Monadic decision processes and hierarchical planning for autonomous systems

David I. Spivak

Department of Mathematics
Massachusetts Institute of Technology

Outline

Autonomous systems

Autonomous systems observe the environment and plan actions.

Autonomous systems

Autonomous systems observe the environment and plan actions.

- This planning requires coordination between subsystems.

Autonomous systems

Autonomous systems observe the environment and plan actions.

- This planning requires coordination between subsystems.
- Coordination requires translations between languages: *communication*.

Autonomous systems

Autonomous systems observe the environment and plan actions.

- This planning requires coordination between subsystems.
- Coordination requires translations between languages: *communication*.
- These translations must negotiate change of context and scale.

Going to the store

Currently in my house, I want to perform the action "go to the store."

Going to the store

Currently in my house, I want to perform the action "go to the store."

- This action is compiled into a lower-level plan, say walking directions.

Going to the store

Currently in my house, I want to perform the action "go to the store."

- This action is compiled into a lower-level plan, say walking directions.
- Each step (left@Elm) compiled into lower-level plan: body movement.
- Each body movement compiled into atoms moving through space.

Going to the store

Currently in my house, I want to perform the action "go to the store."

- This action is compiled into a lower-level plan, say walking directions.
- Each step (left@Elm) compiled into lower-level plan: body movement.
- Each body movement compiled into atoms moving through space.

"Path planning → path management → path following → autopilot → surfaces"

Going to the store

Currently in my house, I want to perform the action "go to the store."

- This action is compiled into a lower-level plan, say walking directions.
- Each step (left@Elm) compiled into lower-level plan: body movement.
- Each body movement compiled into atoms moving through space.

"Path planning $\longrightarrow$ path management $\longrightarrow$ path following $\longrightarrow$ autopilot $\longrightarrow$ surfaces"

Abstractly:

- High-level actions A need to be compiled to low-level action plans A'.

Going to the store

Currently in my house, I want to perform the action "go to the store."

- This action is compiled into a lower-level plan, say walking directions.
- Each step (left@Elm) compiled into lower-level plan: body movement.
- Each body movement compiled into atoms moving through space.

"Path planning → path management → path following → autopilot → surfaces"

Abstractly:

- High-level actions A need to be compiled to low-level action plans A'.
- But lower-level actions are based on the particular lower-level state S'.
  - You say you're in the kitchen and want to walk toward the door.
  - But to plan body movements, it's relevant that you're sitting.

Going to the store

Currently in my house, I want to perform the action "go to the store."

- This action is compiled into a lower-level plan, say walking directions.
- Each step (left@Elm) compiled into lower-level plan: body movement.
- Each body movement compiled into atoms moving through space.

"Path planning → path management → path following → autopilot → surfaces"

Abstractly:

- High-level actions A need to be compiled to low-level action plans A'.
- But lower-level actions are based on the particular lower-level state S'.
  - You say you're in the kitchen and want to walk toward the door.
  - But to plan body movements, it's relevant that you're sitting.
- The low-level state is unknown to the high-level action planner.
- But that low-level state is necessary to develop the low-level plan.

Plan of talk

I'll discuss compositional structures

Plan of talk

I'll discuss compositional structures

- How different planners talk to each other, compositionally.
- Same abstract structure generalizes MDPs in a variety of ways

Plan of talk

I'll discuss compositional structures

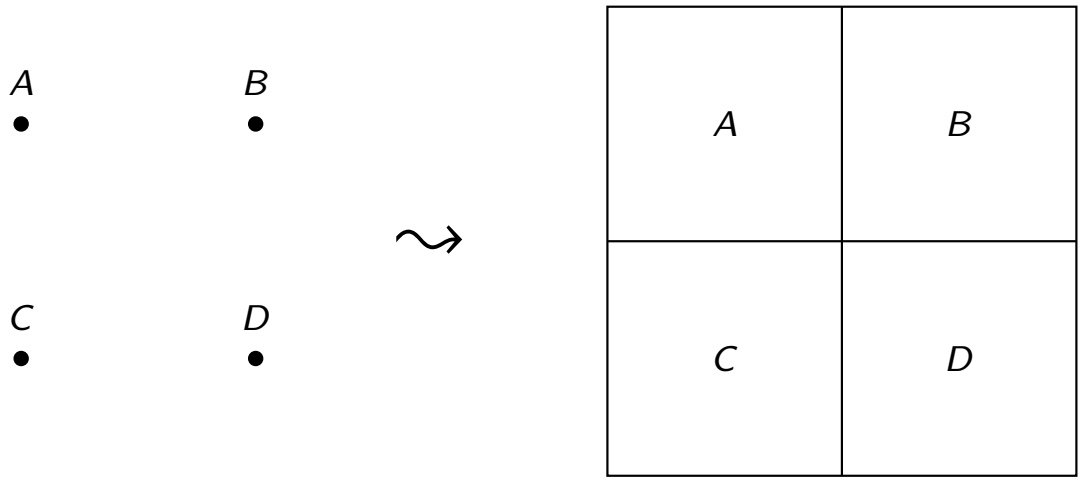
- How different planners talk to each other, compositionally.
- Same abstract structure generalizes MDPs in a variety of ways

I won't discuss optimization or algorithms.

- Future work: see what people are doing that is fully compositional.
- See how we might add our own compositional algorithms.

