Dynamic Epistemic Logic

6. Plan Execution

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Bernhard Nebel and Robert Mattmüller

July 8th, 2019

Credits

UNI FREIBURG

Credits:

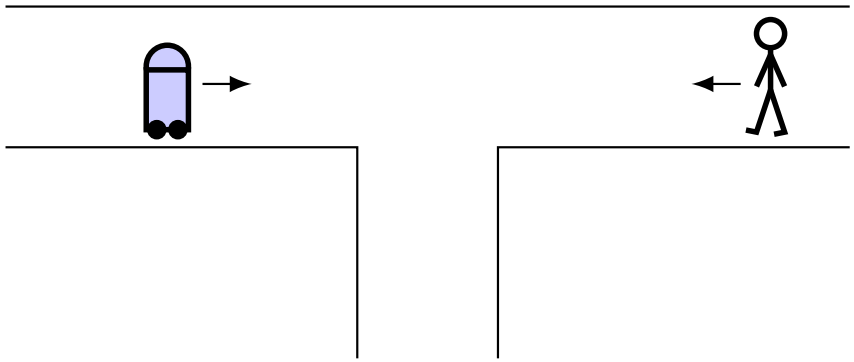
- The first half of the following slides is taken from Thorsten Engesser's KR 2018 presentation on the paper "Better Eager Than Lazy? How Agent Types Impact the Successfulness of Implicit Coordination".
- The second half is taken directly from the paper.

UNI FREIBURG

Implicit Coordination

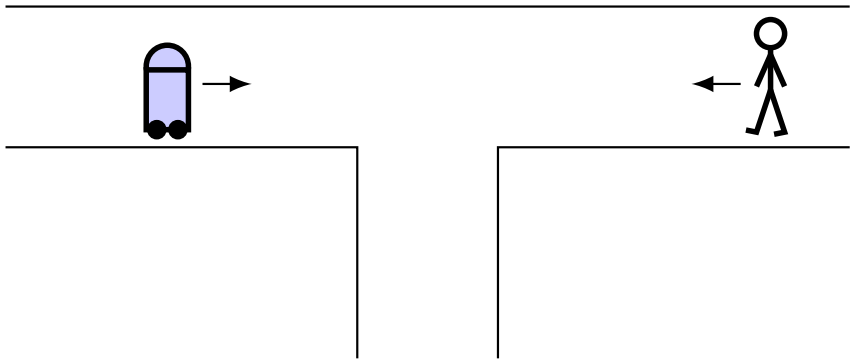
Another Example: Multi-Agent Pathfinding

A robot and a human meet at a narrow corridor intersection



Another Example: Multi-Agent Pathfinding

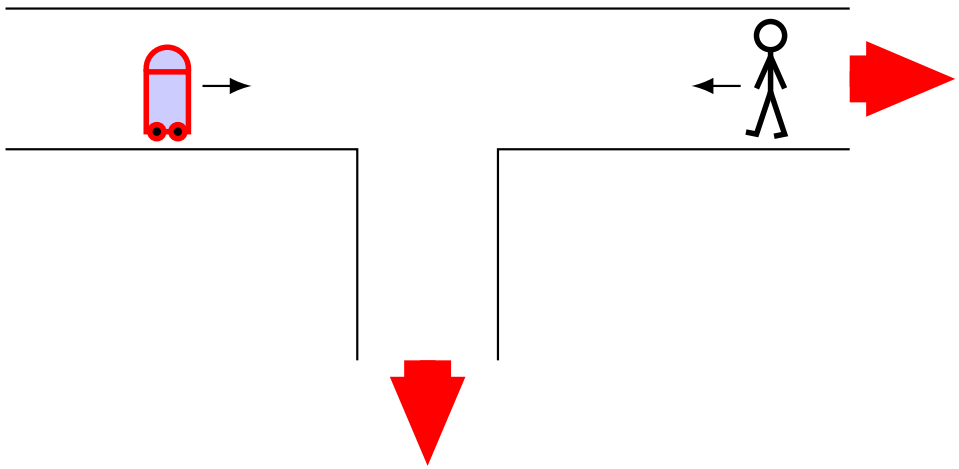
A robot and a human meet at a narrow corridor intersection



- It is common knowledge that

Another Example: Multi-Agent Pathfinding

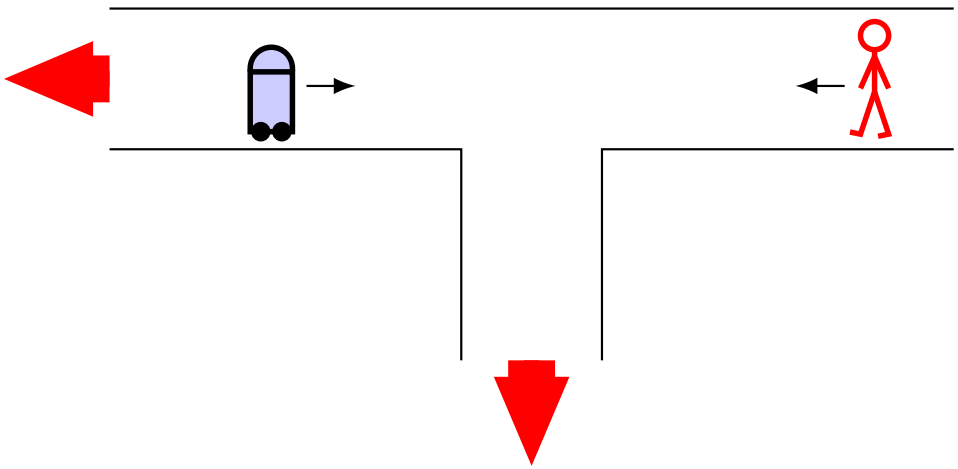
A robot and a human meet at a narrow corridor intersection



- It is common knowledge that
  - the human does not know the robot's goal (east or south)

Another Example: Multi-Agent Pathfinding

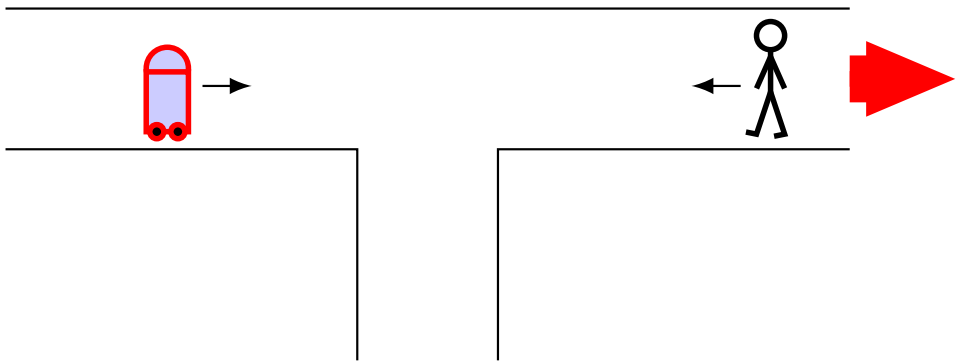
A robot and a human meet at a narrow corridor intersection



