999.
- [Harel *et al.*, 1984] D. Harel, D. Kozen, and J. Tiuryn. Dynamic logic. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic Volume II — Extensions of Classical Logic*, pages 497–604. D. Reidel Publishing Company: Dordrecht, The Netherlands, 1984.
- [Henriksen and Thiagarajan, 1999] J. G. Henriksen and P. S. Thiagarajan. Dynamic linear time temporal logic. *Annals of Pure and Applied Logic*, 96:187–207, 1999.
- [McMillan, 1992] K.L. McMillan. *Symbolic Model Checking*. PhD thesis, CMU University, 1992.
- [Pauly, 2002] M. Pauly. A Modal Logic for Coalitional Power in Games. *Journal of Logic and Computation*, 12:146–166, 2002.
- [Peleg, 1998] B. Peleg. Effectivity functions, game forms, games, and rights. *Social Choice Theory*, 15:67–80, 1998.
- [Sauro *et al.*, 2006] L. Sauro, J. Gerbrandy, W. van der Hoek, and M. Wooldridge. Reasoning about action and cooperation. In *Proceedings of The Fifth International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS’06)*, Hakodate, Japan, 2006.
- [van der Hoek and Wooldridge, 2003] W. van der Hoek and M. Wooldridge. Cooperation, knowledge, and time: Alternating-time temporal epistemic logic and its applications. *Studia Logica*, 75(1):125–157, 2003.
- [van der Hoek *et al.*, 2005] W. van der Hoek, W. Jamroga, and M. Wooldridge. A logic for strategic reasoning. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS’05)*, pages 157–164, Utrecht, The Netherlands, 2005.
- [Wooldridge and Jennings, 1994] M. Wooldridge and J. Jennings. Towards a theory of cooperative problem solving. In *Procs. of the Sixth European Workshop on Modelling Autonomous Agents in Multi-Agent Worlds (MAAMAW-94)*, 1994. □

Appendix: Proofs

Theorem 1 Given an agents module $\langle Ag, Ac, act \rangle$, the corresponding effectivity function E , as defined in definition 6, is playable.

proof: By way of example, consider Ag -maximality. From the definition of effectivity, it follows that $E(\emptyset) = \{\mathbf{A}' \mid act(Ag) \subseteq \mathbf{A}'\}$ and that $E(Ag) = \{(A') \mid \exists A \in act(Ag) \text{ s.t. } A \in \mathbf{A}'\}$. That means that if $(\mathcal{P}(Ac) \setminus \mathbf{A}) \notin E(\emptyset)$, there must be an $A \in \mathbf{A}$ such that $A \in act(Ag)$. But that means that $\mathbf{A} \in E(Ag)$. □

Theorem 2 For any \mathcal{M}^{mas} , if there exists a plan expression Γ such that $\mathcal{M}^{mas}, s \models^m \langle\langle Q \rangle\rangle \Gamma \wedge [\Gamma]\phi$, then $\mathcal{M}^{mas}, s \models^m \langle\langle Q \rangle\rangle \phi$.

proof: Let \mathcal{M}^{mas} be a multi-agent system with a state s , and Γ be a plan. We will show how to find a strategy f_Q that is equivalent to Γ in the sense that it has the same execution traces starting from s : the execution traces in the model $\mathcal{M}^{mas}[\Gamma, s]$, s are isomorphic to those in $\mathcal{M}^{mas}[Q, f_Q, s]$.

Observe first that each sequence s of states in $\mathcal{M}^{mas}[\Gamma, s]$ can be transformed into a sequence of states in the original \mathcal{M}^{mas} by simply removing the programs – s becomes a sequence $redux(s)$ of states in the original model. The function $redux$ is a bijection when restricted to sequences with the same initial state: if two sequences in $\mathcal{M}^{mas}[\Gamma, s]$ with the same initial state are different, they have different $redux$ s. (We can prove this by induction on the length of the sequences. For s of length 1, the statement is trivial. Suppose now that we have two sequences $((\sigma_0, s_0), \dots, (\sigma_n, s_n))$ and $((\sigma'_0, s'_0), \dots, (\sigma'_m, s'_m))$ that have the same $redux$. Then they must be of the same length. By induction hypothesis, we have that $\sigma_i = \sigma'_i$ for each $i < n$. Inspection of the proof rules shows that whenever we can prove $(\sigma, s) \rightarrow_A (\sigma', s')$, the resulting σ' will always be the same. So, σ_n and σ'_n must be equal.)

Define the *head* of a sequence of plans in a state s , $head(\sigma, s)$, to be the first boolean action is executed when

Γ is executed in s – we let the function return the inconsistent action \perp if no such action can be found. Following the proof system of definition 2, we define, e.g. $\text{head}(\alpha) = \alpha$, $\text{head}(\epsilon) = \perp$ (with ϵ the empty sequence), and set $\text{head}(\text{while } \phi \text{ do } \Gamma, s) = \text{head}(\Gamma, s)$ if $s \models \phi$; the value of $\text{head}(\text{while } \phi \text{ do } \Gamma, s) = \perp$ otherwise. It is easy to see that $(\sigma, s) \rightarrow_A (\sigma', s')$ iff $A \models \text{head}(\sigma)$ and $s \rightarrow_A s'$.