Outline

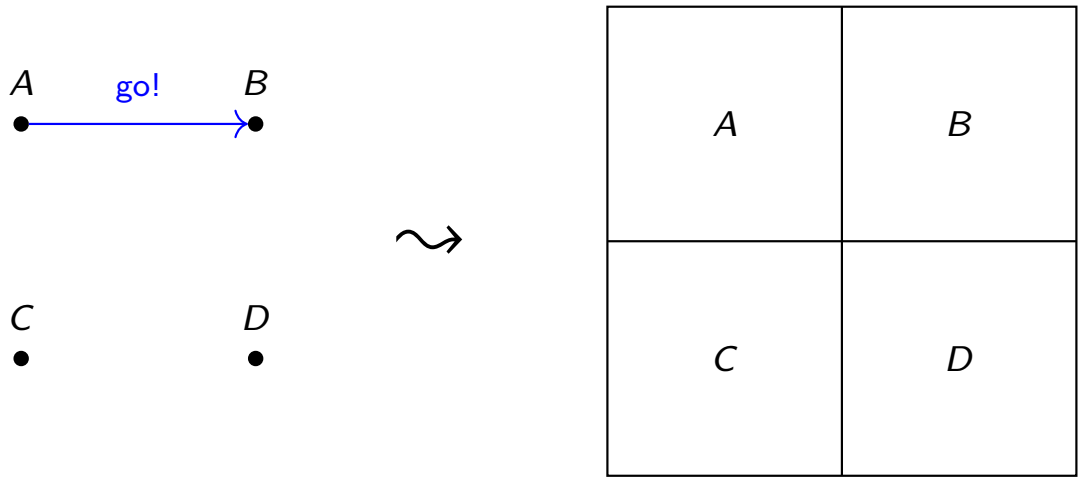
Hierarchical planning in pictures



high-level states
(simpler)

low-level states
(more real)

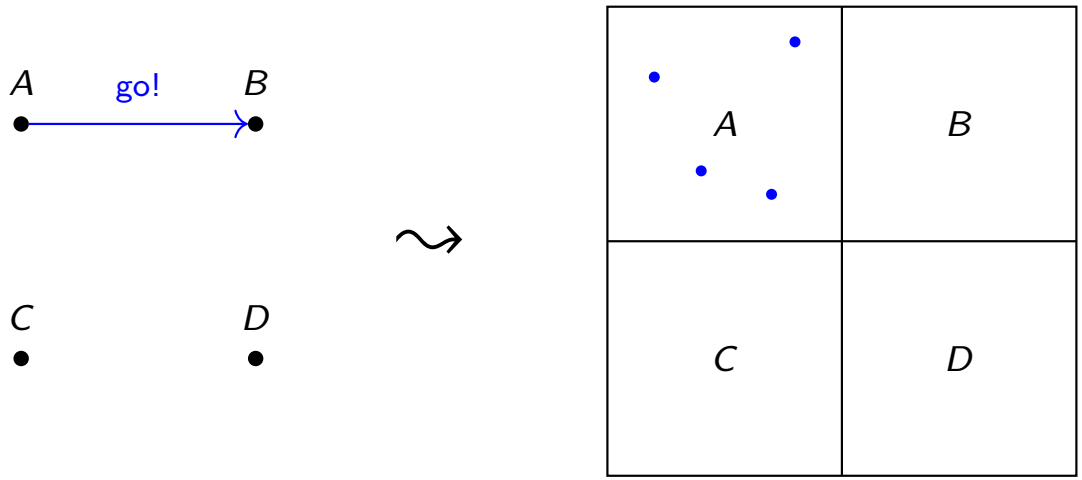
Hierarchical planning in pictures



high-level states
(simpler)

low-level states
(more real)

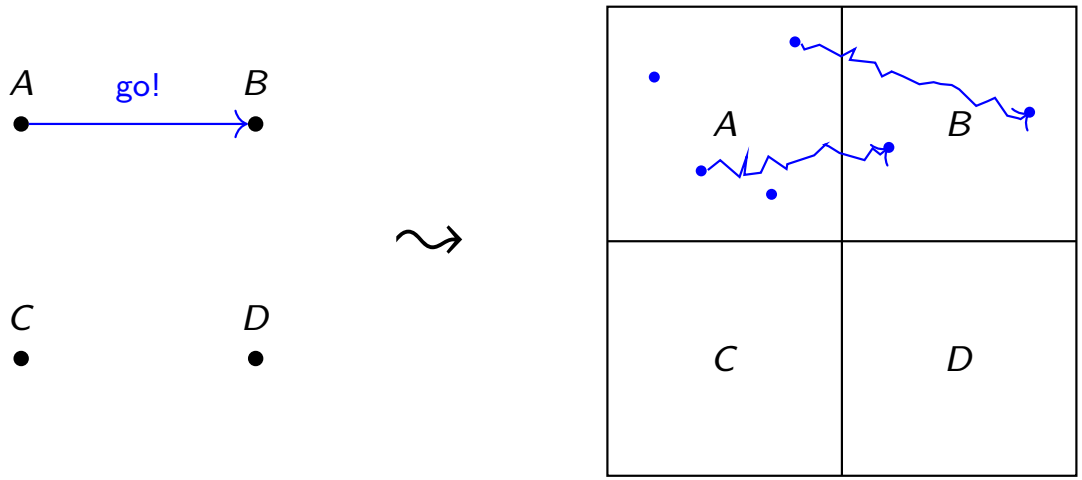
Hierarchical planning in pictures



high-level states
(simpler)

low-level states
(more real)

Hierarchical planning in pictures



high-level states
(simpler)

low-level states
(more real)

Markov decision processes

A common setting for planning is *Markov Decision Processes* (MDPs)

Markov decision processes

A common setting for planning is *Markov Decision Processes* (MDPs)

- Usual mathematical model:
  - A set S of "states", a set A of "actions";
  - For every $(s, a) \in S \times A$ a probability distribution on S "act";
  - For every $(s, a, s') \in S \times A \times S$, a real number "reward".

