JOHNS HOPKINS
APPLIED PHYSICS LABORATORY

11100 Johns Hopkins Road
Laurel, MD 20723-6099

Contextual affordances in context-aware autonomous systems

Angeline Aguinaldo

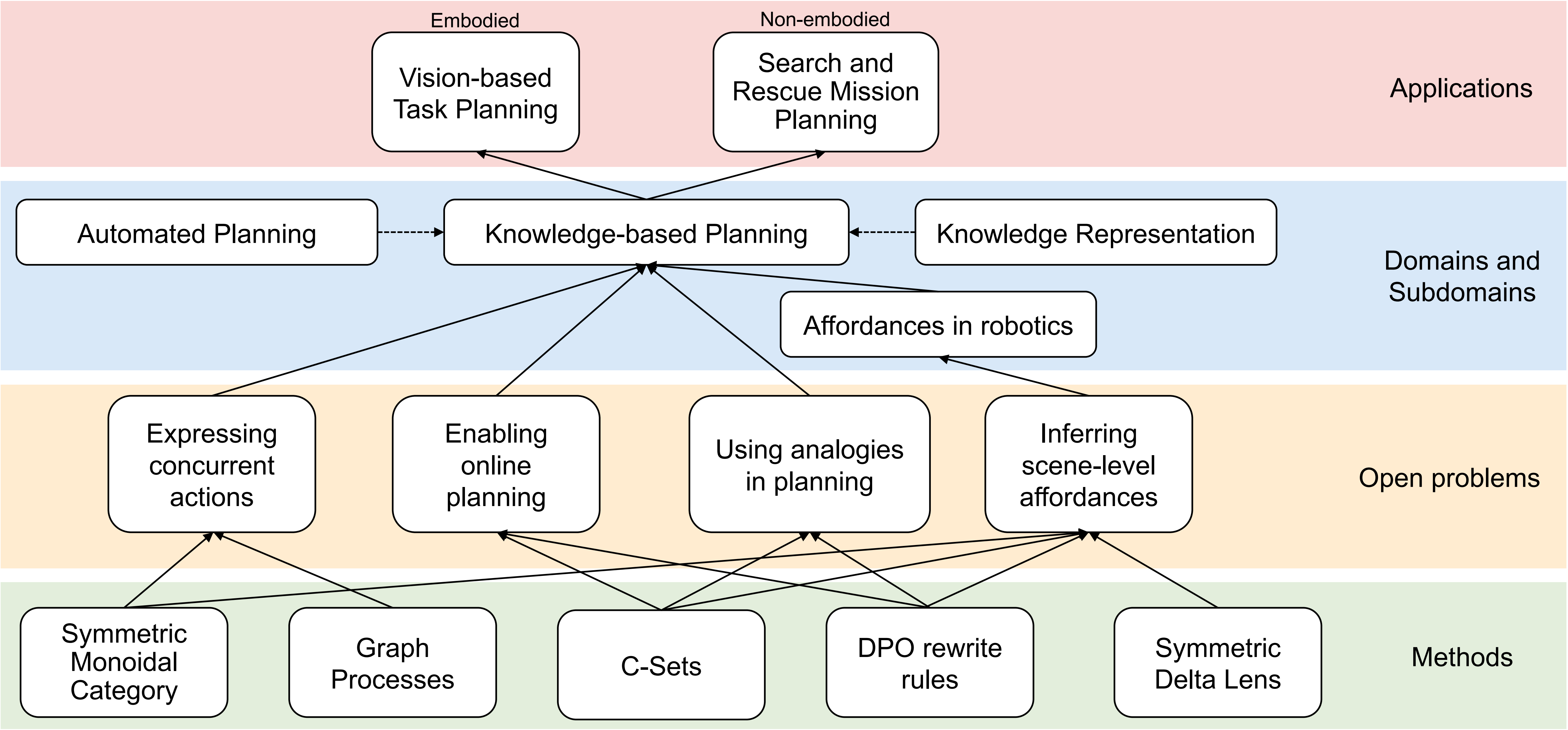
Research Software Engineer, Johns Hopkins University Applied Physics Laboratory
angeline.aguinaldo@jhuapl.edu

Computer Science PhD Student, University of Maryland College Park
aaguinal@cs.umd.edu

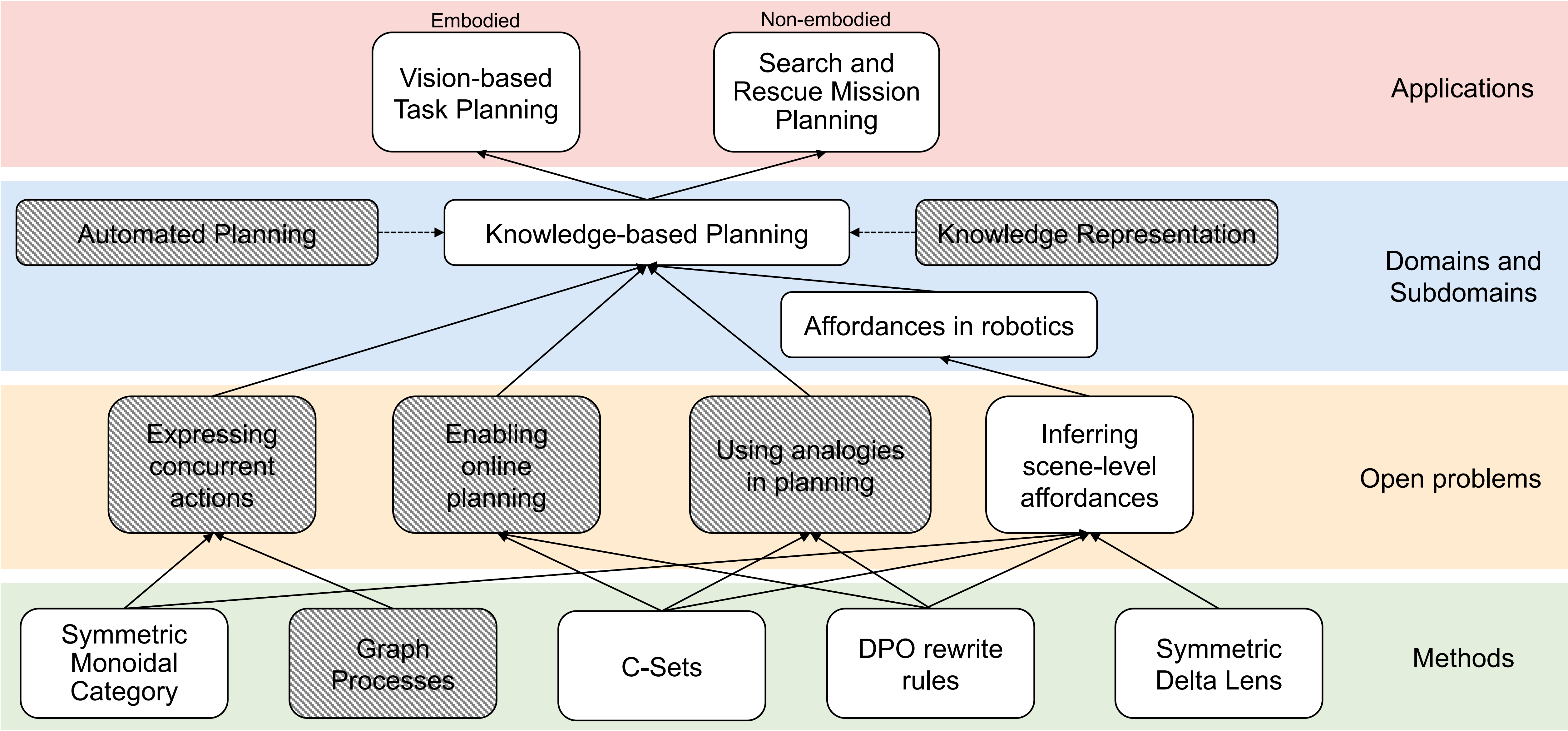
AMS Joint Mathematics Meeting: Applied Category Theory Special Session
Boston, MA
January 2023

