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

NIST Compositional Structures for Systems Engineering and Design Workshop
National Cybersecurity Center of Excellence
November 2022

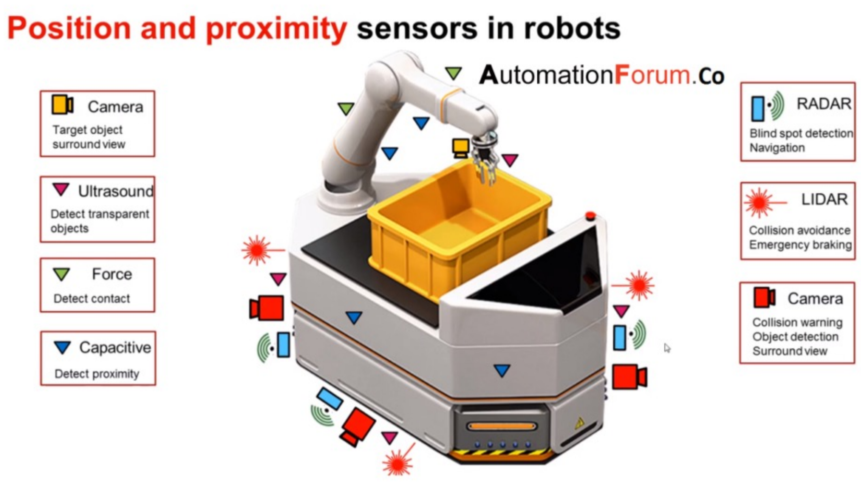
Contents

1. What is contextual affordance?
   - Motivating example
2. Using symmetric delta lenses for the affordance relation
   - What are the structures?
   - What kind of queries can we answer?
3. Ongoing work
   - Developing a categorical database using AlgebraicJulia
   - Test and evaluation plan

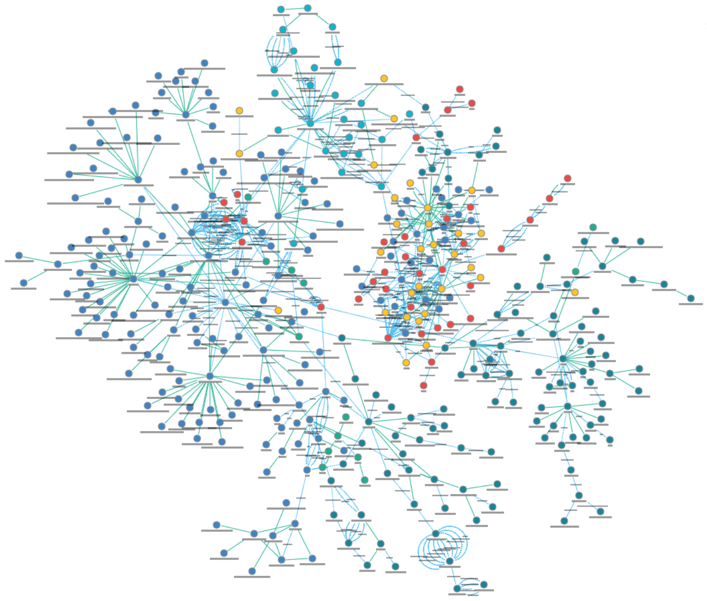
What is contextual affordance?

Background

Making use of context in robotics



https://automationforum.co/what-are-sensors-on-a-robot-and-why-are-sensors-important-to-robots/



A **context-aware autonomous agent** is one that is able to adjust its behavior in response to dynamic context information.

A **knowledge-based agent** makes use of structured representations of knowledge to decide what action to take next.

Affordance relation in robotics

(Barck-Holst 2009), (Cruz 2016), (Kruger 2011), (Montesano 2007)