Markov decision processes

A common setting for planning is *Markov Decision Processes* (MDPs)

- Usual mathematical model:
  - A set S of "states", a set A of "actions";
  - For every $(s, a) \in S \times A$ a probability distribution on S "act";
  - For every $(s, a, s') \in S \times A \times S$, a real number "reward".
- A mild generalization: $T : S \times A \to \mathbf{Dist}(S \times \mathbb{R})$
  - Here $\mathbf{Dist}(X)$ means the set of probability distributions on X.
  - So for every (s, a) we get a joint distribution $T(s, a)$ on $S \times \mathbb{R}$.
  - E.g. $T(s, a) = 30\% \cdot (s_1, \$5) + 60\% \cdot (s_2, -\$7) + 10\% \cdot (s_2, \$2.50)$.

Markov decision processes

A common setting for planning is *Markov Decision Processes* (MDPs)

- Usual mathematical model:
  - A set S of "states", a set A of "actions";
  - For every $(s, a) \in S \times A$ a probability distribution on S "act";
  - For every $(s, a, s') \in S \times A \times S$, a real number "reward".
- A mild generalization: $T : S \times A \to \mathbf{Dist}(S \times \mathbb{R})$
  - Here $\mathbf{Dist}(X)$ means the set of probability distributions on X.
  - So for every (s, a) we get a joint distribution $T(s, a)$ on $S \times \mathbb{R}$.
  - E.g. $T(s, a) = 30\% \cdot (s_1, \$5) + 60\% \cdot (s_2, -\$7) + 10\% \cdot (s_2, \$2.50)$.

Goal: create policies that generate the most reward.

- As mentioned, we'll leave that to others, and for future work.
- For now we'll focus on generalization and hierarchical structure.

Outline

Interlude: category theory

Category theory is the mathematics of structure and compositionality.

- It looks for common structures that exist throughout math and tech.

Interlude: category theory

Category theory is the mathematics of structure and compositionality.

- It looks for common structures that exist throughout math and tech.
- Example: there's something in common between:
  - $A \wedge B$ in boolean logic
  - $A \cap B$ in Venn diagrams
  - $A \times B$ in set theory
  - $A \oplus B$ in vector spaces
  - $\gcd(A, B)$ in number theory

Interlude: category theory

Category theory is the mathematics of structure and compositionality.

- It looks for common structures that exist throughout math and tech.
- Example: there's something in common between:
  - $A \wedge B$ in boolean logic
  - $A \cap B$ in Venn diagrams
  - $A \times B$ in set theory
  - $A \oplus B$ in vector spaces
  - $\gcd(A, B)$ in number theory
  - "it goes into both A and B, more directly than anything else"

Interlude: category theory

Category theory is the mathematics of structure and compositionality.

- It looks for common structures that exist throughout math and tech.
- Example: there's something in common between:
  - $A \wedge B$ in boolean logic
  - $A \cap B$ in Venn diagrams
  - $A \times B$ in set theory
  - $A \oplus B$ in vector spaces
  - $\gcd(A, B)$ in number theory
  - "it goes into both A and B, more directly than anything else"
- CT finds "universal properties" and compositional structures

Interlude: category theory

Category theory is the mathematics of structure and compositionality.

- It looks for common structures that exist throughout math and tech.
- Example: there's something in common between:
  - $A \wedge B$ in boolean logic
  - $A \cap B$ in Venn diagrams
  - $A \times B$ in set theory
  - $A \oplus B$ in vector spaces
  - $\gcd(A, B)$ in number theory
  - "it goes into both A and B, more directly than anything else"
- CT finds "universal properties" and compositional structures

It has made major headway in computer science.

- Abstraction = reusable, readable, maintainable code.
- Functors and monads are found in: Haskell, Scala, C#, Java.

Interlude: category theory

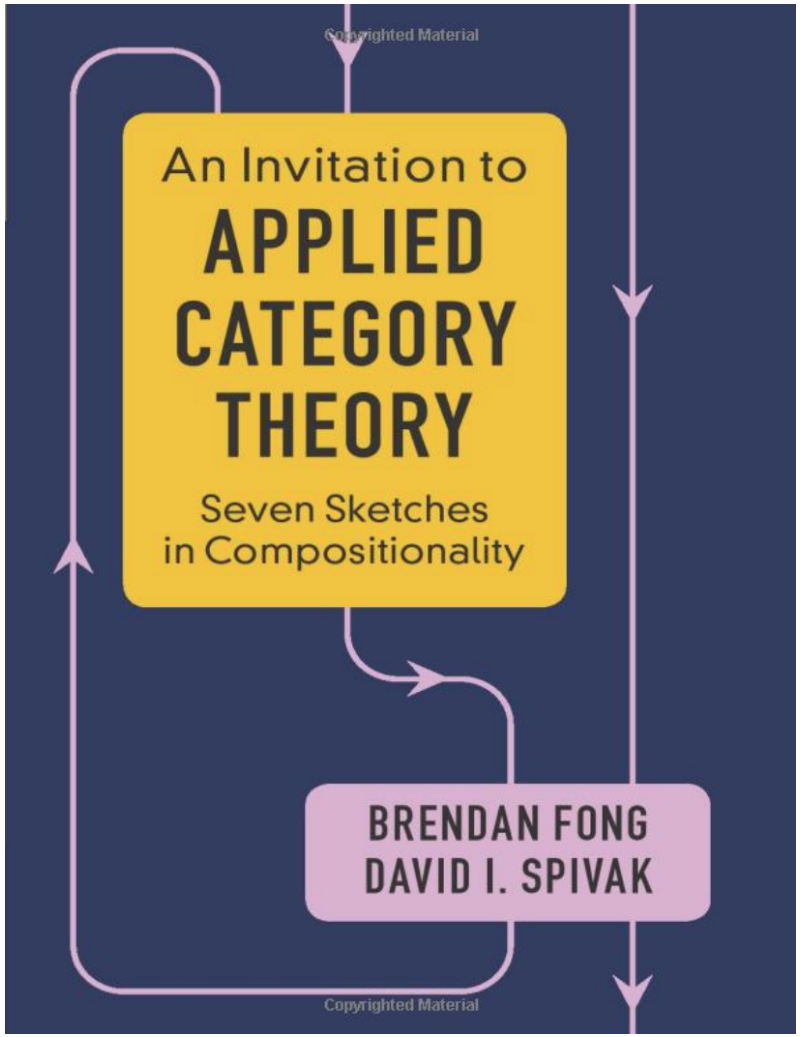
Category theory is the mathematics of structure and compositionality.