Contents

Technical Roadmap



Technical Roadmap



Scene-level affordances in robotics

Mug vs. **Mug with coffee on desk**

https://bootcamp.uxdesign.cc/what-every-game-ux-designer-should-know-about-human-psychology-9f0a325e919e

Object-level

Scene-level

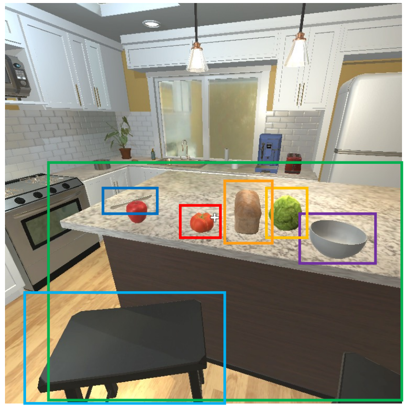
High-level actions often depend on **scene-level arrangements**, as opposed to, object-level features. There is little to no work done towards inferring scene-level affordances. [Lüddecke 2016]

Motivating Example: Kitchen World

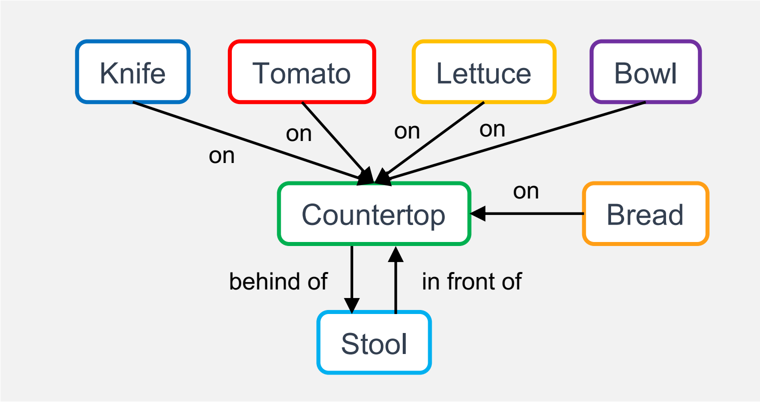
Actions

```
(:action open-object
    :parameters (?obj - Object)
    :precond (not (openness ?obj))
    :effect (openness ?obj))

(:action close-object
    :parameters (?obj - Object)
    :precond (openness ?obj)
    :effect (not (openness ?obj)))

(:action cook-object
    :parameters (?obj - Object)
    :precond (not (cooked ?obj))
    :effect (cooked ?obj))

(:action slice-object
    :parameters (?obj - Object)
    :precond (not (sliced ?obj))
    :effect (sliced ?obj))

(:action pick-up-object
     :parameters (?target-obj - Object
?support-obj - Object ?agent - Agent)
     :precond (and (not (has ?agent ?target-
obj)) (on ?target-obj ?support-obj))
     :effect (and (has ?agent ?target-obj)
(not (on ?target-obj ?support-obj))))

(:action put-object
    :parameters (?target-obj - Object
?support-obj - Object ?agent - Agent)
    :precond (and (has ?agent ?target-obj)
(not (on ?target-obj ?support-obj)))
    :effect (and (on ?target-obj ?support-
obj) (not (has ?agent ?target-obj))))
```

Scenes



Scene Graph



Afforded Actions

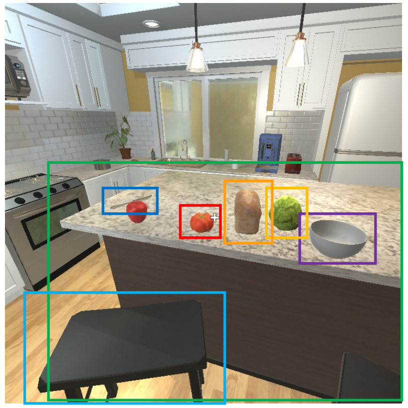
```
(:action pick-up-object
     :parameters (?target-obj - Object
?support-obj - Object ?agent - Agent)
     :precond (and (not (has ?agent ?target-
obj)) (on ?target-obj ?support-obj))
     :effect (and (has ?agent ?target-obj)
(not (on ?target-obj ?support-obj))))
```

Motivating Example: Kitchen World

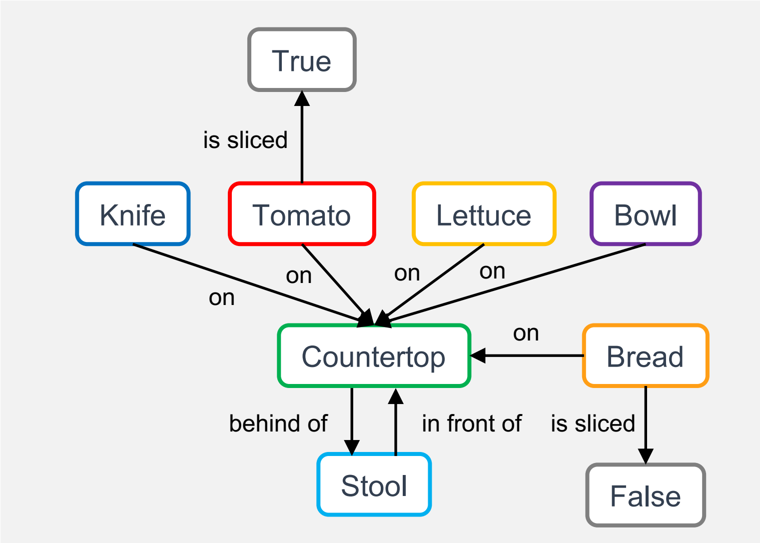
Actions

```
(:action open-object
    :parameters (?obj - Object)
    :precond (not (openness ?obj))
    :effect (openness ?obj))

(:action close-object
    :parameters (?obj - Object)
    :precond (openness ?obj)
    :effect (not (openness ?obj)))

(:action cook-object
    :parameters (?obj - Object)
    :precond (not (cooked ?obj))
    :effect (cooked ?obj))

(:action slice-object
    :parameters (?obj - Object)
    :precond (not (sliced ?obj))
    :effect (sliced ?obj))

(:action pick-up-object
     :parameters (?target-obj - Object
?support-obj - Object ?agent - Agent)
     :precond (and (not (has ?agent ?target-
obj)) (on ?target-obj ?support-obj))
     :effect (and (has ?agent ?target-obj)
(not (on ?target-obj ?support-obj))))

(:action put-object
    :parameters (?target-obj - Object
?support-obj - Object ?agent - Agent)
    :precond (and (has ?agent ?target-obj)
(not (on ?target-obj ?support-obj)))
    :effect (and (on ?target-obj ?support-
obj) (not (has ?agent ?target-obj))))
```