- It is common knowledge that
  - the human does not know the robot's goal (east or south)
  - the robot does not know the human's goal (west or south)

Another Example: Multi-Agent Pathfinding

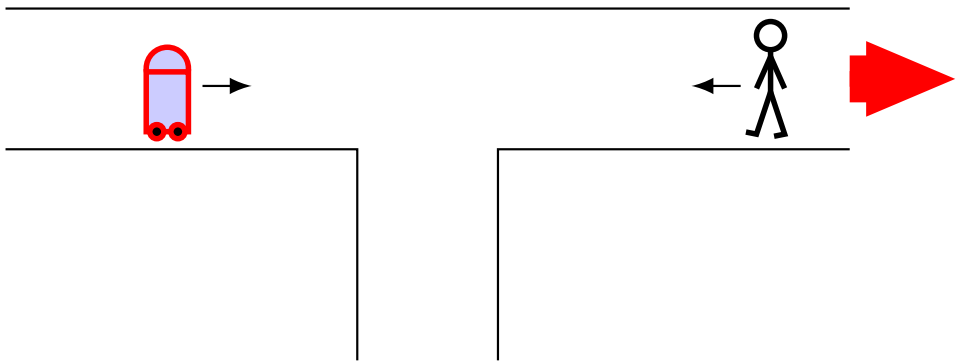
A robot and a human meet at a narrow corridor intersection



- It is common knowledge that
  - the human does not know the robot's goal (east or south)
  - the robot does not know the human's goal (west or south)
- You are the robot and want to go to the east

Another Example: Multi-Agent Pathfinding

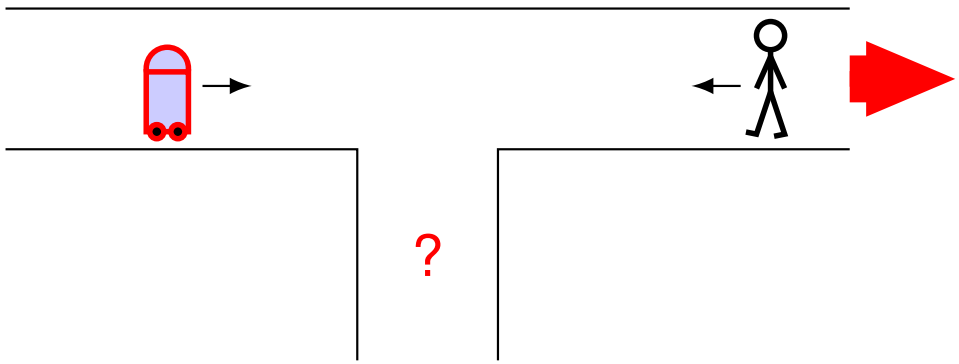
A robot and a human meet at a narrow corridor intersection



- It is common knowledge that
  - the human does not know the robot's goal (east or south)
  - the robot does not know the human's goal (west or south)
- You are the robot and want to go to the east
- You cannot communicate with the human

Another Example: Multi-Agent Pathfinding

A robot and a human meet at a narrow corridor intersection



- It is common knowledge that
  - the human does not know the robot's goal (east or south)
  - the robot does not know the human's goal (west or south)
- You are the robot and want to go to the east
- You cannot communicate with the human

Should you wait or should you go out of the way (south)?

UNI FREIBURG

Agent Types, Executions

Executions of Multiple Agents' Plans

UNI FREIBURG

- If all agents find identical/compatible policies, everything works out fine
- How to deal with incompatible policies?
  1. Do re-planning if something unexpected happens, or
  2. Use maximal strong policies to plan for all contingencies in advance
- Can we guarantee that application of agents' individual policies leads to goal?
- We have to look at the executions of policy profiles

⇝ Characterization of different agent types (= restrictions on allowed policies)

The *Lazy* Agent Type

UNI FREIBURG

An agent is called lazy if he chooses another agents' action whenever allowed
(= it is part of a strong policy).

Example task: Knock, knock! Who gets the door?

The goal, for Jim and John, is to go to the door and let Sarah in.
Both agents are perfectly capable of doing so in one action.

What happens if both agents are lazy?

$\pi_{Jim} = \{s_0 \mapsto \{\textit{john-gets-door}\}\}$, $\pi_{John} = \{s_0 \mapsto \{\textit{jim-gets-door}\}\}$

- There is only one execution which is unsuccessful
- If agents are eager, there are two successful executions

The *Naively Eager* Agent Type

UNI FREIBURG

An agent is called naively eager if he chooses an own action whenever allowed
(= it is part of a strong policy).

The *Naively Eager* Agent Type

UNI FREIBURG

An agent is called naively eager if he chooses an own action whenever allowed
(= it is part of a strong policy).

Prevents deadlocks (situations where all agents wait)

The *Naively Eager* Agent Type

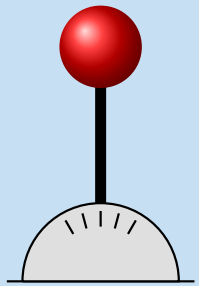
UNI FREIBURG

An agent is called naively eager if he chooses an own action whenever allowed
(= it is part of a strong policy).

Prevents deadlocks (situations where all agents wait)

Example task: Pulling the lever (I)

The goal, for **L**isa and **R**alph, is to pull the lever either fully to the left or to the right. Lisa can only pull left while Ralph can only pull right.



The *Naively Eager* Agent Type

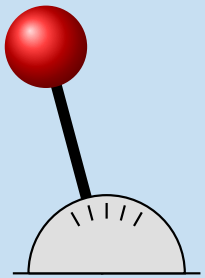
UNI FREIBURG

An agent is called naively eager if he chooses an own action whenever allowed
(= it is part of a strong policy).

Prevents deadlocks (situations where all agents wait)

Example task: Pulling the lever (I)

The goal, for **L**isa and **R**alph, is to pull the lever either fully to the left or to the right. Lisa can only pull left while Ralph can only pull right.



The *Naively Eager* Agent Type

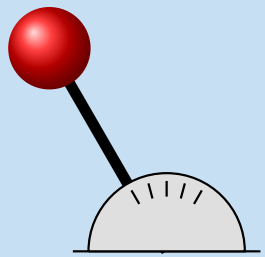
UNI FREIBURG

An agent is called naively eager if he chooses an own action whenever allowed
(= it is part of a strong policy).

Prevents deadlocks (situations where all agents wait)

Example task: Pulling the lever (I)

The goal, for **L**isa and **R**alph, is to pull the lever either fully to the left or to the right. Lisa can only pull left while Ralph can only pull right.



The *Naively Eager* Agent Type

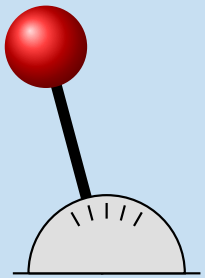
UNI FREIBURG

An agent is called naively eager if he chooses an own action whenever allowed
(= it is part of a strong policy).

Prevents deadlocks (situations where all agents wait)

Example task: Pulling the lever (I)

The goal, for **L**isa and **R**alph, is to pull the lever either fully to the left or to the right. Lisa can only pull left while Ralph can only pull right.



