Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020)
Main Track

Token-based Execution Semantics for Multi-Agent Epistemic Planning

Thorsten Engesser , **Robert Mattmüller** , **Bernhard Nebel** , **Felicitas Ritter**
University of Freiburg, Germany
{engesser,mattmuel,nebel,ritterf}@cs.uni-freiburg.de

Abstract

Epistemic planning has been employed as a means to achieve implicit coordination in cooperative multi-agent systems where world knowledge is distributed between the agents, and agents plan and act individually. However, recent work has shown that even if all agents act with respect to plans that they consider optimal from their own subjective perspective, infinite executions can occur. In this paper, we analyze the idea of using a single token that can be passed around between the agents and which is used as prerequisite for acting. We show that introducing such a token to any planning task will prevent the existence of infinite executions. We furthermore analyze the conditions under which solutions to a planning task are preserved under our tokenization.

1 Introduction

Epistemic implicit coordination planning (Engesser et al. 2017) is a technique for planning and coordination in multi-agent systems in which agents try to collaboratively reach a joint goal. The knowledge and abilities required to reach the goal can be distributed among the agents. With no centralized coordination instance and without the possibility for the agents to agree on a joint plan, the agents need to plan individually and execute their plans in a decentralized way.

A central assumption Engesser et al. (2017) make for their notion of policies and policy execution is that actions are applied in sequence by the agents. However, the order in which agents are allowed to act is not preimposed, i.e., situations where multiple agents have an applicable action are allowed.

The advantage of this kind of sequentiality over joint actions is that for a problem to be solvable, solution existence does not have to be common knowledge between all agents. Instead, only the agent that performs the first action of a plan has to know that the plan leads to the goal and that after the execution of the first action, the next agent who is designated to act will know, and so on. This is helpful since often it is not known in advance which orders of agents will work. In such cases, any agent that finds a plan can begin, with the other agents waiting until they have sufficient information.

One issue with this approach is that agents who each plan for themselves and act according to their own policies may want to act at the same time. These kinds of "conflicts of interest" are not considered at the policy level but at the execution level. Since the idea of planning for implicit coordination is that there should be no centralized instance coordinating the agents, one has to consider all possible executions resulting from each order in which the agents could act, given each of their individual plans. While this *interleaving* semantics is harmless in some cases, in other cases it results in agents inadvertently working against each other.

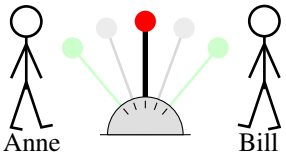

Figure 1: Two agents and a lever.

Consider, for example, the situation depicted in Figure 1, in which a lever can be pulled left or right, with both the leftmost and the rightmost position being a goal state. If Anne's plan is to pull the lever all the way to the left and Bill's plan is to pull the lever all the way to the right, there are executions in which the lever is pulled back and forth indefinitely. An obvious fix that works in this particular instance is to require the agents to act only with respect to optimal plans and thus pull the lever only towards the nearest goal configuration. However, Bolander et al. (2018) have shown that problems with infinite executions can still occur if the agents' knowledge about the world differs. E.g., we could have the situation in which both the leftmost and the rightmost configuration can be, but are not necessarily, goal configurations. Imagine Anne only knows about whether or not the leftmost configuration is a goal configuration and Bill only knows about whether or not the rightmost configuration is one. Using our solution concept, both have to assume the worst case of their configuration being the only goal configuration and we still end up with infinite executions.

Instead of trying to provide success guarantees by only restricting the types of policies which the agents are allowed to take, in the present paper, we try to tackle the problem of infinite executions on the planning task level. The reason why requiring policies to be optimal does not always prevent infinite executions is that optimality is judged from the subjective perspective of each agent. Thus an agent may act because from his perspective the action is part of an optimal

plan, while from the perspective of the agent who acted before it is not, and another agent was expected to act instead. To prevent this from happening, we exclude all agents except for one from performing actions. The acting agent can then specify the agent who is allowed to act next. To model this, we introduce a token that can be passed around by the agents. Only the agent with the token is allowed to perform an action. We show that this approach does not only solve the lever problem from the example but prevents infinite execution in general, while preserving plan existence, given some formal criteria are met.

The remainder of this paper is structured as follows: First, we are going to give a brief overview of related work, and highlight how this paper builds on it. Next, we will introduce the formal framework, dynamic epistemic logic (DEL). In the main part of the paper, we demonstrate how to rewrite a given planning task to include tokens, show that this eliminates infinite executions, and discuss under which conditions plan existence can be preserved. Finally, we conclude and discuss future work.

2 Related Work

In this paper, we attempt to provide success guarantees of interleaved plan executions, just like Bolander et al. (2018) did. Unlike them, we do not study how *agent types* impact the success of implicitly coordinated plans, but rather impose restrictions on allowed behavior by modifying the *rules of the planning task*. Imposing the rule that only an agent that possesses the token may act overcomes a limitation of the agent-types approach: Without tokens, even optimally eager agents are only guaranteed to prevent infinite executions if there is uniform observability.

In recent work orthogonal to the tokenization approach we present here, Nebel et al. (2019) investigated how implicitly coordinated plans without communications can succeed in the *special case* of multi-agent path finding with destination uncertainty, in which agents have to move to different destinations in a collision free manner without communicating, while the agents' individual destinations are *not* common knowledge among them. It was shown that, in such scenarios, eagerness and the capability to perform conservative re-planning, are sufficient to ensure that plans succeed.

Besides planning with the intent to allow agents to self-coordinate, epistemic planning has been mostly applied to finding centralized plans in the presence of knowledge preconditions and goals (Kominis and Geffner 2015; Muise et al. 2015; Huang et al. 2017; Le et al. 2018). In more recent work, Maubert, Pinchinat, and Schwarzentruber (2019) have looked at modeling and synthesizing strategies for reachability games in DEL, which is a setting which is more similar to ours. While in their formalism, agents also act sequentially, there is no interleaving concurrency and it is assumed that it is always commonly known which agent's turn it is.

Tokens have also made an appearance in distributed systems, for example in the work of Loucks and Shaheen (1997). Components of a distributed system are located apart from each other, but still have to communicate and coordinate their actions to achieve a common goal. Makki et al. (1992) used a token queue and semaphore for restricting the use of a mutual resource that can only have a small number of users at a time. In contrast to the approach taken in the present paper, their tokens are used to restrict the access to a limited resource. Their agents do not plan with other agents, and handing the token to the next player is usually done by a waiting queue. In our approach, the idea is that the agent who has the token gets to decide which agent will have the token next. This is not desired in distributed systems because they want all agents to have the same rights of access to a resource with no agent being strategically excluded.

Adding tokens can be seen as implementing a simple and practical *social law* (specifically the law that only allows an agent to act if it possess the token), a concept that has gained increasing interest in multi-agent planning lately. Social laws such as the ones from Karpas, Shleyfman, and Tennenholtz (2017) and Nir and Karpas (2019) have also tried to minimize the problems that arise from multiple agents by trying to force the agents to work together and minimize the amount of "damage" agents can do if they want to prevent other agents from reaching their goal. Unlike their social laws, which have to be designed by a rational person, and specially made to fit one specific planning task, our approach can be applied in a generalized way to given planning tasks.