Scenes



Scene Graph



Afforded Actions

```
(:action slice-object
    :parameters (?obj - Object)
    :precond (not (sliced ?obj))
    :effect (sliced ?obj))

(:action pick-up-object
     :parameters (?target-obj - Object
?support-obj - Object ?agent - Agent)
     :precond (and (not (has ?agent ?target-
obj)) (on ?target-obj ?support-obj))
     :effect (and (has ?agent ?target-obj)
(not (on ?target-obj ?support-obj))))
```

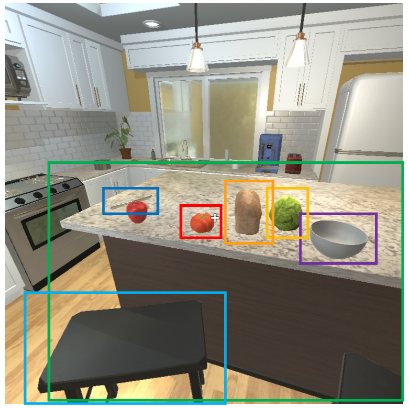
Motivating Example: Kitchen World

Actions

```
(:action open-object
    :parameters (?obj - Object)
    :precond (not (openness ?obj))
    :effect (openness ?obj))

(:action close-object
    :parameters (?obj - Object)
    :precond (openness ?obj)
    :effect (not (openness ?obj)))

(:action cook-object
    :parameters (?obj - Object)
    :precond (not (cooked ?obj))
    :effect (cooked ?obj))

(:action slice-object
    :parameters (?obj - Object)
    :precond (not (sliced ?obj))
    :effect (sliced ?obj))

(:action pick-up-object
     :parameters (?target-obj - Object
?support-obj - Object ?agent - Agent)
     :precond (and (not (has ?agent ?target-
obj)) (on ?target-obj ?support-obj))
     :effect (and (has ?agent ?target-obj)
(not (on ?target-obj ?support-obj))))

(:action put-object
    :parameters (?target-obj - Object
?support-obj - Object ?agent - Agent)
    :precond (and (has ?agent ?target-obj)
(not (on ?target-obj ?support-obj)))
    :effect (and (on ?target-obj ?support-
obj) (not (has ?agent ?target-obj))))
```

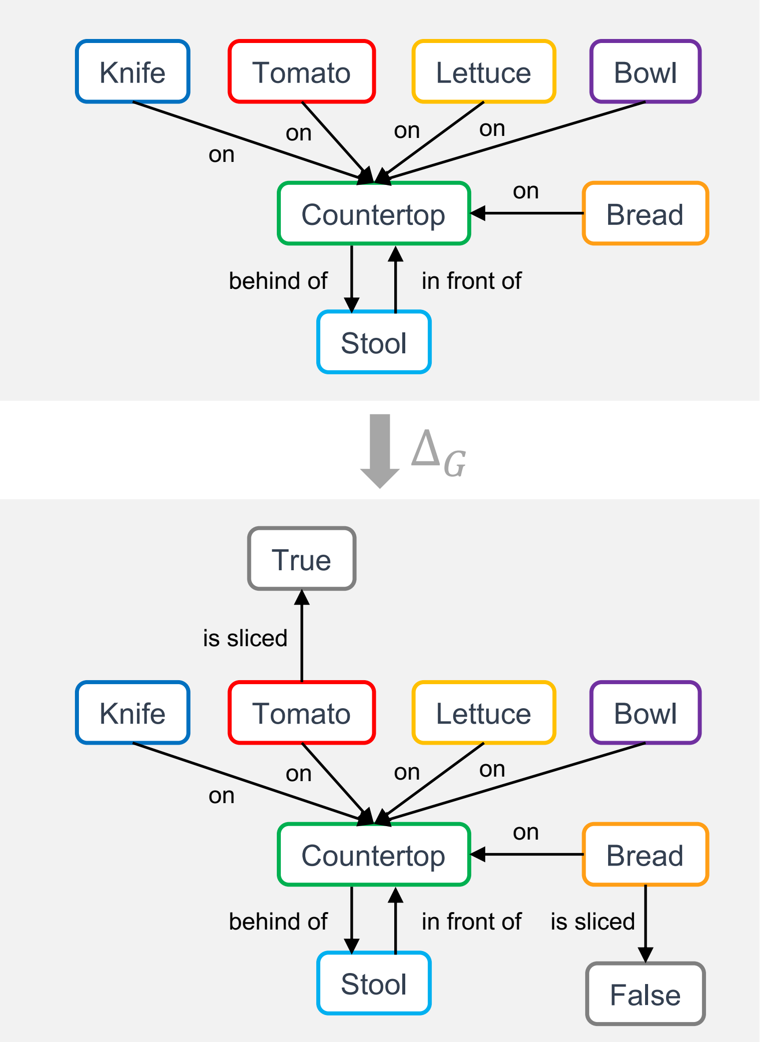
PDDL

Scenes



AI2THOR

Initial Scene Graph



Afforded Actions

```
(:action pick-up-object
     :parameters (?target-obj - Object
?support-obj - Object ?agent - Agent)
     :precond (and (not (has ?agent ?target-
obj)) (on ?target-obj ?support-obj))
     :effect (and (has ?agent ?target-obj)
(not (on ?target-obj ?support-obj))))
```

Δ_A

```
(:action slice-object
    :parameters (?obj - Object)
    :precond (not (sliced ?obj))
    :effect (sliced ?obj))

(:action pick-up-object
     :parameters (?target-obj - Object
?support-obj - Object ?agent - Agent)
     :precond (and (not (has ?agent ?target-
obj)) (on ?target-obj ?support-obj))
     :effect (and (has ?agent ?target-obj)
(not (on ?target-obj ?support-obj))))
```

Using symmetric delta lenses for the affordance relation

Method

Language of Scene Graphs

Scene graphs are a topological representation of objects and their relationships in a scene.

