A Category Theoretic Approach to Planning in a Complex World

Angeline Aguinaldo

University of Maryland, College Park

Johns Hopkins University Applied Physics Laboratory

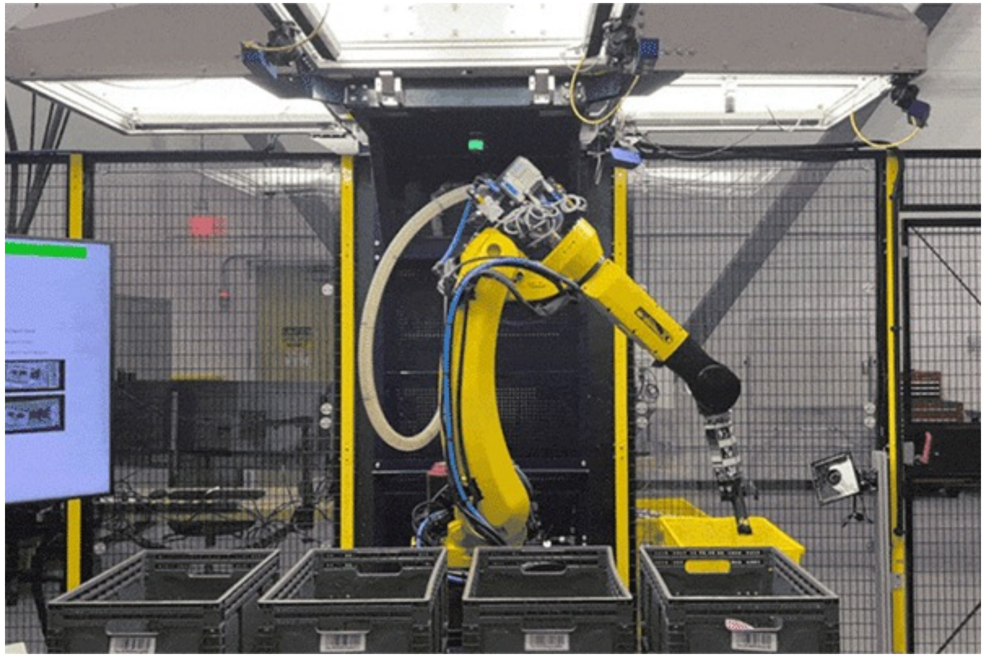
Microsoft Future Leaders in Robotics and AI Seminar Series

April 7, 2023

Planning in robotics

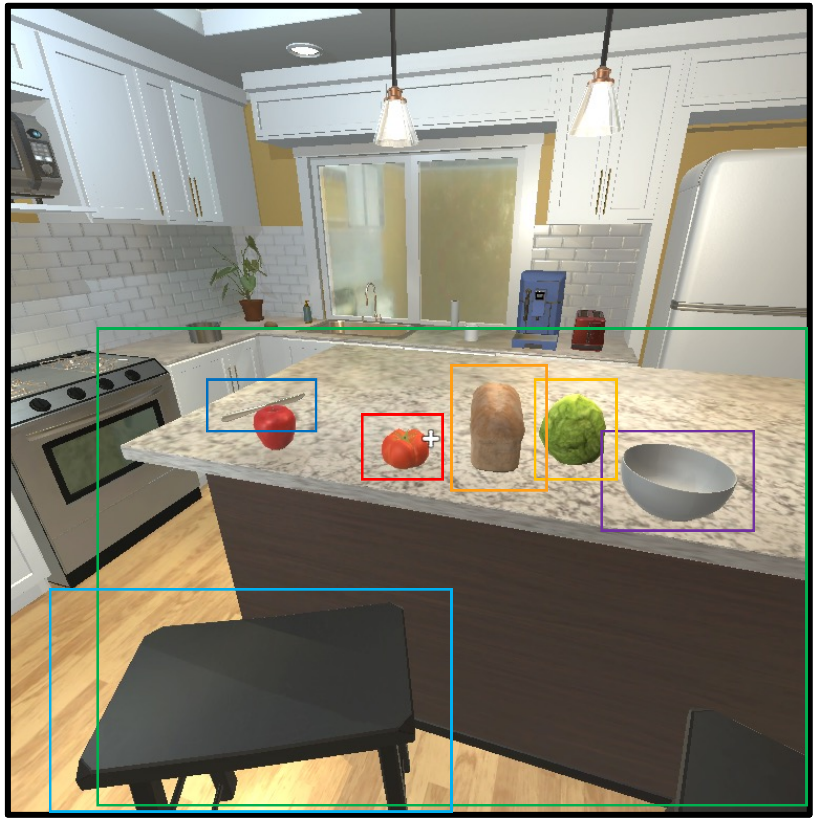
Task Planning

Motion Planning

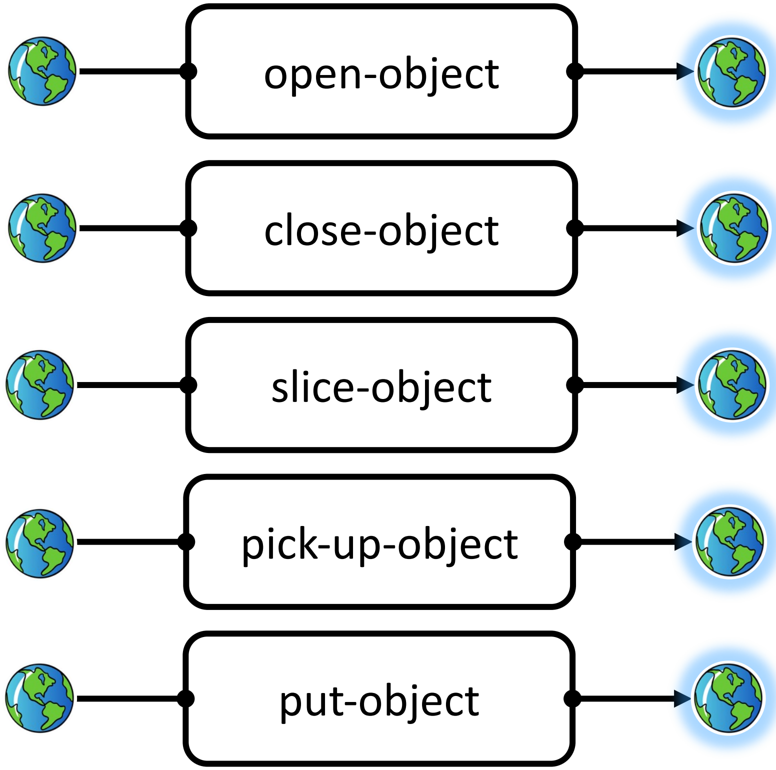


Aguinaldo A., Bunker J., Pollard B., Shukla A., Canedo A., Quiros G., Regli W. RoboCat: A Category Theoretic Framework for Robotic Interoperability Using Goal-Oriented Programming. IEEE Transactions on Automation Science and Engineering, doi: 10.1109/TASE.2021.3094055.
Video by Jacob Bunker