Now, let s be a sequence of states in \mathcal{M}^{mas} . We need to define the value of $f_Q(s)$. If s is not the reduction of some sequence in $\mathcal{M}^{\text{mas}}[\Gamma, s]$, then we leave $f_Q(s)$ undefined. Otherwise, there must be sequence s' in $\mathcal{M}^{\text{mas}}[\Gamma, s]$ such that in $s = \text{redux}(s')$. Let (σ, s') be the last state in s' . Consider $\text{head}(\sigma)$. By inspection of the proof system, it is easy to see that if $\text{head}(\sigma) \neq \perp$, then $\text{head}(\sigma)$ must be occur as a boolean action in Γ . It follows by definition of $\langle\langle Q \rangle\rangle$ that if $\mathcal{M}^{\text{mas}}, s \models^m \langle\langle Q \rangle\rangle \Gamma$, then it holds for each α that occurs as an boolean action in Γ that $\mathcal{M}^{\text{mas}}, s \models^m \langle\langle Q \rangle\rangle \alpha$. By definition of $\langle\langle Q \rangle\rangle$, it follows that $\alpha^{\mathcal{M}} \in E^{\mathcal{M}}(Q)$, and therefore, there must be one (or more) sets $A \in \text{act}(Q)$ such that $A \cup B \in \alpha^{\mathcal{M}}$ for each $B \in \text{act}(Ag \setminus Q)$. We set $f_Q(\text{redux}(s))$ to be the set of all A with this property.

We now claim that the set of traces in $\mathcal{M}^{\text{mas}}[Q, f_Q, s]$ is isomorphic to those in $\mathcal{M}^{\text{mas}}[\Gamma, s]$ – the function redux provides this isomorphism. \square

Theorem 3 *Let M be an ATL model, and $\mathcal{M}^{\text{mas}}_M$ the corresponding MAS-model. Let ϕ be a formula of ATL, and ϕ^* be the translation in our language. Then it holds that:*

$$M, s \models_{\text{atl}} \phi \text{ iff } \mathcal{M}^{\text{mas}}_M, s \models_m \phi^*$$

proof: We first need to be more precise about how we construct a multi-agent system from an ATL model M .

An *ATL model* (a concurrent game structure) is a tuple $\langle Ag, S, P, \pi, d, \delta \rangle$, in which Ag is a set of agents, S a finite set of states, P as set of propositional variables, π a valuation function. The function d assigns to each agent a and each state s a number $d_a(s)$ – we can think of each agent as having actions $\{1 \dots d_a(s)\}$ available at state s . Each agent can, at each state, choose exactly one of these actions; the result is given by the function δ , which assigns to each tuple $(s, j_1 \dots j_n)$ a successor $\delta(s, j_1, \dots, j_n)$.

A concurrent game structure is but a special case of a multi-agent system in the sense of definition 8. To see this, we define a $\mathcal{M}^{\text{mas}} = (S, Ac, (\rightarrow_A)_{A \subseteq Ac}, P, \pi, Ag, \text{act})$ as follows. For each agent $a \in Ag$, we set $\text{act}(\{a\}) = \{\{a_1\} \dots \{a_{n_a}\}\}$, where n_a is the highest value occurring in any of the sets $d_a(s)$ (we can do this since S is finite). We make sure that the sets of actions of different agents are disjoint. This set contains only singleton sets, as the agents can choose to perform just one single action at each point.

For groups Q of agents, we define $\{a_1 \dots a_n\} \in \text{act}(\{i_1 \dots i_n\})$ iff $\{a_1\} \in \text{act}(\{i_1\})$ and ... and $\{a_n\} \in \text{act}(\{i_n\})$.

Finally, we define the transition relation in our \mathcal{M}^{mas} by setting $s_0 \rightarrow_A s_1$ iff $A \in \text{act}(Ag)$ (that is, A contains exactly on action a_i for each of the agents) and $\delta(s_0, a_1 \dots a_n) = s_1$.

We now need to show that this construction preserves the truth of ATL sentences (modulo the translation). This can be

shown by induction on the structure of ϕ , where the only not trivial cases are those which involve these translation rules:

1. $(\langle\langle Q \rangle\rangle \circ \phi)^* = \langle\langle Q \rangle\rangle (AX\phi \wedge EX\top)$
2. $(\langle\langle Q \rangle\rangle \square \phi)^* = \langle\langle Q \rangle\rangle (AG\phi \wedge AGEX\top)$
3. $(\langle\langle Q \rangle\rangle \phi_1 U \phi_2)^* = \langle\langle Q \rangle\rangle (A\phi U \psi \wedge AGEX\top)$

There are two differences between the definitions of strategies $\langle\langle Q \rangle\rangle$ in ATL and that in our system. The first one is that in our system, a strategy may be undefined at a certain state, while in ATL they are defined always. We need the existential ‘ EX ’ operators to guarantee that strategies are defined at each state, and prescribe a course of actions that it indeed possible in the environment. The second difference is that our strategies may be ‘partially defined’: they prescribe a set of possible concurrent actions, while an ATL a strategy gives us (for each agent) one particular action to take at each state. \square