- It looks for common structures that exist throughout math and tech.
- Example: there's something in common between:
  - $A \wedge B$ in boolean logic
  - $A \cap B$ in Venn diagrams
  - $A \times B$ in set theory
  - $A \oplus B$ in vector spaces
  - $\gcd(A, B)$ in number theory
  - "it goes into both A and B, more directly than anything else"
- CT finds "universal properties" and compositional structures

It has made major headway in computer science.

- Abstraction = reusable, readable, maintainable code.
- Functors and monads are found in: Haskell, Scala, C#, Java.
- I once met a low-level Verizon software engineer at a party....

Invitation to applied category theory



Free online (arxiv:1803.05316) or for purchase (Cambridge U. Press).

Generalizing MDPs using monads I.

I won't tell you what monads are now, but I'll give some examples:

Generalizing MDPs using monads I.

I won't tell you what monads are now, but I'll give some examples:

- Probability distributions ("I don't know exactly, but I can guess")

Generalizing MDPs using monads I.

I won't tell you what monads are now, but I'll give some examples:

- Probability distributions ("I don't know exactly, but I can guess")
- Lists ("I don't know exactly, but I know the possibilities")

Generalizing MDPs using monads I.

I won't tell you what monads are now, but I'll give some examples:

- Probability distributions ("I don't know exactly, but I can guess")
- Lists ("I don't know exactly, but I know the possibilities")
- Exceptions ("I do know exactly, but sometimes I get an error")

Generalizing MDPs using monads I.

I won't tell you what monads are now, but I'll give some examples:

- Probability distributions ("I don't know exactly, but I can guess")
- Lists ("I don't know exactly, but I know the possibilities")
- Exceptions ("I do know exactly, but sometimes I get an error")
- Tasks ("depends on what my task is")

Generalizing MDPs using monads I.

I won't tell you what monads are now, but I'll give some examples:

- Probability distributions ("I don't know exactly, but I can guess")
- Lists ("I don't know exactly, but I know the possibilities")
- Exceptions ("I do know exactly, but sometimes I get an error")
- Tasks ("depends on what my task is")
- States ("depends on and updates my state")

Generalizing MDPs using monads I.

I won't tell you what monads are now, but I'll give some examples:

- Probability distributions ("I don't know exactly, but I can guess")
- Lists ("I don't know exactly, but I know the possibilities")
- Exceptions ("I do know exactly, but sometimes I get an error")
- Tasks ("depends on what my task is")
- States ("depends on and updates my state")
- Monoid ("comes equipped with an accumulating reward")

Generalizing MDPs using monads I.

I won't tell you what monads are now, but I'll give some examples:

- Probability distributions ("I don't know exactly, but I can guess")
- Lists ("I don't know exactly, but I know the possibilities")
- Exceptions ("I do know exactly, but sometimes I get an error")
- Tasks ("depends on what my task is")
- States ("depends on and updates my state")
- Monoid ("comes equipped with an accumulating reward")

For any monad M, we often consider functions $f : X \to M(Y)$.

- **Idea:** "For every $x \in X$, I get an element of Y that..." $\langle$ see above $\rangle$.

Generalizing MDPs using monads I.

I won't tell you what monads are now, but I'll give some examples:

- Probability distributions ("I don't know exactly, but I can guess")
- Lists ("I don't know exactly, but I know the possibilities")
- Exceptions ("I do know exactly, but sometimes I get an error")
- Tasks ("depends on what my task is")
- States ("depends on and updates my state")
- Monoid ("comes equipped with an accumulating reward")

For any monad M, we often consider functions $f : X \to M(Y)$.

- **Idea:** "For every $x \in X$, I get an element of Y that..." ⟨ see above ⟩.
- And these form a *category*: you can compose such functions.

Generalizing MDPs using monads I.

I won't tell you what monads are now, but I'll give some examples:

- Probability distributions ("I don't know exactly, but I can guess")
- Lists ("I don't know exactly, but I know the possibilities")
- Exceptions ("I do know exactly, but sometimes I get an error")
- Tasks ("depends on what my task is")
- States ("depends on and updates my state")
- Monoid ("comes equipped with an accumulating reward")

For any monad M, we often consider functions $f : X \to M(Y)$.

- **Idea:** "For every $x \in X$, I get an element of Y that..." ⟨ see above ⟩.
- And these form a *category*: you can compose such functions.
- E.g. given $f : X \to \mathbf{List}(Y)$ and $g : Y \to \mathbf{List}(Z)$ ...

Generalizing MDPs using monads I.

I won't tell you what monads are now, but I'll give some examples:

- Probability distributions ("I don't know exactly, but I can guess")
- Lists ("I don't know exactly, but I know the possibilities")
- Exceptions ("I do know exactly, but sometimes I get an error")
- Tasks ("depends on what my task is")
- States ("depends on and updates my state")
- Monoid ("comes equipped with an accumulating reward")

For any monad M, we often consider functions $f : X \to M(Y)$.

