Applications of category theory to automated planning and program compilation in robotics

Angeline Aguinaldo

University of Maryland, College Park

Johns Hopkins University Applied Physics Laboratory

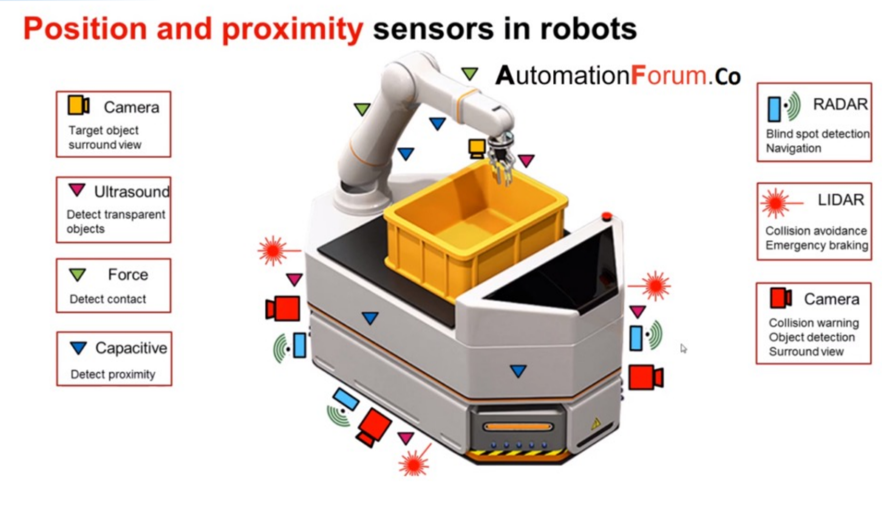
Topos Institute, Berkeley Seminar
August 22, 2022

Contents

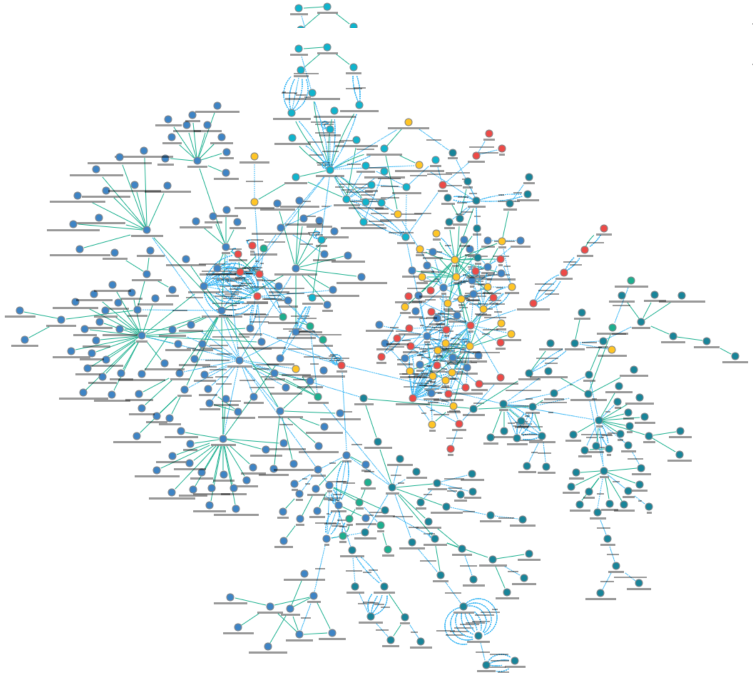
Contents

Making use of context in robotics

Context awareness – mechanism that allows an agent to adjust its behavior in response to dynamic context information such as location and resources; traditionally for mobile and IoT devices



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



Increased availability and capability of sensors results in an increase of information.

This increases the computational demands as more algorithms that use the information get deployed.

Deciding what information is relevant makes knowledge interoperable between tasks.

A general framework for determining what is contextually important (*contextual attention*)

Terminology

Definition (Context). *Context* is a description of the characteristics of the environment an agent must act in.

Definition (Action). An *action* is an operation that changes the state of some or all characteristics of the environment.

Definition (Task plan). A *task plan* is sequence of actions that achieves a specified goal.

Definition (Contextual attention). *Contextual attention* is the identification of context entities that are most important in achieving the task. Importance means that some property of achievable tasks exceeds a given threshold when the context entity is removed or modified.

Related methods for contextual attention

In perception and sensor fusion

- Filling in gaps in images based on context
- Representation learning (find most concise state representation)
- Value of information (VOI) analysis

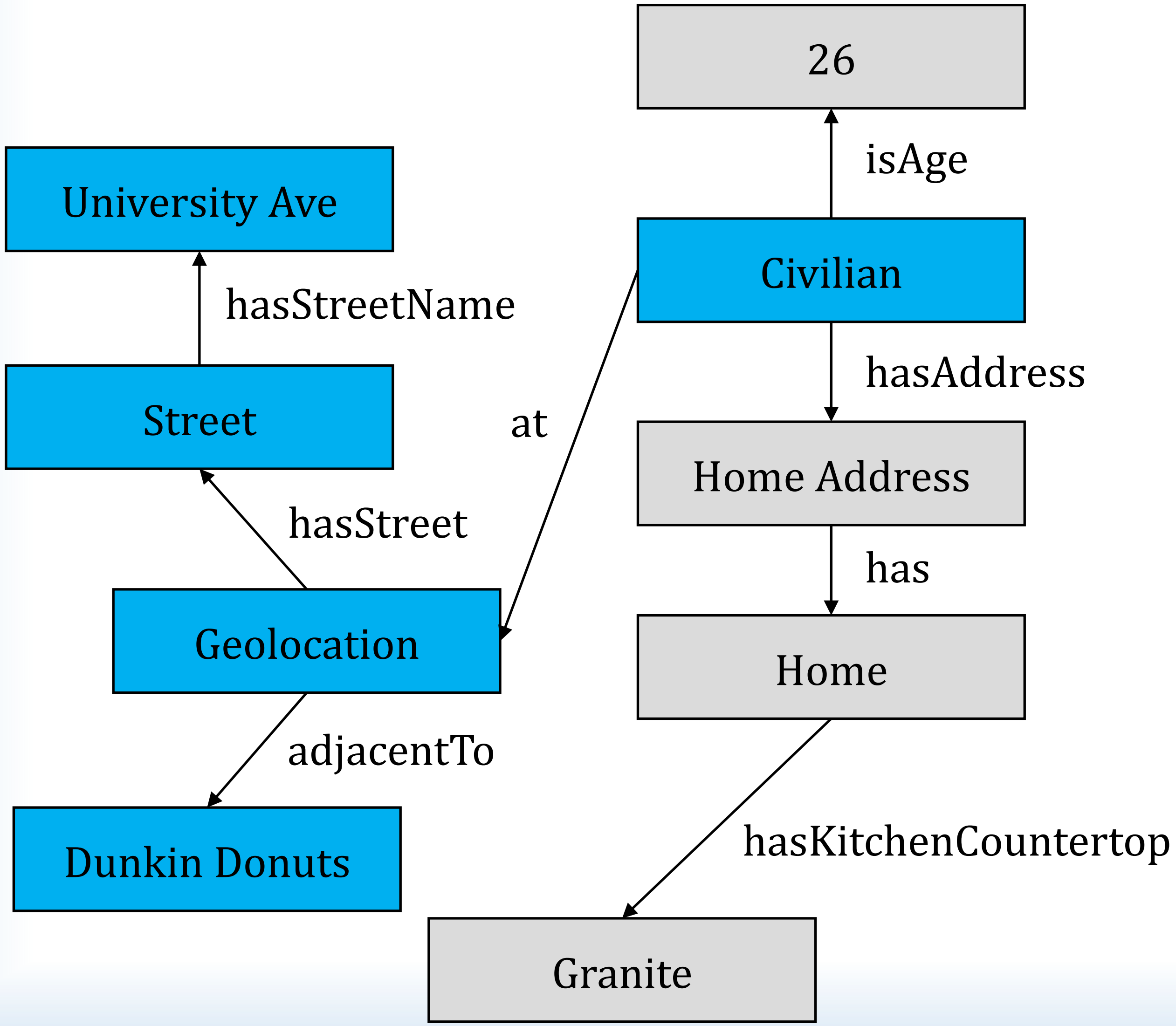
In knowledge representation and planning

- Case-based reasoning (Schank 1982)
- Recommender systems
- Bayesian network, POMDPs, MDPs

Limitations

- Data-driven, requires learning
- Requires attention criteria a priori
- Focused on inferring high-level context from low-level context
- No denotational semantics
- Not tied to capability