3 Epistemic Planning

In the following, we will recapitulate the syntax and semantics of Dynamic Epistemic Logic (DEL) (van Ditmarsch, van der Hoek, and Kooi 2007), which we will use as the formal framework of this paper. We will use the conventions of Bolander et al. (2018), and also use their definitions of planning tasks, policies, agent types and executions.

Let $\mathcal{A}$ be a finite set of agents and P be a finite set of atomic propositions. The *epistemic language* $\mathcal{L}_{\text{KC}}$ is then defined by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_i\varphi \mid C\varphi$$

with $p \in P$ and $i \in \mathcal{A}$. Formula $K_i\varphi$ reads as "agent i knows φ" and $C\varphi$ reads as "it is common knowledge that φ". Furthermore, the operators $\top, \bot, \leftarrow, \rightarrow, \leftrightarrow$ are defined as abbreviations, analogously to their definition in propositional logic. We will refrain from specifying P and $\mathcal{A}$ explicitly, if their values are clear from the context.

Example 1. Recall the lever example from the introduction. If we ignore the position of the lever, we can model the situation using $\mathcal{A} = \{anne, bill\}$, and $P = \{l, r\}$, where l denotes whether there is a goal position to the left and r denotes whether there is a goal position to the right. The formula $\neg K_{anne} r \wedge \neg K_{anne} \neg r$ expresses that Anne does not know whether or not there is a goal to the right. The formula $K_{bill}(\neg K_{anne} r \wedge \neg K_{anne} \neg r)$ expresses that Bill does know that Anne does not know.

Epistemic formulas are evaluated in *epistemic models* $\mathcal{M} = \langle W, (\sim_i)_{i\in\mathcal{A}}, V\rangle$, where W is a non-empty finite set of worlds (called the *domain* of $\mathcal{M}$), $\sim_i \subseteq W \times W$ is an equivalence relation called the *indistinguishability relation* for each agent $i \in \mathcal{A}$, and $V : P \to \mathcal{P}(W)$ is the *valuation function*, assigning to each proposition $p \in P$ a set of worlds $V(p)$ in which the proposition is true.

We depict epistemic models as graphs where the nodes correspond to worlds and the edges correspond to indistinguishability between the worlds. Nodes are labeled with the world name and all propositions which are true in that world. Edges are labeled with the agents for which the worlds are indistinguishable. For better readability, we usually omit reflexive edges and edges that are implied by transitivity.

Example 2. Assuming that it is common knowledge that at least one of the two positions must be a goal position, we can model the situation from our running example as epistemic model $\mathcal{M}_0$ with three worlds: one world w_1 in which just the left position is a goal, one world w_3 in which just the right position is a goal and one world w_2 in which both positions are a goal. The epistemic model, including the indistinguishabilities for the agents is depicted below:

$$\mathcal{M}_0 = \quad \underset{w_1 : l}{\bullet} \overset{anne}{\text{———}} \underset{w_2 : l, r}{\bullet} \overset{bill}{\text{———}} \underset{w_3 : r}{\bullet}$$

For $W_d \subseteq W$, the pair $(\mathcal{M}, W_d)$ is called an *epistemic state* (or simply a state) and the worlds of W_d are called *designated worlds*. A state is called *global* if $W_d = \{w\}$ for some world w (called the *actual world*). We then often write $(\mathcal{M}, w)$ instead of $(\mathcal{M}, \{w\})$. We use $S^{gl}(P, \mathcal{A})$ to denote the set of global states (or simply S^{gl} if P and $\mathcal{A}$ are clear from context). For any state $s = (\mathcal{M}, W_d)$ we let $\mathit{Globals}(s) = \{(\mathcal{M}, w) \mid w \in W_d\}$. A state $(\mathcal{M}, W_d)$ is called a local state for agent i if W_d is closed under $\sim_i$ (that is, if $w \in W_d$ and $w \sim_i v$, then $v \in W_d$).
Given a state $s = (\mathcal{M}, W_d)$ the associated local state of agent i, denoted s^i, is $(\mathcal{M}, \{v | v \sim_i w \text{ and } w \in W_d\})$. Going from s to s^i amounts to a *perspective shift* to the local perspective of agent i.

Example 3. Let $s_0 = (\mathcal{M}_0, w_2)$ be the global state for the lever example. Then Anne sees the local state $s_0^{anne} = (\mathcal{M}_0, \{w_1, w_2\})$, meaning she cannot distinguish whether $(\mathcal{M}_0, w_1)$ or $(\mathcal{M}_0, w_2)$ from $\mathit{Globals}(s_0^{anne})$ is the true global state. Bill sees $s_0^{bill} = (\mathcal{M}_0, \{w_2, w_3\})$.

Let $(\mathcal{M}, W_d)$ be a state with $\mathcal{M} = \langle W, (\sim_i)_{i \in \mathcal{A}}, V \rangle$. For $i \in \mathcal{A}$, $p \in P$ and $\varphi, \psi \in \mathcal{L}_{\text{KC}}$, truth is defined as follows:

$$\begin{array}{lll}
(\mathcal{M}, W_d) \models \varphi & \text{iff} & (\mathcal{M}, w) \models \varphi \text{ for all } w \in W_d \\
(\mathcal{M}, w) \models p & \text{iff} & w \in V(p) \\
(\mathcal{M}, w) \models \neg\varphi & \text{iff} & (\mathcal{M}, w) \not\models \varphi \\
(\mathcal{M}, w) \models \varphi \wedge \psi & \text{iff} & (\mathcal{M}, w) \models \varphi \text{ and } (\mathcal{M}, w) \models \psi \\
(\mathcal{M}, w) \models K_i\varphi & \text{iff} & (\mathcal{M}, v) \models \varphi \text{ for all } v \sim_i w \\
(\mathcal{M}, w) \models C\varphi & \text{iff} & (\mathcal{M}, v) \models \varphi \text{ for all } v \sim^* w
\end{array}$$

where $\sim^*$ is the transitive closure of $\bigcup_{i \in \mathcal{A}} \sim_i$.

Example 4. We can now verify for our running example, that it indeed holds that $s_0 \models \neg K_{anne} r \wedge \neg K_{anne} \neg r$. Note that checking a formula $K_i\varphi$ in a state s amounts to the same as checking the formula φ in s^i. In our example, $s_0 \not\models K_{anne} r$ because $(\mathcal{M}_0, \{w_1, w_2\}) \not\models r$, and $s_0 \not\models K_{anne} \neg r$ because $(\mathcal{M}_0, \{w_1, w_2\}) \not\models \neg r$.

Note that syntactically different states can be epistemically equivalent, i.e., satisfy the exact same set of epistemic formulas. In the following, we assume that such states are identified. In practice, one can do that by checking for *bisimilarity* (Blackburn, de Rijke, and Venema 2001).

3.1 Epistemic Actions and the Product Update

We also need a way do define actions, which can change the facts of the world as well as the knowledge of the agents. The way this is done in the action model logic of DEL is using so-called *event models*.