Task planning in robotics

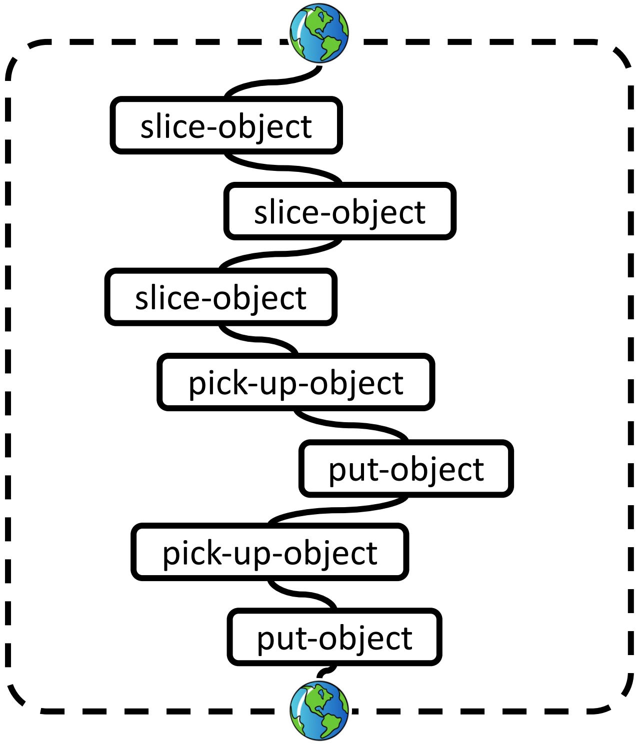


WORLD



ACTIONS

Initial State: There are sandwich ingredients on the countertop

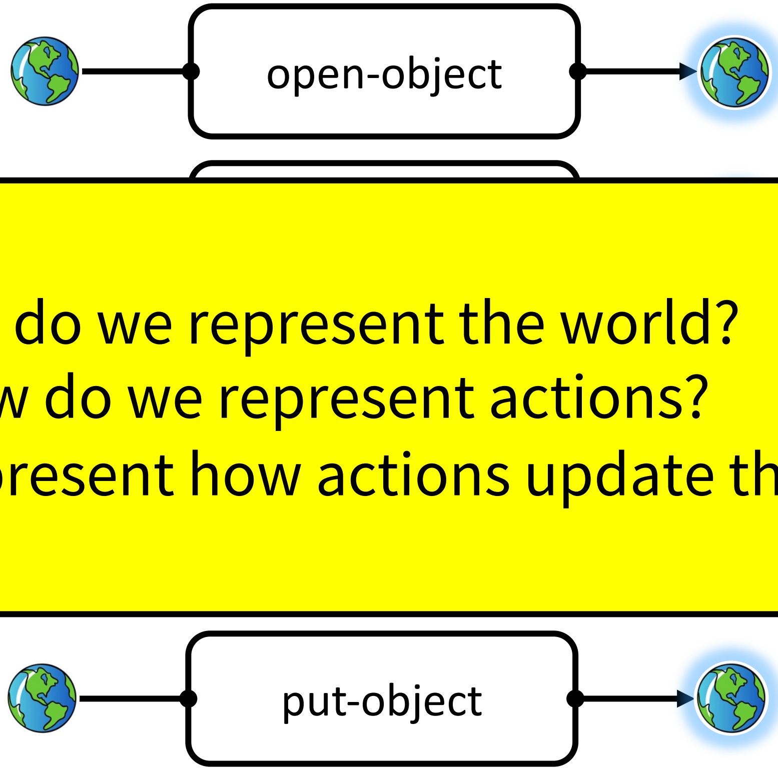


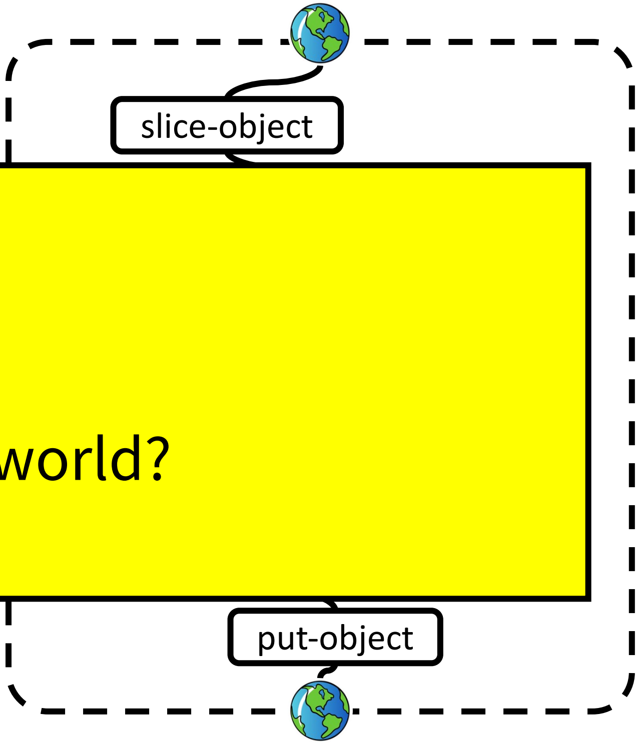
Goal State: There is a tomato and lettuce sandwich on the countertop

PLANNER

Task planning in robotics

Initial State: There are sandwich ingredients on the countertop





How do we represent the world?
How do we represent actions?
How do we represent how actions update the world?

Goal State: There is a tomato and lettuce sandwich on the countertop

WORLD

ACTIONS

PLANNER

Classical approaches to planning

Handling complex world states

Use classical representation with syntax is based on first-order logic Ghallab2004

```
on(apple, countertop)
∧ not(has(robot, apple))
∧ on(knife, countertop)
∧ not(has(robot, knife))
∧ on(tomato, countertop)
∧ not_sliced(tomato))
∧ not(has(robot, tomato))
```

Transferring actions between domains

Specify actions using Planning Domain Description Language (PDDL) McDermott 1998

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

Managing implicit effects

Translate world states as sets and apply set operations (add and subtract) Ghallab2004

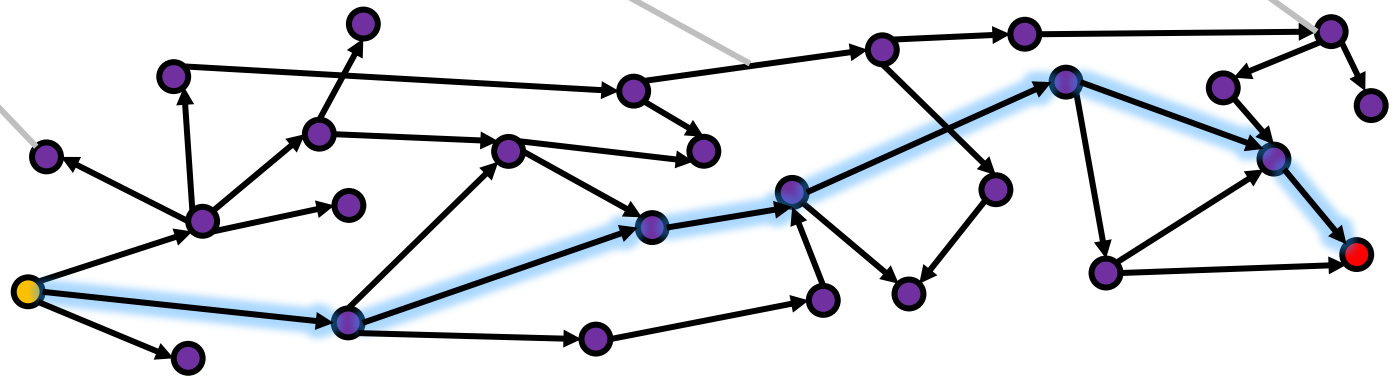
s

↓

s'

```
(:action slice-object
    :parameters (bread – Object)
    :precond (not_sliced bread))
    :effect (and (sliced bread)
(not (not_sliced bread)))
```

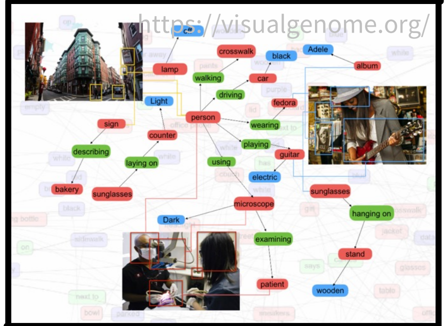
PLANNER



Modern approaches to planning

Handling complex world states

Use scene graphs to model world state Galindo 2008, Agia2021




```
on(apple, countertop)
∧ not(has(robot, apple))
∧ on(knife, countertop)
∧ not(has(robot, knife))
```

Transferring actions between domains

Specify abstract methods using hierarchical task nets (HTNs) Nau 2005

method travel-by-foot
precond: $distance(x,y) \leq 2$
task: travel(a, x, y)
subtasks: walk(a, x, y)

method travel-by-taxi
task: travel(a, x, y)
precond: $cash(a) \geq 1.5 + 0.5 \times distance(x, y)$
subtasks: call-taxi(a, x) $\longrightarrow$ ride(a, x, y) $\longrightarrow$ pay-driver(a, x, y)

operator walk (a, x, y)
precond: $location(a) = x$
effects: $location(a) \leftarrow y$

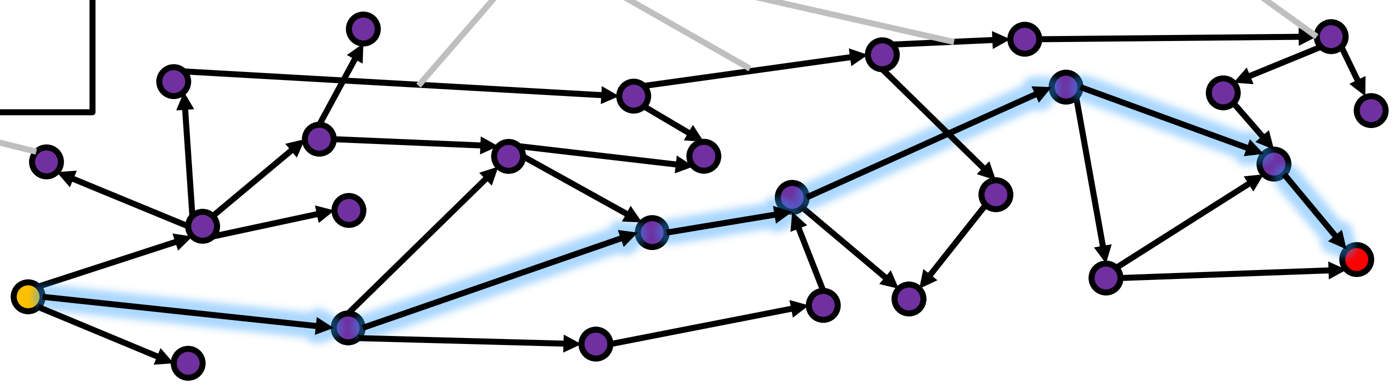
operator call-taxi(a, x)
effects: $location(taxi) \leftarrow x$

Managing implicit effects

Use frame axioms Thiébaux 2005

```
(define (domain bw-axioms)
  (:requirements :strips)
  (:predicates (on-table ?x) (on ?x ?y) ; basic predicates
               (holding ?x) (above ?x ?y) (clear ?x) (handempty)) ;
               derived predicates

(:derived (holding ?x)
      (and (not (on-table ?x))(not (exists (?y) (on ?x ?y)))))

(:derived (above ?x ?y)
    (or (on ?x ?y)
        (exists (?z) (and (on ?x ?z) (above ?z ?y)))))

(:derived (clear ?x)
    (and (not (holding ?x))
         (not (exists (?y) (on ?y ?x)))))

(:derived (handempty) (forall (?x) (not (holding ?x))))

(:action pickup
  :parameters (?ob)
  :precondition (and (clear ?ob) (on-table ?ob) (handempty))
  :effect (not (on-table ?ob)))
```