Example Scenario

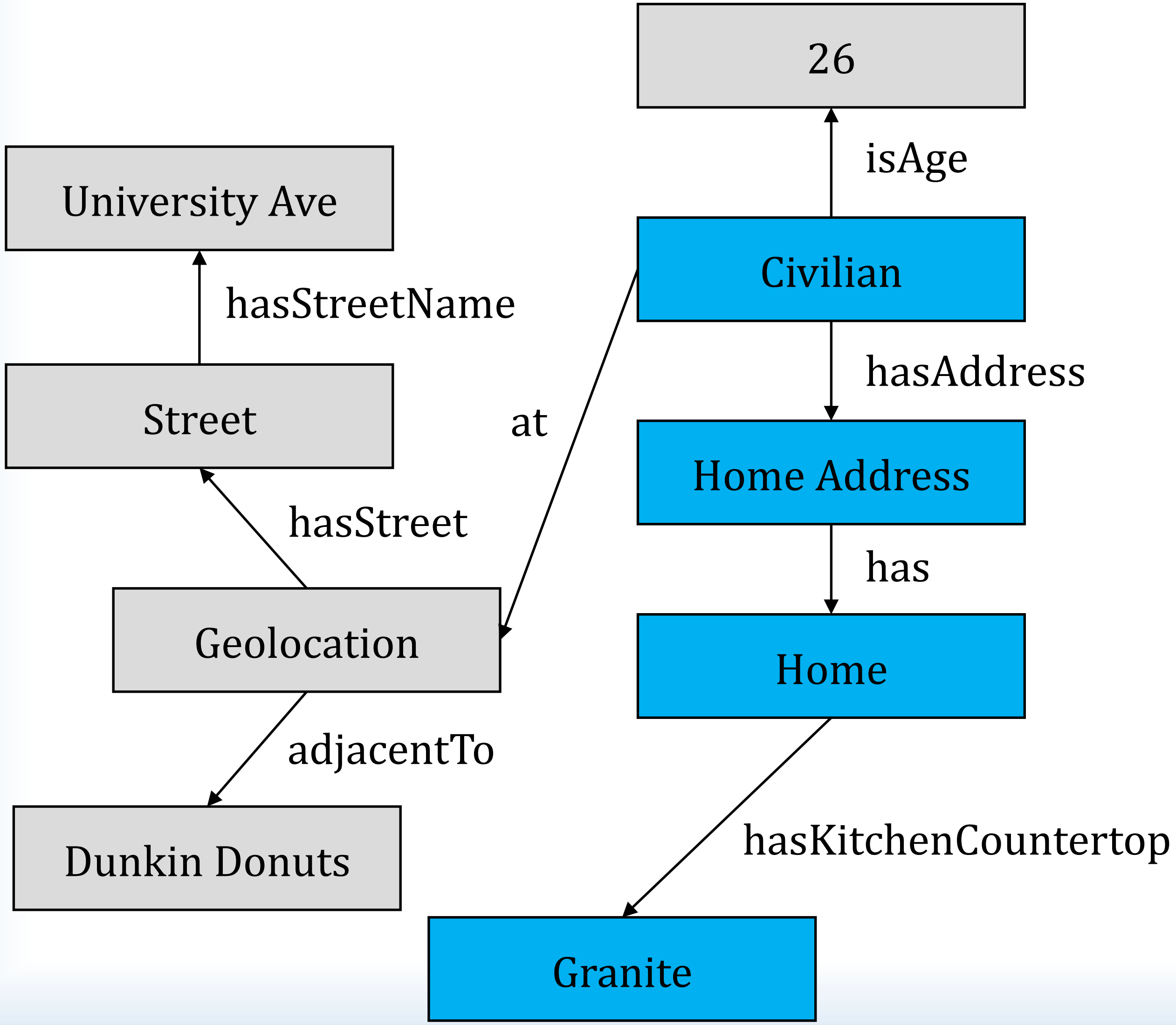


Disaster relief (path planning)

Task: Go to injured civilian

1. **Move forward** until you University Ave
2. **Move right** at the Dunkin Donuts
3. **Locate** civilian on street

Example Scenario

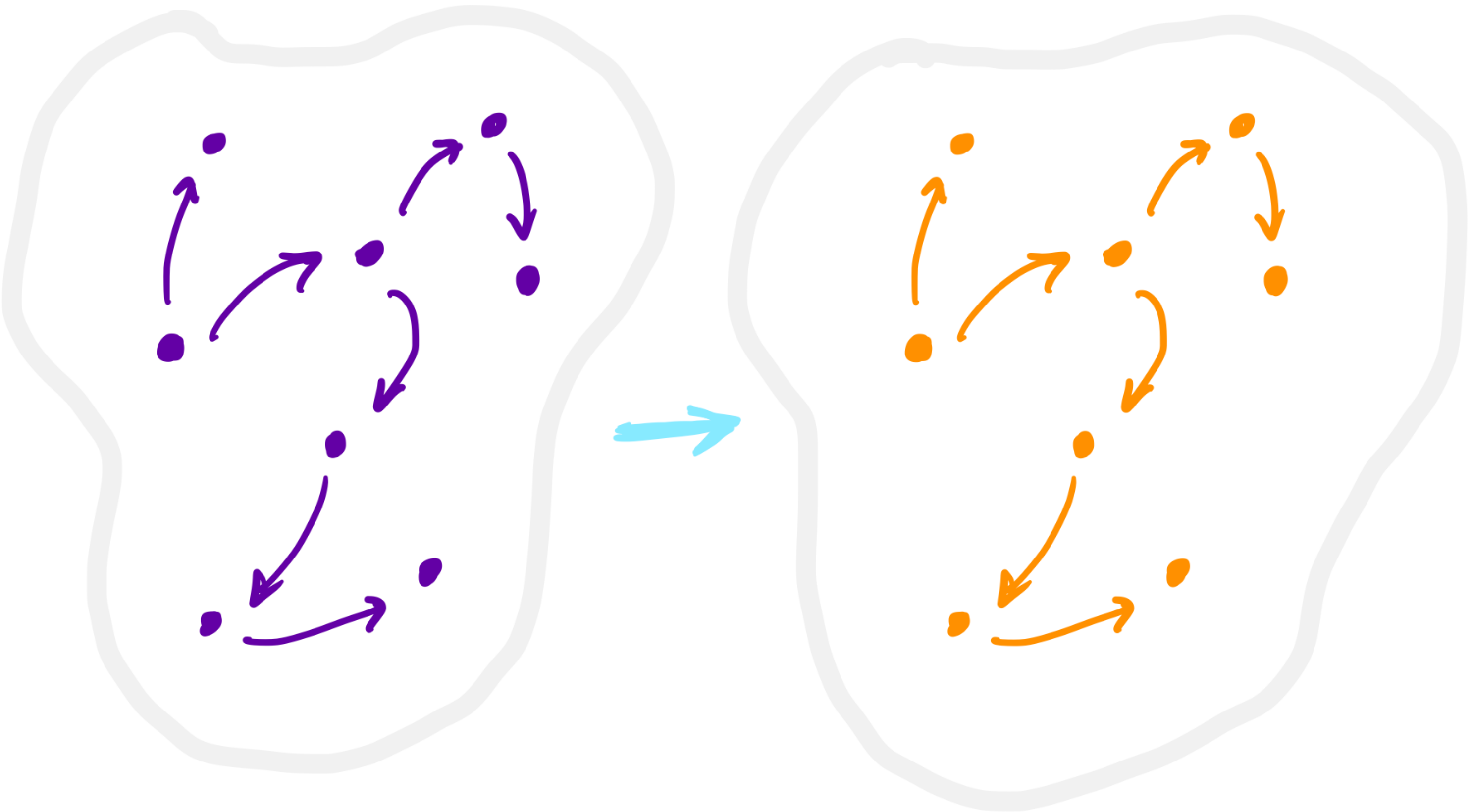


Home renovation (demolition)

Task: Replace granite with ceramic in kitchen

1. **Identify** surfaces with granite
2. **Measure** surface
3. **Remove** surface
4. **Add** ceramic slate of correct size

Informally speaking…



Some system tracking **context**

Some system tracking **achievable tasks**

Some **structure-preserving relationship** between them

Synchronization within robot architectures

All robotic architectures involve some synchronization of knowledge, plan, and control

	Context (knowledge model)	Task Plan (plan, actions)
Description	Environment refers to knowledge about the world such as what objects are present and where they are located	Task plans describe how the robot will achieve a goal by identifying a sequence of actions that symbolically update the state of the world.
Example syntax	ontologies, description logics, first-order predicate logic	hierarchical task nets (HTN), bi-partite directed acyclic graphs (DAGs), Markov decision processes (MDP)
Example semantics	URDF, SDF, KNOWROB	STRIPS, PDDL

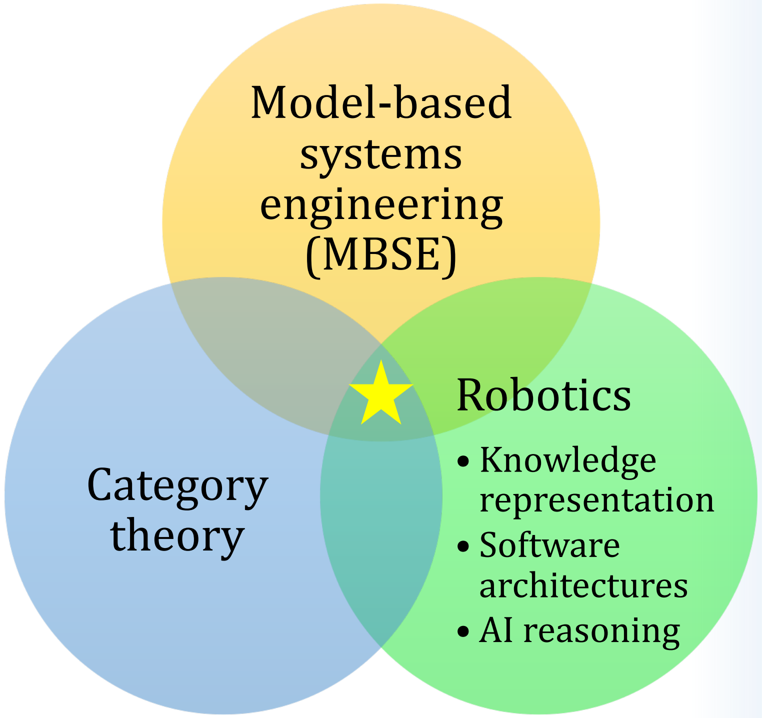
Formal semantic framework requirements

(a)	The ability to encode both **procedural**, such as task plans, and **declarative**, namely context, data.
(b)	The ability to encode **structure-preserving relationships** between task plans and context.
(c)	The ability to describe **composite** (parts of a whole, decomposition, traceability) relationships and **composition** (merging, gluing, synthesis) behaviors.
(d)	The ability to encode **binary relations** such as equivalence and inclusion.
(e)	The ability to describe **constraints** demanded by the semantics of knowledge, plans, and control.

If successful, we can use this formal system to simulate the effect changes in context have on a robot's task capabilities.