An event model is a 4-tuple $\mathcal{E} = \langle E, (\sim_i)_{i \in \mathcal{A}}, \text{pre}, \text{eff} \rangle$ where E is a non-empty finite set of events (called the *domain* of $\mathcal{E}$), $\sim_{\mathcal{A}} \subseteq E \times E$ is an equivalence relation called the indistinguishability relation for each agent $i \in \mathcal{A}$, and the functions $\text{pre} : E \to \mathcal{L}_{\text{KC}}$ and $\text{eff} : E \to \mathcal{L}_{\text{KC}}$ assign *preconditions* and *effects* to each event. While for each event $e \in E$, the precondition $\text{pre}(e)$ can be an arbitrary formula from $\mathcal{L}_{\text{KC}}$, the effect $\text{eff}(e)$ is a conjunction of literals, i.e, of atomic propositions and their negations, including $\top$ and $\bot$.

Each event of an action represents a different possible outcome. By using multiple events $e, e' \in E$ which are indistinguishable ($e \sim_i e'$), for some agent $i \in \mathcal{A}$, it is possible to model actions that are only partially observable.

We depict event models similarly to epistemic models as graphs, where the nodes correspond to events and the edges correspond to the indistinguishability between events. We label each node for an event $e \in E$ with $e : \langle \text{pre}(e), \text{eff}(e) \rangle$. As before, we usually omit reflexive edges and edges that are implied by transitivity for better readability.

For $E_d \subseteq E$, the pair $(\mathcal{E}, E_d)$ is called an *epistemic action*, or simply action. We call $(\mathcal{E}, E_d)$ a local action for agent i when E_d is closed under $\sim_i$.

Example 5. Consider the following event model which we will use to model a *sensing* action for Anne. It contains one event for each possible sensing outcome. Event e_1 occurs if r is true and event e_2 occurs if r is false. Since the action should not change any facts, both events have the effect $\top$. There is no indistinguishability between e_1 and e_2 for Anne. This way, after the action, she will know whether e_1 or e_2 has occurred and thus whether r is true or false. To make the action as general as possible we leave the events indistinguishable to Bill: If he does not know whether or not r is true, he should not learn it as result of Anne's sensing action.

$$\mathcal{E}_1 = \quad \underset{e_1 : \langle r, \top \rangle}{\bullet} \overset{bill}{\text{———}} \underset{e_2 : \langle \neg r, \top \rangle}{\bullet}$$

Anne's sensing action is then $(\mathcal{E}_1, \{e_1, e_2\})$. We need to designate both worlds due to not knowing the outcome of a sensing action in advance. The action $(\mathcal{E}_1, \{e_1\})$ could also make sense, e.g., as an action for a third agent informing Anne that r is true, without letting Bill know about it.

The semantics of action application is given by the *product update*. Let a state $s = (\mathcal{M}, W_d)$ and an action $a = (\mathcal{E}, E_d)$ be given with $\mathcal{M} = \langle W, (\sim_i)_{i \in \mathcal{A}}, V \rangle$ and $\mathcal{E} = \langle E, (\sim_i)_{i \in \mathcal{A}}, \text{pre}, \text{eff} \rangle$. Then the product update of s with a is defined as $s \otimes a = (\langle W', (\sim'_i)_{i \in \mathcal{A}}, V' \rangle, W'_d)$ where

- each world is paired up with all applicable events, i.e., $W' = \{(w, e) \in W \times E \mid \mathcal{M}, w \models \text{pre}(e)\}$;

- new worlds are indistinguishable if the old worlds were indistinguishable and the events are indistinguishable, i.e., $(w, e) \sim_i' (w', e')$ iff $w \sim_i w'$ and $e \sim_i e'$;
- propositions become true if they occur positively in the effect of the event, or if they don't occur negatively and have already been true before, i.e., $(w, e) \in V'(p)$ iff $\mathrm{eff}(e) \models p$ or $(\mathcal{M}, w \models p$ and $\mathrm{eff}(e) \not\models \neg p)$;
- worlds are designated if both predecessor world and event are designated, i.e., $(w, e) \in W_d'$ iff $w \in W_d$ and $e \in E_d$.

We say that an action $a = (\mathcal{E}, E_d)$ is *applicable* in a state $s = (\mathcal{M}, W_d)$ if for all $w \in W_d$ there is an applicable event $e \in E_d$, meaning $\mathcal{M}, w \models \mathrm{pre}(e)$.

Example 6. In our running example, the action a_1 is applicable in s_0, and after the action application, we obtain the state $s_1 = (\mathcal{M}_1, \{(w_2, e_1)\})$ where $\mathcal{M}_1$ is depicted below:

$$\mathcal{M}_1 = \quad \bullet \qquad \bullet \overset{bill}{\text{———}} \bullet$$
$$(w_1, e_2) : l \quad (w_2, e_1) : l, r \quad (w_3, e_1) : r$$

We can see that in s_1, Anne now knows that r. Also, while Bill has not learned anything new about l, he knows now that Anne knows about r. The sensing action of Anne is *public* in the sense that other agents will know that the sensing has taken place.

3.2 Planning Tasks, Policies and Executions

We now have everything that is needed to define multi-agent epistemic planning tasks in DEL. For a fixed set of agents $\mathcal{A}$, a *planning task* $\Pi = \langle s_0, A, \omega, \gamma \rangle$ consists of a global state s_0 called the *initial state*; a finite set of actions A; an *owner function* $\omega : A \to \mathcal{A}$; assigning each action to its owner and a *goal formula* $\gamma \in \mathcal{L}_{\mathrm{KC}}$.

Example 7. For the lever problem, the planning task is a tuple $\langle s_0, A, \omega, \gamma \rangle$ such that s_0 is defined as before plus additional propositions that indicate the position of the lever (e.g., $p_l \in P$ to indicate that the lever is at the leftmost position, $p_r \in P$ to indicate that the lever is at the rightmost position, and other propositions for the positions in between). For each non-leftmost position of the lever, we could then have an action $a \in A$ for pulling the lever to the left. These actions are owned by Anne, i.e., $\omega(a) = \mathit{anne}$. And for all non-rightmost positions we could have actions $a \in A$ for Bill pulling the lever to the right, i.e., with $\omega(a) = \mathit{bill}$. Since these pulling actions can be fully observed by all agents, they could be defined using only one single event with an appropriate precondition and effect. Finally, the goal formula would be $\gamma = (l \wedge p_l) \vee (r \wedge p_r)$.

A policy π for $\Pi = \langle s_0, A, \omega, \gamma \rangle$ is a partial mapping $\pi : S^{gl} \hookrightarrow \mathcal{P}(A)$ satisfying the conditions applicability (appl), uniformity (unif), and single-agent determinism (det).

(appl) Actions are applicable in states they are assigned to: for all $s \in S^{gl}, a \in \pi(s) : a$ is applicable in s.

(unif) If in some state the policy prescribes an action to an agent, it should prescribe the action also in states that the agent cannot distinguish: for all $s, t \in S^{gl}$ such that $s^{\omega(a)} = t^{\omega(a)}$, from $a \in \pi(s)$ follows $a \in \pi(t)$.