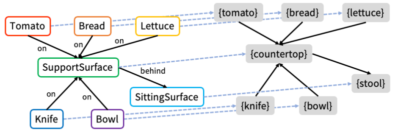
PLANNER



My research contributions

Handling complex world states

Use C-set representation of scene graphs to represent world state



$$X : C \to \mathrm{Set}$$

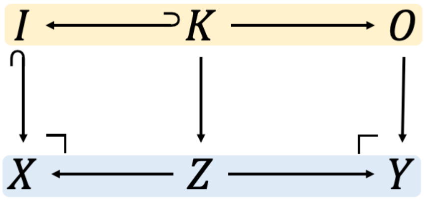
Transferring actions between domains

Translate actions between domains according to isomorphic relations

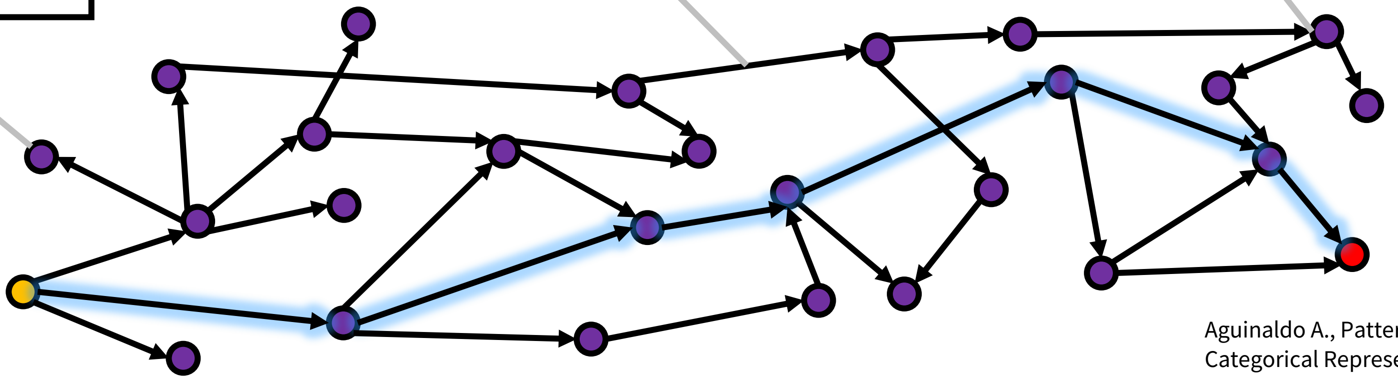
$$\cong$$

Managing implicit effects

View world state updates as generalized graph rewrites that preserve composition



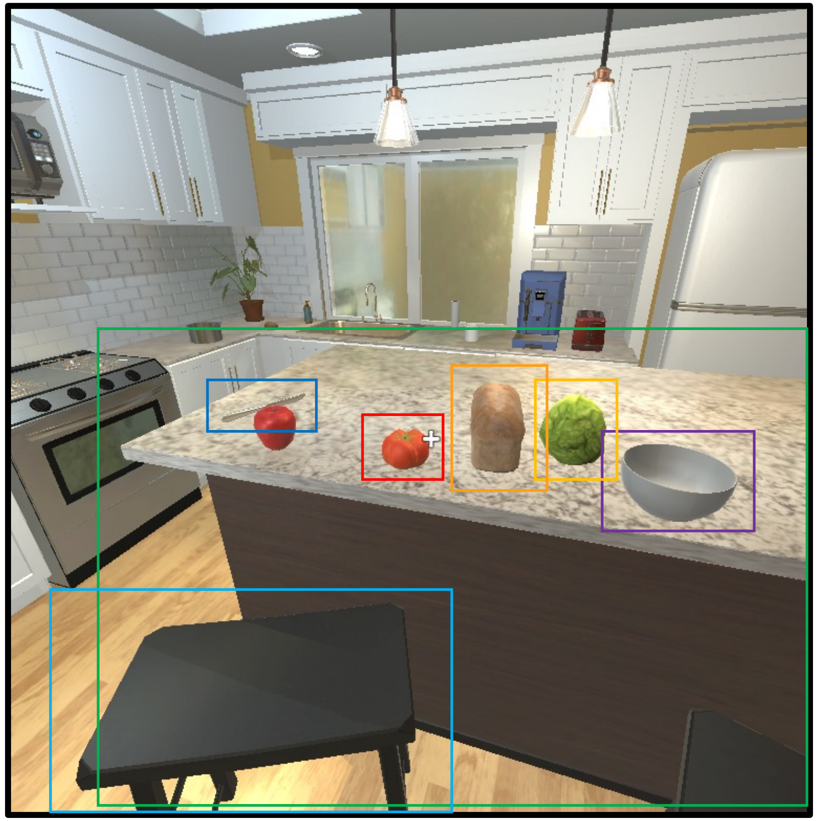
PLANNER

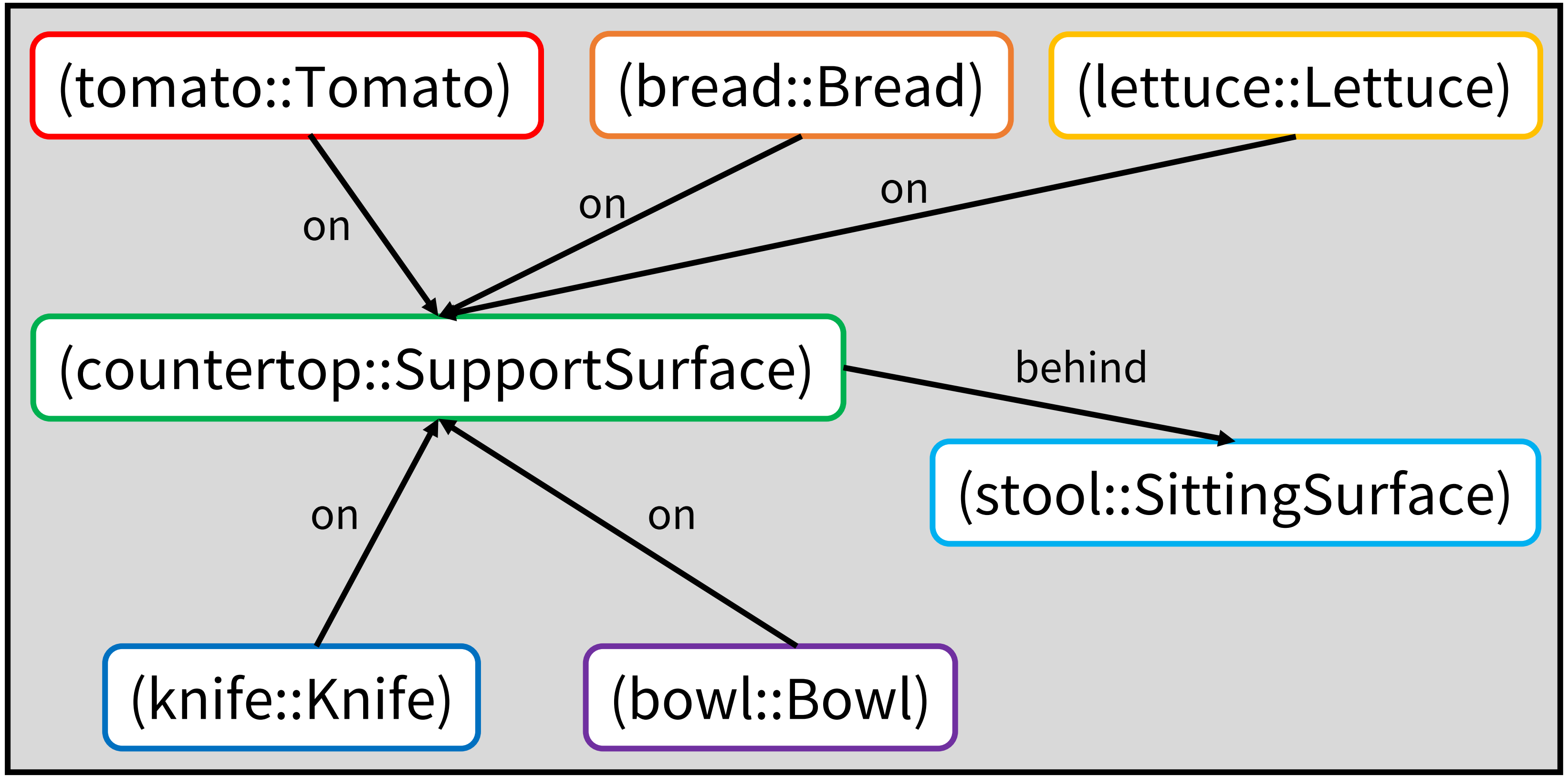


Aguinaldo A., Patterson E., Fairbanks J., Ruiz J. (2023). A Categorical Representation Language and Computational System for Knowledge-Based Planning. In review.