Def. A ***scene graph***, S, consists of:

i. A schema, or ontology, consisting of classes (C), primitive types (T), relations (R, R_a) between classes and types, and inference rules

   e.g. $(\text{person}, \text{driving}, \text{car}) \Rightarrow \neg(\text{person}, \text{walking}, \text{crosswalk})$

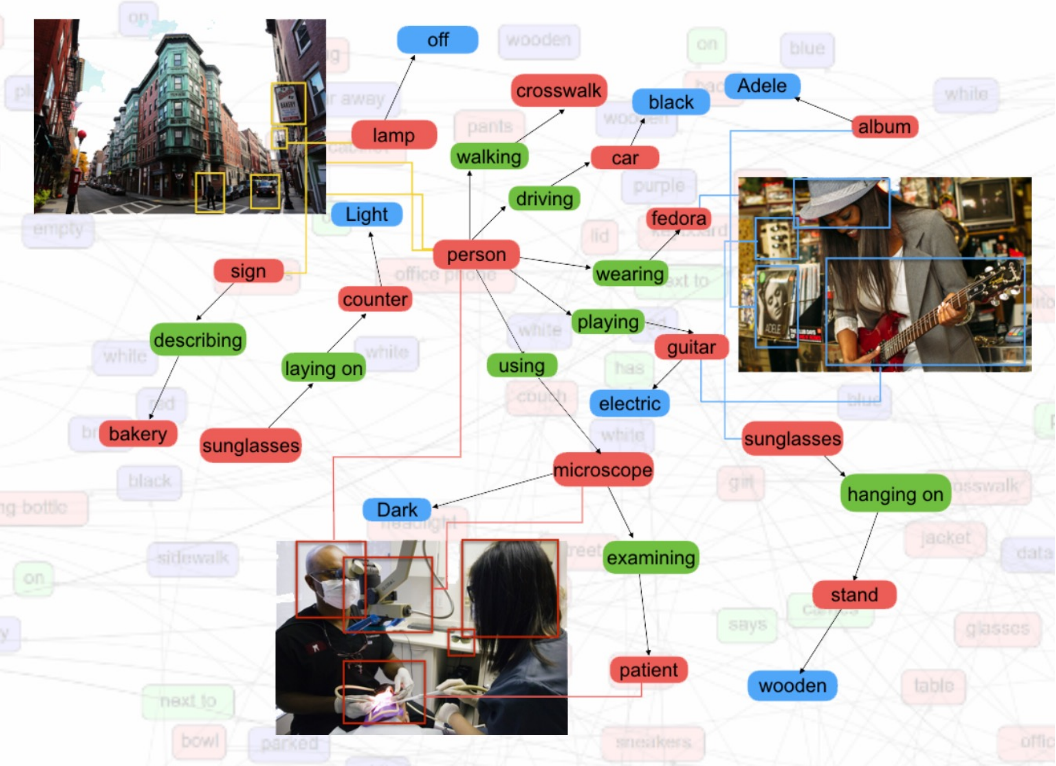
ii. A set of object-object relations

$$(x :: c_1, r, x' :: c_2)$$

iii. A set of object-attribute relations

$$(x :: c_1, r_a, b :: t)$$

Categorically, a scene graph can be represented as a **co-presheave** ($\mathbb{C}$**-Set**) where the schema category, $\mathbb{C}$, is the ontology and the target sets are the specific instances of each class. The arrows are natural transformations.



https://visualgenome.org/

$\mathbb{C} - \mathbf{Set}$ and Attributed $\mathbb{C} - \mathbf{Set}$

Def. A (finite) $\mathbb{C} - \mathbf{Set}$ is a functor from $\mathbb{C} \to \mathrm{FinSet}$.

For computable examples, we assume finitely presented categories.

Def. A (finite) **attribute** $\mathbb{C} - \mathbf{Set}$ is a functor, F, from a finitely presented schema category, $\mathbb{C}$, to Set, where $\mathbb{C}$ is partitioned using a map $S: \mathbb{C} \to \mathbf{2}$.

The preimage $S^{-1}(0)$ isolates the combinatorial structure.

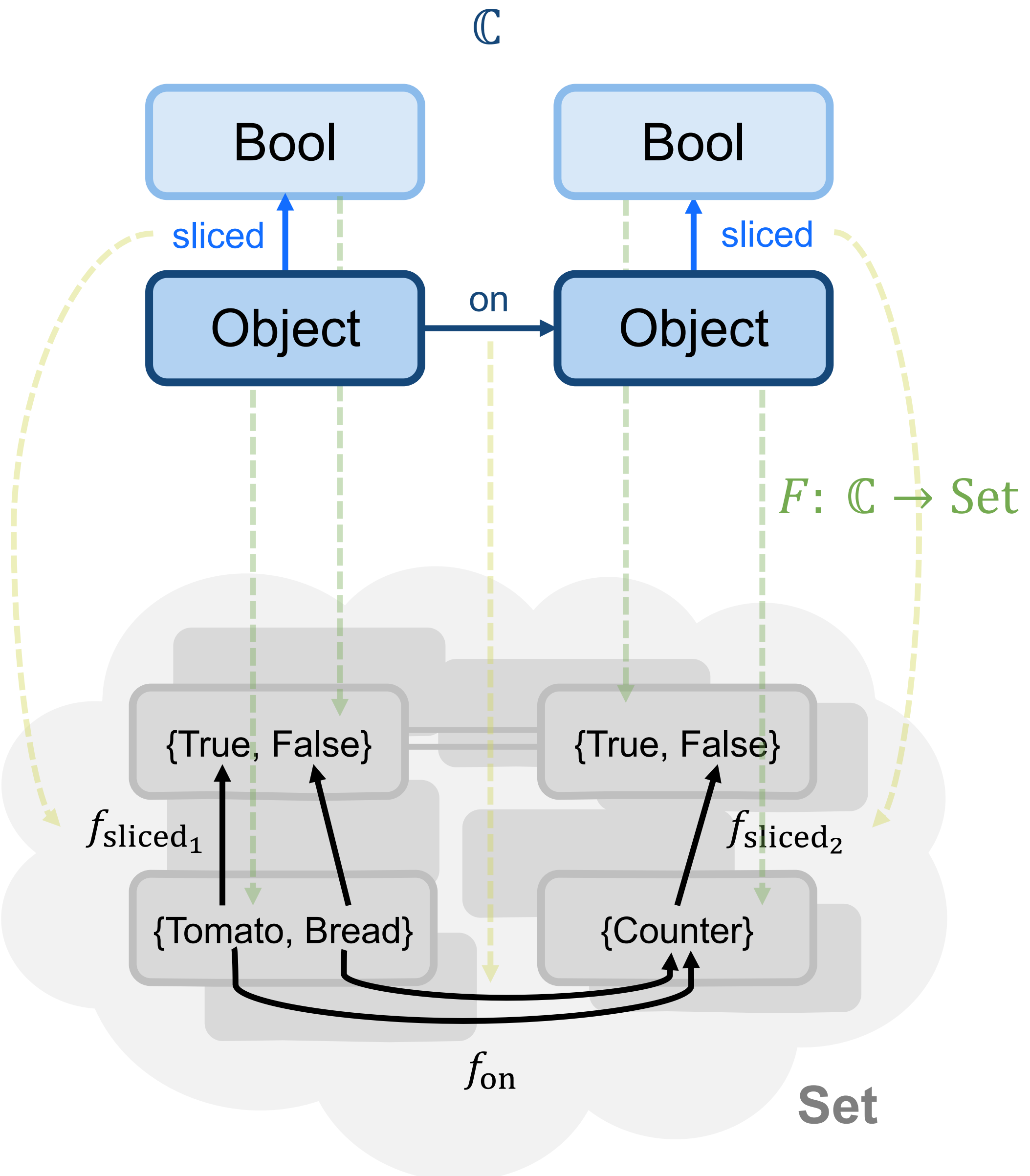
The preimage $S^{-1}(1)$ isolates the attribute structure.

The preimage $S^{-1}(0 \to 1)$ isolates the arrows between the combinatorial structure and the attribute structure.