Related work

- **MBSE** & **Robotics**
  - Platform independent model (PIM) and/or platform specific model (PSM) with model-to-model and model-to-text transformation methods to synthesize robotic implementations *(Heinzemann 2018, Bocciarelli 2019, Brugali 2016, Ruscio 2016, Bruyninckx 2013, Ringert 2015, Nordmann 2015, Wigand 2017, Steck 2011, Schlegel 2010, Hochgeschwender 2016)*
- **MBSE** & **Category theory**
  - Bidirectional model synchronization, state-based and delta-based lenses *(Diskin 2008, Diskin 2011, Diskin 2012)*
  - Model transformations with constraints *(Rutle 2010, Rutle 2012)*
  - Program synthesis using metamodels *(Batory 2008)*
- **Robotics** & **Category theory**
  - Symmetric monoidal categories for modeling robot program abstractions *(Aguinaldo 2020)*
  - Co-design applied to autonomous system design *(Zardini 2021 ECC, Zardini 2021 IROS)*



Contents

What is category theory?

Category theory is a branch of mathematics that provides mathematical structures whose properties are attentive to **composition of relationships**.

A category ($\mathbb{C}$) is:

- A set of **objects** $\{A, B, C, \dots\}$
- A set of **morphisms** $\{f, g, h, \dots\}$ that map objects to objects
  - Where every object has an identity morphism, id_A
- **Composition operator**, $\circ$, between morphisms that is *associative* and has identity morphisms as *units*

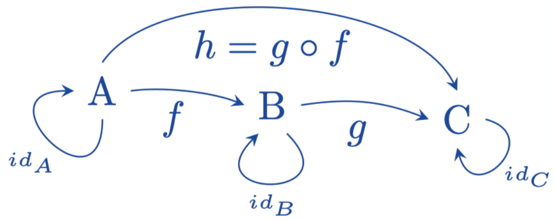
$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[rr, bend left, "h = g \circ f"] \arrow[loop left, "id_A"] & B \arrow[r, "g"] \arrow[loop below, "id_B"] & C \arrow[loop right, "id_C"]
\end{tikzcd}$$

What is category theory?

Category theory is a branch of mathematics that provides mathematical structures whose properties are attentive to **composition of relationships**.

A category ($\mathbb{C}$) is:

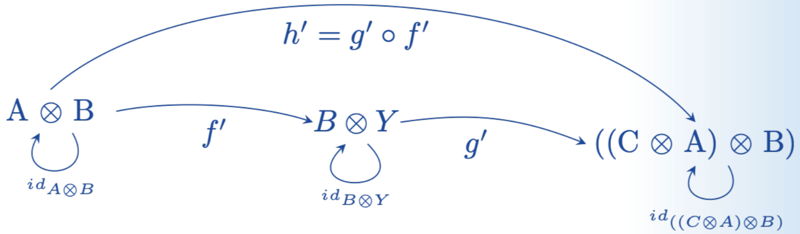
- A set of **objects** $\{A, B, C, \ldots\}$
- A set of **morphisms** $\{f, g, h, \ldots\}$ that map objects to objects
  - Where every object has an identity morphism, id_A
- **Composition operator**, $\circ$, between morphisms that is *associative* and has identity morphisms as *units*



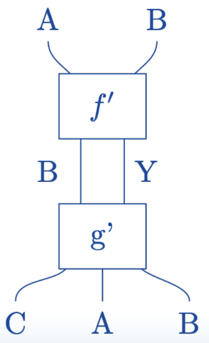
A symmetric monoidal category ($\mathbb{M}$), adds:

\+ **Tensor product**, $\otimes$, which is a product on $\mathbb{M}$ (objects and morphisms) that has *associator* and *unitor isomorphisms*

\+ Braiding isomorphism where $B_{\{X,Y\}}: X \otimes Y \to Y \otimes X$



A string diagram is the graphical syntax for symmetric monoidal categories, where **boxes are morphisms** and **strings are objects**.

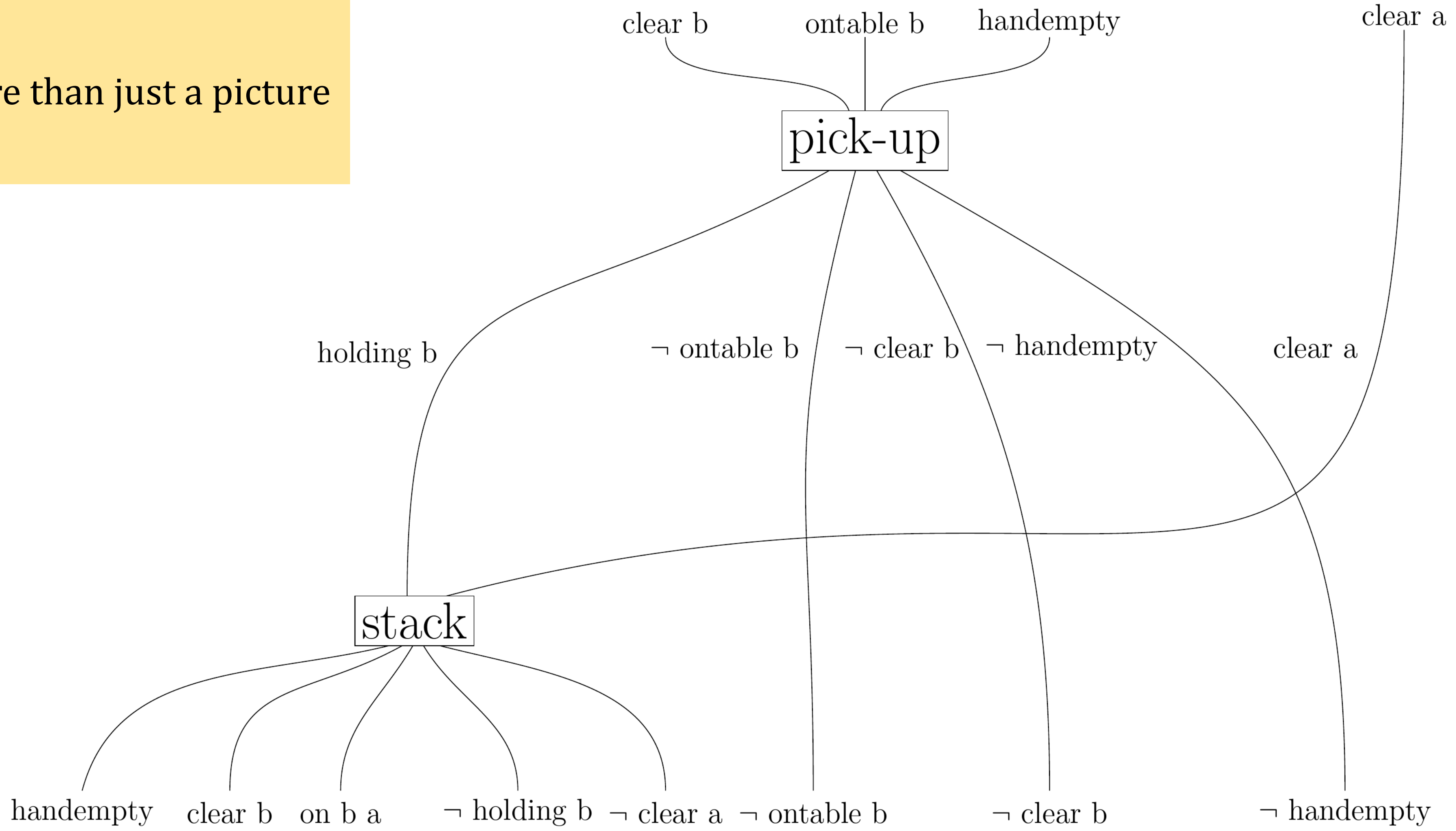


$$(id_C \otimes id_B \otimes id_A) \circ$$
$$g' \circ$$
$$(id_B \otimes id_Y) \circ$$
$$f' \circ$$
$$(id_A \otimes id_B)$$