Approach

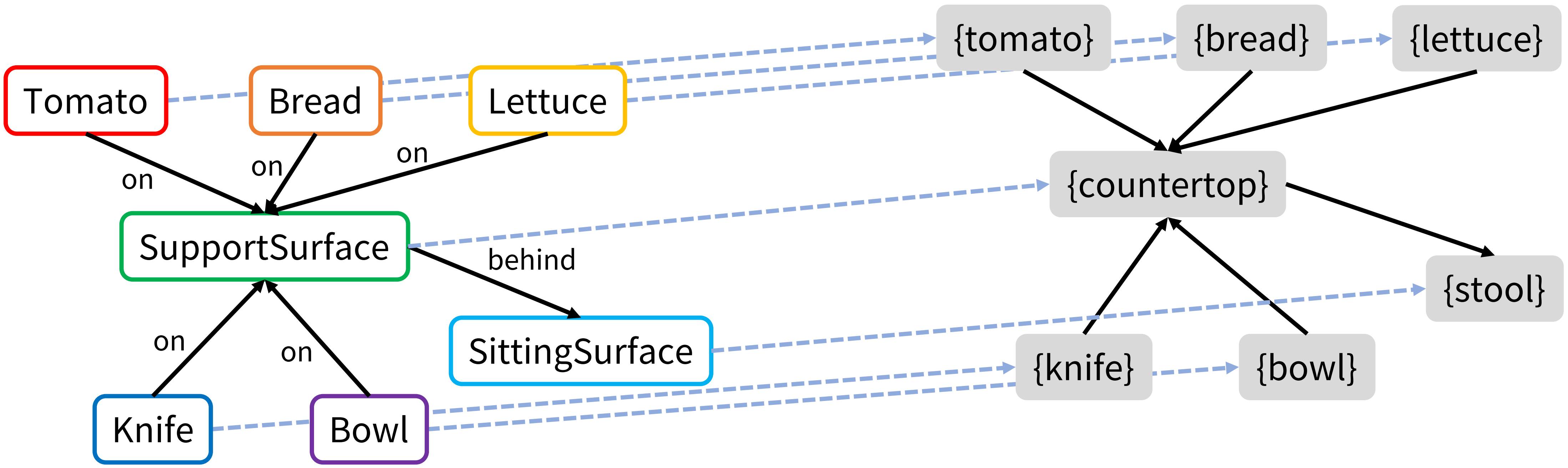
Our scene graph





Scene graph as a typed graph

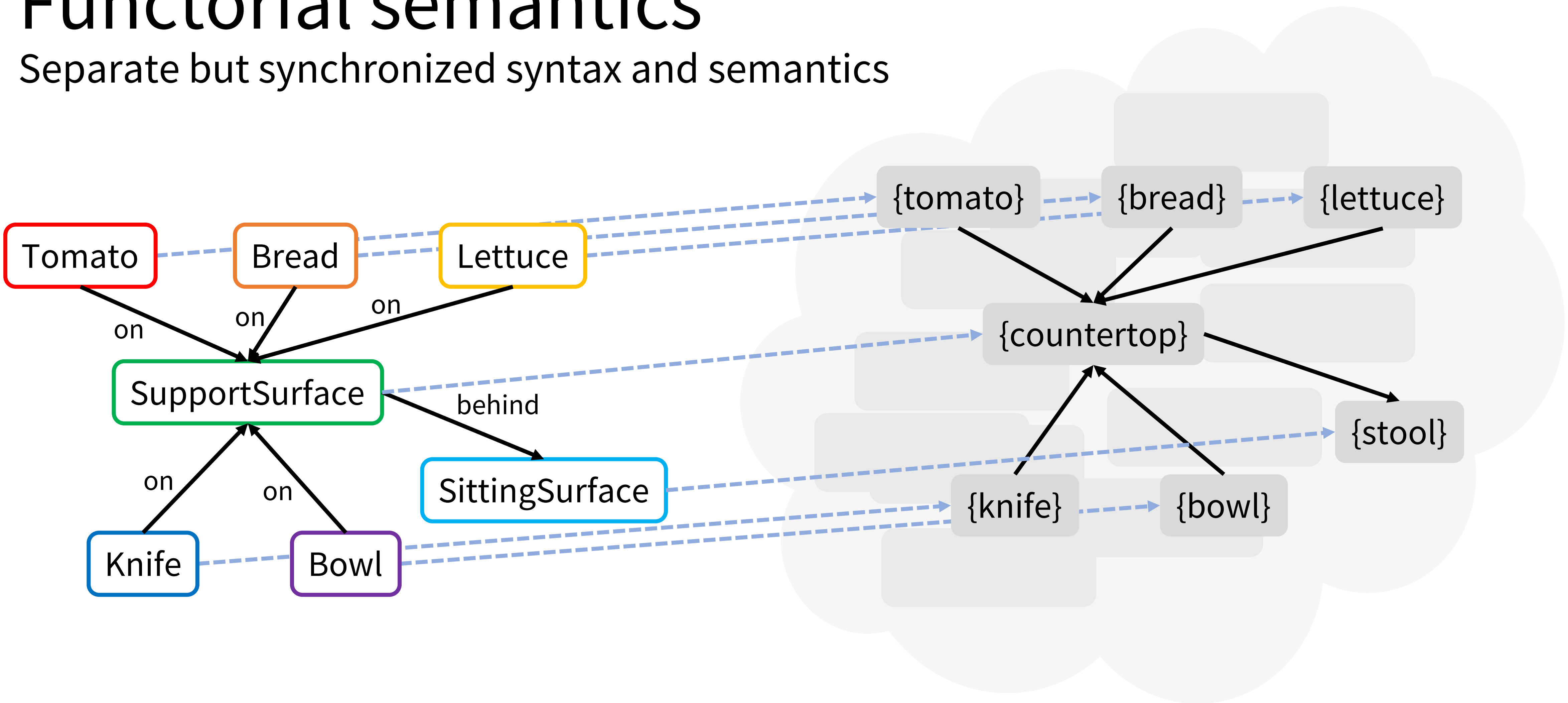
Our scene graph



Every typed graph is a C-set Brown2021

Functorial semantics

Separate but synchronized syntax and semantics

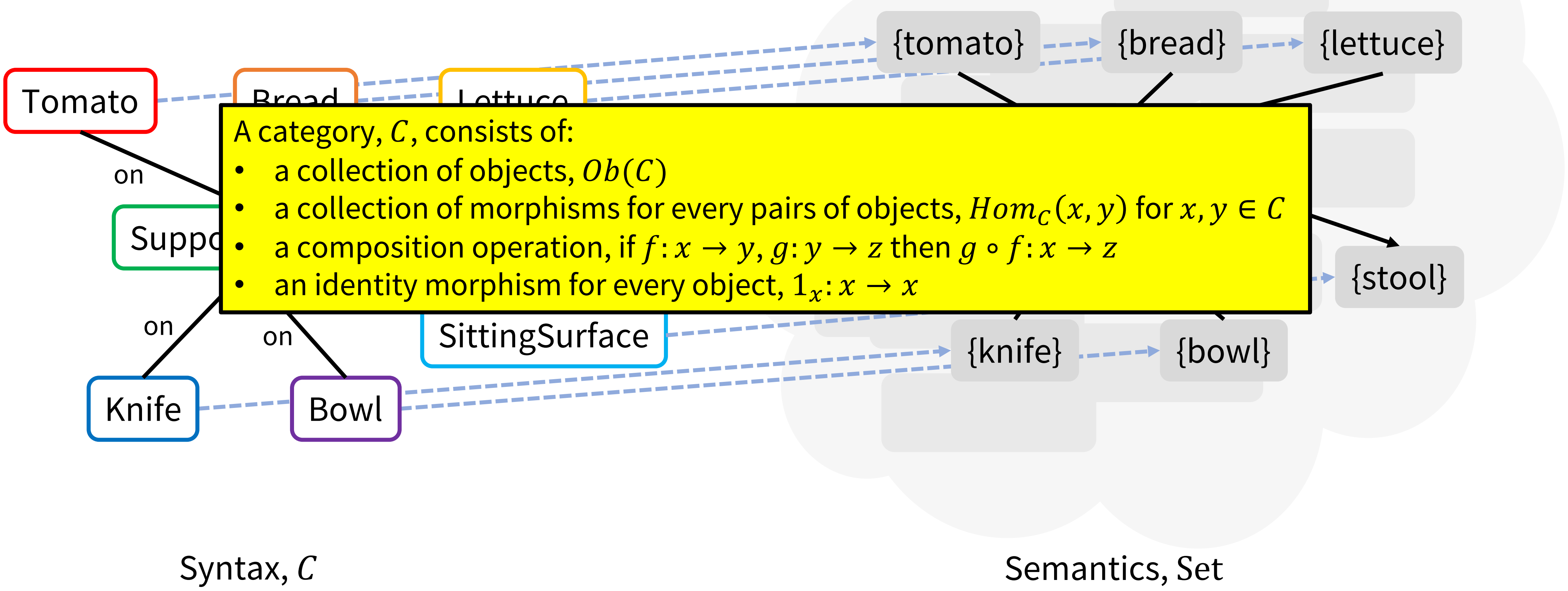


Syntax, C

Semantics, Set

Functorial semantics

Separate but synchronized syntax and semantics

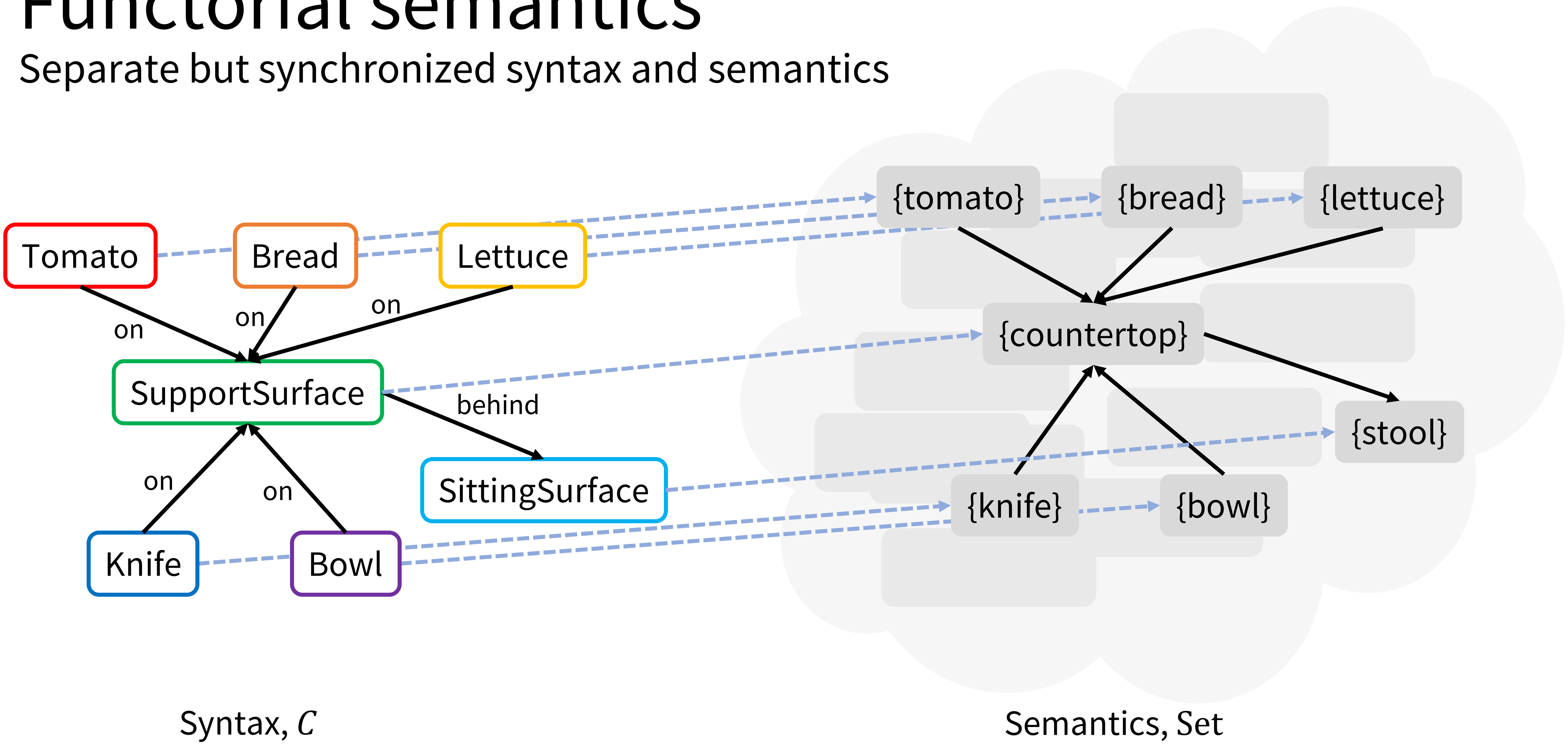


A category, C, consists of:

- a collection of objects, $Ob(C)$
- a collection of morphisms for every pairs of objects, $Hom_C(x, y)$ for $x, y \in C$