Objects in environment ∼ Actions of agent

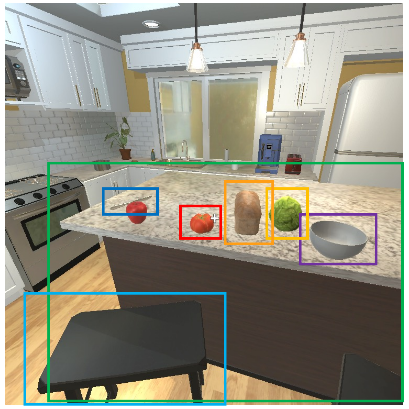
Motivating Example

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))))
```

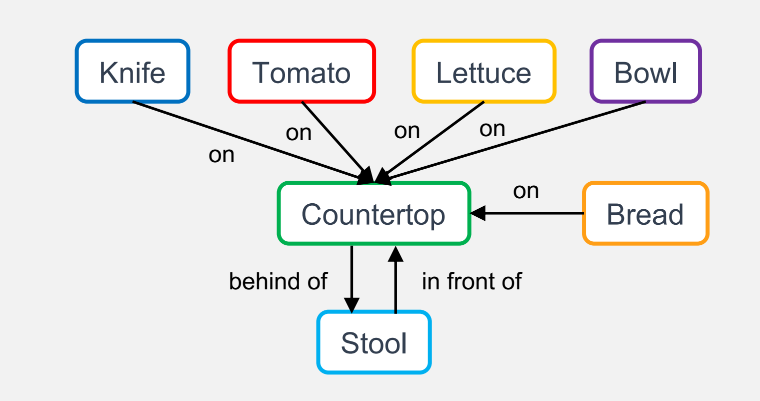
PDDL

Scenes



AI2THOR

Initial Scene Graph



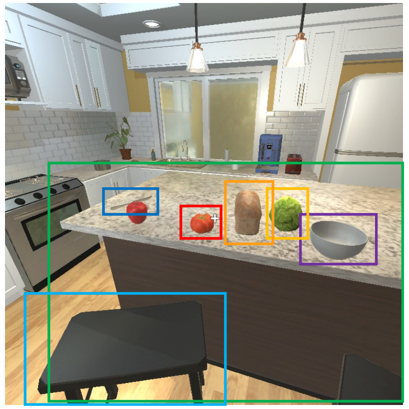
Motivating Example

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))))
```

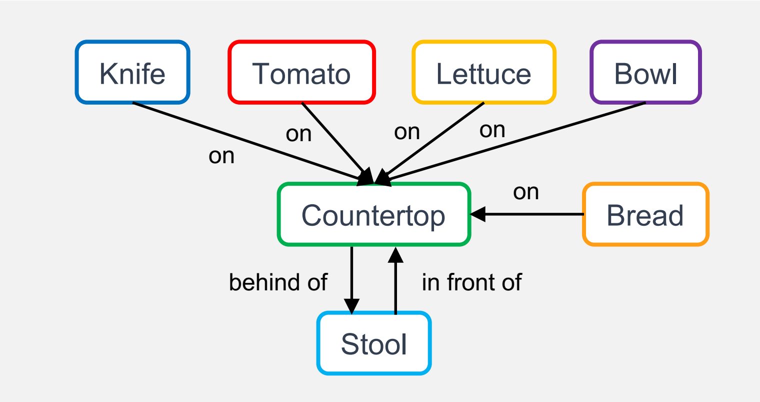
PDDL

Scenes



AI2THOR

Initial Scene Graph



Afforded Task Plans

```
make-veggie-sandwich :=
    slice-object Lettuce
    slice-object Tomato
    slice-object Bread
    put-object Lettuce Bread MyRobo
    put-object Tomato Lettuce MyRobo
    put-object Bread Tomato MyRobo
```

```
make-salad :=
    slice-object Lettuce
    slice-object Tomato
    put-object Lettuce Bowl MyRobo
    put-object Tomato Bowl MyRobo
```