The *Naively Eager* Agent Type

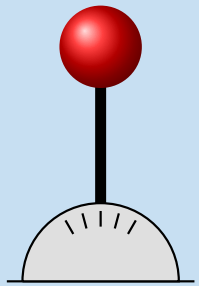
UNI FREIBURG

An agent is called naively eager if he chooses an own action whenever allowed
(= it is part of a strong policy).

Prevents deadlocks (situations where all agents wait)

Example task: Pulling the lever (I)

The goal, for **L**isa and **R**alph, is to pull the lever either fully to the left or to the right. Lisa can only pull left while Ralph can only pull right.



The *Naively Eager* Agent Type

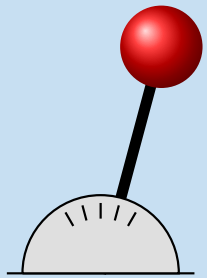
UNI FREIBURG

An agent is called naively eager if he chooses an own action whenever allowed
(= it is part of a strong policy).

Prevents deadlocks (situations where all agents wait)

Example task: Pulling the lever (I)

The goal, for **L**isa and **R**alph, is to pull the lever either fully to the left or to the right. Lisa can only pull left while Ralph can only pull right.



The *Naively Eager* Agent Type

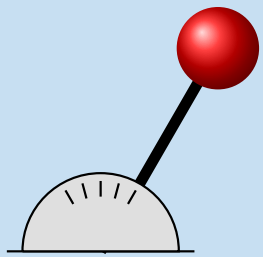
UNI FREIBURG

An agent is called naively eager if he chooses an own action whenever allowed
(= it is part of a strong policy).

Prevents deadlocks (situations where all agents wait)

Example task: Pulling the lever (I)

The goal, for **L**isa and **R**alph, is to pull the lever either fully to the left or to the right. Lisa can only pull left while Ralph can only pull right.



The *Naively Eager* Agent Type

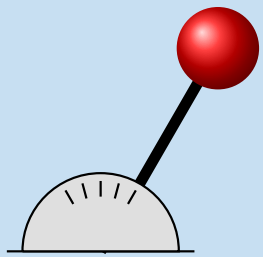
UNI FREIBURG

An agent is called naively eager if he chooses an own action whenever allowed
(= it is part of a strong policy).

Prevents deadlocks (situations where all agents wait)

Example task: Pulling the lever (I)

The goal, for **L**isa and **R**alph, is to pull the lever either fully to the left or to the right. Lisa can only pull left while Ralph can only pull right.



What happens if both agents are naively eager?

Infinite Executions

for naively eager agents

Infinite Executions

for naively eager agents

Ralph's (maximal) strong policy

Infinite Executions

for naively eager agents

Lisa's (maximal) strong policy

Infinite Executions

for naively eager agents

Ralph's (maximal) strong policy Lisa's (maximal) strong policy

Infinite Executions

for naively eager agents

Ralph's (maximal) strong policy Lisa's (maximal) strong policy

- There are many possible infinite executions

Infinite Executions

for naively eager agents

Ralph's (maximal) strong policy Lisa's (maximal) strong policy

- There are many possible infinite executions
- Solution here: Allow only "optimal" policies

Infinite Executions

for naively eager agents

Ralph's (maximal) strong policy Lisa's (maximal) strong policy

- There are many possible infinite executions
- Solution here: Allow only "optimal" policies
- For both agents, there is a unique maximal such policy

The *Optimally Eager* Agent Type

UNI FREIBURG

An agent is called optimally eager if he plans optimally and chooses an own action whenever this action is part of such a optimal strong policy (= of minimal depth).

The *Optimally Eager* Agent Type

UNI FREIBURG

An agent is called optimally eager if he plans optimally and chooses an own action whenever this action is part of such a optimal strong policy (= of minimal depth).

Prevents infinite executions if problem is uniformly observable

The *Optimally Eager* Agent Type

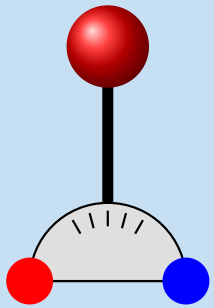
UNI FREIBURG

An agent is called optimally eager if he plans optimally and chooses an own action whenever this action is part of such a optimal strong policy (= of minimal depth).

Prevents infinite executions if problem is uniformly observable

Example task: Pulling the lever (II)

Same problem as before, but **L**isa only knows about the leftmost setting being a goal setting, while **R**alph only knows about the rightmost setting being one.



The *Optimally Eager* Agent Type

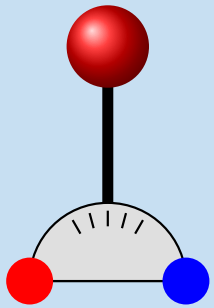
UNI FREIBURG

An agent is called optimally eager if he plans optimally and chooses an own action whenever this action is part of such a optimal strong policy (= of minimal depth).

Prevents infinite executions if problem is uniformly observable

Example task: Pulling the lever (II)

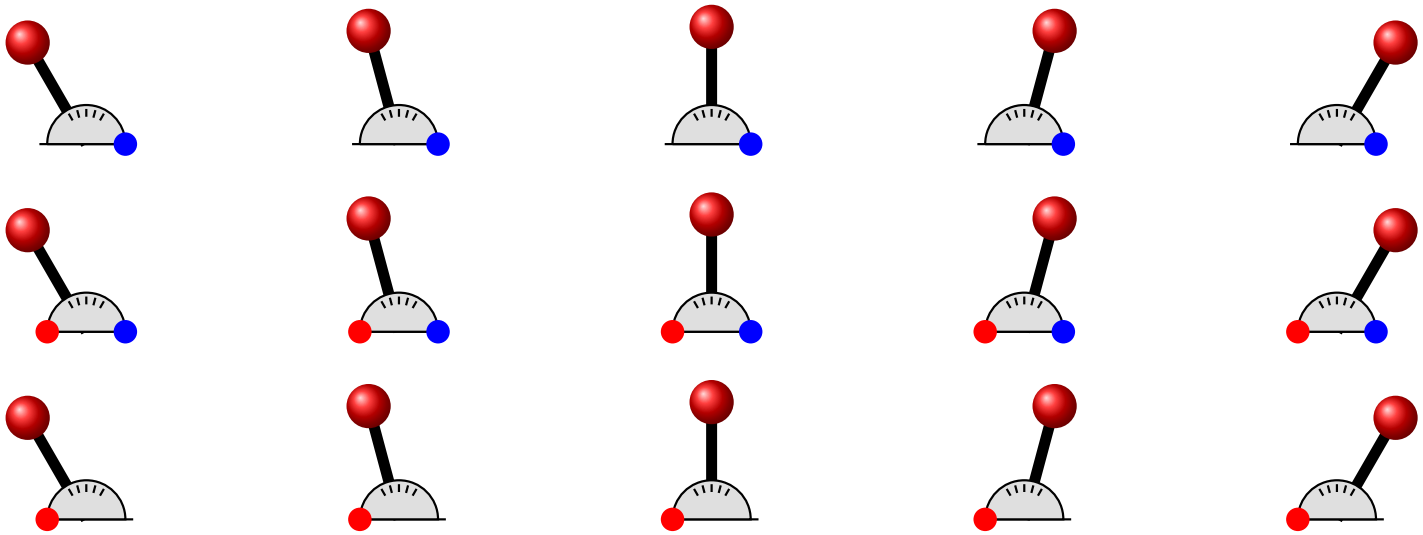
Same problem as before, but **L**isa only knows about the leftmost setting being a goal setting, while **R**alph only knows about the rightmost setting being one.