- a composition operation, if $f: x \to y, g: y \to z$ then $g \circ f: x \to z$
- an identity morphism for every object, $1_x: x \to x$

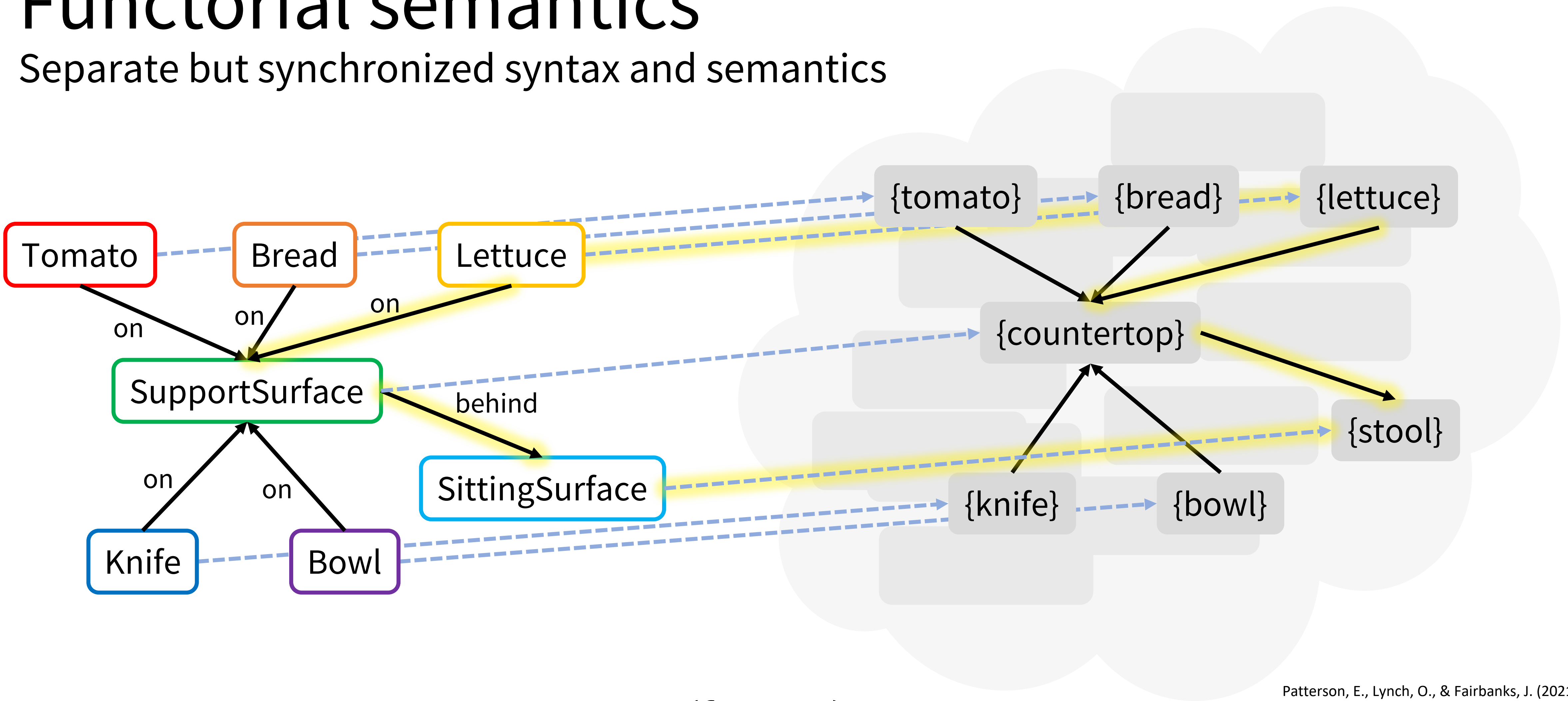
Functorial semantics

Separate but synchronized syntax and semantics



Functorial semantics

Separate but synchronized syntax and semantics



Syntax, C

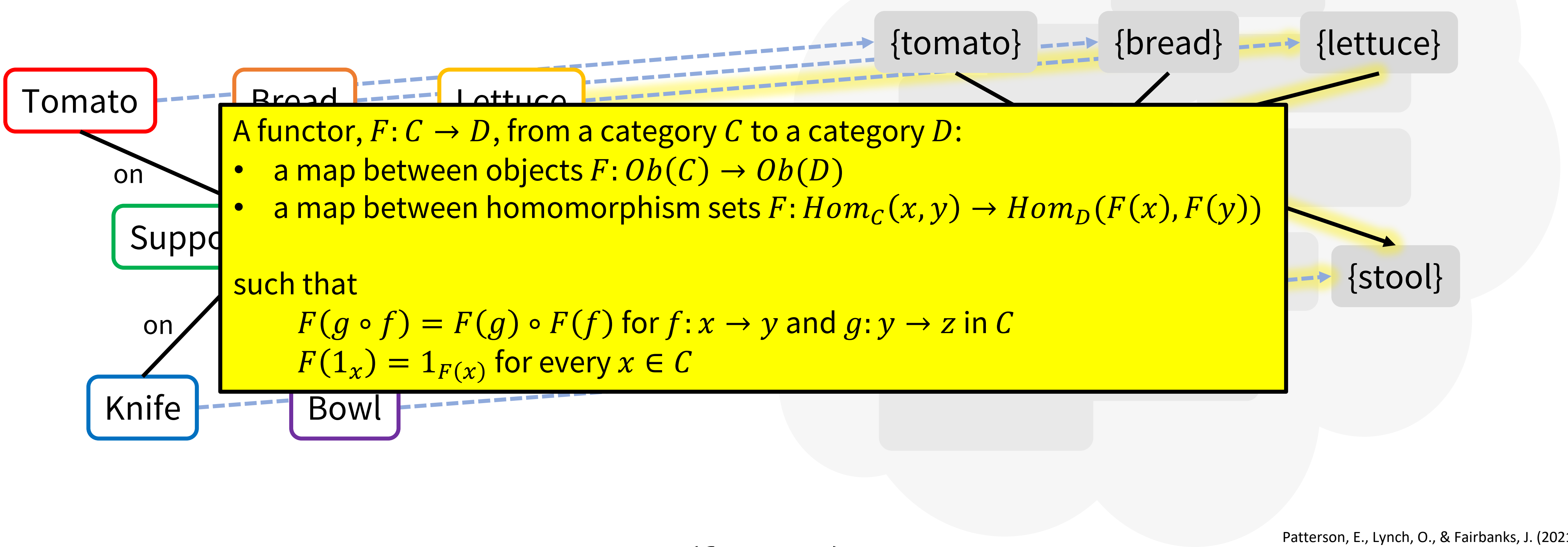
$X: C \to \mathrm{Set}$ (functor)

Semantics, Set

Patterson, E., Lynch, O., & Fairbanks, J. (2021). Categorical Data Structures for Technical Computing. *Compositionality*, *4*(5), 1–27.