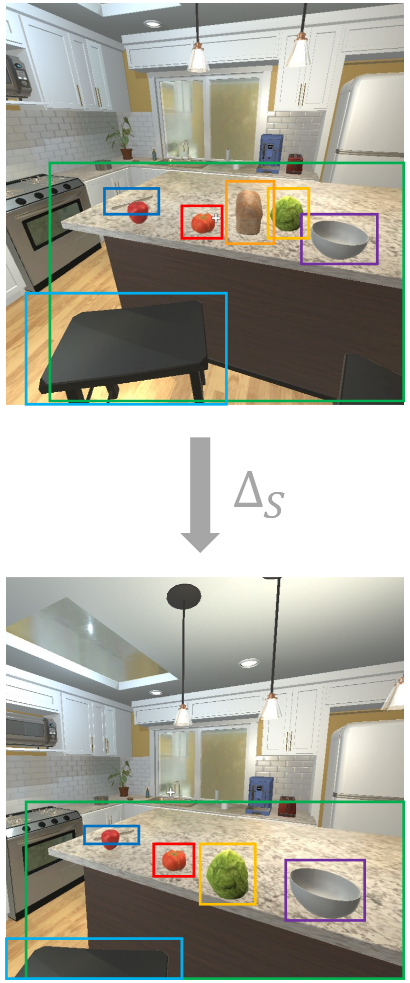
Motivating Example

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))))
```

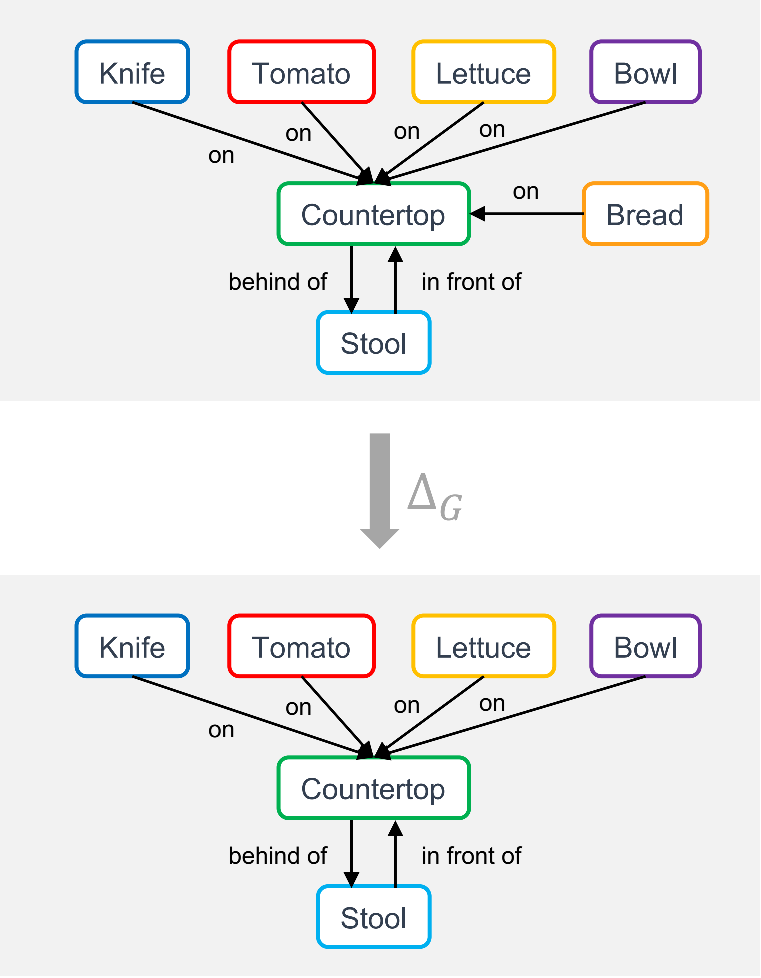
PDDL

Scenes



AI2THOR

Initial Scene Graph



Afforded Task Plans

```
make-veggie-sandwich :=
    slice-object Lettuce
    slice-object Tomato
    slice-object Bread
    put-object Lettuce Bread MyRobo
    put-object Tomato Lettuce MyRobo
    put-object Bread Tomato MyRobo
```

```
    put-object Lettuce Bread MyRobo
```

```
make-salad :=
    slice-object Lettuce
    slice-object Tomato
    put-object Lettuce Bowl MyRobo
    put-object Tomato Bowl MyRobo
```

Motivating Example

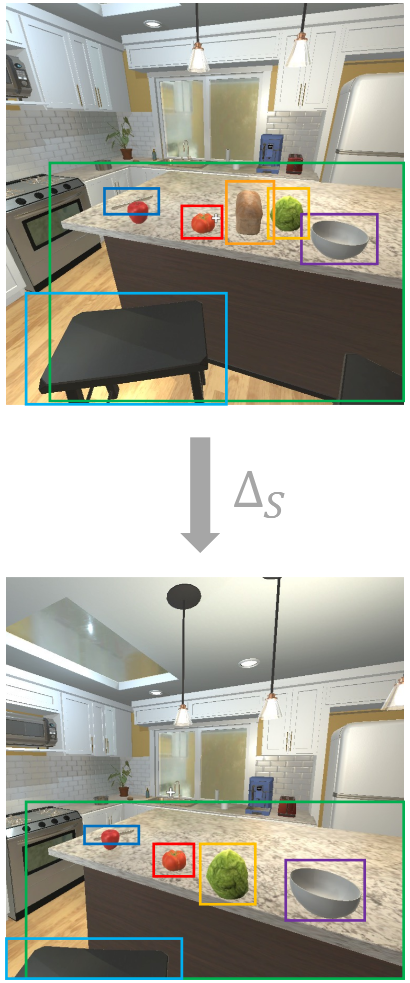
Q-A. Given a change in the environment, what changes in the afforded task plans?

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))))
```