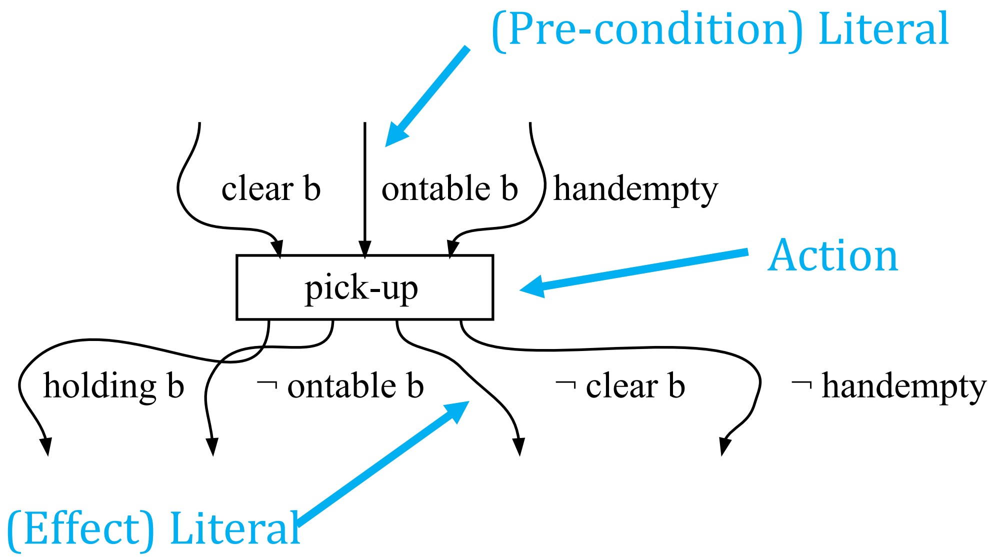
Contents

String Diagrams from Category Theory

! More than just a picture



String Diagrams for PDDL

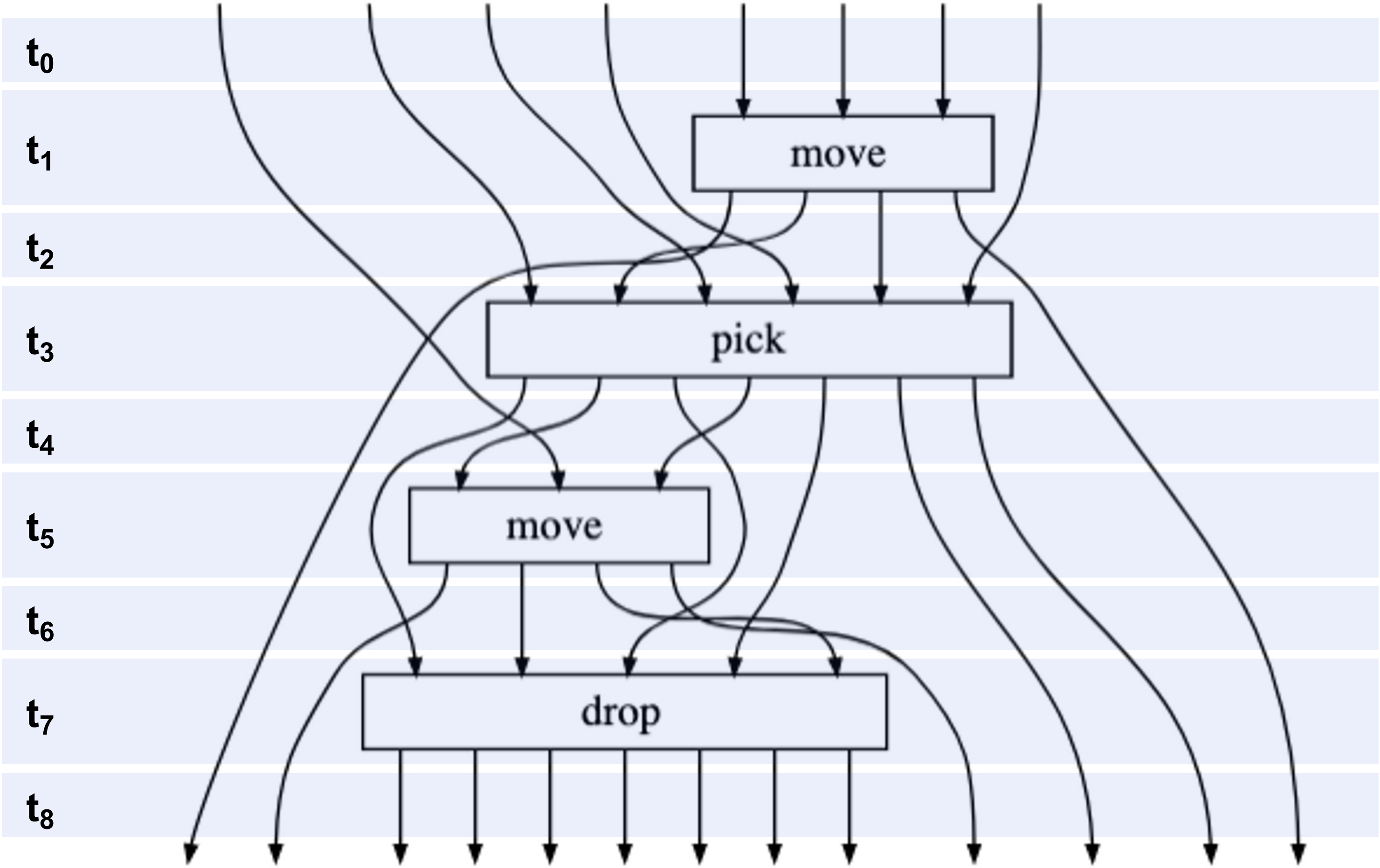


Categorification of Planning Solution

- Objects are **literals**
- Morphisms are **actions**
- Composition (∘) **chains actions**
- Tensor product (⊗) implies **parallel actions** or **conjunction of literals**

PDDL – Planning Domain Definition Language (McDermott 1998)

String Diagrams for Resource Tracking



String diagram with arbitrary time slices (t_0 - t_8) overlayed. At every time slice, we have complete knowledge of the data resources and/or function(s) running. Each slice can be re-interpreted in a linear mathematical syntax (not shown). Note, this is only one sample, discovered by the PDDL solver, from the larger valid solution space.

Visualize Classical AI Planning Solutions

Domain file

```
(define (domain BLOCKS)
(:requirements :strips)
(:predicates (on ?x ?y)
(ontable ?x)
(clear ?x)
(handempty)
(holding ?x)
)

(:action pick-up
:parameters (?x)
:precondition (and (clear ?x)
(ontable ?x) (handempty))
:effect
(and (not (ontable ?x))
(not (clear ?x))
(not (handempty))
(holding ?x)))
...
```

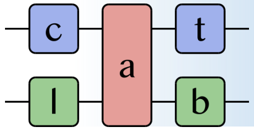
Problem file

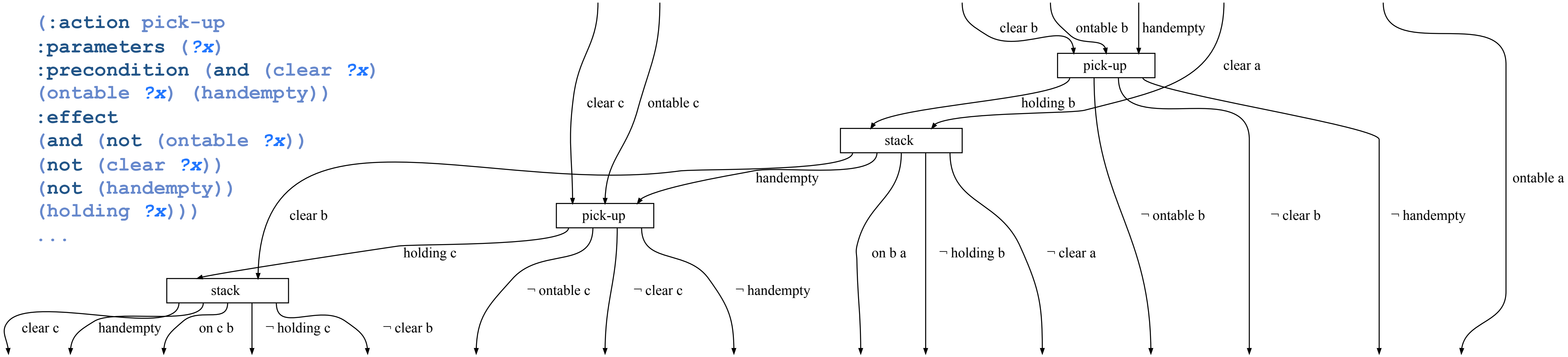
```
(define (problem BLOCKS-3-0)
(:domain BLOCKS)
(:objects a b c)
(:init (clear c) (clear a) (clear b)
(ontable c) (ontable a) (ontable b)
(handempty))
(:goal (AND (on c b) (on b a)))
)
```

Solution

```
pick-up b
stack b a
pick-up c
stack c b
```

Angeline Aguinaldo and William Regli. *Encoding Compositionality in Classical Planning Solutions.* International Joint Conference for Artificial Intelligence (IJCAI) Generalization in Planning Workshop. 2021.





Contents

What is a functor?

C. Translating Using Functors

The ability to translate the context from one abstraction level to another is a necessary step when compiling goal-oriented description to motion primitives. Category theory permits this concept via *functors*. Functors map objects and arrows between pairs of categories. If $\mathbf{X}$ and $\mathbf{Y}$ are categories, a functor, $F : \mathbf{X} \to \mathbf{Y}$ maps an object in $\mathbf{X}$ to some object in $\mathbf{Y}$ and maps each arrow between two objects in $\mathbf{X}$ to an arrow in $\mathbf{Y}$, such that (9) and (10) are satisfied:

$$F(\mathrm{id}_{\mathbf{X}}) = \mathrm{id}_{F\mathbf{X}} \tag{9}$$

$$F(g \circ f) = Fg \circ Ff \tag{10}$$

where f and g are composable arrows in $\mathbf{X}$.

Structure-preserving map between categories

Example

X and **Y** are categories where,

Objects($\boldsymbol{X}$) =
$\{A, B, C, D\}$
Arrows($\boldsymbol{X}$) =
$\{f: A \to \mathrm{B}, g: B \to \mathrm{C}, h: B \to D\}$

Objects($\boldsymbol{Y}$) =
{dog, cat, mouse, rabbit}
Arrows($\boldsymbol{Y}$) =
$\{f': \mathrm{dog} \to \mathrm{cat}, g': \mathrm{cat} \to \mathrm{mouse}, h': \mathrm{cat} \to \mathrm{rabbit}\}$