- **Idea:** "For every $x \in X$, I get an element of Y that..." ⟨ see above ⟩.
- And these form a *category*: you can compose such functions.
- E.g. given $f : X \to \mathbf{List}(Y)$ and $g : Y \to \mathbf{List}(Z)$ ...
  ... for each $x \in X$, get list of lists in Z, and then flatten.
- Or given $f : X \to \mathbf{Dist}(Y)$ and $g : Y \to \mathbf{Dist}(Z)$ ...
  ... for each $x \in X$, get dist. of distributions, and then integrate.

This category is called the *Kleisli category* of M.

Generalizing MDPs using monads

For any monad M, define a *monadic Markov decision process*, an M-MDP.

- Example: monad $=$ $Y \mapsto \mathbf{Dist}(Y \times \mathbb{R})$.
- A morphism in the Kleisli category is $X \to \mathbf{Dist}(Y \times \mathbb{R})$.

Generalizing MDPs using monads

For any monad M, define a *monadic Markov decision process*, an M-MDP.

- Example: monad $=$ $Y \mapsto \mathbf{Dist}(Y \times \mathbb{R})$.
- A morphism in the Kleisli category is $X \to \mathbf{Dist}(Y \times \mathbb{R})$.

Def: an *M-MDP* is: a functor from a monoid A to the Kleisli category.

- This is a very compressed formula, but basically it means:
- ... a set S and a function $A \times S \to \mathbf{Dist}(S \times \mathbb{R})$.

Generalizing MDPs using monads

For any monad M, define a *monadic Markov decision process*, an M-MDP.

- Example: monad $=\quad Y \mapsto \mathbf{Dist}(Y \times \mathbb{R})$.
- A morphism in the Kleisli category is $X \to \mathbf{Dist}(Y \times \mathbb{R})$.

Def: an *M-MDP* is: a functor from a monoid A to the Kleisli category.

- This is a very compressed formula, but basically it means:
- ... a set S and a function $A \times S \to \mathbf{Dist}(S \times \mathbb{R})$.

M-MDPs are parametric in the monad M.

- Probabilities, lists, exceptions, tasks, arbitrary monoids, states
- E.g. $A \times S \to S$ actions deterministically result in states
- $A \times S \to S + E$ actions sometimes result in exceptions
- $A \times S \to S^{\mathrm{Task}}$ results are based on tasks
- $A \times S \to \mathbf{List}(S \times \mathbb{R}_{\geq 0})$ possibilistic results with nonnegative rewards

Generalizing MDPs using monads

For any monad M, define a *monadic Markov decision process*, an M-MDP.

- Example: monad $=\quad Y \mapsto \mathbf{Dist}(Y \times \mathbb{R})$.
- A morphism in the Kleisli category is $X \to \mathbf{Dist}(Y \times \mathbb{R})$.

Def: an *M-MDP* is: a functor from a monoid A to the Kleisli category.

- This is a very compressed formula, but basically it means:
- ... a set S and a function $A \times S \to \mathbf{Dist}(S \times \mathbb{R})$.

M-MDPs are parametric in the monad M.

- Probabilities, lists, exceptions, tasks, arbitrary monoids, states
- E.g. $A \times S \to S$ actions deterministically result in states
- $A \times S \to S + E$ actions sometimes result in exceptions
- $A \times S \to S^{\mathrm{Task}}$ results are based on tasks
- $A \times S \to \mathbf{List}(S \times \mathbb{R}_{\geq 0})$ possibilistic results with nonnegative rewards

Your choice! We work abstractly, so you can plug and play.

Hierarchical planning with M-MDPs I.

Fix a monad M. We define a *category* of M-MDPs.

Hierarchical planning with M-MDPs I.

Fix a monad M. We define a *category* of M-MDPs.

- Objects are M-MDPs, i.e. $\{(A, S, T) \mid T : A \times S \to M(S)\}$.

Hierarchical planning with M-MDPs I.

Fix a monad M. We define a *category* of M-MDPs.

- Objects are M-MDPs, i.e. $\{(A, S, T) \mid T : A \times S \to M(S)\}$.
  - If $M(S) = \mathbf{Dist}(S \times \mathbb{R})$ this is our generalized MDP from before.
  - If $M(S) = \mathbf{List}(S)$, we get possibilities rather than probabilities.
  - etc.

Hierarchical planning with M-MDPs I.

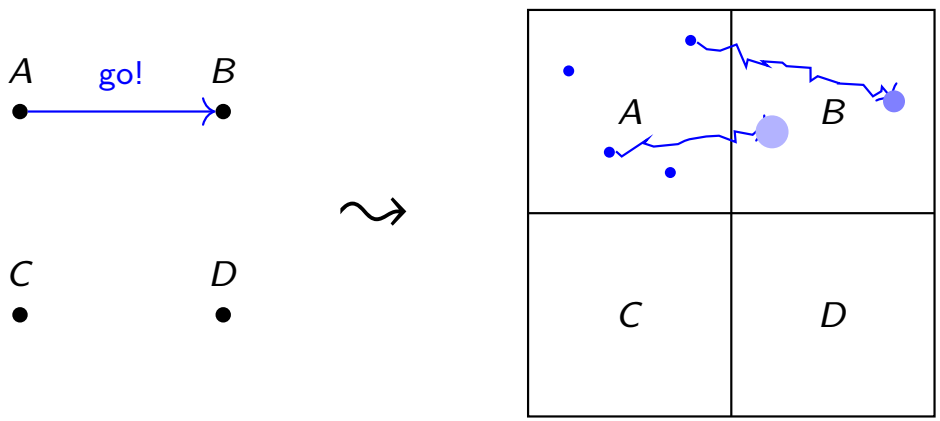
Fix a monad M. We define a *category* of M-MDPs.