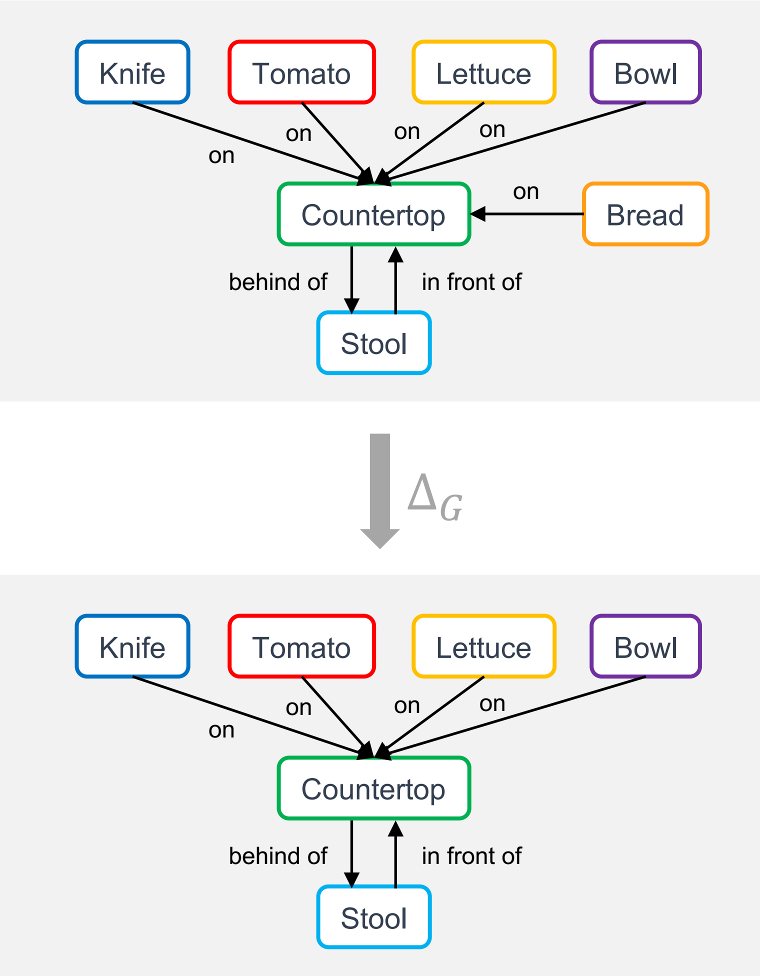
PDDL

Scenes



AI2THOR

Initial Scene Graph



Afforded Task Plans

```
make-veggie-sandwich :=
    slice-object Lettuce
    slice-object Tomato
    slice-object Bread
    put-object Lettuce Bread MyRobo
    put-object Tomato Lettuce MyRobo
    put-object Bread Tomato MyRobo
```

```
    put-object Lettuce Bread MyRobo
```

```
make-salad :=
    slice-object Lettuce
    slice-object Tomato
    put-object Lettuce Bowl MyRobo
    put-object Tomato Bowl MyRobo
```

Δ_P

```
make-salad :=
    slice-object Lettuce
    slice-object Tomato
    put-object Lettuce Bowl MyRobo
    put-object Tomato Bowl MyRobo
```

```
 put-object Tomato Bowl MyRobo
 slice-object Lettuce
 slice-object Tomato
 put-object Lettuce Bowl MyRobo
 put-object Tomato Bowl MyRobo
```

Motivating Example

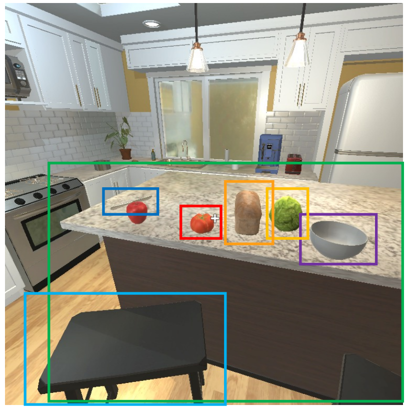
Q-B. Given a change in the afforded task plans, what changes are necessary in the environment?

Actions

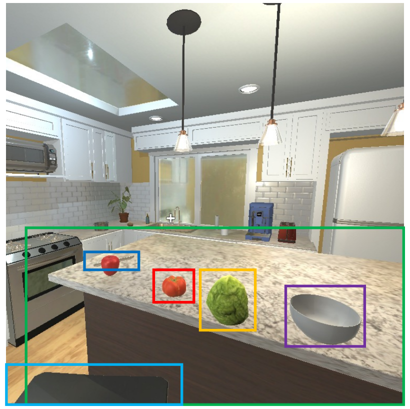
```
(:action open-object
    :parameters (?obj – Object)
    :precond (not (openness ?obj))
    :effect (openness ?obj))

(: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

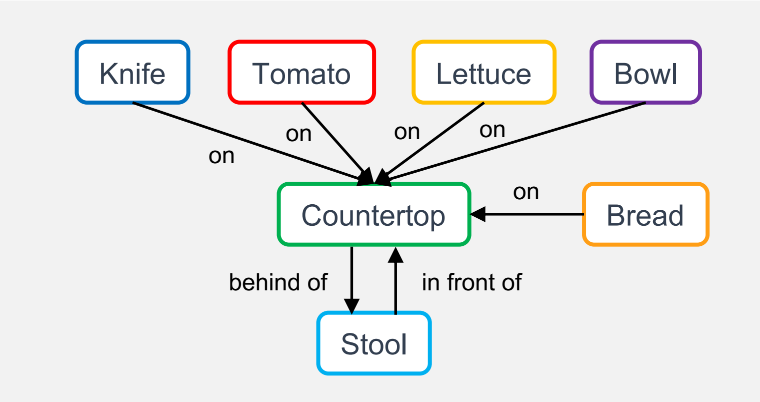


Δ_S

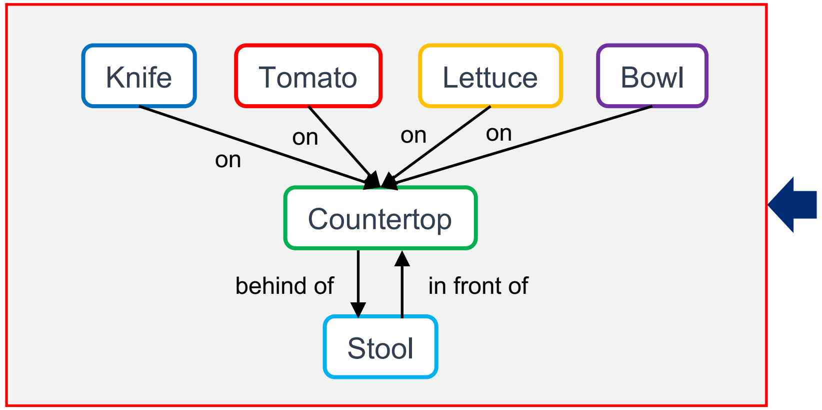


AI2THOR

Initial Scene Graph



Δ_G



Afforded Task Plans