What happens if both agents are optimally eager?

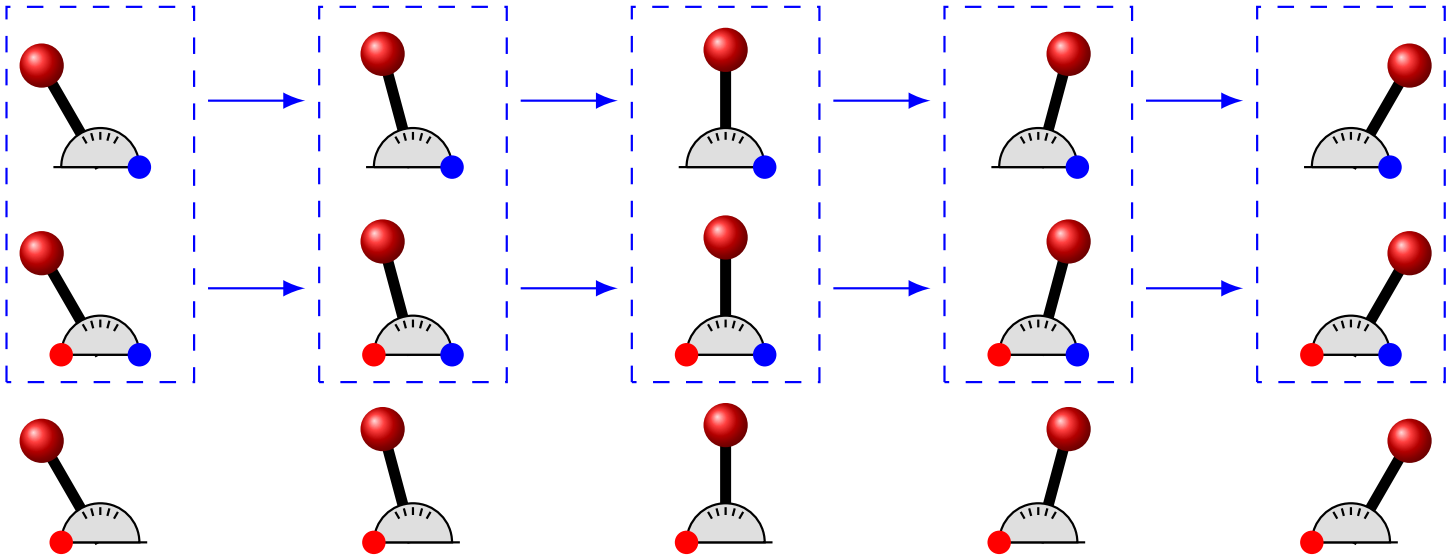
Infinite Executions

for optimally eager agents



Infinite Executions

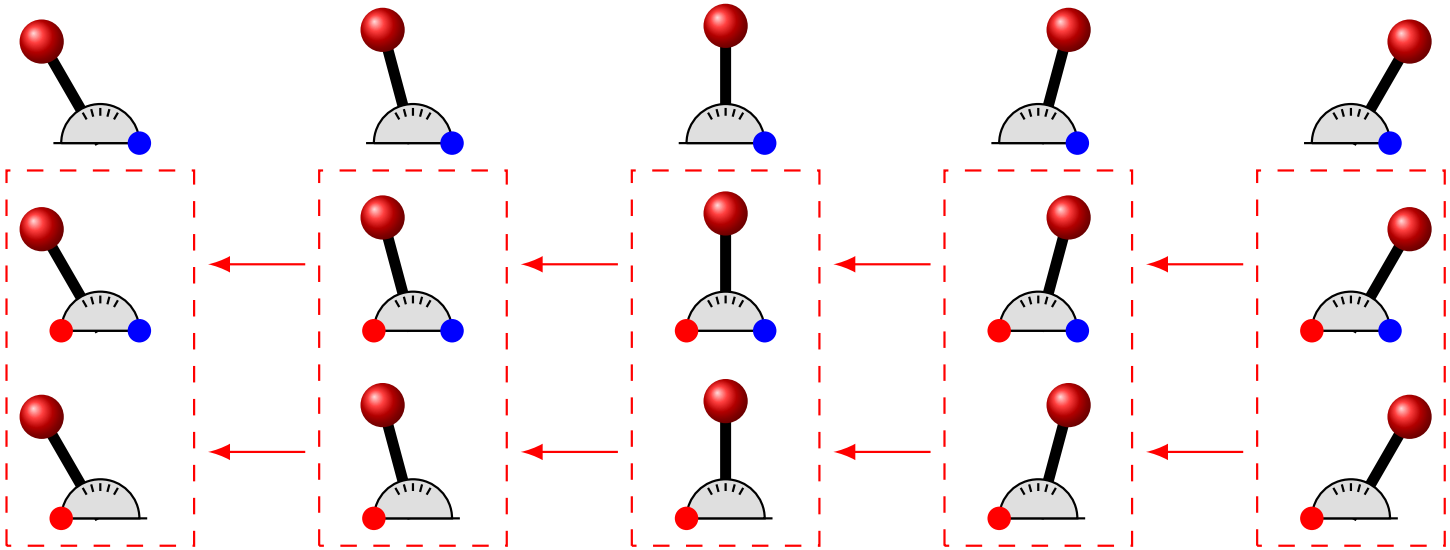
for optimally eager agents



Ralph's (maximal) strong policy

Infinite Executions

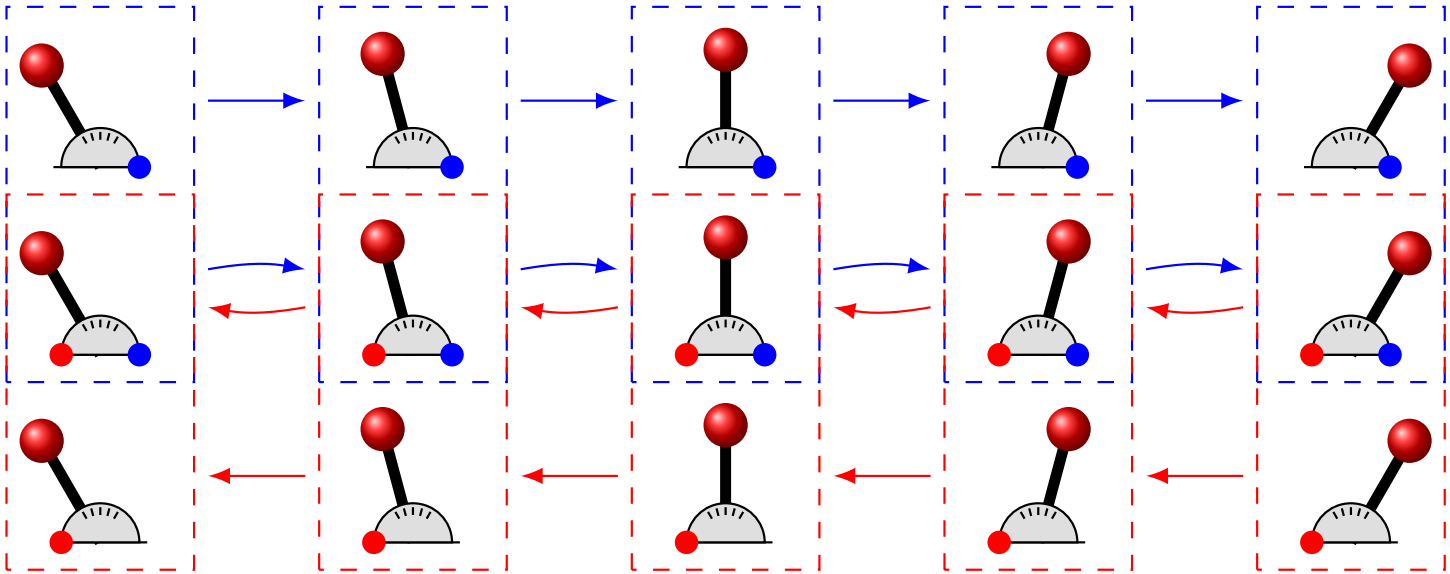
for optimally eager agents



Lisa's (maximal) strong policy

Infinite Executions

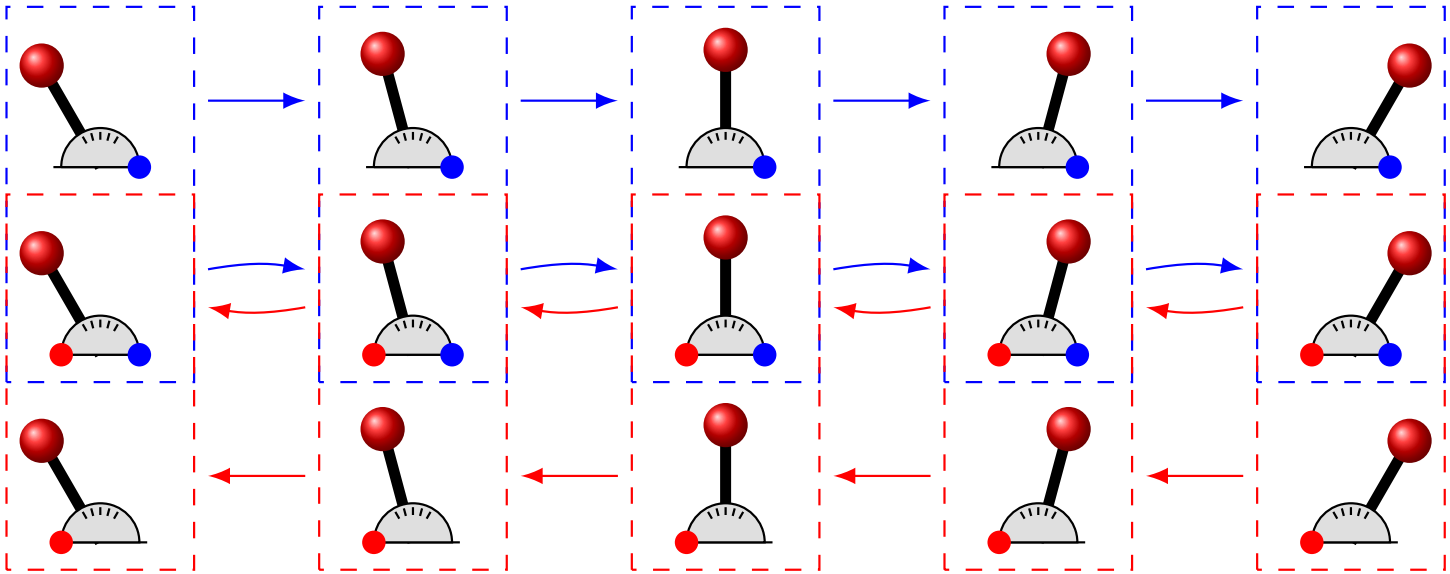
for optimally eager agents



Ralph's (maximal) strong policy Lisa's (maximal) strong policy

Infinite Executions

for optimally eager agents



Ralph's (maximal) strong policy Lisa's (maximal) strong policy

- Problem: Optimality has to be judged subjectively and doesn't help us here

UNI FREIBURG

Formalization

Formalization

UNI FREIBURG

Note:

The following definitions and propositions are taken directly from the paper. Notation differs a bit between the paper and this class.

- States are denoted by s, actions by a.
- A means action set.
- Agents are denoted by i, the set of agents by $\mathcal{A}$.
- $\mathrm{Dom}(\pi)$ is the domain of definition of π, i. e., the set of states where π is defined.

Executions of Policies

UNI FREIBURG

Definition 3. 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 $(s_0, a_1, s_1, a_2, s_2, \ldots)$, 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 $(s_0, a_1, s_1, \ldots, a_n, s_n)$ such that $s_n \models \gamma$.

Strong Policies

UNI FREIBURG