- Objects are M-MDPs, i.e. $\{(A, S, T) \mid T : A \times S \to M(S)\}$.
  - If $M(S) = \mathbf{Dist}(S \times \mathbb{R})$ this is our generalized MDP from before.
  - If $M(S) = \mathbf{List}(S)$, we get possibilities rather than probabilities.
  - etc.
- Morphisms: translations between M-MDPs

Hierarchical planning with M-MDPs I.

Fix a monad M. We define a *category* of M-MDPs.

- Objects are M-MDPs, i.e. $\{(A, S, T) \mid T : A \times S \to M(S)\}$.
  - If $M(S) = \mathbf{Dist}(S \times \mathbb{R})$ this is our generalized MDP from before.
  - If $M(S) = \mathbf{List}(S)$, we get possibilities rather than probabilities.
  - etc.
- Morphisms: translations between M-MDPs
  - Recall the notion of hierarchical planning; now add the monad M.



high-level states (simpler)

low-level states (more real)

Hierarchical planning with M-MDPs II.

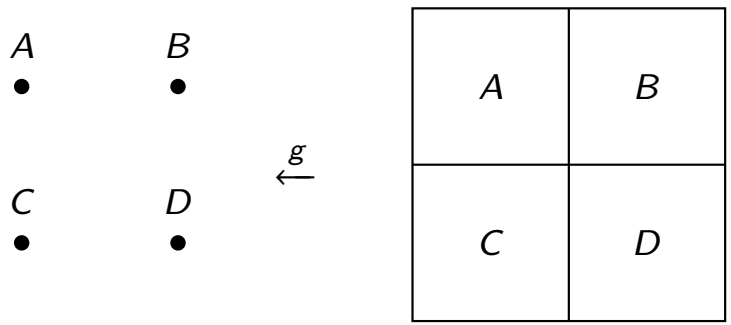
What's a morphism $(A, S, T) \longrightarrow (A', S', T')$, more precisely?

- Think of the unprimed as higher-level, the primed as lower-level

Hierarchical planning with M-MDPs II.

What's a morphism $(A, S, T) \longrightarrow (A', S', T')$, more precisely?

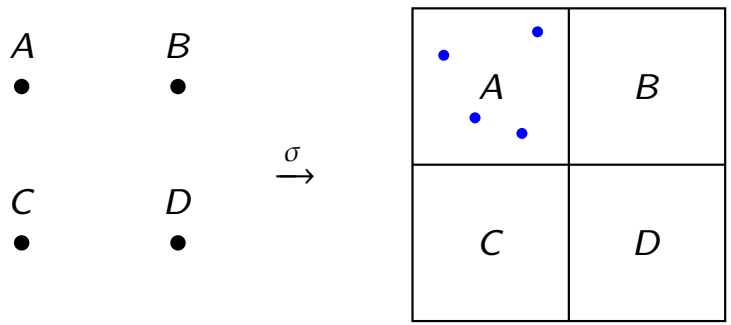
- Think of the unprimed as higher-level, the primed as lower-level
- A morphism as above consists of the following:
  - a morphism $g : S' \longrightarrow S$ "coarse-grain"



Hierarchical planning with M-MDPs II.

What's a morphism $(A, S, T) \longrightarrow (A', S', T')$, more precisely?

- Think of the unprimed as higher-level, the primed as lower-level
- A morphism as above consists of the following:
  - a morphism $g\colon S' \longrightarrow S$ "coarse-grain"

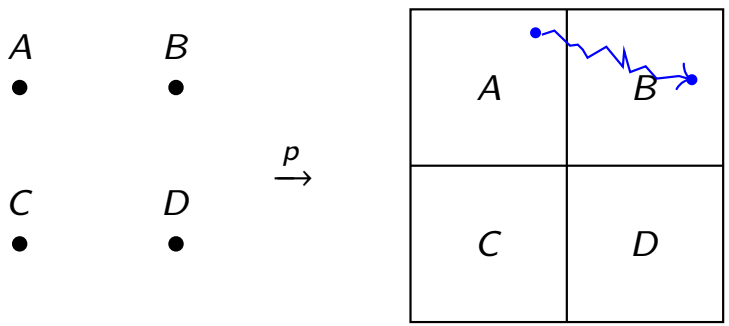


  - a morphism $\sigma\colon S \longrightarrow M(S')$ "sample"

Hierarchical planning with M-MDPs II.

What's a morphism $(A, S, T) \longrightarrow (A', S', T')$, more precisely?

- Think of the unprimed as higher-level, the primed as lower-level
- A morphism as above consists of the following:
  - a morphism $g\colon S' \longrightarrow S$ "coarse-grain"

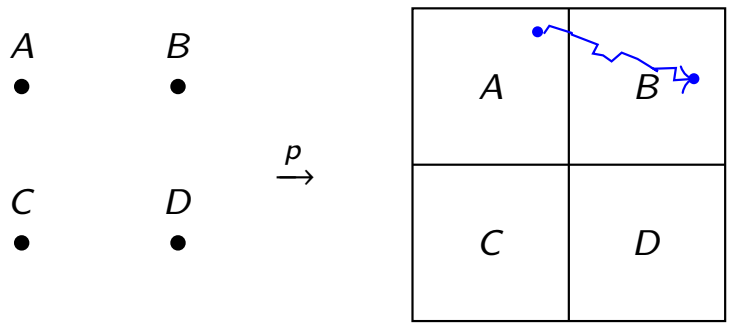


  - a morphism $\sigma\colon S \longrightarrow M(S')$ "sample"
  - a morphism $p\colon A \times S' \longrightarrow A'$ "plan"

Hierarchical planning with M-MDPs II.

What's a morphism $(A, S, T) \longrightarrow (A', S', T')$, more precisely?