```
make-veggie-sandwich :=
    slice-object Lettuce
    slice-object Tomato
    slice-object Bread
    put-object Lettuce Bread MyRobo
    put-object Tomato Lettuce MyRobo
    put-object Bread Tomato MyRobo
```

```
make-salad :=
    slice-object Lettuce
    slice-object Tomato
    put-object Lettuce Bowl MyRobo
    put-object Tomato Bowl MyRobo
```

Δ_P

```
make-salad :=
    slice-object Lettuce
    slice-object Tomato
    put-object Lettuce Bowl MyRobo
    put-object Tomato Bowl MyRobo
```

```
slice-object Lettuce
slice-object Tomato
put-object Lettuce Bowl MyRobo
put-object Tomato Bowl MyRobo
```

Objective

Describe a general framework for identifying

(i) **what knowledge is necessary** given desired capabilities,

(ii) **how an agent's capabilities change** when knowledge of the environment changes

(iii) **what capabilities an agent has** given knowledge of the environment, and

(iv) **how knowledge of the environment should change** when the desired capabilities change

in knowledge-based, context-aware autonomous agents.

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

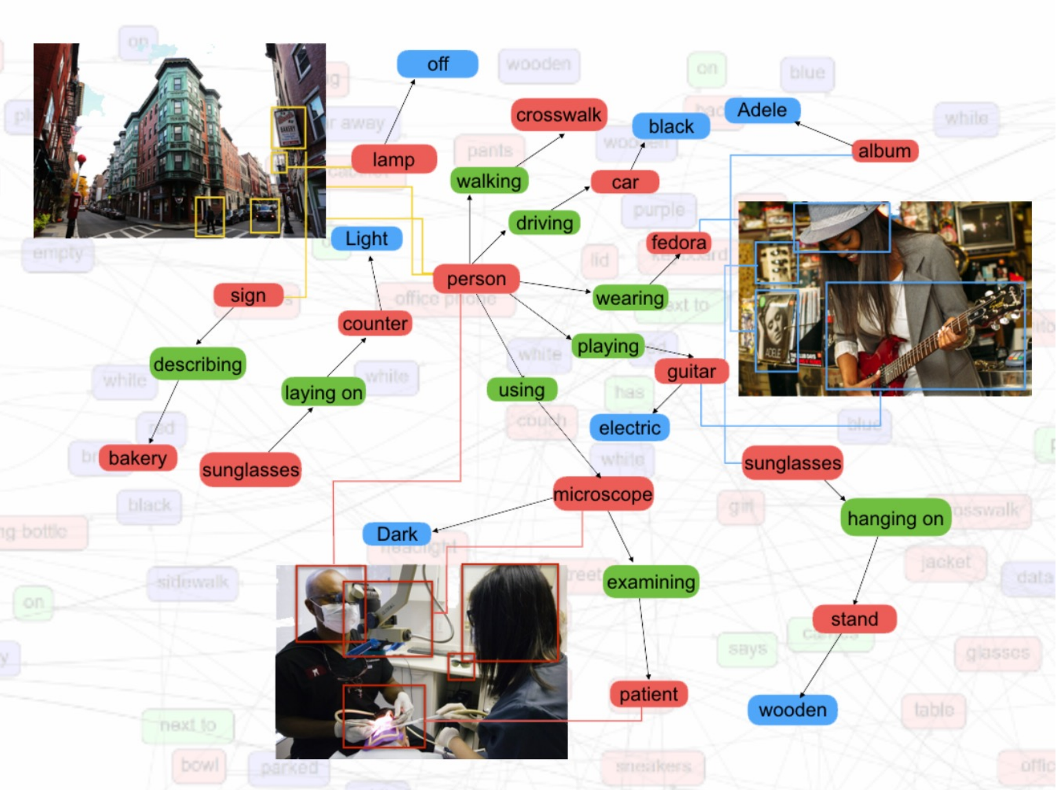
ii. A set of object-object relations

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

iii. A set of object-attribute relations

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

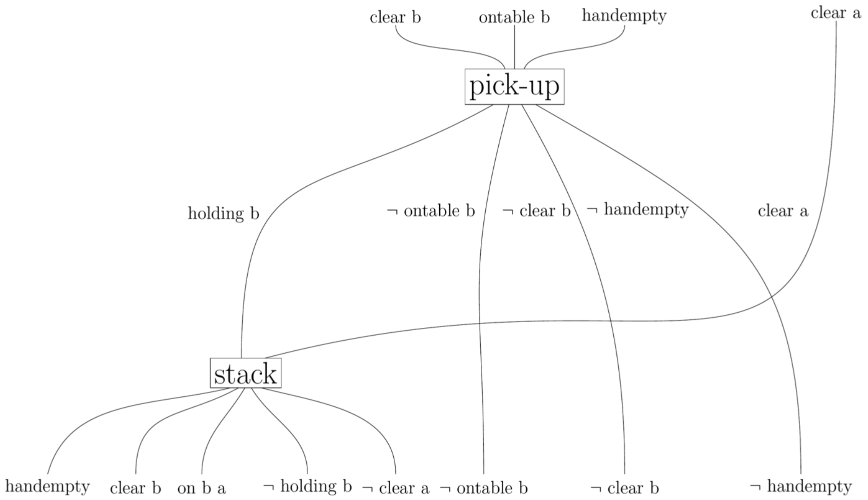
Categorically, a scene graph can be represented as a **co-presheave** ($\mathbb{C}$**-Set**) where the schema category, $\mathbb{C}$, is the ontology and the target sets are the specific instances of each class. The arrows are natural transformations.



https://visualgenome.org/

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.

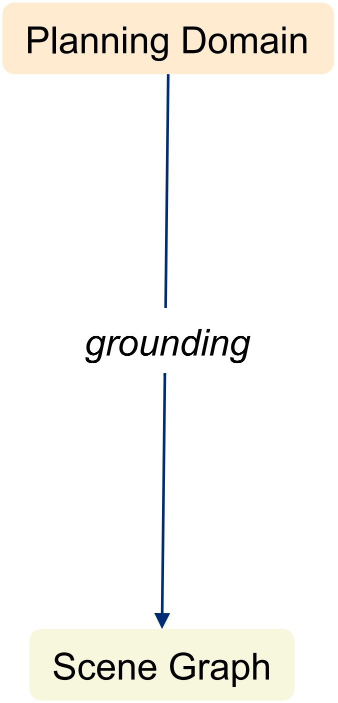
```
(:action pick-up-object
     :precond (and (not (has MyRobo Tomato)) (on Tomato
Counter))
     :effect (and (has MyRobo Tomato) (not (on Tomato
Counter))))
```

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))))
```