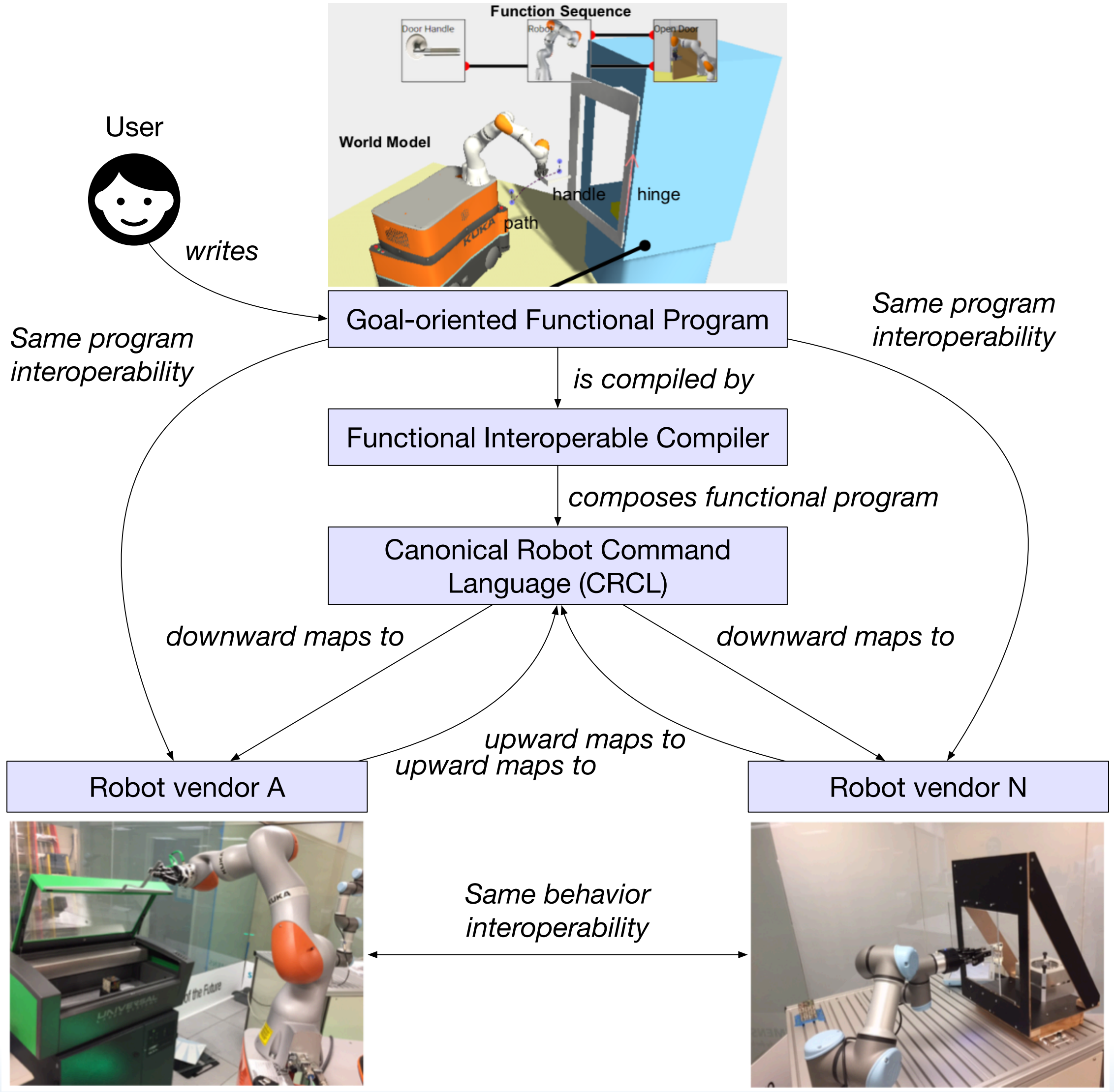
A possible functor, F, could be

objects	arrows	identities
$A \longmapsto$ dog	$f \longmapsto f'$	$id_A \longmapsto id_{\mathrm{dog}}$
$B \longmapsto$ cat	$g \longmapsto g'$	$id_B \longmapsto id_{\mathrm{cat}}$
$C \longmapsto$ mouse	$h \longmapsto h'$	$id_C \longmapsto id_{\mathrm{mouse}}$
$D \longmapsto$ rabbit		$id_D \longmapsto id_{\mathrm{rabbit}}$

Check

$$F(g \circ f) = Fg \circ Ff = g' \circ f'$$

Goal-Oriented Robot Programming

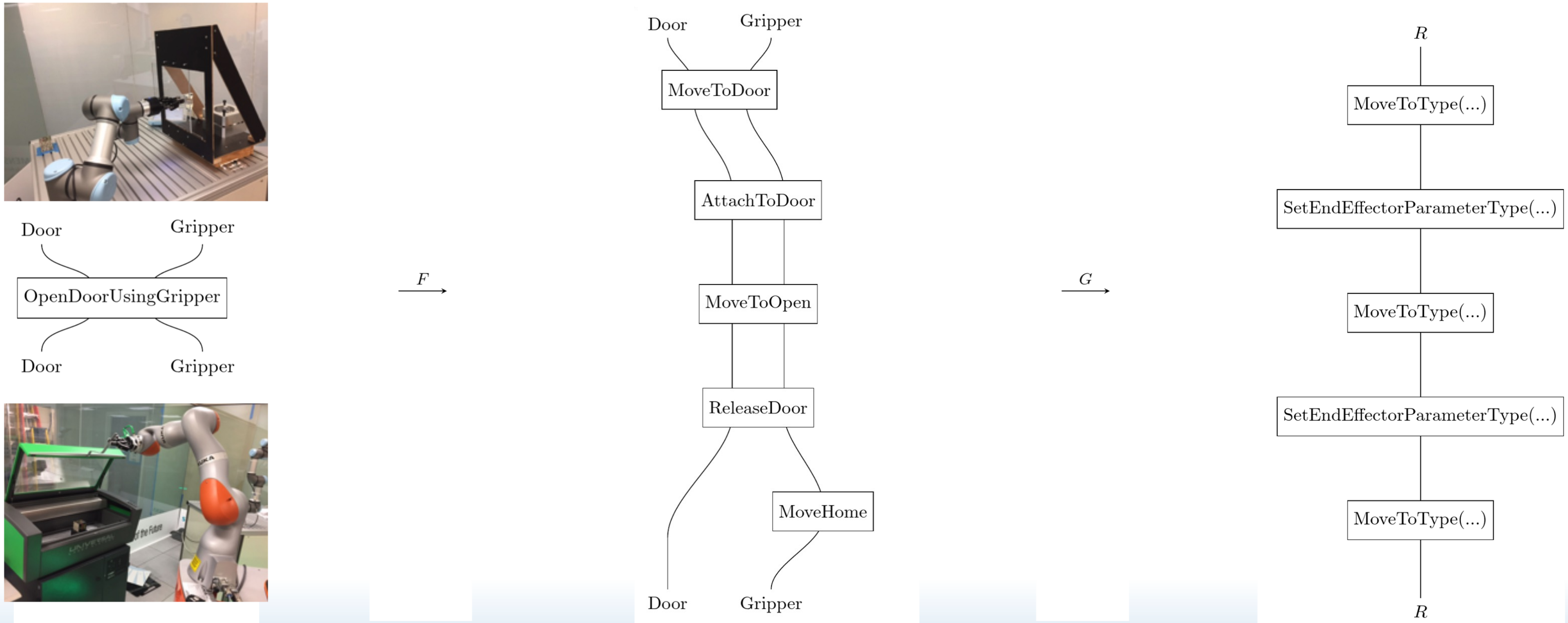


Goal-Oriented Robot Programming

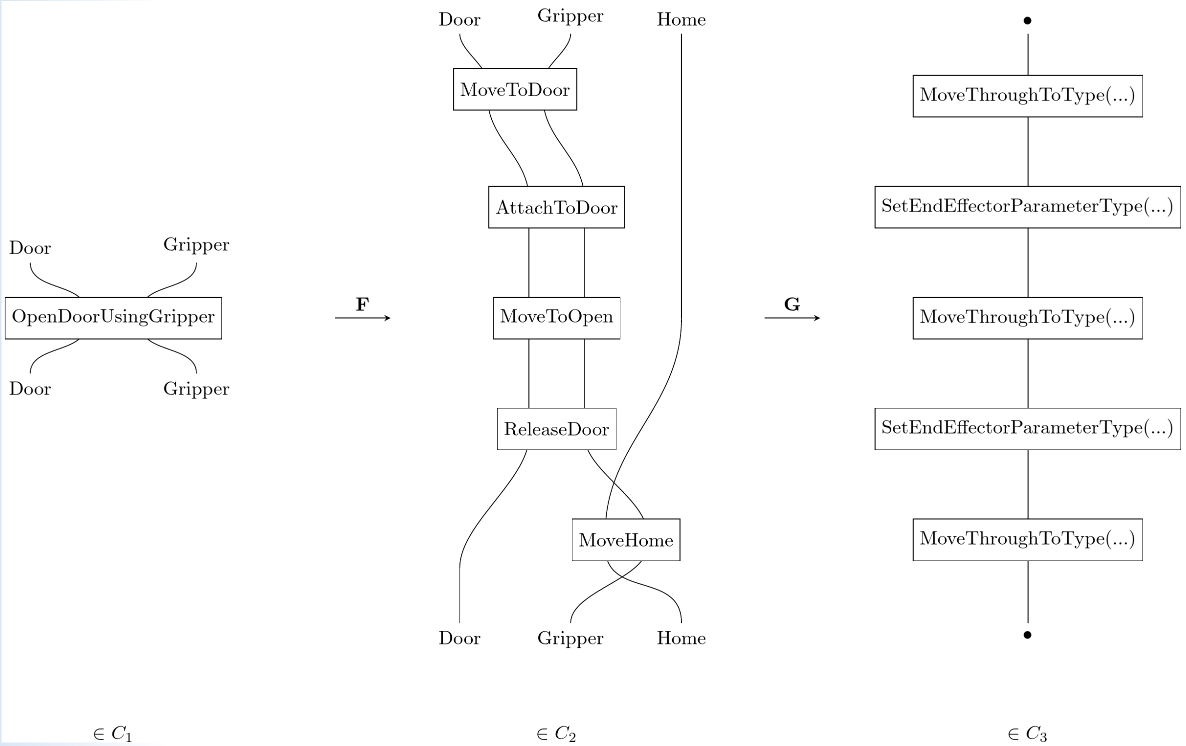
Physical
Identify types of physical resources needed to execute program. Name the program.

Software
Identify skills necessary to complete the desired action. Identify informational inputs and outputs for each skill.

Specification
Identify available command types and their possible parameters according to target robot command specification.