(det) For each state, the policy assigns at most one action per agent. I.e., there are no $s \in S^{\mathrm{gl}}$ and $a, a' \in \pi(s)$ with $a \neq a'$ and $\omega(a) = \omega(a')$.

Note that our definition of uniformity works because we consider bisimilar states to be equal. For two epistemically equivalent states s and s', it thus holds that $\pi(s) = \pi(s')$. The properties uniformity and applicability together imply *knowledge of preconditions*, the property that in each state, an agent who is supposed to perform a particular action must also know that the action is applicable in that state.

Note also that because of the uniformity we must allow policies to sometimes prescribe multiple actions of different owners to the same state. Imagine that from some state s the goal can be only reached via action a of agent i and from some state s' the goal can be only reached via action b of agent j. If there is a state s'' which is indistinguishable to s for agent i and to s' for agent j, then the policy should assign both actions a and b to state s''. To characterize the different outcomes of agents acting according to a common policy, we define the notion of policy executions:

An execution of a policy π from a global state s_0 is a maximal (finite or infinite) sequence of alternating global states and actions $(s_0, a_1, s_1, a_2, s_2, ...)$, such that for all $m \geq 0$,

(1) $a_{m+1} \in \pi(s_m)$, and

(2) $s_{m+1} \in \mathit{Globals}(s_m \otimes a_{m+1})$.

An execution is called successful for a planning task $\Pi = \langle s_0, A, \omega, \gamma \rangle$, if it is a finite execution $(s_0, a_1, s_1, ..., a_n, s_n)$ such that $s_n \models \gamma$.

Example 8. In the lever example, a policy could be, starting in the position with the lever being in the middle, Bill pulls the lever to the right and then to the right again. This policy satisfies all policy properties and its only execution is successful since the goal formula is satisfied in the end. However, while from the perspective of Bill, this is a reasonable policy, Anne cannot verify that the policy is successful because she does not know whether or not the right position is a goal position.

We now want to restrict our focus to policies that are guaranteed to achieve the goal after a finite number of steps. More formally, all of their executions must be successful. As in nondeterministic planning, such policies are called strong (Cimatti et al. 2003). For a planning task $\Pi = \langle s_0, A, \omega, \gamma \rangle$, a policy π is called strong if $s_0 \in \mathrm{Dom}(\pi) \cup \{s \in S^{gl} \mid s \models \gamma\}$ and for each $s \in \mathrm{Dom}(\pi)$, any execution of π from s is successful for Π. A planning task is called solvable if a strong policy for Π exists. For $i \in \mathcal{A}$, we call a policy i-strong if it is strong and $\mathit{Globals}(s_0^i) \subseteq \mathrm{Dom}(\pi) \cup \{s \in S^{gl} \mid s \models \gamma\}$.

When a policy is i-strong it means that the policy is strong and defined on all the global states that agent i cannot distinguish between in the initial state. It follows directly from the definition that any execution of an i-strong policy from any of those initially indistinguishable states will be successful. So if agent i comes up with an i-strong policy, then agent i knows the policy to be successful.

Example 9. The policy from above, with Bill pulling the lever to the right twice is *bill*-strong but not *anne*-strong.

Sometimes the agents cannot coordinate their plans but rather have to come up with them individually. Their policies can differ substantially, as agents often have different knowledge about the current states, applicable actions and action outcomes. To deal with agents having differing policies, we will define executions for *policy profiles*. A policy profile for a planning task Π is a family of policies $(\pi_i)_{i\in\mathcal{A}}$ where each π_i is a policy for Π. We assume actions to be instantaneous and executed asynchronously. This leads to the following definition of executions:

An execution of a policy profile $(\pi_i)_{i\in\mathcal{A}}$ is a maximal (finite or infinite) sequence of alternating global states and actions $(s_0, a_1, s_1, ...)$, such that for all $m \geq 0$,

(1) $a_{m+1} \in \pi_i(s_m)$ where $i = \omega(a_{m+1})$, and

(2) $s_{m+1} \in \mathit{Globals}(s_m \otimes a_{m+1})$.

Note that there are two different sources of nondeterminism: the nondeterminism resulting from the possibility of multiple policies prescribing actions for their respective agents (in condition 1) and the nondeterminism from nondeterministic action outcomes (in condition 2).

If all agents have one strong policy in common which all of them follow, then at execution time, the goal is guaranteed to be eventually reached. If, however, each agent acts on its individual strong policy, then the incompatibility of the individual policies may prevent the agents from reaching the goal, even though each individual policy is strong.

Bolander et al. (2018) have studied this in detail. They looked at different types of *planning agents*, which they defined as pairs (i, T), where $i \in \mathcal{A}$ is an agent name and T is a mapping from planning tasks to policies such that $T(\Pi)$ must be an i-strong policy for Π, whenever such a policy exists. The question they investigated was whether we can impose restrictions on a groups of agents $(i, T_i)_{i\in\mathcal{A}}$ so that all executions generated by this groups can be guaranteed to be successful. To simplify things, Bolander et al. (2018) have only looked at cases where all agents find *maximal strong policies* in the initial states of the planning task. These policies must be defined on all states (1) which are reachable from the initial state by arbitrary sequences of actions and (2) from which a strong policy exists. If all agents act with respect to such plans, re-planning is unnecessary and does not have to be considered.

A positive result was obtained in the general case for avoiding deadlocks (i.e., executions which end in a non-goal state where agents are waiting for each other to act). This was achieved by requiring planning agents to be *eager* and prefer own actions over other agents' actions in their plans whenever possible.

Concerning infinite executions, they have shown that there exists no type of planning agent that can prevent situations similar to the lever example. However, in cases where all agents have uniform knowledge, both deadlocks and infinite executions can be avoided if all agents are *optimally eager*, meaning if the planning agents only generate policies which are *subjectively optimal* (which means that these policies must have minimal *perspective-sensitive costs*), and that they prefer own actions over other agents' actions whenever possible without increasing the costs.

Let π be a strong policy for a planning task Π. The perspective-sensitive cost (or simply cost) of π from a state $s \in \mathrm{Dom}(\pi)$, denoted $\kappa_\pi(s)$ is defined as:

$$\kappa_\pi(s) = \begin{cases} 0 & \text{if there exists no } a \in \pi(s) \\ 1 + \max_{a\in\pi(s), s'\in \mathit{Globals}(s^{\omega(a)}\otimes a)} \kappa_\pi(s') & \text{else.} \end{cases}$$

The positive results that we get for our token-based approach will be based on the assumption that the planning agents act with respect to subjectively optimal policies.

4 Planning with Tokens

In the example from the introduction, the problem is that both agents want to pull the lever to the different goal configurations they know about, resulting in infinite executions. This problem can be eliminated by introducing a token such that only the agent that possesses the token is allowed to act. The goal of this tokenization is that only one agent gets to make a move at any time. Once the agent is done with their own actions, they can pass on the token to the next agent.