Note: The Grothendieck construction, $\int F$, translates to RDF triples, e.g.

(Tomato :: Object, f_{on} :: on, Counter :: Object)

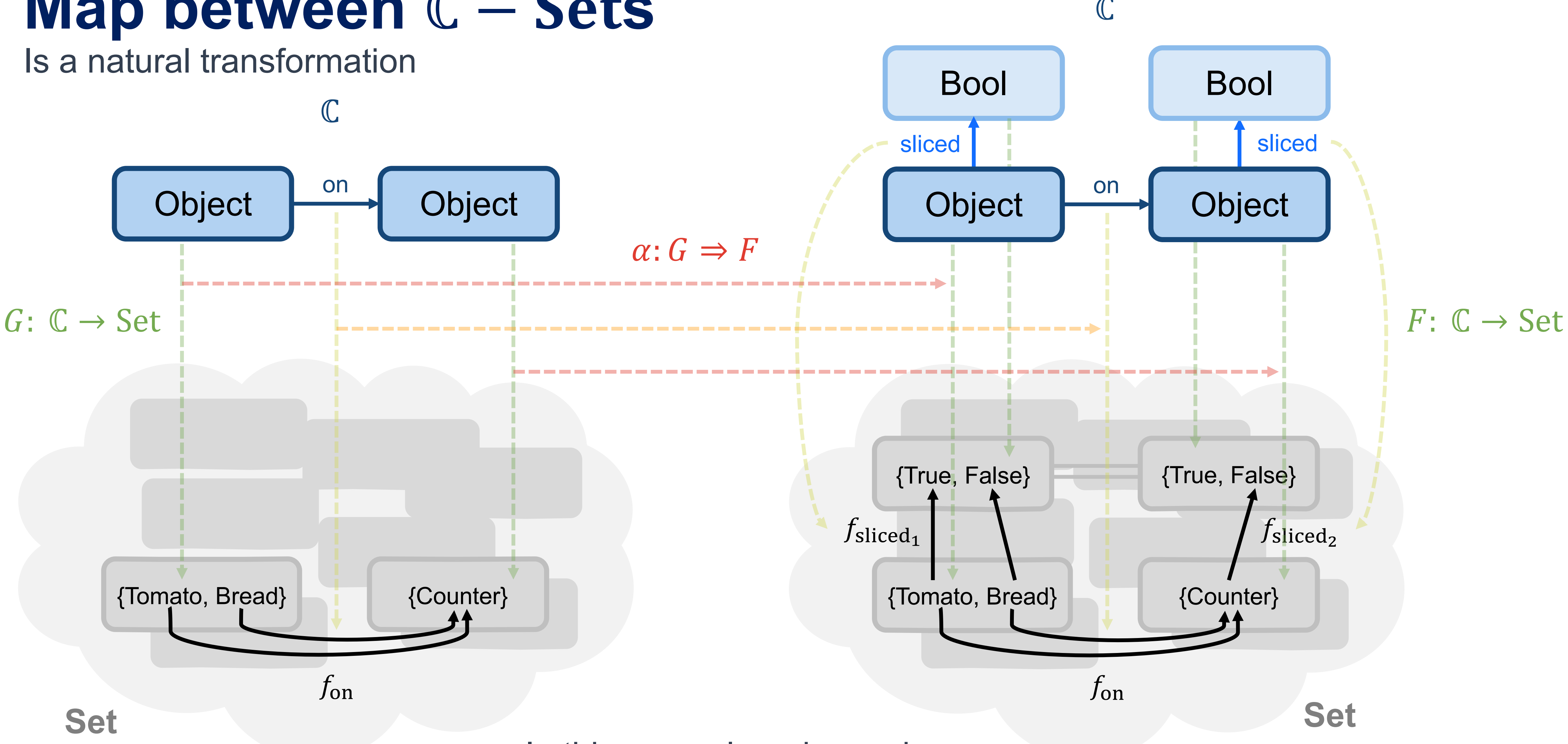
(Tomato :: Object, $f_{sliced1}$ :: sliced, True :: Bool)



Patterson, E., Lynch, O., & Fairbanks, J. (2021). *Categorical Data Structures for Technical Computing*. *4*(5), 1–27. http://arxiv.org/abs/2106.04703

Map between $\mathbb{C}-\mathbf{Sets}$

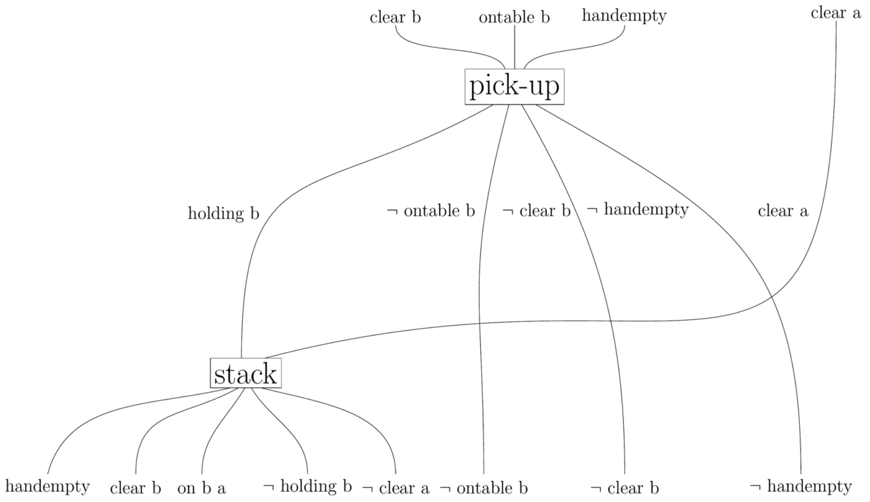
Is a natural transformation



In this example, α is monic.

Language of Planning Domains

Planning domains are a set of atomic action operators that can be composed to form a sequence of actions, or task plan.



Categorically, a STRIPS-based planning domain can be represented as a **symmetric monoidal category** where the generating objects are literals, the generating arrows are action operators, and the tensor product is conjunction. Positive and negated sentences are considered unique objects with no relation.

Aguinaldo A., Regli W. Encoding Compositionality in Classical Planning Solutions. IJCAI Workshop on Generalization in Planning 2021.

Def. A ***planning domain***, P, consists of a set of action schemas with parameters (`parameters`), preconditions (`precond`), effects (`effect`).