Functorial semantics

Separate but synchronized syntax and semantics



A functor, $F: C \to D$, from a category C to a category D:

- a map between objects $F: Ob(C) \to Ob(D)$
- a map between homomorphism sets $F: Hom_C(x, y) \to Hom_D(F(x), F(y))$

such that

$F(g \circ f) = F(g) \circ F(f)$ for $f: x \to y$ and $g: y \to z$ in C

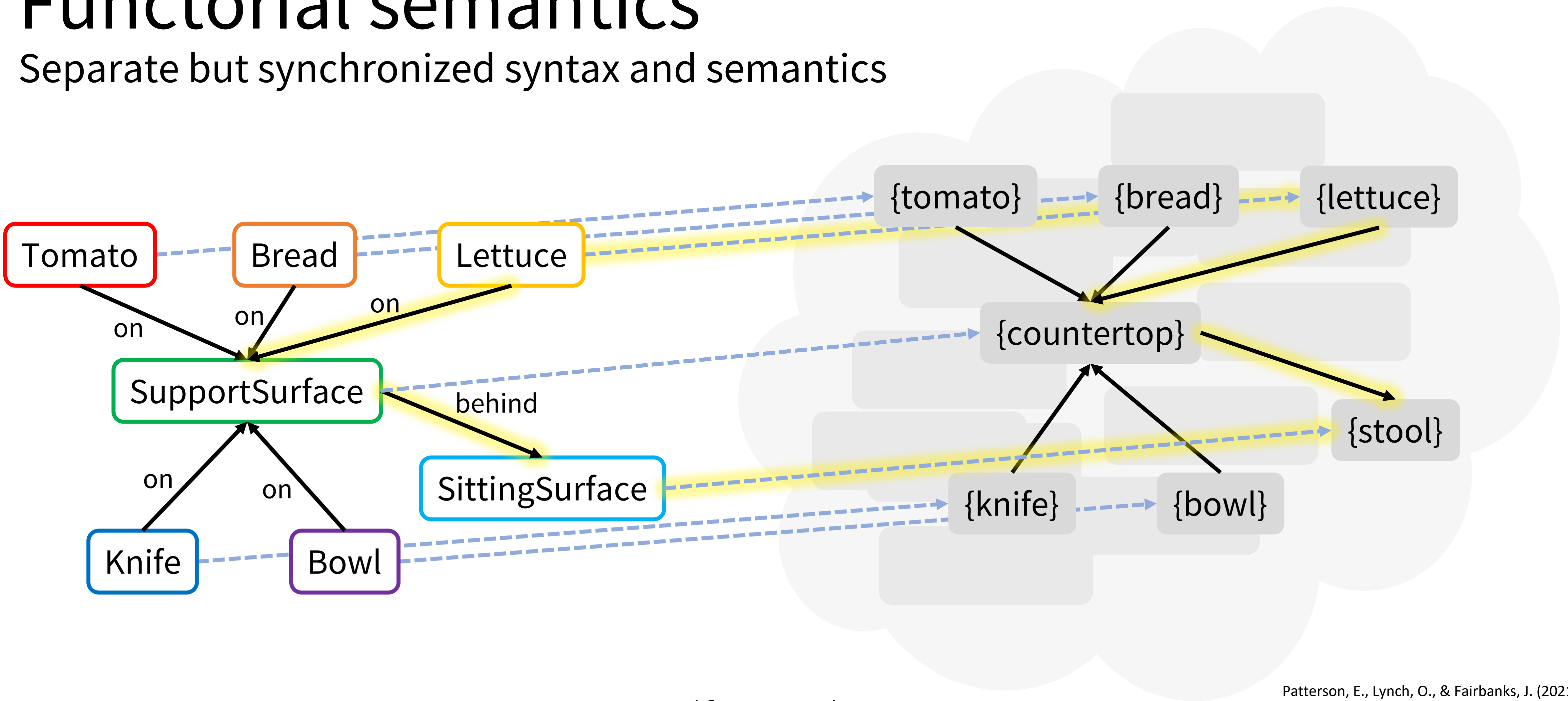
$F(1_x) = 1_{F(x)}$ for every $x \in C$

Syntax, C $\quad$ $X: C \to \text{Set (functor)}$ $\quad$ Semantics, Set

Patterson, E., Lynch, O., & Fairbanks, J. (2021). Categorical Data Structures for Technical Computing. *Compositionality*, *4*(5), 1–27.

Functorial semantics

Separate but synchronized syntax and semantics



Syntax, C

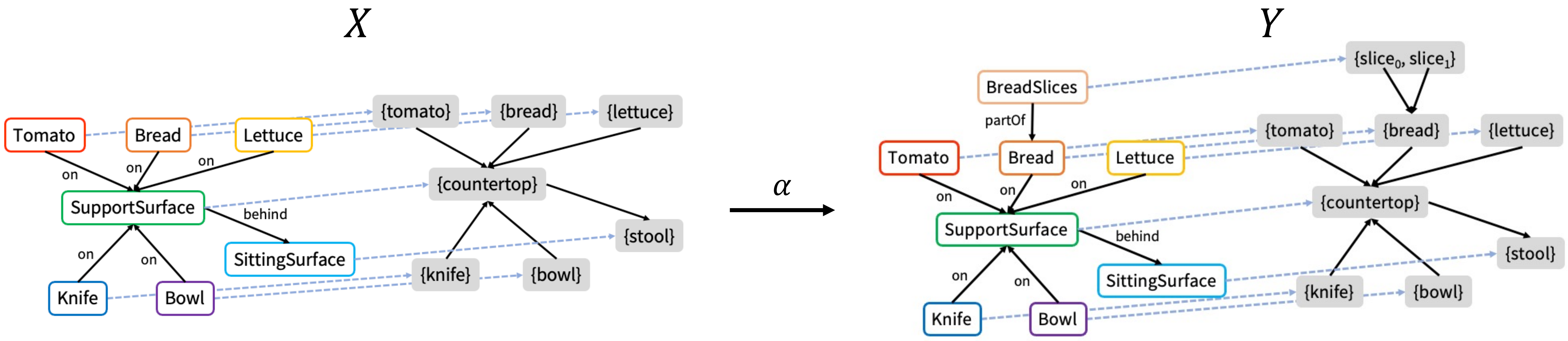
$X: C \to \text{Set}$ (functor)

Semantics, Set

Patterson, E., Lynch, O., & Fairbanks, J. (2021). Categorical Data Structures for Technical Computing. *Compositionality*, *4*(5), 1–27.

C-Sets

A category of C-set functors



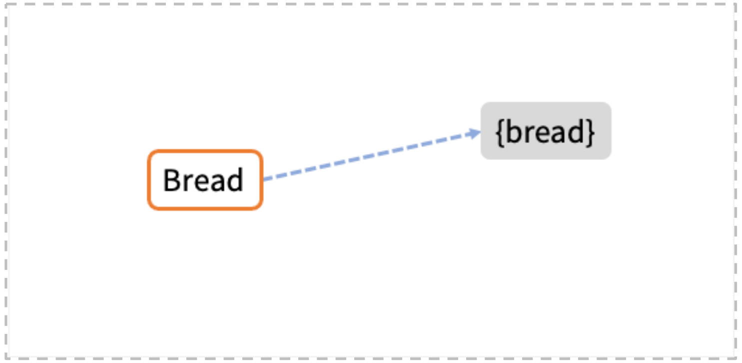
Finding an assignment can be formulated as a **typed CSP** (only consider assignment that satisfies type relations).
The typed CSP search space grows by $O(n^k)$ where n is the size of the target (Y) and k is the size of the source (X).
For reference, a generic graph homomorphism matching problem is NP-complete.

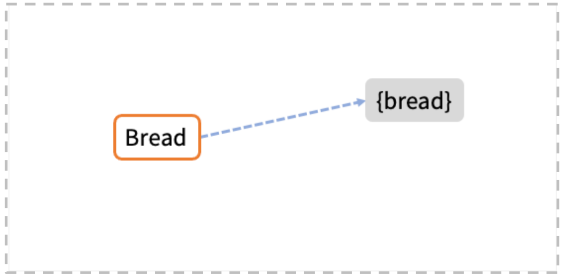
A transformation, α, between X and Y is a typed CSP solution if it is natural.

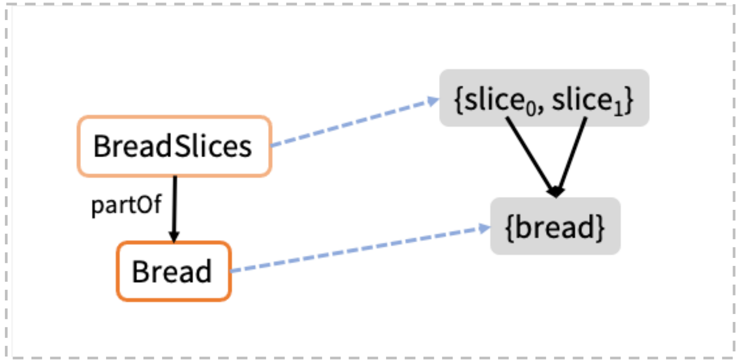
CSP: constraint satisfaction problem

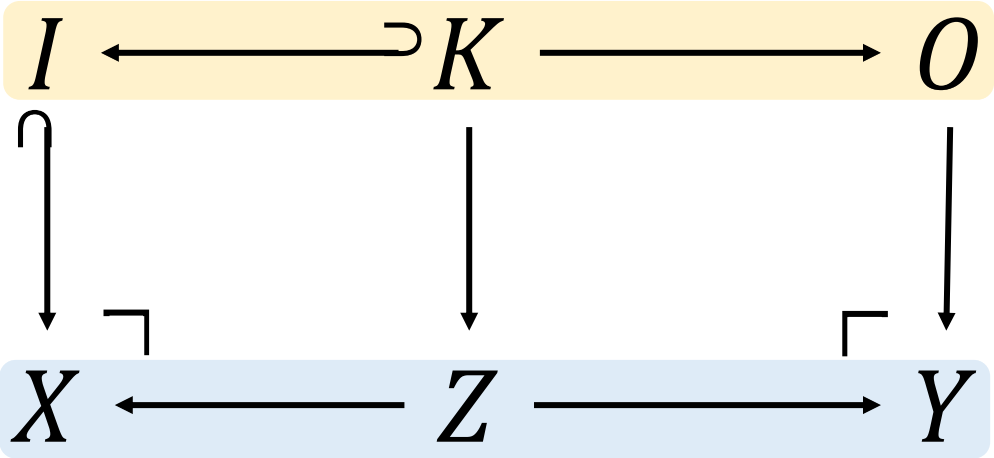
Brown, K., Patterson, E., & Hanks, T. (2022). Computational Category-Theoretic Rewriting (Vol. 1). Springer International Publishing. https://doi.org/10.1007/978-3-031-09843-7