There are different ways to implement such a token. First, we have to add actions with which agents can pass the token to the next agent. Furthermore, we have to define how the first agent obtains the token. The way we decided to implement this is via an action with which any agent can take the token initially. Note that there are other possibilities. For example, we could initially assign the token randomly to one of the agents. The disadvantage with this approach is that there are planning tasks where only some but not all of the agents find a plan from their local perspectives. In these cases, assigning the token randomly would lead to an unsolvable task. We can also leave the burden of designating the first agent to act to the modeler of the planning task. But then again, to be sure to not make a solvable planning task unsolvable, the modeler would have to already know which of the agents can find a plan and which of the agents cannot.

4.1 Tokenization of Planning Tasks

We will now formally describe a function that tokenizes any given planning task. The idea is that we introduce additional predicates $\mathtt{t}_i$ for all agents $i \in \mathcal{A}$ with the intuitive meaning of "agent i possesses the token". We can then use these propositions in the preconditions of our tokenized actions. We first define the tokenization of epistemic states.

Definition 1. Let $s = (\langle W, (\sim_i)_{i\in\mathcal{A}}, V\rangle, W_d)$ be an epistemic state with proposition set P. Then the *tokenization of state* s is a new epistemic state $tok(s) = (\langle W, (\sim_i)_{i\in\mathcal{A}}, V'\rangle, W_d)$, which has the proposition set $P \cup \{\mathtt{t}_i \mid i \in \mathcal{A}\}$ and where $V' = V \cup \{\mathtt{t}_i \mapsto \emptyset \mid i \in \mathcal{A}\}$.

Furthermore, for any agent $i \in \mathcal{A}$, we define the i-tokenization $tok^i(s)$ of state s analogously, but with $V' = V \cup \{\mathtt{t}_i \mapsto W\} \cup \{\mathtt{t}_j \mapsto \emptyset \mid j \in \mathcal{A}, j \neq i\}$.

This means in $tok(s)$ no agent has the token yet and in $tok^i(s)$, the token is owned by agent i. Token ownership is always common knowledge between the agents.

We can now define the tokenization of epistemic actions.

Definition 2. Let $a = (\langle E, (\sim_i)_{i\in\mathcal{A}}, \mathrm{pre}, \mathrm{eff}\rangle, E_d)$ be an epistemic action and $i \in \mathcal{A}$ be an agent. Then

the *i-tokenization of action* a is a new epistemic action $tok^i(a) = (\langle E, (\sim_i)_{i\in\mathcal{A}}, \mathrm{pre}', \mathrm{eff}\rangle, E_d)$ with new preconditions $\mathrm{pre}'(e) = \mathsf{t}_i \wedge \mathrm{pre}(e)$ for all $e \in E$.

Since tokenized actions use the token propositions, they can only be applied to tokenized states and their successors. We can finally define the tokenization of planning tasks.

Definition 3. Let $\Pi = \langle s_0, A, \omega, \gamma\rangle$ be a planning task. We define the *tokenization of task* Π as $tok(\Pi) = \langle s_0', A', \omega', \gamma\rangle$ as follows:

- $s_0' = tok(s_0)$ where $tok(s_0)$ is the tokenization of s_0,
- $A' = \{tok^{\omega(a)}(a) \mid a \in A\} \cup \{\mathsf{takeTok}^i \mid i \in \mathcal{A}\} \cup \{\mathsf{giveTok}^{ij} \mid i, j \in \mathcal{A}, i \neq j\}$, where for all $i \neq j \in \mathcal{A}$
  - $\mathsf{takeTok}^i$ is an action consisting of a single event with precondition $\bigwedge_{k\in\mathcal{A}} \neg\mathsf{t}_k$ and effect t_i,
  - $\mathsf{giveTok}^{ij}$ is an action consisting of a single event with precondition t_i and effect $\neg\mathsf{t}_i \wedge \mathsf{t}_j$, and
- $\omega'(a') = \begin{cases} \omega(a) & \text{if } a' = tok(a) \text{ for some } a \in A \\ i & \text{if } a' = \mathsf{takeTok}^i \text{ for some } i \in \mathcal{A} \\ i & \text{if } a' = \mathsf{giveTok}^{ij} \text{ for some } i, j \in \mathcal{A}. \end{cases}$

Note that the tokenization of a planning task with proposition set P will have the proposition set $P \cup \{\mathsf{t}_i \mid i \in \mathcal{A}\}$. When talking about tokenized planning tasks, we will from now on use $S^{\mathrm{gl}}_{\mathrm{tok}}$ to denote the set of states that we get by tokenizing all the states in S^{gl}, i.e., $S^{\mathrm{gl}}_{\mathrm{tok}} = \{tok(s) \mid s \in S^{\mathrm{gl}}\} \cup \{tok^i(s) \mid s \in S^{\mathrm{gl}}, i \in \mathcal{A}\}$. Furthermore, we will use the notation $tok(A) = A'$ for the set A' of all tokenized actions from the action set, as defined above.

While we assume unit action costs in Π, it is not immediately obvious which costs we should assign to giveTok actions. Assigning them unit costs as well may be the most obvious option, but comes at the risk of introducing an unwarranted bias towards few token passings at the expense of overly costly subplans consisting of "proper" (non-token-passing) actions. On the other hand, assigning them costs of zero makes token passing free and preserves optimality. Unfortunately, with zero-cost token passings, our main theorem (Theorem 1) that states that introducing tokens prevents infinite executions, becomes invalid, since then, there can be maximal subjectively optimal i-strong policies that still lead to infinite executions, more specifically, infinite rounds of token passing. As a compromise between zero costs and unit costs, we may also assign costs of a sufficiently small $\varepsilon > 0$ to all giveTok actions. In the following, we will not study the question of those action costs further, but rather assume that all actions have unit costs.

4.2 No More Infinite Executions

Given this formalization, we can now show that we can prevent the existence of infinite executions by transforming the planning task into a tokenized planning task and requiring the agents to use subjectively optimal policies.

Theorem 1. *Let Π be a tokenized planning task and let $(\pi_i)_{i\in\mathcal{A}}$ be a profile of maximal subjectively optimal i-strong policies for Π. Then all executions of $(\pi_i)_{i\in\mathcal{A}}$ are finite.*

Proof sketch. For the case where the initial state is already a goal state and no agent takes the token, we do not have to prove anything. For the other case, let us assume that agent j has the token in some given state s with costs $\kappa_{\pi_j}(s) = c$. We can distinguish the following cases:

- If $c = 0$, then j will not perform any more action and the execution is finished.
- If $c > 0$ and $\pi_j(s) \notin \{\mathsf{giveTok}^{jk} \mid k \in \mathcal{A}\}$ then j will still have the token in each of the subjective successor states $s' \in \mathit{Globals}(s^j \otimes \pi_j(s))$ of s. By the definition of subjective costs, we have $\kappa_{\pi_j}(s') \leq c - 1$.
- If $c > 0$ and $\pi_j(s) = \mathsf{giveTok}^{jk}$ for some agent $k \in \mathcal{A}$ then agent k will have the token in each of the subjective successor states $s' \in \mathit{Globals}(s^j \otimes \pi_j(s))$ of s. By the definition of subjective costs we have $\kappa_{\pi_j}(s') \leq c - 1$. Since k plans optimally, $\kappa_{\pi_j}(s')$ must be an overestimate of $\kappa_{\pi_k}(s')$ and we thus have $\kappa_{\pi_k}(s') \leq \kappa_{\pi_j}(s') \leq c - 1$.