Preconditions and effects in an action operator consist of a conjunction of fluents.

```
(:action pick-up-object
     :precond (and (not (has MyRobo Tomato)) (on Tomato
Counter))
     :effect (and (has MyRobo Tomato) (not (on Tomato
Counter))))
```

A set of action operators can be lifted to be universally quantified over all variables to form an action schema. Preconditions and effects in an action operator consist of a conjunction of literals.

```
(:action pick-up-object
     :parameters (?target-obj – Object ?support-obj – Object
?agent – Agent)
     :precond (and (not (has ?agent ?target-obj)) (on
?target-obj ?support-obj))
     :effect (and (has ?agent ?target-obj)
(not (on ?target-obj ?support-obj))))
```

Symmetric monoidal categories

Def. A ***symmetric monoidal category,*** $\mathbb{M}$**,** is a category with the following additional properties:

- A unit object, $I \in \mathbb{M}$
- A tensor product, $\otimes : \mathbb{M} \times \mathbb{M} \rightarrow \mathbb{M}$
- An associative isomorphism,

$$\alpha_{X,Y,Z} : (X \otimes Y) \otimes Z \rightarrow X \otimes (Y \otimes Z)$$

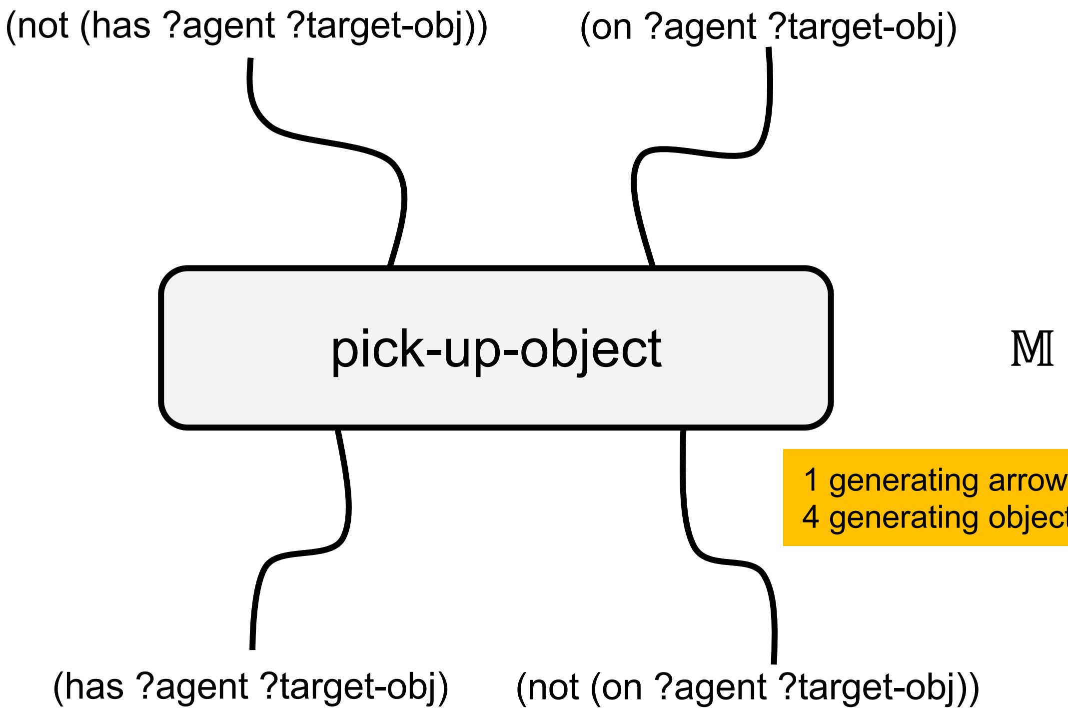
- Left and right unitor isomorphisms,

$$\rho_l : I \otimes X \rightarrow X \text{ and } \rho_R : X \otimes I \rightarrow X$$

- And a braiding isomorphism,

$$B_{X,Y} : X \otimes Y \rightarrow Y \otimes X$$

```
(:action pick-up-object
     :parameters (?target-obj - Object ?support-obj - Object
?agent - Agent)
     :precond (and (not (has ?agent ?target-obj)) (on
?target-obj ?support-obj))
     :effect (and (has ?agent ?target-obj)
(not (on ?target-obj ?support-obj))))
```



1 generating arrow
4 generating objects

Joyal, A.; and Street, R. 1991. The geometry of tensor calculus, I. Advances in Mathematics 88(1): 55 – 112. ISSN 0001-8708. doi:https://doi.org/10.1016/0001- 8708(91)90003-P. URL http://www.sciencedirect.com/ science/article/pii/000187089190003P.

Functor between symmetric monoidal categories

Symmetric Monoidal Category, $\mathbb{M}$

Objects:

- $\mathrm{X} = (\mathrm{not}\ (\mathrm{has}\ ?\mathrm{agent}\ ?\mathrm{target}-\mathrm{obj}))$
- $\mathrm{Y} = (\mathrm{on}\ ?\mathrm{agent}\ ?\mathrm{target}-\mathrm{obj})$
- $\mathrm{Z} = (\mathrm{has}\ ?\mathrm{agent}\ ?\mathrm{target}-\mathrm{obj})$
- $\mathrm{U} = (\mathrm{not}\ (\mathrm{on}\ ?\mathrm{agent}\ ?\mathrm{target}-\mathrm{obj}))$

Arrows:

- pick−up−object: $\mathrm{X} \otimes \mathrm{Y} \rightarrow \mathrm{Z} \otimes \mathrm{U}$

Symmetric Monoidal Category, $\mathbb{M}'$

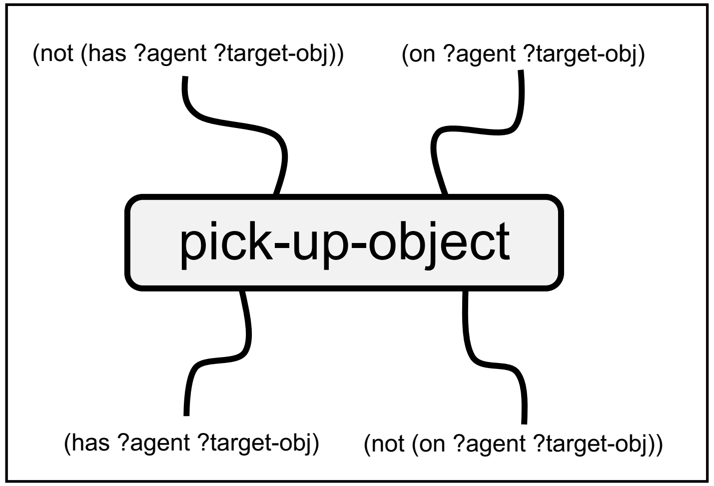
Objects:

- $\mathrm{X} = (\mathrm{not}\ (\mathrm{has}\ ?\mathrm{agent}\ ?\mathrm{target}-\mathrm{obj}))$
- $\mathrm{Y} = (\mathrm{on}\ ?\mathrm{agent}\ ?\mathrm{target}-\mathrm{obj})$
- $\mathrm{Z} = (\mathrm{has}\ ?\mathrm{agent}\ ?\mathrm{target}-\mathrm{obj})$
- $\mathrm{U} = (\mathrm{not}\ (\mathrm{on}\ ?\mathrm{agent}\ ?\mathrm{target}-\mathrm{obj}))$
- $\mathrm{T} = (\mathrm{not}\ (\mathrm{sliced}\ ?\ \mathrm{obj}))$
- $\mathrm{R} = (\mathrm{sliced}\ ?\ \mathrm{obj})$