Affordance relation using functors

Functor *G*

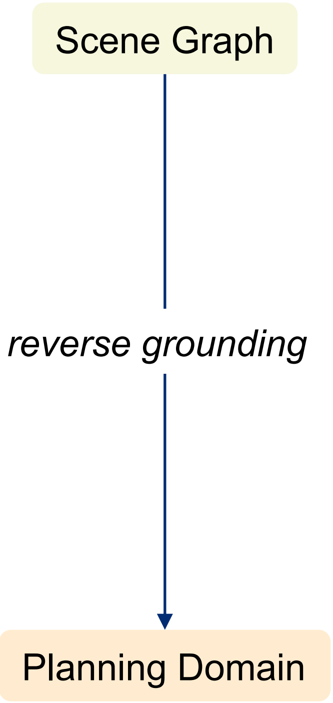


```
(:action pick-up-object
     :parameters (?target-obj - Object ?support-
obj - Object)
     :precond (on ?target-obj ?support-obj)
     :effect(not (on ?target-obj ?support-obj)))
```

↧

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

Functor *G'*



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

↧

```
(: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

Change propagation using symmetric delta lens

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

<u>Axioms</u>
Lifting operations, ϕ, ϕ', preserve compositions and identities

Functors, G, G', are arbitrary functors

Johnson 2016

$$\begin{tikzcd}
\mathbb{A} \arrow[d, "G"] & a \arrow[r, "\phi_{(w,a)}"] \arrow[d, dash, dashed] & \phi_{(w,a)}(a) \arrow[d, dash, dashed] \\
\mathbb{B} & G(a) \arrow[r, "w"] & b
\end{tikzcd}$$

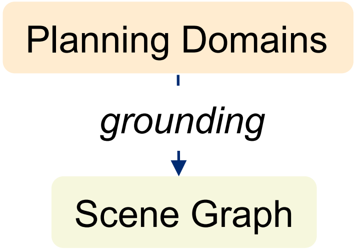
$$\begin{tikzcd}
\mathbb{B} \arrow[d, "G'"] & b' \arrow[r, "\phi'_{(u,b')}"] \arrow[d, dash, dashed] & \phi'_{(u,b')}(b') \arrow[d, dash, dashed] \\
\mathbb{A} & G'(b') \arrow[r, "u"] & a'
\end{tikzcd}$$

$\mathbb{A}$ ~ 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)

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

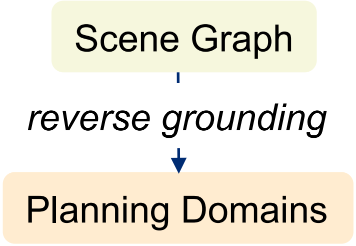
What kind of queries can we answer?


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

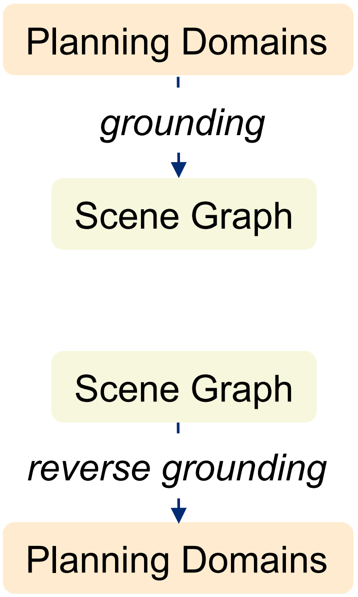
Planning Domains

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

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?”

What kind of queries can we answer?



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

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

Queries

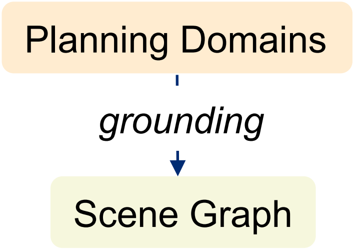