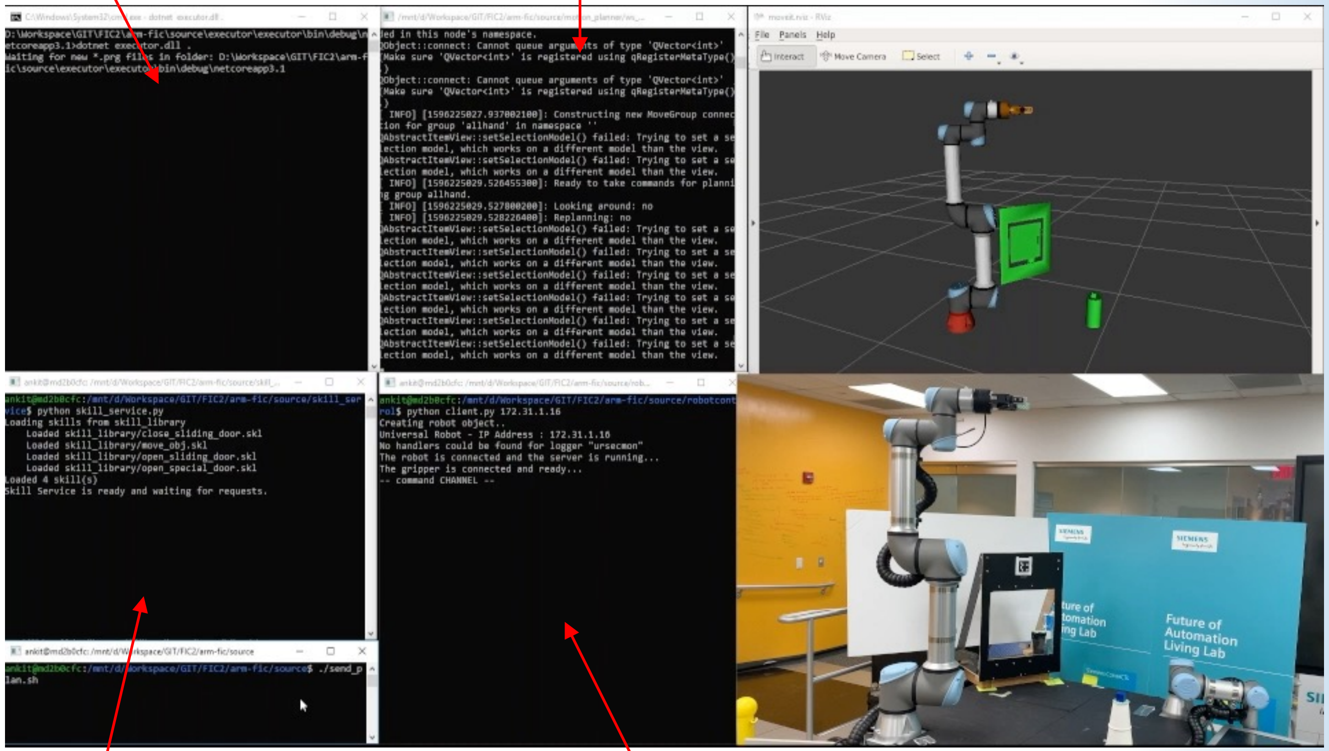


Goal-Oriented Robot Programming



CRCL Message Broker

Motion planning



Invoking skill library

Robot specific controller

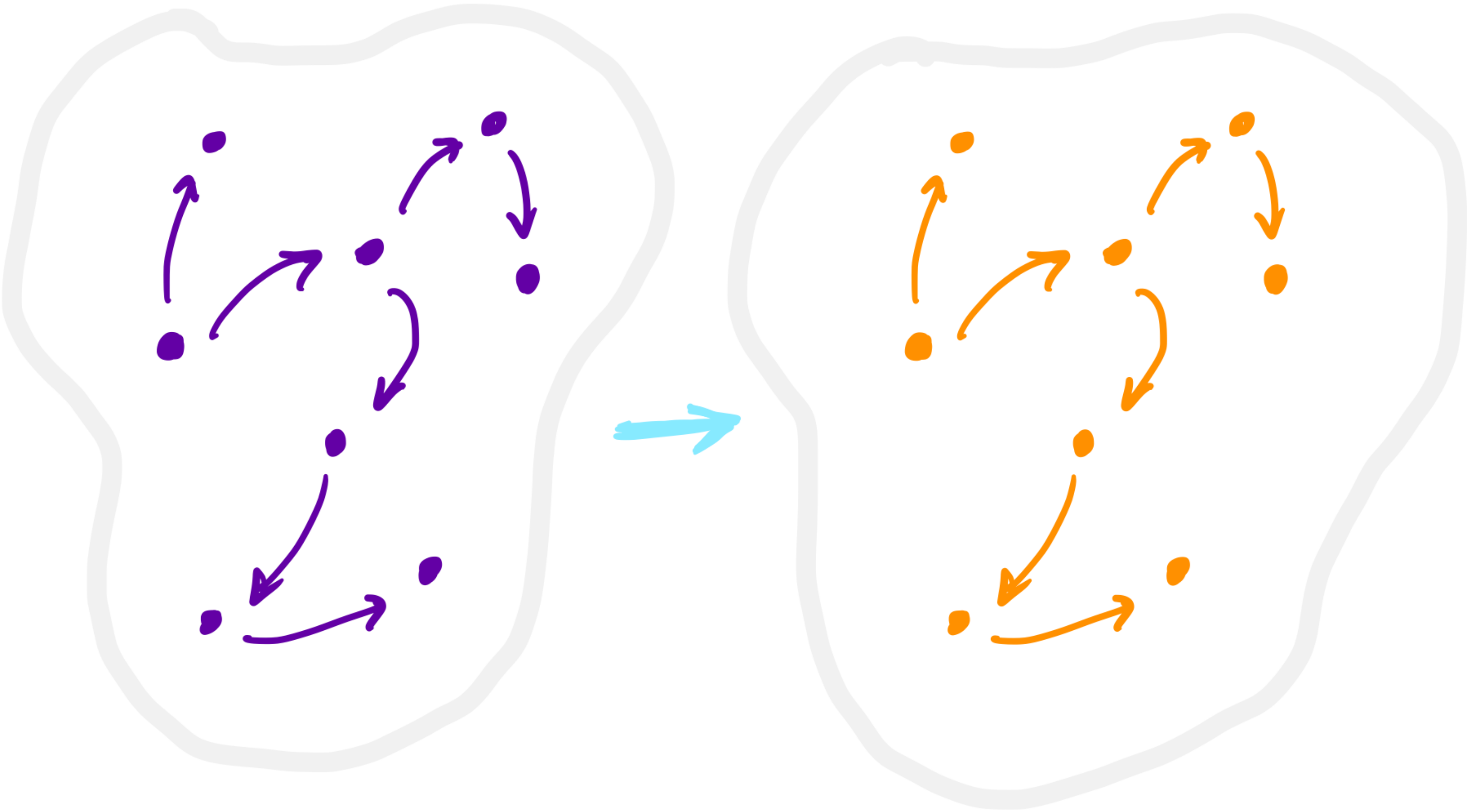
A. Aguinaldo, J. Bunker, B. Pollard, A. Canedo, G. Quiros, W. Regli. *RoboCat: A category theoretic framework for robotic interoperability using goal-oriented programming.* IEEE Transactions for Automated Science and Engineering. 2022.