Definition 4. For a planning task $\Pi = \langle s_0, A, \omega, \gamma \rangle$, a policy π is called *strong* if $s_0 \in \mathrm{Dom}(\pi) \cup \{s \in S^{\mathrm{gl}} \mid s \models \gamma\}$ and for each $s \in \mathrm{Dom}(\pi)$, any execution of π from s is successful for Π. A planning task Π is called *solvable* if a strong policy for Π exists. For $i \in \mathcal{A}$, we call a policy π *i-strong* if it is strong and $\mathit{Globals}(s_0^i) \subseteq \mathrm{Dom}(\pi) \cup \{s \in S^{\mathrm{gl}} \mid s \models \gamma\}$.

So, if agent *i* comes up with an *i*-strong policy, it means that agent *i* knows the policy to be successful.

Reachability

UNI FREIBURG

Definition 5. Given global states s_0 and s, we call s *reachable* from s_0 if there are sequences of actions $a_1, \ldots, a_n$ and states $s_1, \ldots, s_n = s$ such that a_{m+1} is applicable in s_m and $s_{m+1} \in \mathit{Globals}(s_m \otimes a_{m+1})$ for all $m = 0, \ldots, n-1$. We call s reachable from s_0 *by following* a policy π if it is part of an execution $(s_0, a_1, \ldots, s, \ldots)$ of π.

Implicit Coordination

UNI FREIBURG

Proposition 1. *Let π be a strong policy for $\langle s_0, A, \omega, \gamma \rangle$ and let s be a non-goal state reachable from s_0 by following π. Then for some $i \in \mathcal{A}$: $\pi(s) \cap \{a \mid \omega(a) = i\} \neq \emptyset$ and π is an i-strong policy for $\langle s, A, \omega, \gamma \rangle$.*

Note that there are planning tasks where *i*-strong policies exist only for some of the agents. If these policies require other agents to act in some future states, Proposition 1 implies that these agents are able to find their own *i*-strong policies when re-planning from these states.