i. What is $\mathrm{G}(a)$? “What scene graph is afforded by this planning domain?”

ii. What is $\phi_{(w,a)}$? “Given a change in the scene graph, what changes in the afforded planning domains?”

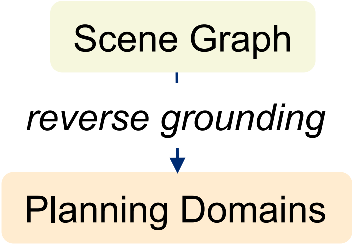
iii. What is $\mathrm{G}'(b')$? “What planning domain is afforded by this scene graph?”

iv. What is $\phi'_{(u,b')}$? “Given a change in the afforded planning domains, what changes are necessary in the scene graph?”

What kind of queries can we answer?



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

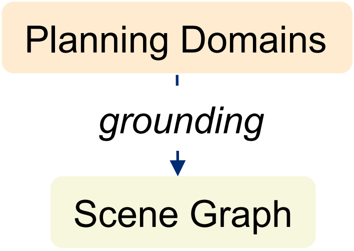


$$\begin{tikzcd}
\mathbb{B} \arrow[d, "G'"] & b' \arrow[r, "\phi'_{(u,b')}"] \arrow[d, dashed, no head] & \phi'_{(u,b')}(b') \arrow[d, dashed, no head] \\
\mathbb{A} & G'(b') \arrow[r, "u"] & a'
\end{tikzcd}$$

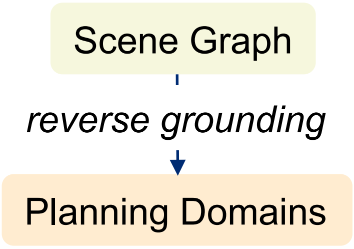
Queries

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?"

What kind of queries can we answer?



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

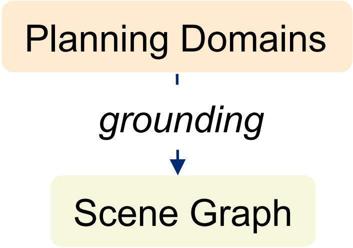


$$\begin{tikzcd}
\mathbb{B} \arrow[d, "G'"] & b' \arrow[r, "\phi'_{(u,b')}"] \arrow[d, dashed, no head] & \phi'_{(u,b')}(b') \arrow[d, dashed, no head] \\
\mathbb{A} & G'(b') \arrow[r, "u"] & a'
\end{tikzcd}$$

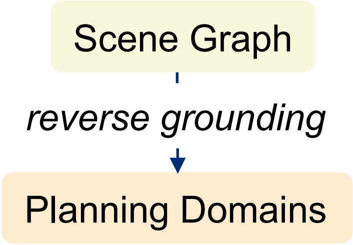
Queries

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?"

What kind of queries can we answer?



$$\begin{tikzcd}
\mathbb{A} \arrow[d, "G"] & a \arrow[r, "\phi_{(w,a)}"] \arrow[d, dash, dashed] & \phi_{(w,a)}(a) \arrow[d, dash, dashed] \\
\mathbb{B} & G(a) \arrow[r, "w"] & b
\end{tikzcd}$$



$$\begin{tikzcd}
\mathbb{B} \arrow[d, "G'"] & b' \arrow[r, "\phi'_{(u,b')}"] \arrow[d, dash, dashed] & \phi'_{(u,b')}(b') \arrow[d, dash, dashed] \\
\mathbb{A} & G'(b') \arrow[r, "u"] & a'
\end{tikzcd}$$