Since the value of c decreases by at least 1 after each action and can never fall below 0, any execution must eventually stop. □

Example 10. It is easy to see how the tokenization works with our lever example: Initially, both agents want to take the token. After one of the agents has obtained the token (which is decided nondeterministically), the agent moves the lever all the way to its goal position. The token remains with that agent and the execution is finished.

4.3 Policy Existence for Tokenized Tasks

Whereas tokenization makes sure that infinite executions disappear, unfortunately, it does not preserve all i-strong policies from the original task. An issue arises in scenarios such as the following.

Example 11. Assume that agent 1 initially holds an object and knows that either agent 2 or agent 3 desires that object, but does not know who. Without tokens, agent 1 can place the object on the table and expect the agent desiring the object to pick it up. With tokens, agent 1 can still place the object on the table, but then has to pass the token to the correct agent. Without knowing who that is, the policy cannot be tokenized in a straight forward way. Assuming that one of the agents does not even know that the other agent desires the object, giving the token to this agent would even lead to a dead-end state. In this case, while there exists a 1-strong policy for the original task, there is no 1-strong policy for the tokenized version of the task.

The underlying problem can be characterized by a property of agent 1's policy π_1: The state s in which agent 1 puts the object on the table has a global successor state s' in which one of the other agents must act, but which is indistinguishable for agent 1 to another state s'' in which the third agent must act. Formally, we have $\pi_1(s') = \{\mathsf{pickUp}\}$ and $\pi_1(s'') = \{\mathsf{pickUp}'\}$ with $\omega(\mathsf{pickUp}) \neq \omega(\mathsf{pickUp}')$.

Intuitively, to avoid this problem, we need to require that after each action of an agent i leading to some non-terminal state s', the agent i can identify a unique agent $next(i, s')$ which can act next. In the tokenized version of a policy, the

first agent would pass the token to exactly this agent or, in the case where $next(i, s') = i$, keep it.

Definition 4. We say a policy $\pi : S^{gl} \hookrightarrow \mathcal{P}(A)$ satisfies the *knows-the-next-agent* property (KNA) if there exists a function $next : \mathcal{A} \times S^{\text{gl}} \hookrightarrow \mathcal{A}$, which we call the *next agent function*, with the following property: For each state $s \in S^{gl}$, action $a \in \pi(s)$ owned by agent $i = \omega(a)$, non-terminal successor state $s' \in \mathit{Globals}(s \otimes a)$, i.e. for which $\pi(s')$ is defined and nonempty, and all states $s'' \in \mathit{Globals}(s'^{i})$ which are indistinguishable to s' for i, there is an action $a' \in \pi(s'')$ such that $\omega(a') = next(i, s')$.

Note that for some policies there is more than one next agent function. Given a finite policy, it is easy to either construct a next agent function or to prove that no such function exists. This can be done by successively looking at each triple $(s, a, s') \in S^{\text{gl}} \times A \times S^{\text{gl}}$ such that $a \in \pi(s)$ and $s' \in \mathit{Globals}(s \otimes a)$. If there is an agent $i \in \mathcal{A}$ who owns for all states $s'' \in \mathit{Globals}(s'^{\omega(a)})$ an action $a' \in \pi(s'')$, we can assign it to $next(\omega(a), s')$. If there is no such agent, we know that there is no next action function and that the policy thus does not satisfy the KNA property. However, if there is always such an agent and thus the KNA property is satisfied, this means that the acting agents will be able to identify the agents to which the token can be passed next from their own perspectives and without the need of external coordination.

4.4 Tokenization of Policies

We can now tokenize *policies* which satisfy the KNA property in the obvious way: before each regular action, we add a token-passing action to the owner of the next action if necessary.

Definition 5. Let $\pi : S^{\text{gl}} \hookrightarrow \mathcal{P}(A)$ be a policy that satisfies the KNA property and let $next$ be a next agent function for π. Then we define the *tokenization of the policy* π with $next$ as $tok_{next}(\pi) : S^{\text{gl}}_{\text{tok}} \hookrightarrow \mathcal{P}(tok(A))$, where

(1) For all states $s' = tok(s) \in S^{\text{gl}}_{\text{tok}}$, i.e., in which no agent has the token, we have $tok_{next}(\pi)(s') = \{\mathsf{takeTok}^{\omega(a)} \mid a \in \pi(s)\}$.

(2) For all states $s' = tok^i(s)$, i.e., such that some agent i has the token, in the case where $next(i, s') = j \neq i$, we have $tok_{next}(\pi)(s') = \{\mathsf{giveTok}^{ij}\}$.

(3) For all states $s' = tok^i(s)$, i.e., such that some agent i has the token, in the case where $next(i, s') = i$, we have $tok_{next}(\pi)(s') = \{tok(a) \mid a \in \pi(s), \omega(a) = i\}$.

We are now ready to prove that, under the assumption of the KNA property, tokenization does indeed preserve i-strong policies.

Theorem 2. *Let π be a strong policy for a planning task Π which satisfies the KNA property and let $next$ be a next agent function for π. Then the tokenized policy $\pi' = tok_{next}(\pi)$ is also a strong policy for the tokenized planning task $\Pi' = tok(\Pi)$.*

Proof sketch. We first have to show that the policy properties are satisfied for π'. For the token taking and passing actions from the first two cases of Definition 5, the applicability condition is trivially satisfied: The token can be taken because it is still on the table, and it can be passed along because it is possessed by the correct agent. For the third case, the applicability condition is also satisfied, since all actions that are assigned to the state belong to the agent that possesses the token. Uniformity is slightly more complicated: In the first case, uniformity follows directly from the uniformity of the original policy. In the second case, uniformity follows from our definition of next agent functions which guarantees that $next(i, s') = next(i, s'')$ for all states s'' which are indistinguishable to s' for agent i. Finally, in the third case, uniformity follows for the same reason, together with the fact that all actions that were assigned to each equivalence class of indistinguishable states in the original policy are now again assigned to the corresponding states in the tokenized policy.

We can now prove that the policy is strong. We first have to show that the initial state is again either already a goal state or that it is contained in the domain of the policy. This follows directly from strength of the original policy: Either the initial state of Π is a goal state, in which case the initial state of Π' will be a goal state for Π' as well (since the tokens cannot be part of the goal formula). Or otherwise, if the initial state of Π is contained in π, its tokenized version, which is the initial state for Π' will also be contained in π'.

We then have to show that each execution of π' is successful. We first have to note that for each execution of π' in Π', there is a corresponding execution of $\pi \in \Pi$ using the untokenized actions and omitting the token taking and passing actions. Note that we cannot have any dead ends in π', because there are no dead ends in π and because our tokenization ensures that whenever there are transitions from one state to another in π, at least one of these transitions must have a corresponding transition in π' (which might include a token passing action before the actual action). Since any execution of π ends in a goal state, also any execution of π' must end in a goal state. □

5 Tokenization and Re-Planning

Note that in our analysis, so far we have always assumed that all agents act with respect to maximal strong policies which they form in the initial state of the planning task. In practice, this is a big limitation.