Double-pushout (DPO) rewriting







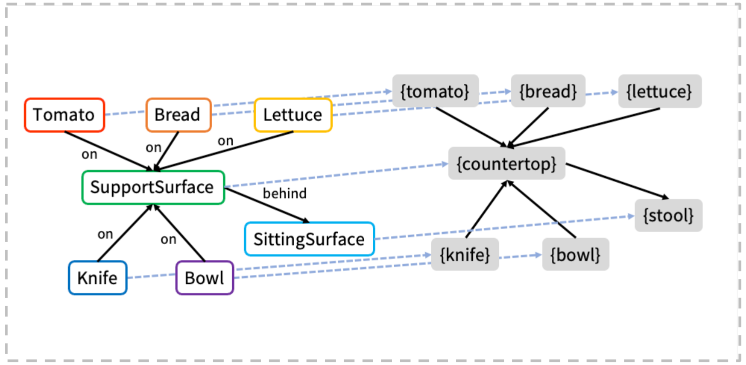


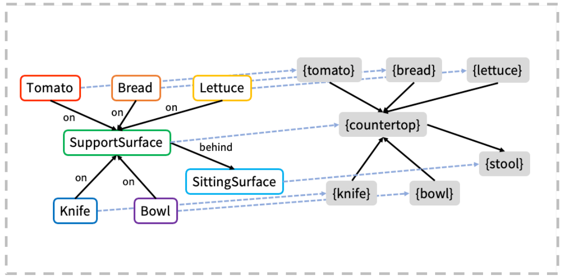
Applicability is determined by a monomorphism, $I \rightarrow X$

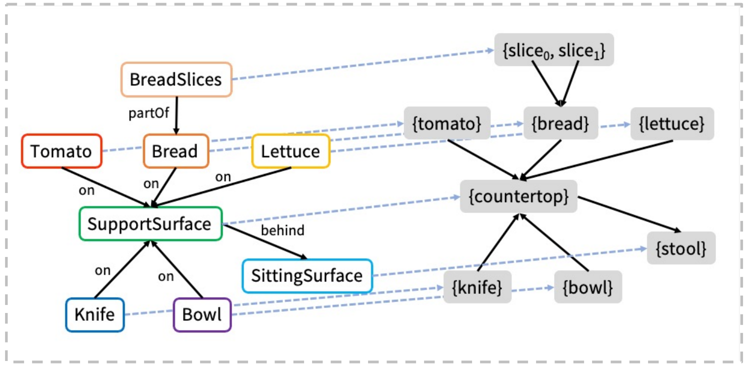
e.g. A monomorphism in Set is an injective function

DPO rewrite rule (action)

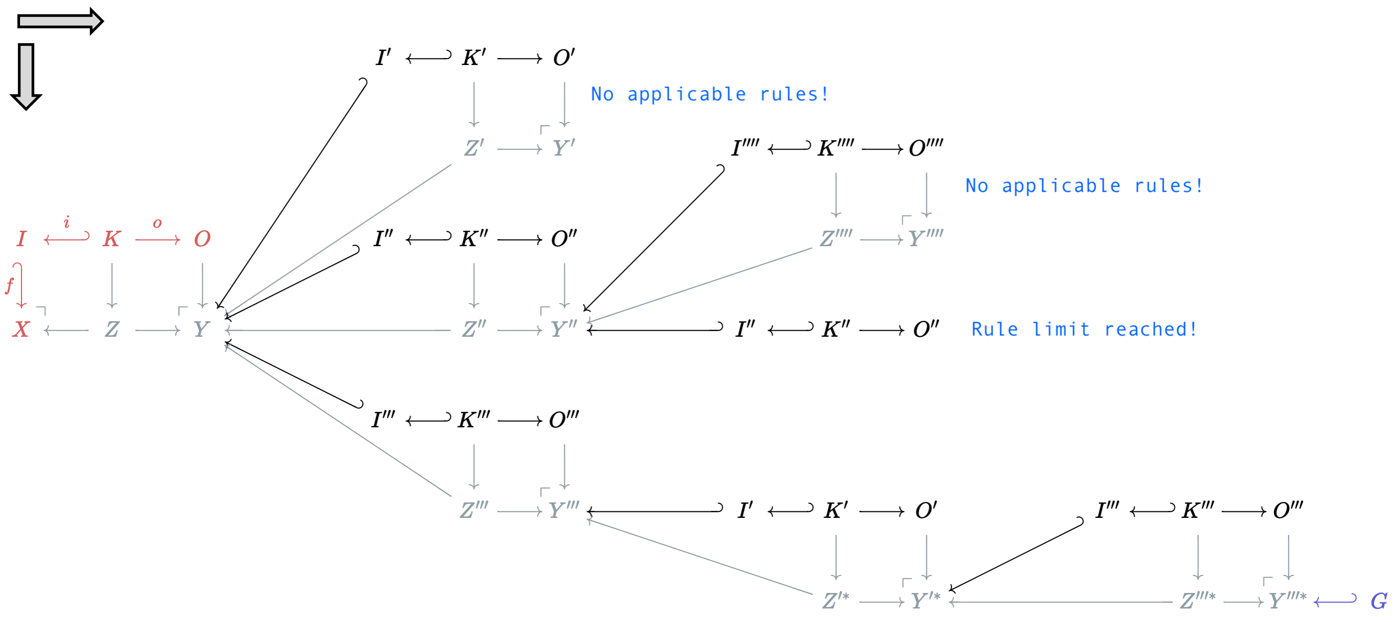
World state







Forward planning algorithm with DPO



Algorithm: Forward Planning with Backtracking

Procedure: ForwardPlan(Y world, G goal, `r` rules, `r_usage` rule usage, `r_limits` rule limits, `p` plan)

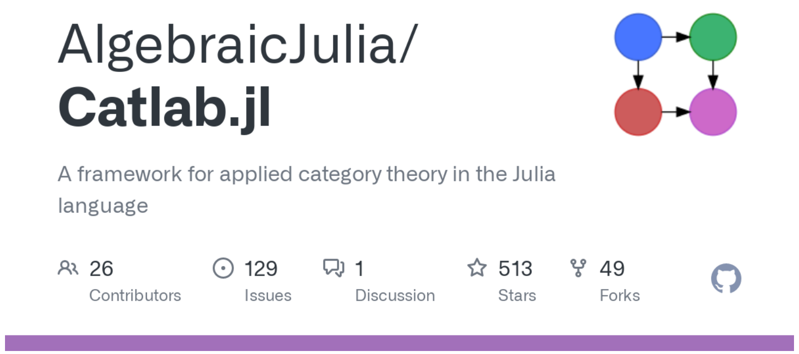
1. (*Exit criteria*) **If** monomorphism $G \hookrightarrow Y$ exists
   - 1a. **Return** Plan `p`
2. Initialize applicable rules list, `applicable`
3. **For** `rule` in `r` **do**
   - 3a. Get the input object of `rule`, r_I
   - 3b. Check if monomorphism $r_I \hookrightarrow Y$ exists
   - 3c. **If** exists, append `rule` to `applicable`
4. (*Backtrack criteria*) **If** `applicable` is empty, "No applicable rules!" **ThrowException**
5. **For** `a` in `applicable` **do**
   - 5a. (*Backtrack criteria*) **If** `r_usage[a]` >= `r_limits[a]`, "Rule limit reached!" **continue**
   - 5b. Y = DPO(Y, representable(`a`))
   - 5c. Append `a` to `p`
   - 5d. ForwardPlan(Y, G, `r`, `r_usage`, `r_limits`, `p`)

A. Aguinaldo. *"Using categorical logic for AI planning"*. AlgebraicJulia Blog [https://blog.algebraicjulia.org/post/2022/09/ai-planning-cset/]. 2022

Future Work

EVALUATION

- Implement a planning package within the **AlgebraicJulia** ecosystem
  - Leverage the C-set structure and DPO rewriting procedure developed in Catlab.jl



- Implement existing planning algorithms and compare plan qualities

DIRECTIONS

I. Analogies in planning
Abstracts all domains to a topological setting which allows for transfer of actions between domains that are isomorphic to rewrite rules

II. Online planning
Abstracts world state updates to a common language that can be expressed by an AI planner, a human, or a machine

III. Scene affordance relations
All applicable actions can be thought of as actions afforded by the scene

Thank you for listening!