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?”

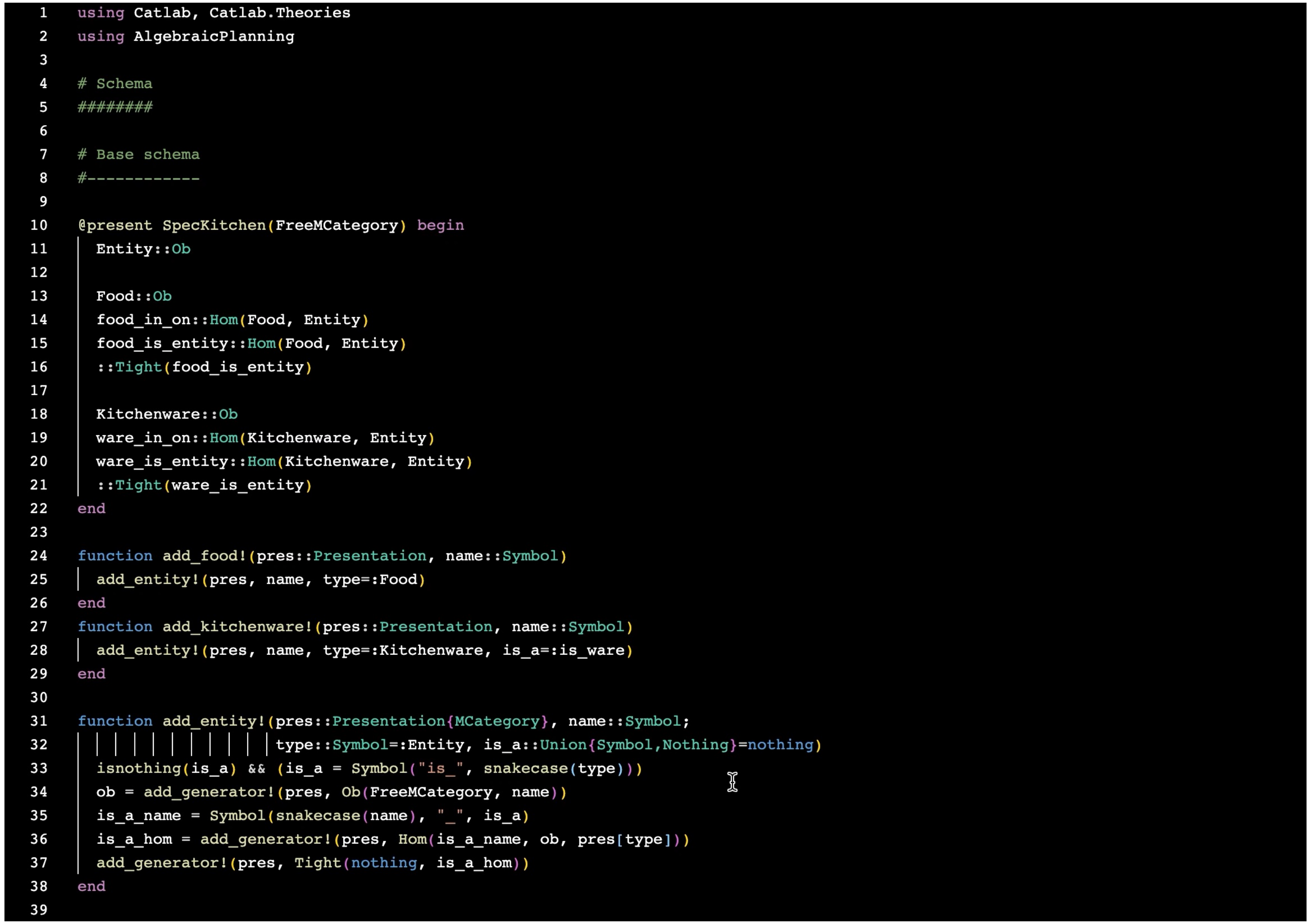
iii. What is $G'(b')$? “What planning domain is afforded by this scene graph?”

iv. What is $\phi'_{(u,b')}$? “Given a change in the afforded planning domains, what changes are necessary in the scene graph?”

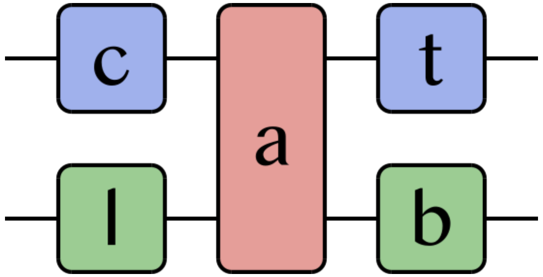
Ongoing Work

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/

Test and Evaluation Plan

Materials: VEQA dataset (Kim 2020)

- Uses AI2THOR simulator and scene graph generator to generate 3,916 candidate scene graphs as RDF
    - Contains ~13,000 objects, ~26,000 attributes, ~25,500 relations in total
- Contains 200 action scenarios (task plans) in PDDL syntax
    - Average plan length of 77

Plan for results:

I. Theoretical proof that queries (i) – (iv) are answerable by the framework.

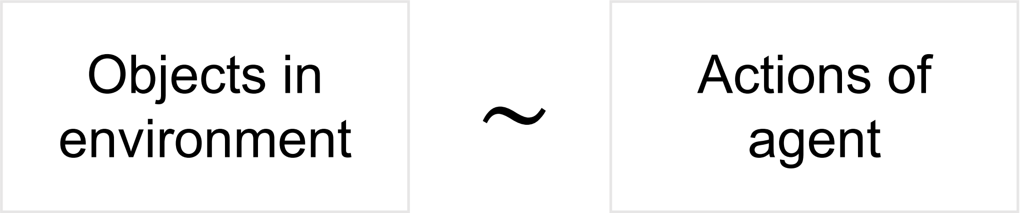
II. Evaluate performance of (a) grounding and (b) reverse grounding method against ground truth.

III. Evaluate accuracy of query responses of types (i) – (iv) against ground truth.

IV. Evaluate speed of query as scene graphs scale, by (a) number of objects, (b) number of relations.

Future work: Compositional Affordance

Affordance relations in robotics



Objects in environment ~ Actions of agent

e.g. set function map

(Barck-Holst 2009), (Cruz 2016), (Kruger 2011), (Montesano 2007)

Compositional affordances, hierarchical affordances, “behavior affords behavior”

- Task plans are a composition of action operators
- Objects in the environment are a composition of other objects
  - e.g. A sandwich is composition of bread, ham, and cheese
- Little work done to formalize an affordance relation that incorporates composition of objects and composition of actions *(Zech 2017)*

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