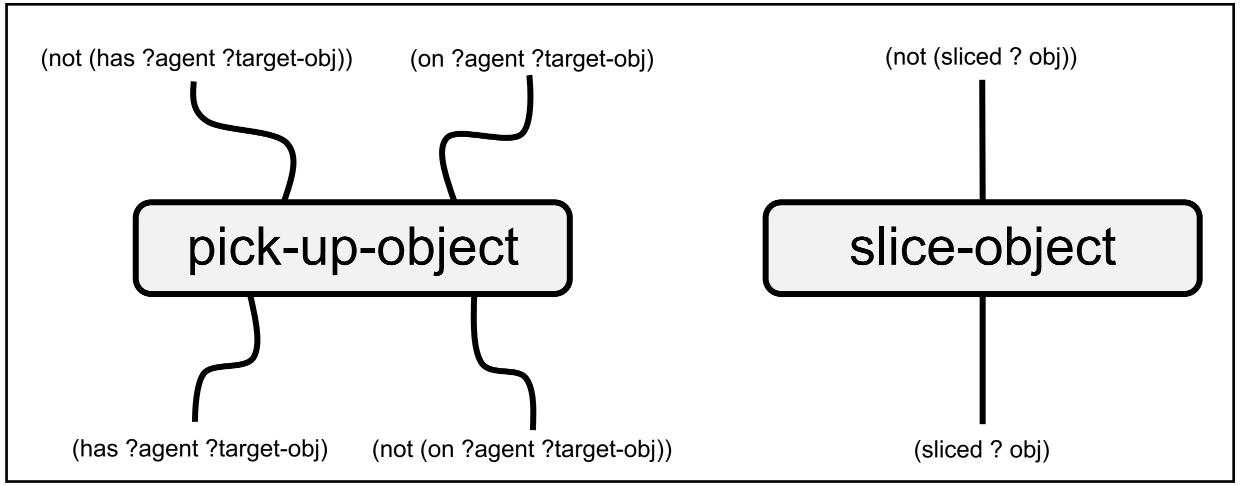
Arrows:

- pick−up−object: $\mathrm{X} \otimes \mathrm{Y} \rightarrow \mathrm{Z} \otimes \mathrm{U}$
- slice−object: $\mathrm{T} \rightarrow \mathrm{R}$

Symmetric monoidal functor, H, is a functor that preserves monoidal and braiding isomorphisms.



$H\colon \mathbb{M} \rightarrow \mathbb{M}'$



Affordance relation using functors: Object maps

Functor G

Planning Domain

grounding, G

Scene Graph

```
(:action pick-up-object
     :parameters (?target-obj - Object ?support-
obj - Object)
     :precond (on ?target-obj ?support-obj)
     :effect(not (on ?target-obj ?support-obj)))
```

$\downarrow$

$$\begin{tikzcd} \mathbb{C} \arrow[d, "F"'] & \text{Object} \arrow[r, "\text{on}"] \arrow[d, dash, dashed] & \text{Object} \arrow[d, dash, dashed] \\ \textbf{Set} & \{\text{Tomato}\} \arrow[r, "\text{on}"'] & \{\text{Counter}\} \end{tikzcd}$$

Functor G'

Scene Graph

reverse grounding, G'

Planning Domain

$$\begin{tikzcd} & \text{Bool} & \text{Bool} \\ \mathbb{C} \arrow[d, "F"'] & \text{Object} \arrow[u, "\text{sliced}"'] \arrow[r, "\text{on}"] \arrow[d, dash, dashed] & \text{Object} \arrow[u, "\text{sliced}"'] \arrow[d, dash, dashed] \\ \textbf{Set} & \{\text{Tomato}\} \arrow[r, "\text{on}"'] \arrow[d, "\text{sliced}"] & \{\text{Counter}\} \arrow[d, "\text{sliced}"] \\ & \{\text{True}, \text{False}\} & \{\text{True}, \text{False}\} \end{tikzcd}$$

$\downarrow$

```
(:action pick-up-object
     :parameters (?target-obj - Object ?support-
obj - Object)
     :precond (on ?target-obj ?support-obj)
     :effect(not (on ?target-obj ?support-obj)))

(:action slice-object
    :parameters (?obj - Object)
    :precond (not (sliced ?obj))
    :effect (sliced ?obj))
```

Showing only object maps

Affordance relation using functors: Arrow maps

Planning Domain

$$\Big\downarrow \textit{grounding}, G$$

Scene Graph

```
(:action pick-up-object
      :parameters (?target-obj – Object
?support-obj – Object)
      :precond (on ?target-obj ?support-
obj)
      :effect(not (on ?target-obj
?support-obj)))
```

```
(:action pick-up-object
      :parameters (?target-obj – Object
?support-obj – Object)
      :precond (on ?target-obj ?support-
obj)
      :effect(not (on ?target-obj
?support-obj)))
```

```
(:action pick-up-object
      :parameters (?target-obj – Object
?support-obj – Object)
      :precond (on ?target-obj ?support-obj)
      :effect(not (on ?target-obj ?support-
obj)))

(:action slice-object
    :parameters (?obj – Object)
    :precond (not (sliced ?obj))
    :effect (sliced ?obj))
```

$$\begin{tikzcd} \mathbb{C} \arrow[d, "F"'] & \text{Object} \arrow[r, "\text{on}"] \arrow[d, dashed, no head] & \text{Object} \arrow[d, dashed, no head] \\ \mathbf{Set} & \{\text{Tomato}\} \arrow[r, "\text{on}"'] & \{\text{Counter}\} \end{tikzcd}$$

$$\begin{tikzcd} \mathbb{C} \arrow[d, "H"'] & \text{Object} \arrow[r, "\text{on}"] \arrow[d, dashed, no head] & \text{Object} \arrow[d, dashed, no head] \\ \mathbf{Set} & \{\text{Tomato}\} \arrow[r, "\text{on}"'] & \{\text{Counter}\} \end{tikzcd}$$

$$\begin{tikzcd} & \text{Bool} & \text{Bool} \\ \mathbb{C} \arrow[d, "G"'] & \text{Object} \arrow[u, "\text{sliced}"'] \arrow[r, "\text{on}"] \arrow[d, dashed, no head] & \text{Object} \arrow[u, "\text{sliced}"'] \arrow[d, dashed, no head] \\ \mathbf{Set} & \{\text{Tomato}\} \arrow[r, "\text{on}"'] \arrow[d, "\text{sliced}"] & \{\text{Counter}\} \arrow[d, "\text{sliced}"] \\ & \{\text{True}, \text{False}\} & \{\text{True}, \text{False}\} \end{tikzcd}$$

Affordance relation using functors: Arrow maps

Scene Graph

reverse grounding, G'