First of all, finding maximally strong policies can be very expensive since the state space can be huge and maximally strong policies must be defined for all reachable states from which a strong policy exists. Even in decidable fragments of DEL planning it stands to reason that finding maximally strong plans can be much more difficult than finding optimal plans. Also, we cannot deal with situations yet where only one of the agents has an i-strong policy from the start, and where the other agents will find their plans and join taking part in the execution only after a few actions of agent i, which is one of the main advantages of i-strong policies.

An alternative to using maximal strong policies is to employ a *re-planning* regime. This approach has been taken by Nebel et al. (2019) in the specialized setting of *multi-agent path finding with destination uncertainty*, which has been

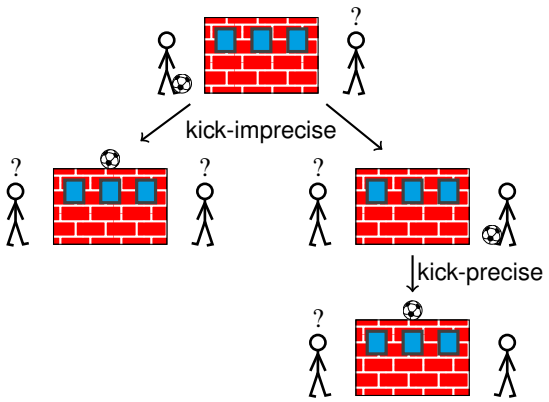

Figure 2: Two agents, a ball and a roof.

originally formalized as DEL planning task (Bolander et al. 2018). Nebel et al. showed that even in this restricted class of implicit-coordination problems, and using a re-planning regime, optimally eager agents can still produce infinite executions. In the following, we formalize re-planning executions for the general case and analyze whether tokenization helps to prevent infinite executions.

Definition 6. A *re-planning execution* of a group of planning agents $(i, T_i)_{i\in\mathcal{A}}$ is a maximal (finite or infinite) sequence of alternating global states and actions $(s_0, a_1, s_1, \ldots)$, such that for all $m \geq 0$,

(1) $a_{m+1} \in T_i(\Pi|_{s_m})(s_m)$ where $i = \omega(a_{m+1})$, and
(2) $s_{m+1} \in \mathit{Globals}(s_m \otimes a_{m+1})$,

where $\Pi|_{s_m}$ is the same planning task as Π, but where the initial state has been replaced by s_m. We call such an execution successful if it is a finite execution $(s_0, a_1, s_1, \ldots, s_n)$ such that $s_n \models \gamma$.

Unfortunately, Theorem 1 can not be transferred directly to re-planning agents. Consider the following counter-example, as depicted in Figure 2.

Example 12. There are two agents standing at opposing sides of a building. Since the building is between them, they cannot see each other. The goal of the planning task is that a football, which is initially on the side of the left agent, ends up on top of the building. The agents can observe the ball only if it is on their respective side of the building. In our depiction, an agent that has a question mark on the top of its head is unsure whether the ball is already on the roof or with the other agent. Both agents have an action to kick the ball. However, the left agent cannot aim as accurately as the right agent. If he kicks the ball, it might either land on the roof or on the side of the other agent. The right agent has sufficient aim, so after she kicks the ball, she will be certain that the ball is on the rooftop. The left agent will not know when this happens, since he cannot see the shot.

Figure 2 shows a policy that is i-strong for both agents. Interestingly, it contains a goal state in which both agents do not know that the goal is satisfied. Since the figure is also a depiction of the entire state space of the planning task (and since in the goal states, no agent can act), it is clear that re-planning agents who act with respect to i-strong policies will always generate successful executions.

However, in the tokenization of the planning task, the fact that the agents now do re-planning leads to an infinite execution in the case where the imprecise kicking action of the left agent already succeeds landing the ball on the roof. This is because the agent must pass the token to the agent on the right to cover for the contingency where the ball went over the roof. The agent on the right must then re-plan and cover for the contingency where the ball is still with the left agent and pass the token directly back, which leads to the same state as before. Thus the agents will effectively pass the token back and forth.

The reason for the infinite execution is that in the end, none of the agents knows that the goal has been reached, and both of the agents consider it possible that the other agent still has some work to do. If we allowed the agents to do some kind of forward induction, i.e., reasoning about the motive behind the other agent's action, they could maybe infer from the fact that the token has been passed to them that the ball must already be on the roof. However, it is unclear how a reasonable solution concept for implicit coordination with forward induction would look like and this is a major topic for future research.

A more direct way to prevent such situations and which works with our existing notion of strong policies is to ensure that each policy is *stable*, in the sense that in terminal states, all agents who have applicable actions know that the goal has already been reached.

Definition 7. We say that a policy $\pi : S \hookrightarrow \mathcal{P}(A)$ is *stable* for a goal γ, if for all states $s \in S$, actions $a \in \pi(s)$ and successor states $s' \in \mathit{Globals}(s \otimes a)$ for which there is no action $a' \in \pi(s')$ (i.e., terminal states), we have $s' \models K_i\gamma$ for all agents i who have actions that are applicable in s'^i.

Although this approach seems to be very restrictive in general, it makes a lot of sense for tokenized planning tasks because in any given state of an execution, with the exception of the initial state, there is only one agent that has applicable actions anyway. We can now verify that there are no infinite executions for tokenized planning tasks, given all agents are optimally eager and produce only stable plans.

Theorem 3. *Given a tokenized planning task $tok(\Pi)$ with goal γ and a group $(i, T_i)_{i\in\mathcal{A}}$ of optimally eager planning agents, such that for arbitrary states $s \in S^{gl}_{tok}$, the agents produce only policies $T_i(tok(\Pi)|_s)$ which are stable for γ. Then all re-planning executions of $(i, T_i)_{i\in\mathcal{A}}$ are finite.*

Proof sketch. The proof works analogously to the proof of Theorem 1, with the difference that agents can have a different policy for each state. The first two cases are identical: If the agent who has the token finds a policy with cost 0, the execution is finished. And if the agent who has the token has an action that is not a token passing action, then in the successor state he will have the token again and be guaranteed to find a policy where the cost decreases by at least 1.

The difference is in the third case where the token is passed from an agent j to another agent k. Here, the proof of Theorem 1 relies on the assumption that if agent k wants

to apply an action after obtaining the token, then this action must also be part of a policy that is subjectively optimal from the initial state. For example, in the rooftop example with tokens, there is no policy starting from the initial state such that if the ball lands on the roof immediately and the token is passed to the right agent, this agent will try to give the token back (because this would create a cycle in the policy). Instead, the execution would stop with the agent not knowing that the goal has been reached. With re-planning and without the stability property, the right agent would find a new plan in which the token is given back and then, if the goal has already been reached, the left agent does nothing, or otherwise the policy proceeds as in Figure 2. However, in the execution, after the token has been given back, the left agent would also re-plan and pass the token straight back.