- Think of the unprimed as higher-level, the primed as lower-level
- A morphism as above consists of the following:
  - a morphism $g\colon S' \longrightarrow S$ "coarse-grain"



  - a morphism $\sigma\colon S \longrightarrow M(S')$ "sample"
  - a morphism $p\colon A \times S' \longrightarrow A'$ "plan"
- All of these are required to satisfy several properties, ensuring:
  - High-level actions do as planned and executed by low-level.

Combining planners

Assume M is a monoidal monad

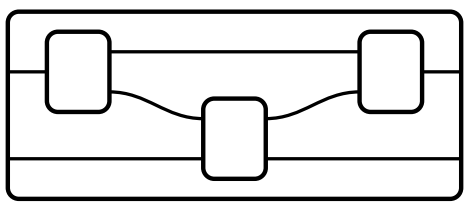
- Technical; all the above work except exceptions
- Exceptions needs minor tweak: must be able to combine exceptions

Combining planners

Assume M is a monoidal monad

- Technical; all the above work except exceptions
- Exceptions needs minor tweak: must be able to combine exceptions

Then the category of M-MDPs is itself a monoidal category. That means:



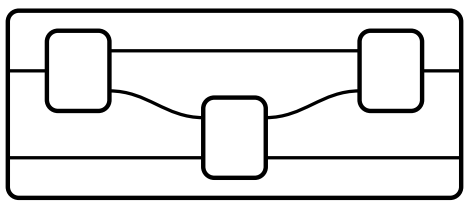
How to read this (very abstract) wiring diagram (WD):

Combining planners

Assume M is a monoidal monad

- Technical; all the above work except exceptions
- Exceptions needs minor tweak: must be able to combine exceptions

Then the category of M-MDPs is itself a monoidal category. That means:



How to read this (very abstract) wiring diagram (WD):

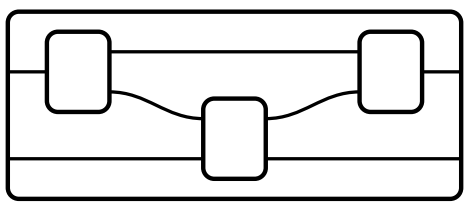
- Wire = M-MDP: provide a monadic function: action $\rightsquigarrow$ state update

Combining planners

Assume M is a monoidal monad

- Technical; all the above work except exceptions
- Exceptions needs minor tweak: must be able to combine exceptions

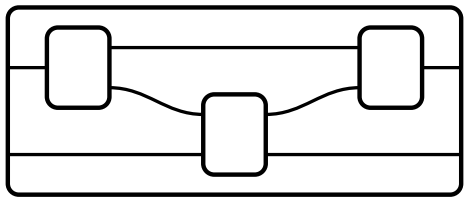
Then the category of M-MDPs is itself a monoidal category. That means:



How to read this (very abstract) wiring diagram (WD):

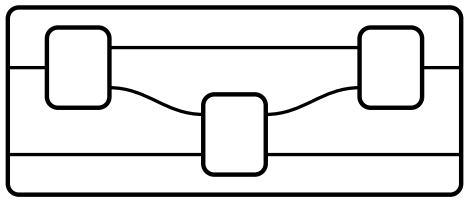
- Wire = M-MDP: provide a monadic function: action $\rightsquigarrow$ state update
- Box = planner: translate higher-level intentions to lower-level actions.
- WD = planning hierarchy, distributing work in a composable way.

Hierarchical planning is associative



- Wire = M-MDP: provides a monadic function: action $\rightsquigarrow$ state update
- Box = planner: translate higher-level intentions to lower-level actions.
- WD = planning hierarchy, distributing work in a composable way.

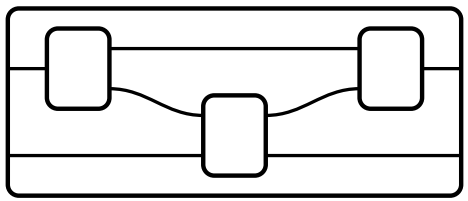
Hierarchical planning is associative



- Wire = M-MDP: provides a monadic function: action $\rightsquigarrow$ state update
- Box = planner: translate higher-level intentions to lower-level actions.
- WD = planning hierarchy, distributing work in a composable way.

Can draw arbitrary wiring diagrams. "It's a monoidal category."

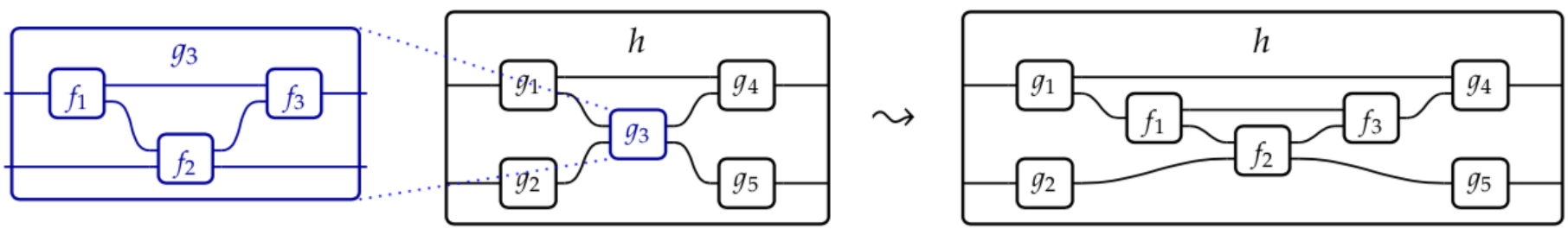
Hierarchical planning is associative



- Wire = M-MDP: provides a monadic function: action $\rightsquigarrow$ state update
- Box = planner: translate higher-level intentions to lower-level actions.
- WD = planning hierarchy, distributing work in a composable way.

Can draw arbitrary wiring diagrams. "It's a monoidal category."

- Theorem: these diagrams satisfy associativity (arbitrary clustering)
- This means you can cluster the WD in any way—well-defined.