Maximal *i*-strength

The notion of “planning for all contingencies” in this setting is captured by maximality of strong policies.

Definition 6. We call an i-strong policy π for planning task Π a *maximal i-strong policy* for Π if π is a strong policy for $\langle s, A, \omega, \gamma \rangle$ for all states s such that: (1) s is reachable from some $s'_0 \in \mathit{Globals}(s^i_0)$, and (2) $\langle s, A, \omega, \varphi \rangle$ is solvable.

Formalization

UNI FREIBURG

What if agents come up with different (incompatible) plans from their local perspectives?

A policy profile for Π is a family $(\pi_i)_{i \in \mathcal{A}}$, where each π_i is a policy for Π.

We assume actions to be instantaneous and executed asynchronously.

Executions of Policy Profiles

UNI FREIBURG

Definition 7. 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 $(s_0, a_1, s_1, \ldots)$, such that for all $m \geq 0$,

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

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

We call such an execution successful if it is a finite execution $(s_0, a_1, s_1, \ldots, a_n, s_n)$ such that $s_n \models \gamma$.

Closedness

UNI FREIBURG

Proposition 2. *Let $(\pi_i)_{i\in\mathcal{A}}$ be a policy profile where each π_i is a maximal i-strong policy for task Π. Then $s \in \mathrm{Dom}(\pi_i)$ for all agents $i \in \mathcal{A}$ and non-goal states $s \in S^{gl}$ occurring in arbitrary executions $(s_0, a_1, \ldots, s, \ldots)$ of $(\pi_i)_{i\in\mathcal{A}}$.*

What can go wrong?

UNI FREIBURG

Deadlocks!

Example: “Who gets the door” problem from above.

Problem: Laziness of agents.

Agent Types

UNI FREIBURG

So, let's define agents that are not lazy.

Definition 8. A *planning agent* (or simply *agent*) is a pair (i, T), where i is an agent name and T is a mapping from planning tasks to policies, such that $T(\Pi)$ is an i-strong policy for Π, whenever such a policy exists.

Definition 9. Let $(i, T_i)_{i \in \mathcal{A}}$ be a group of agents and let Π be a planning task. Then the *executions* by $(i, T_i)_{i \in \mathcal{A}}$ of Π are the executions of the policy profile $(T_i(\Pi))_{i \in \mathcal{A}}$.

Agent Types and Preferences

UNI FREIBURG

Definition 10. For a state s, a policy π, and a set of actions A', we say that π *uses* A' in s if $\pi(s) \cap A' \neq \emptyset$. Then we say that agent (i, T) *has preference*

(1) *for* (the actions in) A' if for all Π and all $s \in \mathrm{Dom}(T(\Pi))$, policy $T(\Pi)$ uses A' in s unless no i-strong policy for Π uses A' in s, and

(2) *against* (the actions in) A' if for all Π and all $s \in \mathrm{Dom}(T(\Pi))$, policy $T(\Pi)$ does not use A' in s unless every i-strong policy for Π uses A' in s.

Lazy Agents

UNI FREIBURG

Unfortunately, *preference against* a set of actions is not the same as *preference for* its complement, which is why we need both notions. We can now define laziness as preference against one's own actions, that is, we call an agent (i, T) *lazy* if it has preference against the actions in $\{a \in A \mid \omega(a) = i\}$.

Deadlocks

UNI FREIBURG

A deadlock occurs if (1) something still needs to be done, (2) it is known that something can be done, but where (3) nothing will be done because of incompatible individual policies.

Definition 11. A *deadlock* for a policy profile $(\pi_i)_{i \in \mathcal{A}}$ is a global state s such that (1) s is not a goal state, (2) $s \in \mathrm{Dom}(\pi_i)$ for some $i \in \mathcal{A}$, and (3) $\omega(a) \neq i$ for all $i \in \mathcal{A}$ and $a \in \pi_i(s)$.

Proposition 3. *There are solvable planning tasks for which all executions by lazy agents result in a deadlock.*

Naively Eager Agents

UNI FREIBURG

To avoid deadlocks, we define (naively) eager agents as agents who have a preference *for* their own actions. That is, we call an agent (i, T) *naively eager* if it has a preference for the actions in $\{a \in A \mid \omega(a) = i\}$. They are called

Proposition 4. *Let Π be a planning task and $(i, T_i)_{i \in \mathcal{A}}$ be a group of naively eager agents. If each $\pi_i = T_i(\Pi)$ is a maximal i-strong policy, then all executions of $(\pi_i)_{i \in \mathcal{A}}$ are deadlock-free.*

Naively Eager Agents and Infinite Executions

UNI FREIBURG

Proposition 5. *There are solvable planning tasks for which some executions by naively eager agents are infinite.*

Example: Lever example where the agents sabotage each other by being overly eager, although they know that there is also a goal position on the opposite end.

Perspective-sensitive Costs

UNI FREIBURG

Let's try to make the agents only want to act themselves if it appears optimal to them (instead of trying to act whenever they reasonably can).

Definition 12. 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:

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

We extend this to local states s with $\mathit{Globals}(s) \subseteq \mathrm{Dom}(\pi)$ by letting $\kappa_\pi(s) := \max_{s' \in \mathit{Globals}(s)} \kappa_\pi(s')$.

Subjective Optimality

UNI FREIBURG

A policy is subjectively optimal if it always looks optimal from the perspective of the agent that is to act.

Definition 13. A policy π for a planning task $\Pi = \langle s_0, A, \omega, \gamma \rangle$ is called *subjectively optimal* if for all $s \in \mathrm{Dom}(\pi)$, all $a \in \pi(s)$ and all $\omega(a)$-strong policies π' for $\langle s, A, \omega, \gamma \rangle$ we have $\kappa_{\pi'}(s^{\omega(a)}) \geq \kappa_{\pi}(s^{\omega(a)})$.

Optimally Eager Agents

UNI FREIBURG

Definition 14. Given a set of actions A', we say that agent (i, T) is *subjectively optimal with preference for the actions in* A', if for all Π: (1) $T(\Pi)$ is an i-strong subjectively optimal policy if such a policy exists, and (2) $T(\Pi)$ uses A' in each $s \in \mathrm{Dom}(\pi)$ unless no i-strong subjectively optimal policy for Π uses A' in s.

We call an agent that is subjectively optimal with preference for its own actions *optimally eager*. That is, a planning agent (i, T) is called *optimally eager* if it is subjectively optimal with preference for the actions in $\{a \in A \mid \omega(a) = i\}$.

Deadlock-Freedom with Optimally Eager Agents

UNI FREIBURG

Good news:

Proposition 7. *Let Π be a planning task and $(i, T_i)_{i \in \mathcal{A}}$ be a group of optimally eager agents. If each $\pi_i = T_i(\Pi)$ is a maximal i-strong policy, then all executions of $(\pi_i)_{i \in \mathcal{A}}$ are deadlock-free.*

Finite Executions with Optimally Eager Agents

UNI FREIBURG

More good news:

Proposition 8. *Let Π be a uniformly observable and solvable planning task and let $(i, T_i)_{i \in \mathcal{A}}$ be a group of optimally eager agents. Then all executions by $(i, T_i)_{i \in \mathcal{A}}$ of Π are finite.*

Proposition 9. *Let Π be a uniformly observable planning task and $(i, T_i)_{i \in \mathcal{A}}$ be a group of optimally eager agents. If each $\pi_i = T_i(\Pi)$ is a maximal i-strong policy, then all executions of $(\pi_i)_{i \in \mathcal{A}}$ are successful.*

Example: In the lever problem, the agents will only pull towards their end if the lever is not already closer to the opposite end.