We will now look at what happens with policies satisfying our stability property. As before, we assume that the token is passed from agent j to k, leading from a state s to a state s' in which only agent k has applicable actions. We distinguish between the following cases:

- If $\pi_j = T_j(tok(\Pi)|_s)$ assigns some action to state s', this action must be owned by agent k and it is thus clear that π_j must also be a k-strong policy for s'. Since agent k plans optimally, $\pi'_k = T_k(tok(\Pi)|_{s'})$ will be a policy with $\kappa_{\pi'_k}(s') \leq \kappa_{\pi_j}(s') \leq \kappa_{\pi_j}(s) - 1$.
- If π_j assigns no action $a' \in \pi_j(s')$ to s', then because of the stability condition (since at least the token passing actions are applicable), agent k must know that the goal is satisfied in s' and thus $\pi'_k = T_k(tok(\Pi)|_{s'})$ will be a policy with $\kappa_{\pi'_k}(s') = 0$.

Since, as before, the cost of the policy belonging to the agent who acts in each state of the execution decreases in each step by at least 1 and can never fall below 0, any execution must eventually stop. □

Instead of defining a new agent type that also requires generated policies to be stable, we can modify the planning tasks to enforce that each strong policy for the planning task must be stable. This is especially easy if the planning task is already tokenized: If the propositions $\mathtt{t}_i$ denote that an agent i has the token, we can simply change the goal formula from γ to $\gamma' = \gamma \wedge \bigwedge_{i \in \mathcal{A}}(\mathtt{t}_i \rightarrow K_i\gamma)$. Thus, in each new goal state, an agent who has the token must know that γ is satisfied and, by introspection, also that γ' itself is satisfied. Therefore, any strong policy trivially satisfies the stability condition, and thus agents never have reason to pass on the token to another agent in goal states.

Example 13. Imagine that in our football example, the left agent has an additional action using which also he can pass the ball to the right agent with certainty, e.g., by throwing the ball instead of kicking it. Then, if we encode our stability condition into the goal formula, and given both agents are optimally eager, a successful execution is guaranteed. The left agent will throw the ball to the right agent who will then kick it onto the rooftop. Without the stability condition, the left agent might just as well decide to kick the ball, which might result in both agents passing the tokens back and forth, as seen previously.

6 Conclusion

We have looked at how the tokenization of epistemic planning tasks can be used to mitigate the problem of infinite executions. We have shown that in the case of agents acting with respect to maximally strong policies, tokenization successfully prevents infinite executions. However, there are planning tasks for which, despite the existence of a strong policy, there is no strong policy for the tokenized task. The strong policies of the original task which can be tokenized are characterized by the knows-the-next-agent property.

So far, we have considered only a tokenization in which the token is passed directly from one agent to another agent. This is a big restriction since we can easily construct planning tasks where the first agent to act knows that one of the other agents can finish the plan, but not which of the agents. Even if for this task infinite executions are not possible, the fact that no policy satisfies the KNA property implies that there are no strong policies for the tokenization. A possible remedy could be to allow the agents to pass the token to multiple agents at once, one of which then has to decide to take the next action and continue passing the token. However, without additional constraints (e.g., on the subsets of agents that the token can be passed to), using such a tokenization we can easily end up with infinite executions again. This is because after each action the token can be passed back to all of the agents, and thus each strong policy of the original task also corresponds to a strong policy for the tokenization. Thus, if we can get infinite executions for the original task, we can also get infinite executions for this kind of tokenization. Note that the solution policies for most of the tasks considered in literature, including multi-agent path finding with destination uncertainty, satisfy the KNA property meaning that the simple tokenization from this paper is sufficient to solve these tasks while avoiding infinite executions.

Importantly, if there is a policy for the original planning task that can be tokenized, it can be found by directly by planning for the tokenized task, which is not a more complex problem than planning for the original task. This is because the tokenization increases the size of the input task only linearly. Note that while in the general case the strong policy existence problem is undecidable, there exist tractable fragments (Engesser and Miller 2020).

We have furthermore shown that for re-planning agents, tokenization alone is not sufficient to prevent infinite executions. However, we can employ an additional stability condition that can be encoded into the goal formula.

For future work, we plan to investigate tokenized planning tasks in the context of forward induction. This would mean allowing the agents to infer knowledge by reasoning about the motive of other agents' actions. For example, an agent could pass the token to another agent to signal that that agent has an action available which progresses towards the goal, even if the agent does not know that yet. However, forward induction has not been formalized in the context of epistemic planning for implicit coordination so far. Supporting such reasoning would arguably require a solution concept that goes beyond what is possible with strong policies.

References

Blackburn, P.; de Rijke, M.; and Venema, Y. 2001. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press.

Bolander, T.; Engesser, T.; Mattmüller, R.; and Nebel, B. 2018. Better eager than lazy? How agent types impact the successfulness of implicit coordination. In *Proceedings of KR 2018*, 445–453.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence* 147(1–2):35–84.

Engesser, T., and Miller, T. 2020. Implicit coordination using FOND planning. In *Proceedings of AAAI 2020*, 7151–7159.

Engesser, T.; Bolander, T.; Mattmüller, R.; and Nebel, B. 2017. Cooperative epistemic multi-agent planning for implicit coordination. In *Proceedings of M4M 2017*, 75–90.

Huang, X.; Fang, B.; Wan, H.; and Liu, Y. 2017. A general multi-agent epistemic planner based on higher-order belief change. In *Proceedings of IJCAI 2017*, 1093–1101.

Karpas, E.; Shleyfman, A.; and Tennenholtz, M. 2017. Automated verification of social law robustness in STRIPS. In *Proceedings of ICAPS 2017*, 163–171.

Kominis, F., and Geffner, H. 2015. Beliefs in multiagent planning: From one agent to many. In *Proceedings of ICAPS 2015*, 147–155.

Le, T.; Fabiano, F.; Son, T. C.; and Pontelli, E. 2018. EFP and PG-EFP: epistemic forward search planners in multi-agent domains. In *Proceedings of ICAPS 2018*, 161–170.

Loucks, L. K., and Shaheen, A. A. 1997. System and method for multi-level token management for distributed file systems. US Patent 5,634,122.

Makki, K.; Banta, P.; Been, K.; Pissinou, N.; and Park, E. K. 1992. A token based distributed K mutual exclusion algorithm. In *Proceedings of SPDP 1992*, 408–411.

Maubert, B.; Pinchinat, S.; and Schwarzentruber, F. 2019. Reachability games in dynamic epistemic logic. In *Proceedings of IJCAI 2019*, 499–505.

Muise, C.; Belle, V.; Felli, P.; McIlraith, S.; Miller, T.; Pearce, A. R.; and Sonenberg, L. 2015. Planning over multi-agent epistemic states: A classical planning approach. In *Proceedings of AAAI 2015*, 3327–3334.

Nebel, B.; Bolander, T.; Engesser, T.; and Mattmüller, R. 2019. Implicitly coordinated multi-agent path finding under destination uncertainty: Success guarantees and computational complexity. *Journal of Artificial Intelligence Research* 64:497–527.

Nir, R., and Karpas, E. 2019. Automated verification of social laws for continuous time multi-robot systems. In *Proceedings of AAAI 2019*, 7683–7690.

van Ditmarsch, H.; van der Hoek, W.; and Kooi, B. 2007. *Dynamic Epistemic Logic*, volume 337 of *Synthese Library*. Springer.