SIEMENS

ARM ADVANCED ROBOTICS FOR MANUFACTURING

Contents

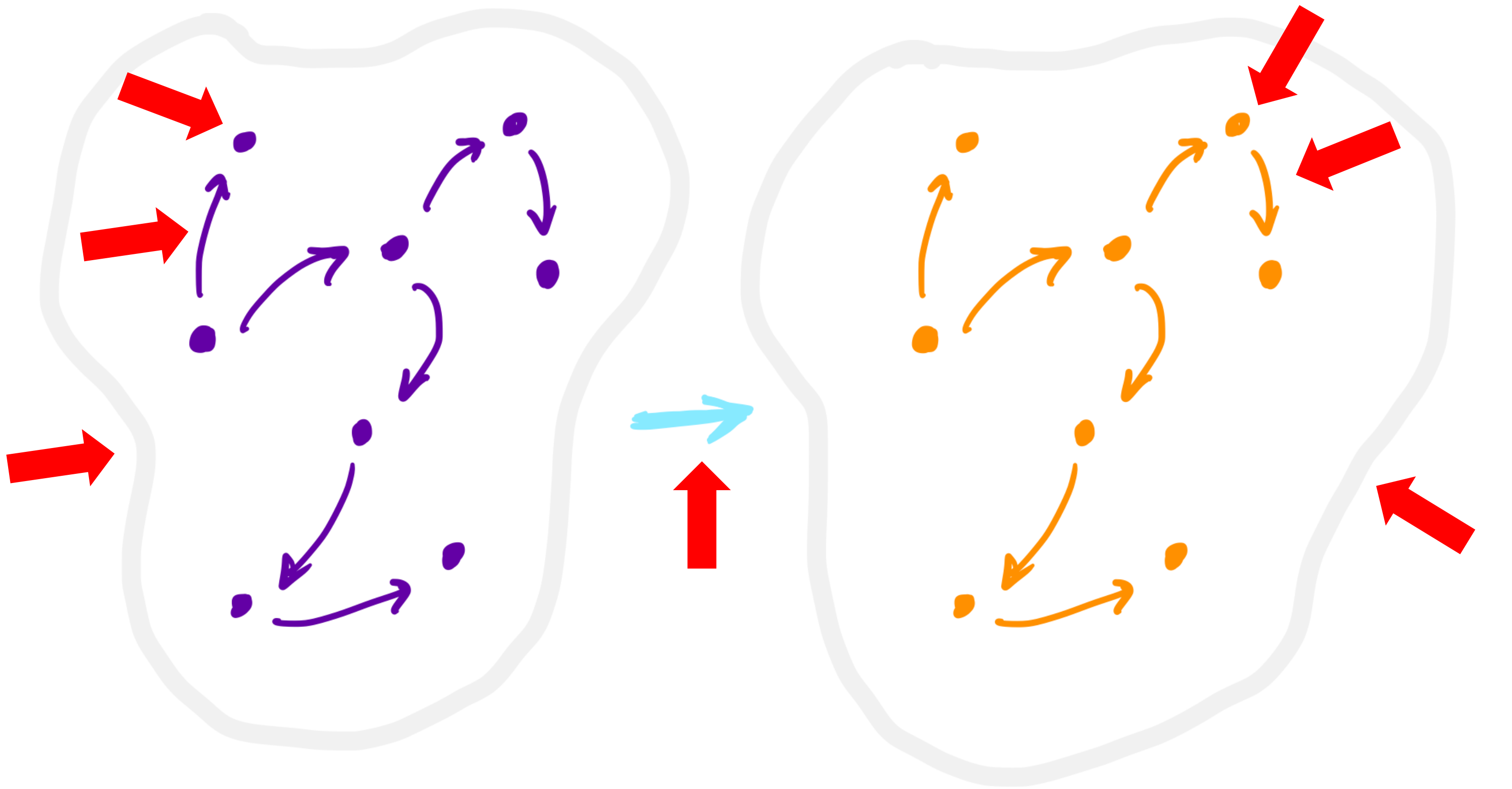
Contextual Attention Categorical Model



Some system tracking **context** Some system tracking **achievable tasks**

Some **structure-preserving relationship** between them

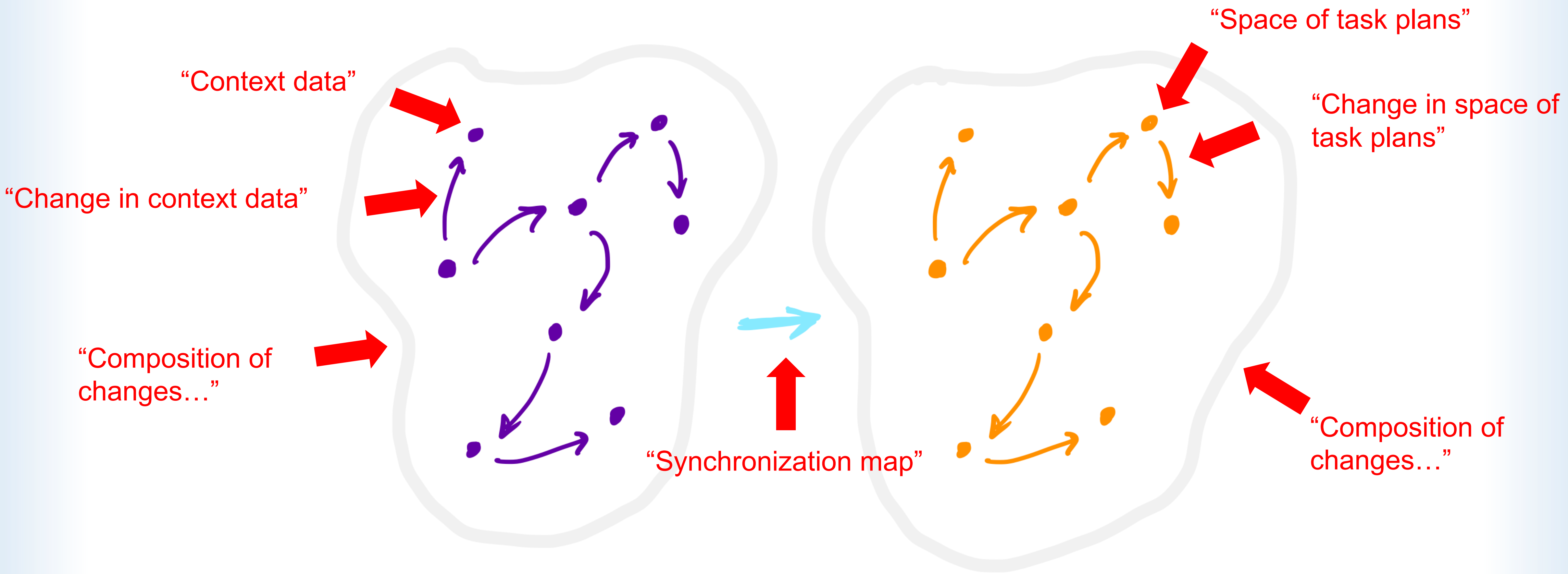
Contextual Attention Categorical Model



Some system tracking **context** Some system tracking **achievable tasks**

Some **structure-preserving relationship** between them

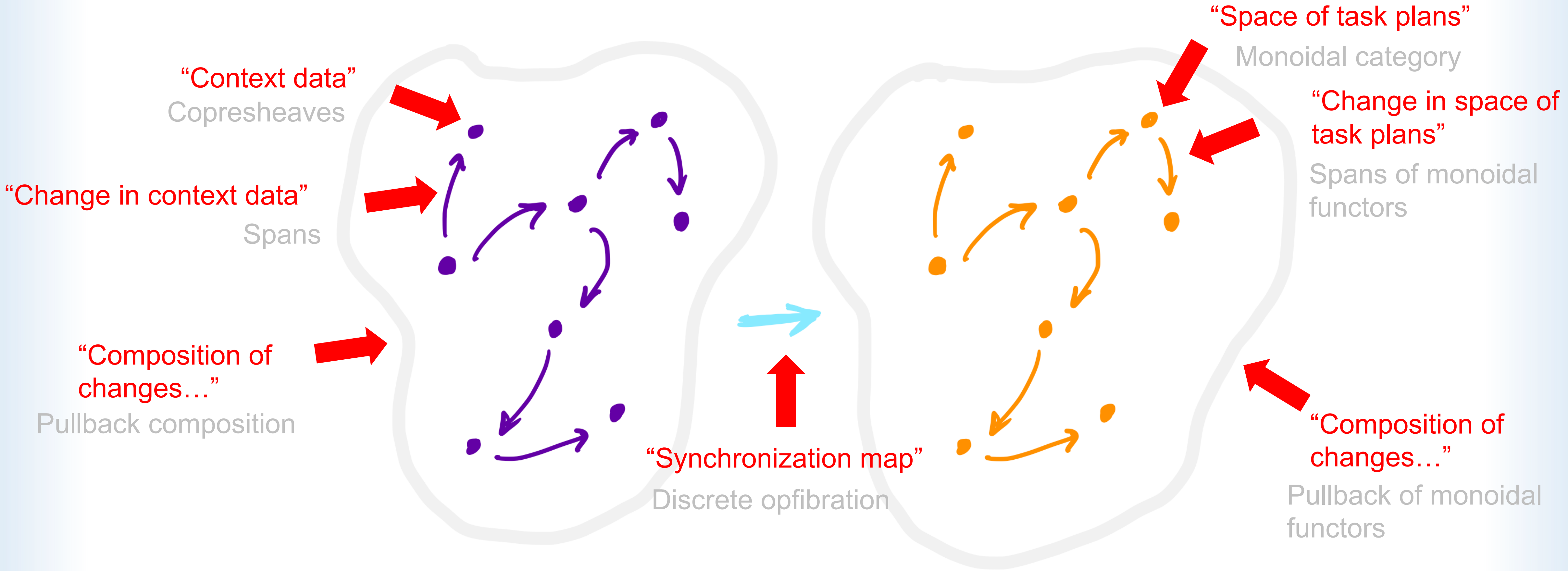
Contextual Attention Categorical Model



Some system tracking **context** Some system tracking **achievable tasks**

Some **structure-preserving relationship** between them

Contextual Attention Categorical Model



Some system tracking **context** Some system tracking **achievable tasks**

Some **structure-preserving relationship** between them

Contextual Attention Categorical Model

Delta lenses (Diskin 2011)
Morphism in the category of small categories, **Cat**
- Discrete opfibration functor, G

$$
\begin{tikzcd}
\text{Task plans} \quad \mathbb{A} \arrow[d, "G"] & A \arrow[r, "\phi_{(w,A)}"] \arrow[d, dash, dashed] & \phi_{(w,A)}(A) \arrow[d, dash, dashed] \\
\text{Context} \quad \mathbb{B} & GA \arrow[r, "w"] & B
\end{tikzcd}
$$

Such that,
1. $G\left(\phi_{(w,A)}\right) = w$
2. Lifting, ϕ, respects composition and identities

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

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

Modeling capabilities
- **Traceability** is instantiated via functor, G
- **Change information** is captured via the span, or delta, construction for arrows
- **Synthesis** of spaces of task plans is provided by the lifting property of discrete opfibrations.

Aguinaldo A., Regli W. Modeling traceability, change information, and synthesis in autonomous system design using symmetric delta lenses. ICRA Compositional Robotics Workshop 2022.

Category of Context

$\mathbb{D}$ is an Olog category (Spivak 2012), the syntactic category for databases, where objects are types and arrows are is-a relations and properties.

Objects

$$I\colon \mathbb{D} \to \mathbf{Set}$$

$$J\colon \mathbb{D} \to \mathbf{Set}$$

$$\dots$$

$$K\colon \mathbb{D} \to \mathbf{Set}$$

where $I, J, \dots K \in \mathbf{D} - \mathbf{Inst}$ map to sets with the empty element. Otherwise known as copresheaves.