Outline

Future work

What makes a high-level planner $T : A \times S \to \mathbf{Dist}(S)$ "powerful"?

Future work

What makes a high-level planner $T : A \times S \to \mathbf{Dist}(S)$ "powerful"?

- Examples and non-examples
  - Non-example: $T(a, s)$ uniform for all $a \in A$, $s \in S$.

Future work

What makes a high-level planner $T : A \times S \to \mathbf{Dist}(S)$ "powerful"?

- Examples and non-examples
  - Non-example: $T(a, s)$ uniform for all $a \in A$, $s \in S$.
  - Non-example: $T(a, s) = T(a', s)$ for all $a, a' \in A$, $s \in S$.

Future work

What makes a high-level planner $T : A \times S \to \mathbf{Dist}(S)$ "powerful"?

- Examples and non-examples
  - Non-example: $T(a, s)$ uniform for all $a \in A$, $s \in S$.
  - Non-example: $T(a, s) = T(a', s)$ for all $a, a' \in A$, $s \in S$.
  - Example: $T(a, s)$ is Dirac for each a, s and $T(-, s)$ injective.

Future work

What makes a high-level planner $T : A \times S \to \mathbf{Dist}(S)$ "powerful"?

- Examples and non-examples
  - Non-example: $T(a, s)$ uniform for all $a \in A$, $s \in S$.
  - Non-example: $T(a, s) = T(a', s)$ for all $a, a' \in A$, $s \in S$.
  - Example: $T(a, s)$ is Dirac for each a, s and $T(-, s)$ injective.
- Empowerment: (intrinsic motivation)
- "Maximize channel capacity between actuators now and sensors in future.

Future work

What makes a high-level planner $T : A \times S \to \mathbf{Dist}(S)$ "powerful"?

- Examples and non-examples
  - Non-example: $T(a, s)$ uniform for all $a \in A$, $s \in S$.
  - Non-example: $T(a, s) = T(a', s)$ for all $a, a' \in A$, $s \in S$.
  - Example: $T(a, s)$ is Dirac for each a, s and $T(-, s)$ injective.
- Empowerment: (intrinsic motivation)
- "Maximize channel capacity between actuators now and sensors in future.
  - Which translators $A \times S' \to A'$ lead to high empowerment?

Future work

What makes a high-level planner $T : A \times S \to \mathbf{Dist}(S)$ "powerful"?

- Examples and non-examples
  - Non-example: $T(a, s)$ uniform for all $a \in A$, $s \in S$.
  - Non-example: $T(a, s) = T(a', s)$ for all $a, a' \in A$, $s \in S$.
  - Example: $T(a, s)$ is Dirac for each a, s and $T(-, s)$ injective.
- Empowerment: (intrinsic motivation)
- "Maximize channel capacity between actuators now and sensors in future.
  - Which translators $A \times S' \to A'$ lead to high empowerment?
  - Find compositional notions of empowerment for planner / translators.

Summary

Hierarchical planning is ubiquitous

- Autonomy: from mission to task execution ... to basic control
- Any team situation: from general to colonel ... to private

Summary

Hierarchical planning is ubiquitous

- Autonomy: from mission to task execution ... to basic control
- Any team situation: from general to colonel ... to private

We model the effects of actions in different ways

Summary

Hierarchical planning is ubiquitous

- Autonomy: from mission to task execution ... to basic control
- Any team situation: from general to colonel ... to private

We model the effects of actions in different ways

- Monads capture many different sorts of models.
- Probability, possibility, task-based, exceptions, rewards, etc.
- Value of parametricity in the monad: kill many birds w/ one stone.

Summary

Hierarchical planning is ubiquitous

- Autonomy: from mission to task execution ... to basic control
- Any team situation: from general to colonel ... to private

We model the effects of actions in different ways

- Monads capture many different sorts of models.
- Probability, possibility, task-based, exceptions, rewards, etc.
- Value of parametricity in the monad: kill many birds w/ one stone.

Formalizing translation systems

- Translation is generally implemented informally (in engineers' heads).

Summary

Hierarchical planning is ubiquitous

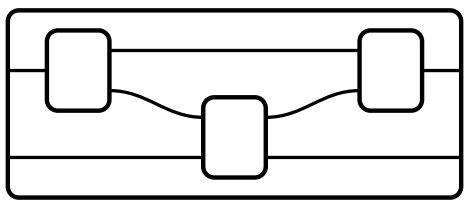
- Autonomy: from mission to task execution ... to basic control
- Any team situation: from general to colonel ... to private

We model the effects of actions in different ways

- Monads capture many different sorts of models.
- Probability, possibility, task-based, exceptions, rewards, etc.
- Value of parametricity in the monad: kill many birds w/ one stone.

Formalizing translation systems

- Translation is generally implemented informally (in engineers' heads).
- CT structure suggests the above generalizations (monad and WDs).



Summary

Hierarchical planning is ubiquitous

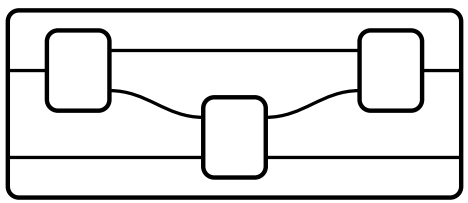
- Autonomy: from mission to task execution ... to basic control
- Any team situation: from general to colonel ... to private

We model the effects of actions in different ways

- Monads capture many different sorts of models.
- Probability, possibility, task-based, exceptions, rewards, etc.
- Value of parametricity in the monad: kill many birds w/ one stone.

Formalizing translation systems

- Translation is generally implemented informally (in engineers' heads).
- CT structure suggests the above generalizations (monad and WDs).



Thanks; comments and questions welcome!