One More Negative Result under Non-Uniform Observability

UNI FREIBURG

But . . . also some bad news, again:

Proposition 10. *For every group of at least two agents* $(i, T_i)_{i \in A}$ *there exists a partially observable and solvable planning task* Π *that has unsuccessful executions by* $(i, T_i)_{i \in A}$ *of* Π.

Example: Lever problem with non-uniform observability.

This can serve as a motivation to look for special cases where this problem does not arise
$\rightsquigarrow$ multi-agent path finding under destination uncertainty (MAPF-DU), Wednesday

UNI FREIBURG

Summary

Results Overview

UNI FREIBURG

- Deadlocks can be avoided by (naively/optimally) eager agents

Results Overview

UNI FREIBURG

- Deadlocks can be avoided by (naively/optimally) eager agents
- If there is uniform knowledge, optimally eager agents avoid infinite executions

Results Overview

UNI FREIBURG

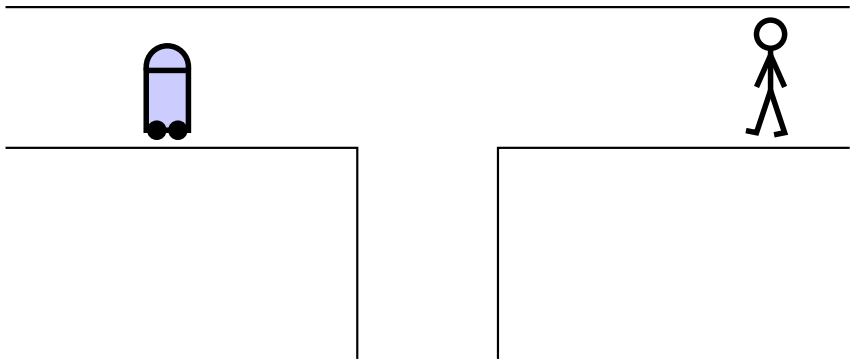
- Deadlocks can be avoided by (naively/optimally) eager agents
- If there is uniform knowledge, optimally eager agents avoid infinite executions
- In the general case, using our history-independent policies, this is not possible

Results Overview

UNI FREIBURG

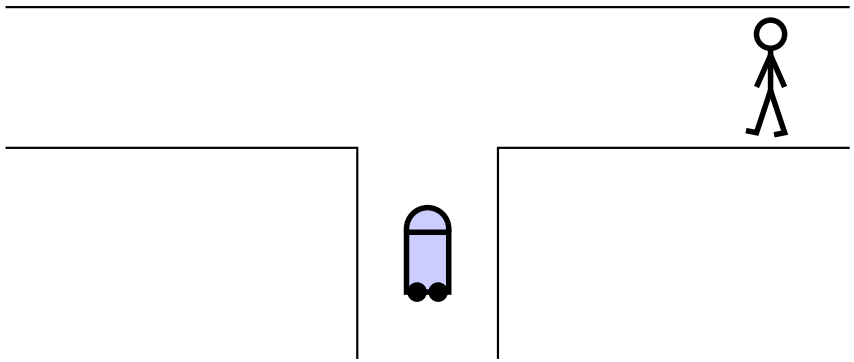
- Deadlocks can be avoided by (naively/optimally) eager agents
- If there is uniform knowledge, optimally eager agents avoid infinite executions
- In the general case, using our history-independent policies, this is not possible
- Of course, some problems are unproblematic . . .

Multi-Agent Pathfinding Example – Solution



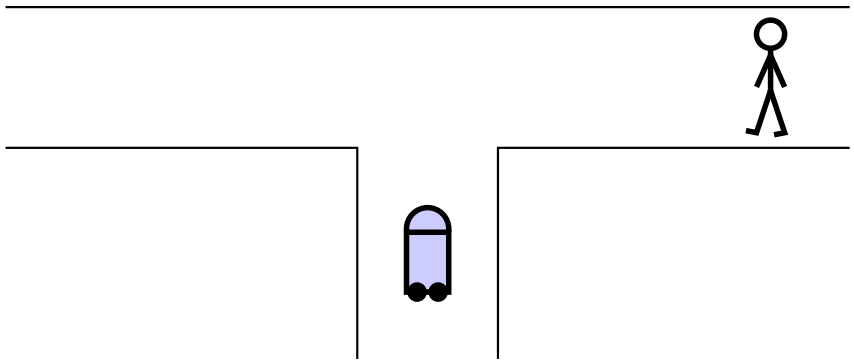
- Lazy agents who both stay won't solve the problem

Multi-Agent Pathfinding Example – Solution



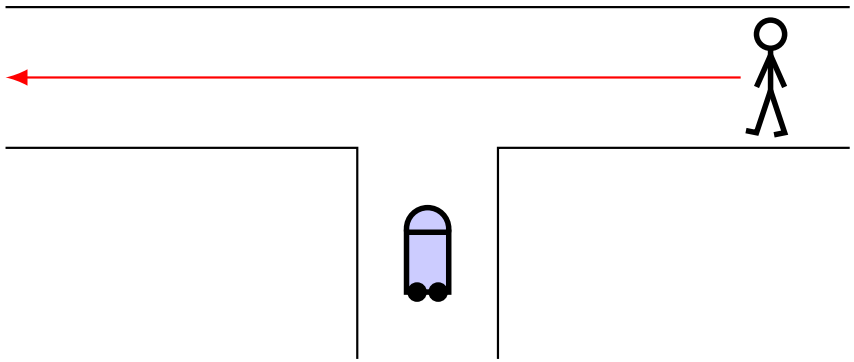
- Lazy agents who both stay won't solve the problem
- Going south is an advancement towards the goal

Multi-Agent Pathfinding Example – Solution



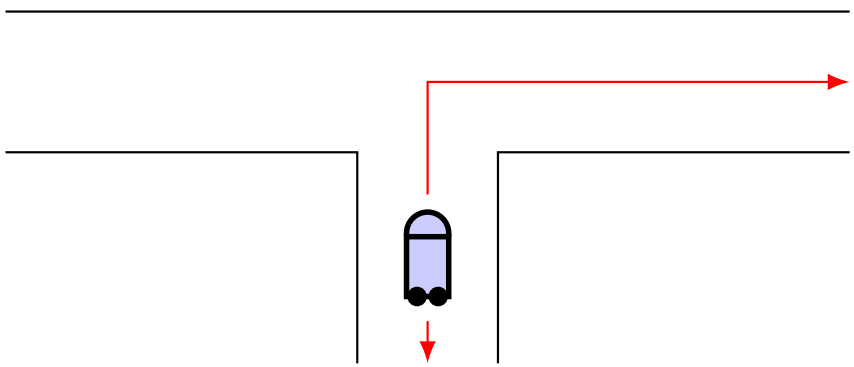
- Lazy agents who both stay won't solve the problem
- Going south is an advancement towards the goal
- Case 1: Human wants to go west:

Multi-Agent Pathfinding Example – Solution



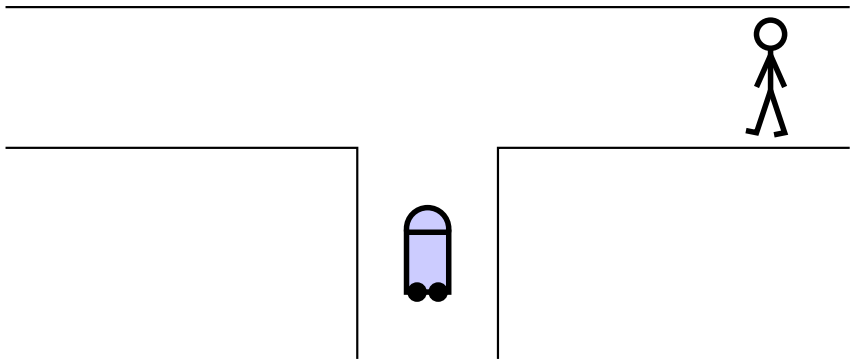
- Lazy agents who both stay won't solve the problem
- Going south is an advancement towards the goal
- Case 1: Human wants to go west:
  - Human can walk directly to his goal (west)

Multi-Agent Pathfinding Example – Solution



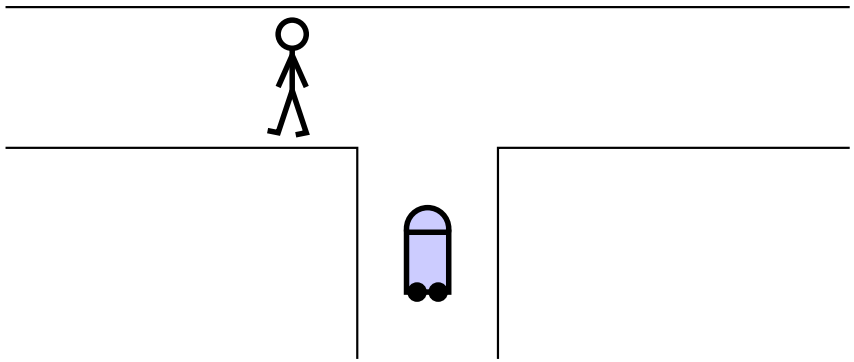
- Lazy agents who both stay won't solve the problem
- Going south is an advancement towards the goal
- Case 1: Human wants to go west:
  - Human can walk directly to his goal (west)
  - enabling the robot to reach both potential goals

Multi-Agent Pathfinding Example – Solution



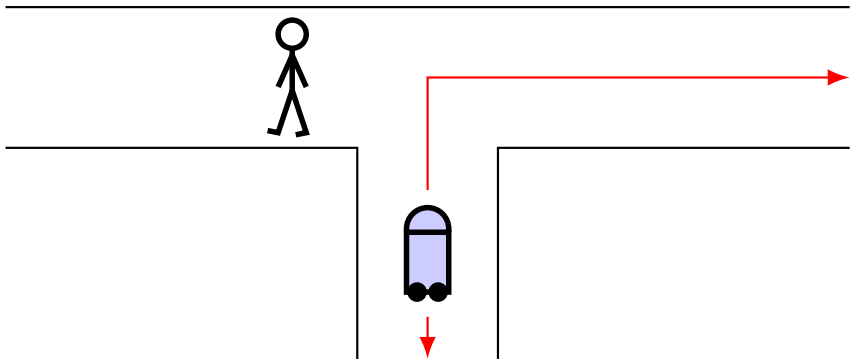
- Lazy agents who both stay won't solve the problem
- Going south is an advancement towards the goal
- Case 2: Human wants to go south:

Multi-Agent Pathfinding Example – Solution



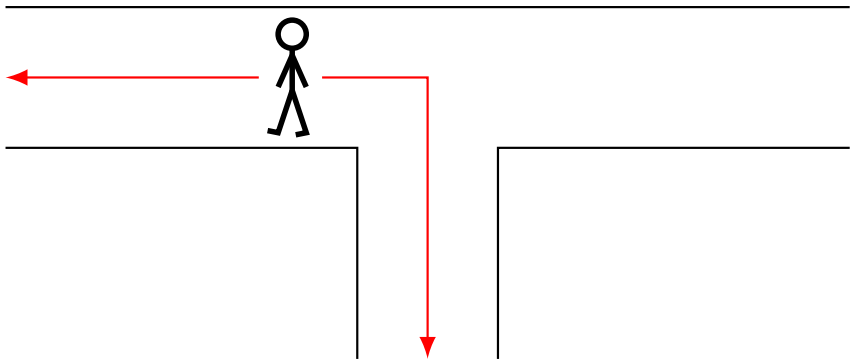
- Lazy agents who both stay won't solve the problem
- Going south is an advancement towards the goal
- Case 2: Human wants to go south:
  - Human can go out of the way (west)

Multi-Agent Pathfinding Example – Solution



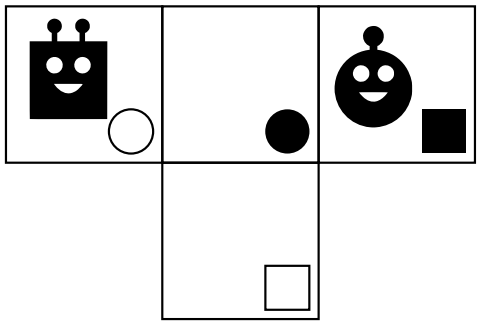
- Lazy agents who both stay won't solve the problem
- Going south is an advancement towards the goal
- Case 2: Human wants to go south:
  - Human can go out of the way (west)
  - enabling the robot to reach both potential goals

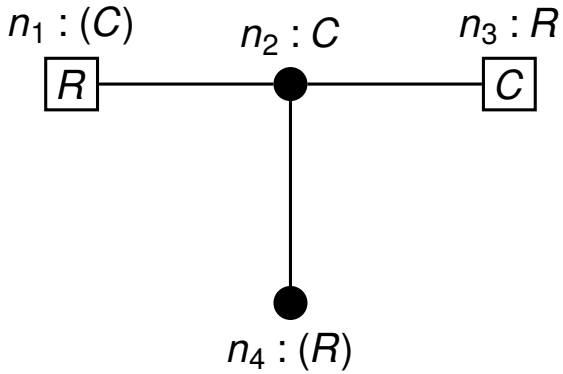
Multi-Agent Pathfinding Example – Solution



- Lazy agents who both stay won't solve the problem
- Going south is an advancement towards the goal
- Case 2: Human wants to go south:
  - Human can go out of the way (west)
  - enabling the robot to reach both potential goals
  - enabling the human to reach both potential goals

Multi-Agent Pathfinding Example – Solution





Implicitly Coordinated Multi-Agent Path Finding under Destination Uncertainty
(next lesson)