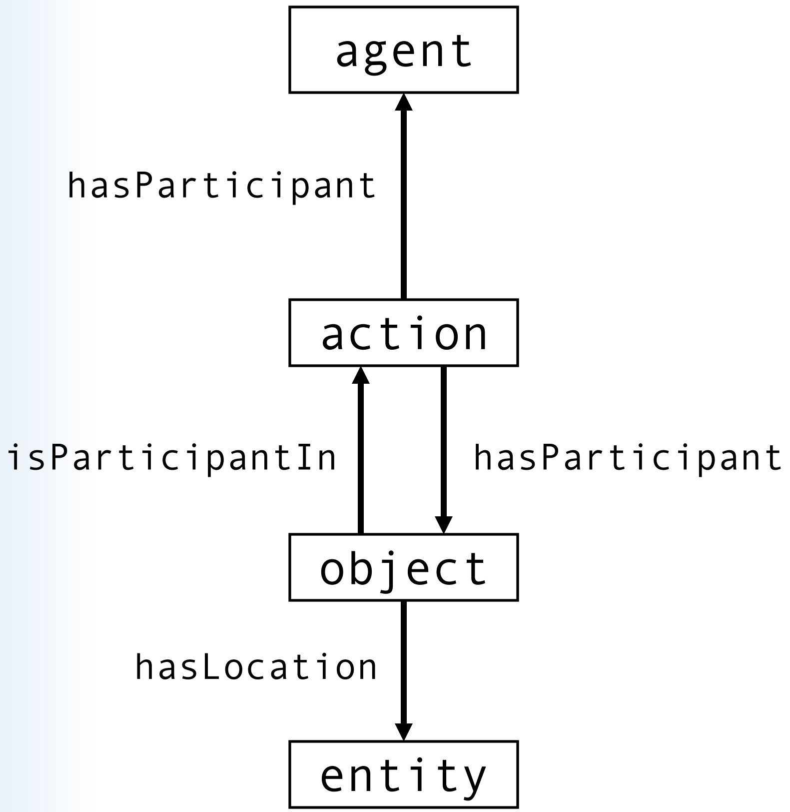
Arrows

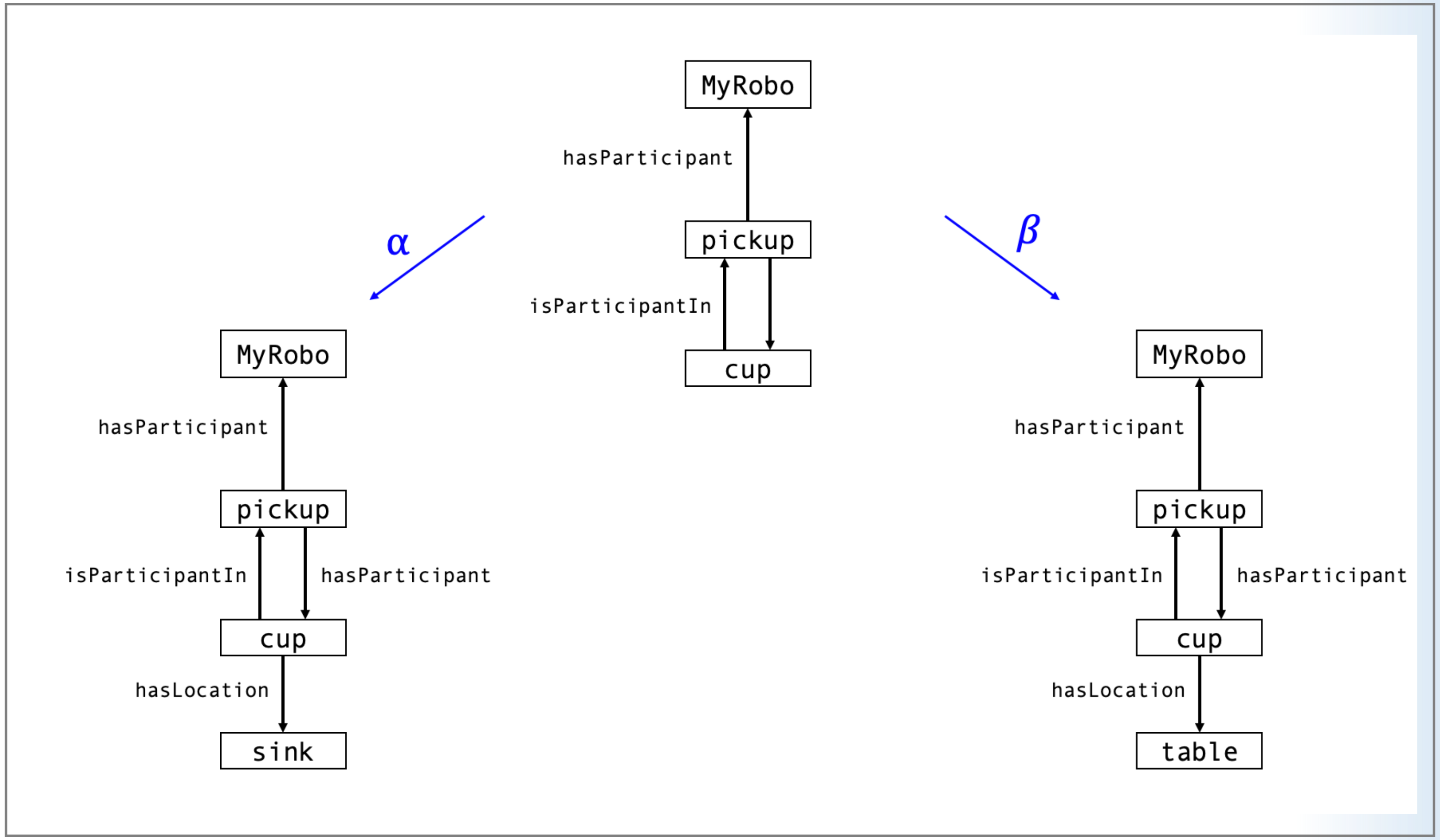
$$f : I \to J$$

$$\begin{tikzcd} & K \arrow[dl, "\alpha : K \to I"'] \arrow[dr, "\beta : K \to J"] & \\ I & & J \end{tikzcd}$$

where,
composition is the pullback

Category of Context (Example)





Based on KNOWROB ontology (Tenorth 2015)

Category of Achievable Tasks

$\mathbb{T}$ is the category of monoidal categories. Functors between monoidal categories preserve the monoidal structure.

Objects

$\mathrm{Monoidal}(X_1, A_1, \otimes)$

$\mathrm{Monoidal}(X_2, A_2, \otimes)$

...

$\mathrm{Monoidal}(X_n, A_n, \otimes)$

where,

$X_i \in$ Generating set of logical predicates

and

$A_i \in$ Generating set of actions

and

$\otimes$ is the conjunction of predicates and actions

Arrows

$$f : \mathrm{Monoidal}(X_i, A_i, \otimes) \to \mathrm{Monoidal}(X_j, A_j, \otimes)$$

$$\begin{tikzcd}
 & \mathrm{Monoidal}(X_k, A_k, \otimes) \arrow[dl, "g"'] \arrow[dr, "h"] & \\
\mathrm{Monoidal}(X_i, A_i, \otimes) & & \mathrm{Monoidal}(X_j, A_j, \otimes)
\end{tikzcd}$$

where,

composition is the pullback of a span of functors

Future Work

- What **functor, *G*,** can be defined between the proposed categories?
  - Does G meet the requirements of delta lenses?
- Are all items in the **formal semantic framework requirements** met in this framework?
- What **other properties** does this framework afford us? What other properties *should* we be modeling?
  - Can we make a statement about whether an automated decision-maker is more capable than another given the same information using this framework?
- How might we **implement this framework on a computer**? What is the computational complexity of these queries?

Contents

Flexible and adaptable formal semantics

Category theory as conceptual stem-cell

Category theory (CT) can differentiate into many forms:

- All forms of pure math... (we'll briefly discuss this)
- Databases and knowledge representation (categories and functors)
- Functional programming languages (cartesian closed categories)
- Universal algebra (finite-product categories)
- Dynamical systems and fractals (operad-algebras, co-algebras)
- Hierarchical planning (lenses and monads)
- Shannon Entropy (operad of simplices)
- Partially-ordered sets and metric spaces (enriched categories)
- Higher order logic (toposes = categories of sheaves)
- Measurements of diversity in populations (magnitude of categories)
- Collaborative design (enriched categories and profunctors)
- Petri nets and chemical reaction networks (monoidal categories)
- Quantum processes and NLP (compact closed categories)

2 / 32

David Spivak: https://applied-compositional-thinking.engineering/wp-content/uploads/2021/01/ETH2021-ACT4E.pdf

Tooling in development

Autogenerate PDDL String Diagrams

String diagrams are a graphical language used to describe symmetric monoidal categories (SMCs) from category theory. They can be seen as mathematical rigorous expressions to describe processes and their dependencies. In this notebook, we use string diagrams to express the solutions to Planning Domain Definition Language (PDDL) problems. More specifically, we seek to observe if the string diagram representation can elucidate interesting properties of robot manipulator program plans in a manufacturing work cell. This code uses the WiringDiagram Catlab Julia library to construct the string diagrams. In these examples, the objects are considered to be Boolean expressions and the arrows, or morphisms, are the PDDL actions.

In [1305]:

```
# SOFTWARE PRE-REQ
#
# Julia 1.3.1
# Catlab 0.5.3
# Latex

using Catlab.WiringDiagrams
using Catlab.Doctrines
using Catlab

using Catlab.Graphics
import Catlab.Graphics: Graphviz

import TikzPictures
using Catlab.Graphics
```

In [1306]:

```
EXAMPLE = "blocksworld";
```

Process PDDL Files and PDDL solution

To run this notebook, you must provide the name of directory (in `examples/`) containing domain.pddl, problem.pddl, and solution.txt in the `EXAMPLE` variable (above), then run *all* cells. The composed string diagram is shown as the output of the last cell. It can also be seen as an SVG in `smc.dot.svg` .

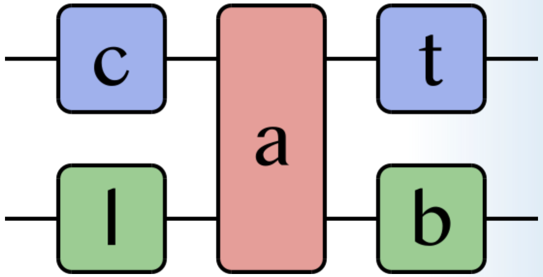
About files

The `domain.pddl` and `problem.pddl` files must adhere to PDDL specifications following the `:strips` requirement.

The `solution.txt` file should be a newline for each action with parameters provided by a PDDL planner of choice. An example is shown below:

```
move lochome locbox2
pick boxa locbox2grippera
drop boxa locbox2 grippera
```

One possible way to obtain a PDDL solution is to run PDDL4j solver, using



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

Thank you for listening!

Angeline Aguinaldo

aaguinal@cs.umd.edu