Planning Domain

$$\begin{tikzcd} \mathbb{C} \arrow[d, "H"'] & \text{Object} \arrow[r, "\text{on}"] \arrow[d, dashed, no head] & \text{Object} \arrow[d, dashed, no head] \\ \mathbf{Set} & \{\text{Tomato}\} \arrow[r, "\text{on}"'] & \{\text{Counter}\} \end{tikzcd}$$

$$\begin{tikzcd} \mathbb{C} \arrow[d, "F"'] & \text{Object} \arrow[r, "\text{on}"] \arrow[d, dashed, no head] & \text{Object} \arrow[d, dashed, no head] \\ \mathbf{Set} & \{\text{Tomato}\} \arrow[r, "\text{on}"'] & \{\text{Counter}\} \end{tikzcd}$$

$$\begin{tikzcd} & \text{Bool} & \text{Bool} \\ \mathbb{C} \arrow[d, "G"'] & \text{Object} \arrow[u, "\text{sliced}"'] \arrow[r, "\text{on}"] \arrow[d, dashed, no head] & \text{Object} \arrow[u, "\text{sliced}"'] \arrow[d, dashed, no head] \\ \mathbf{Set} & \{\text{Tomato}\} \arrow[r, "\text{on}"'] \arrow[d, "\text{sliced}"] & \{\text{Counter}\} \arrow[d, "\text{sliced}"] \\ & \{\text{True}, \text{False}\} & \{\text{True}, \text{False}\} \end{tikzcd}$$

```
(:action pick-up-object
     :parameters (?target-obj - Object
?support-obj - Object)
     :precond (on ?target-obj ?support-
obj)
     :effect(not (on ?target-obj
?support-obj)))
```

```
(:action pick-up-object
     :parameters (?target-obj - Object
?support-obj - Object)
     :precond (on ?target-obj ?support-
obj)
     :effect(not (on ?target-obj
?support-obj)))
```

```
(:action pick-up-object
     :parameters (?target-obj - Object
?support-obj - Object)
     :precond (on ?target-obj ?support-obj)
     :effect(not (on ?target-obj ?support-
obj)))

(:action slice-object
    :parameters (?obj - Object)
    :precond (not (sliced ?obj))
    :effect (sliced ?obj))
```

Inferring affordances using symmetric delta lens

Claim. Symmetric delta lens construct the affordance relation.

Def. ***Symmetric delta lenses***
1. Delta lens (G, ϕ): $\mathbb{A} \rightarrow \mathbb{B}$
2. Delta lens (G', ϕ'): $\mathbb{B} \rightarrow \mathbb{A}$

<u>Axioms</u>
Lifting operations, ϕ, ϕ', provides unique lifts. They preserve compositions and identities.

Functors, G, G', are arbitrary functors.

Johnson 2016

$$\begin{tikzcd}
\mathbb{A} \arrow[d, "G"] & a \arrow[r, "\phi_{(w,a)}"] \arrow[d, dashed, no head] & \phi_{(w,a)}(a) \arrow[d, dashed, no head] \\
\mathbb{B} & G(a) \arrow[r, "w"] & b
\end{tikzcd}$$

$$\begin{tikzcd}
\mathbb{B} \arrow[d, "G'"] & b' \arrow[r, "\phi'_{(u,b')}"] \arrow[d, dashed, no head] & \phi'_{(u,b')}(b') \arrow[d, dashed, no head] \\
\mathbb{A} & G'(b') \arrow[r, "u"] & a'
\end{tikzcd}$$

$\mathbb{A}$ ~ category of planning domains
$\mathbb{B}$ ~ category of scene graphs

Within each category, $(\mathbb{A}, \mathbb{B})$
- Objects are models
- Arrows, f, are model updates (deltas)

$$f : L \rightarrow R \quad := \quad \begin{tikzcd}
& A = \mathrm{Shared}(L, R) \arrow[dl, "\mathrm{Diff}(A, L)"'] \arrow[dr, "\mathrm{Diff}(A, R)"] & \\
L & & R
\end{tikzcd}$$

What kind of queries can we answer?

Planning Domains

grounding

Scene Graph

$$\begin{tikzcd} \mathbb{A} \arrow[d, "G"] & a \arrow[r, "\phi_{(w,a)}"] \arrow[d, dashed, no head] & \phi_{(w,a)}(a) \arrow[d, dashed, no head] \\ \mathbb{B} & G(a) \arrow[r, "w"] & b \end{tikzcd}$$

Scene Graph

reverse grounding

Planning Domains

$$\begin{tikzcd} \mathbb{B} \arrow[d, "G'"] & b' \arrow[r, "\phi'_{(u,b')}"] \arrow[d, dashed, no head] & \phi'_{(u,b')}(b') \arrow[d, dashed, no head] \\ \mathbb{A} & G'(b') \arrow[r, "u"] & a' \end{tikzcd}$$

Queries

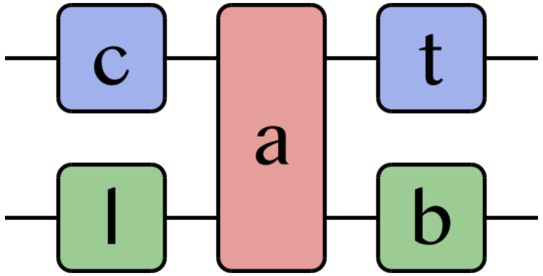
i.	What is $G(a)$?	"What scene graph is afforded by this planning domain?"
ii.	What is $\phi_{(w,a)}$?	"Given a change in the scene graph, what changes in the afforded planning domains?"
iii.	What is $G'(b')$?	"What planning domain is afforded by this scene graph?"
iv.	What is $\phi'_{(u,b')}$?	"Given a change in the afforded planning domains, what changes are necessary in the scene graph?"

Ongoing Work

Operationalization and evaluation

Computational categories in development

```julia
using Catlab, Catlab.Theories
using AlgebraicPlanning

# Schema
########

# Base schema
#------------

@present SpecKitchen(FreeMCategory) begin
  Entity::Ob

  Food::Ob
  food_in_on::Hom(Food, Entity)
  food_is_entity::Hom(Food, Entity)
  ::Tight(food_is_entity)

  Kitchenware::Ob
  ware_in_on::Hom(Kitchenware, Entity)
  ware_is_entity::Hom(Kitchenware, Entity)
  ::Tight(ware_is_entity)
end

function add_food!(pres::Presentation, name::Symbol)
  add_entity!(pres, name, type=:Food)
end
function add_kitchenware!(pres::Presentation, name::Symbol)
  add_entity!(pres, name, type=:Kitchenware, is_a=:is_ware)
end

function add_entity!(pres::Presentation{MCategory}, name::Symbol;
                     type::Symbol=:Entity, is_a::Union{Symbol,Nothing}=nothing)
  isnothing(is_a) && (is_a = Symbol("is_", snakecase(type)))
  ob = add_generator!(pres, Ob(FreeMCategory, name))
  is_a_name = Symbol(snakecase(name), "_", is_a)
  is_a_hom = add_generator!(pres, Hom(is_a_name, ob, pres[type]))
  add_generator!(pres, Tight(nothing, is_a_hom))
end
```



https://github.com/AlgebraicJulia/Catlab.jl

Features

- ☑ C-sets (copresheaves)
- ☑ Symmetric monoidal categories
- ☑ Categorical database migration
- ❑ RDF to C-set serialization
- ❑ PDDL to SMC serialization
- ❑ Lenses

In collaboration with Evan Patterson, James Fairbanks, Owen Lynch, Kris Brown, Sophie Libkind

A. Aguinaldo. Using categorical logic for AI planning. 2022. Blogpost: https://www.algebraicjulia.org/blog/post/2022/09/ai-planning-cset/

Thanks for listening!

Please feel free to reach out with questions, suggestions, or related projects.

Angeline Aguinaldo
aaguinal@cs.umd.edu

JOHNS HOPKINS

APPLIED PHYSICS LABORATORY