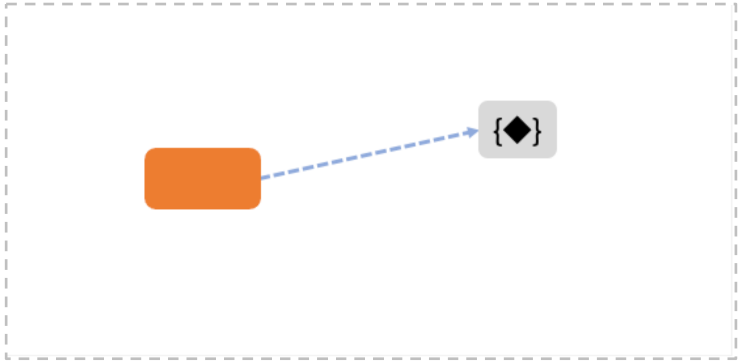
Angeline Aguinaldo

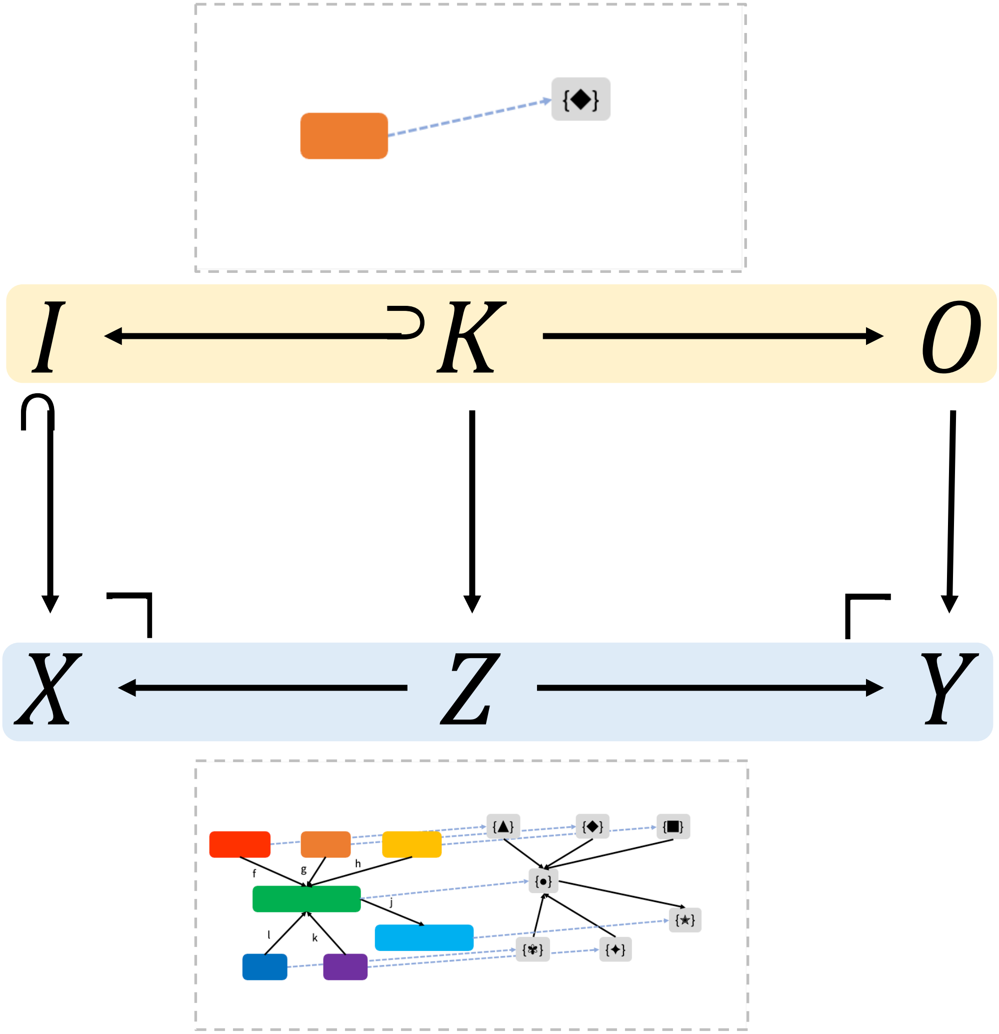
aaguinal@cs.umd.edu

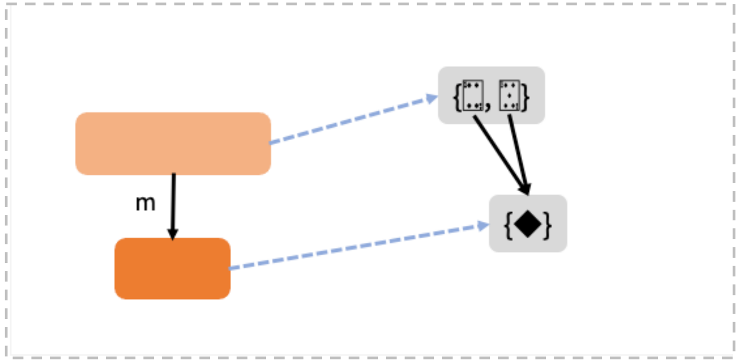
Summary

- Explained a toy planning example for how to make a tomato and lettuce sandwich
- Explored the applications of C-sets and DPO rewriting as the basis of a scene graph planning framework
- Touched on future work regarding analogies in planning, online planning, and scene affordance relations

Double-pushout (DPO) rewriting





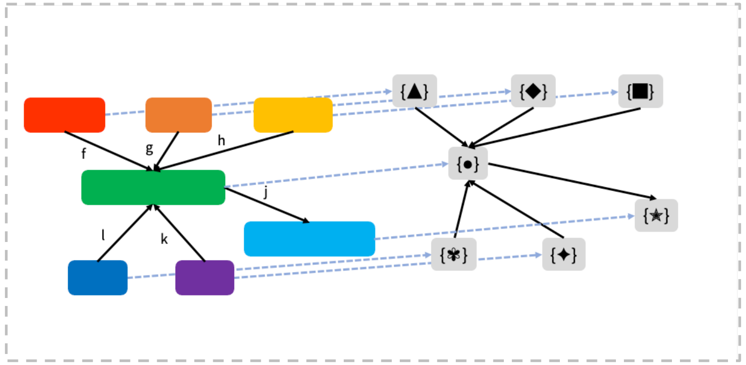


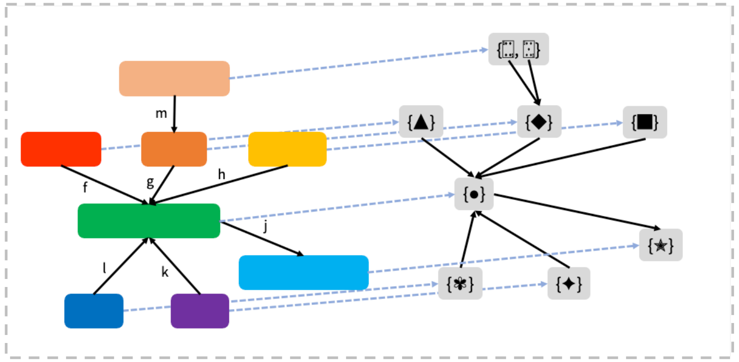
Applicability is determined by a monic transformation, $I \rightarrow X$

e.g. A monomorphism in Set is an injective function

DPO rewrite rule (action)

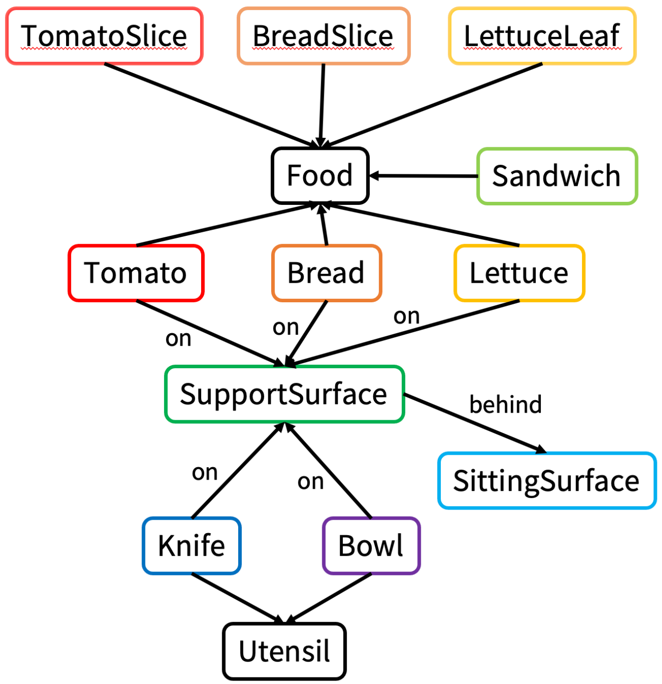
World state

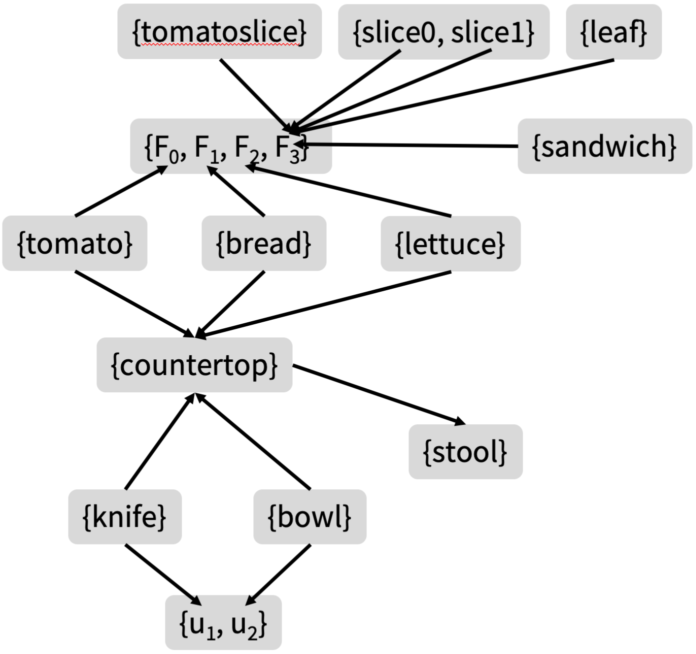




Merging and gluing operations

Depicting last action and last state in the plan






```
:close_sandwich=> @migration(SchDB, begin

      I => @join begin
      sandwich::Sandwich
      slice1::BreadSlice
      end

      K => @join begin
      sandwich::Sandwich
      slice1::BreadSlice
      end


      O => @join begin
      sandwich::Sandwich
      slice1::BreadSlice
      sandwich_is_food(sandwich) ==
      breadslice_is_food(slice1)
      end

end)
```

Syntax provided by Catlab.jl

A **conjunctive (merging)** operation is a **limit** in the category of representable functors.
A **gluing** operation is a **colimit** in the category of representable functors.