Modeling traceability, change information, and synthesis in autonomous system design using symmetric delta lenses

Angeline Aguinaldo[1, 2] and William Regli[1]

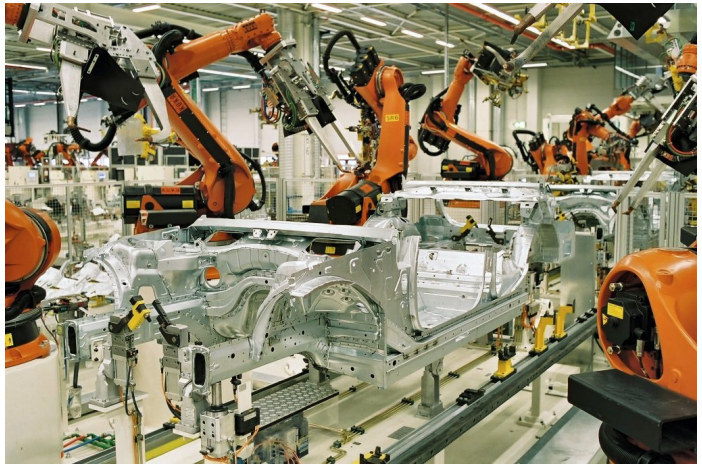
International Conference on Robotics and Automation (ICRA) 2022
Compositional Robotics: Mathematics and Tools Workshop

1 DEPARTMENT OF COMPUTER SCIENCE

2 JOHNS HOPKINS APPLIED PHYSICS LABORATORY

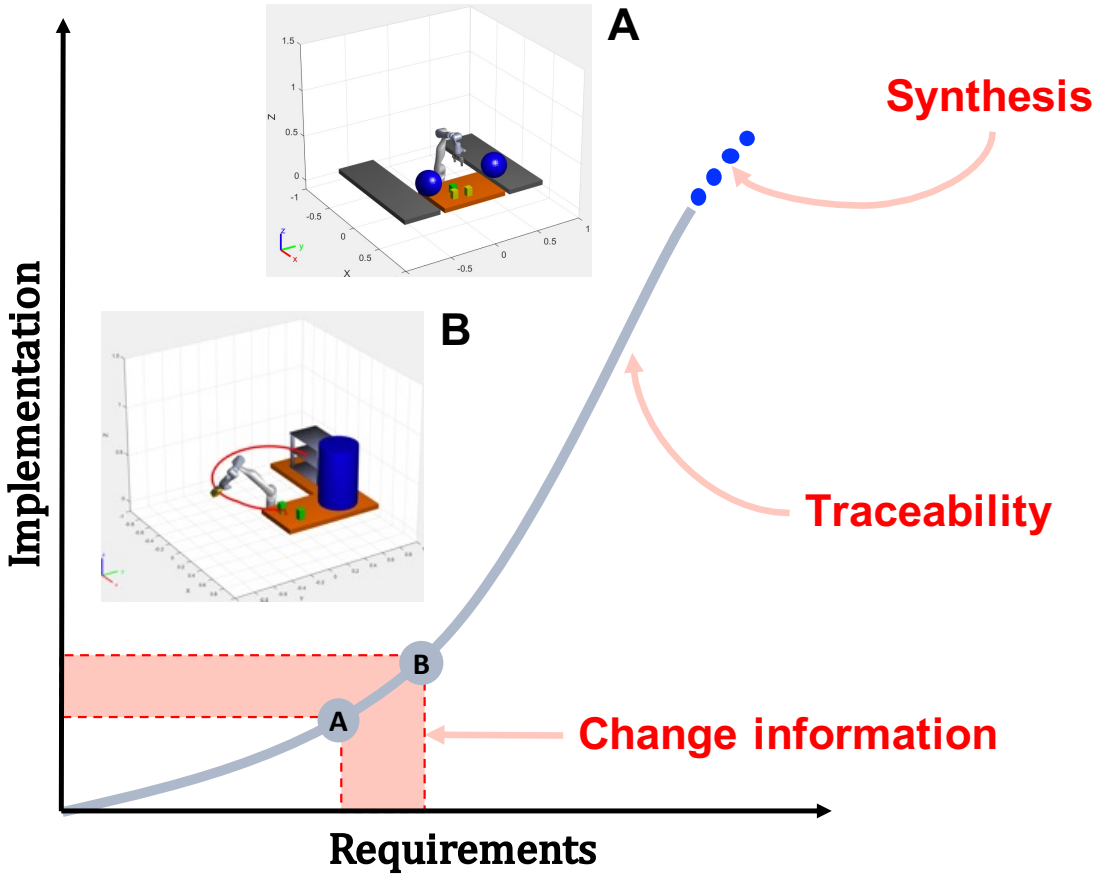
Re-use in engineering design

- *Reusability* is the ability to minimally change existing solutions to meet a change in requirements
- Engineers want their autonomous systems to be reusable when confronted with changes in requirements because there is a cost to wholesale re-engineering:
  - Labor costs, and
  - Downtime
    - Low product output (in manufacturing)
    - Missed deliveries (in supply chain)
    - Vulnerabilities (in defense)



Motivation: Tracking what parts of the solution has changed when requirements have changed can provide the information needed to synthesize new solutions from old ones

Traceability, change information, synthesis



Synthesis
Synthesis is the process by which functional architectures and their requirements are translated into implementations.

Traceability
Traceability is the ability to identify relationships between artifacts generated throughout the engineering design process.

Change information
Change information captures how artifacts have been altered according to some data model.

Autonomous System Components

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

	Knowledge (world model, skills)	**Plan (symbolic)**	**Control (programs, waypoints)**
Description	Knowledge refers to the world models such as what objects are present and where they are located, the symbolic actions the robot can accomplish, the goals, and its kinematic design.	Task and motion plans describe how the robot will achieve a goal by identifying a sequence of operations that symbolically update the state of the world.	Control refers to the low-level instructions given to the agent that tell it how to actuate.
Example semantics	ontologies, description logics, first-order predicate logic	hierarchical task nets (HTN), bi-partite directed acyclic graphs (DAGs), Markov decision processes (MDP)	finite state machines, directed graphs (control flow graphs, abstract syntax trees), petri nets
Example syntax	URDF, SDF, KNOWROB	STRIPS, PDDL plans	General purpose languages (C, C++, Python), robot controller languages (Kuka KRL, ABB RAPIDS, etc.)

How much does it **cost** to adapt to **change**?

What is the change we need to adapt to?

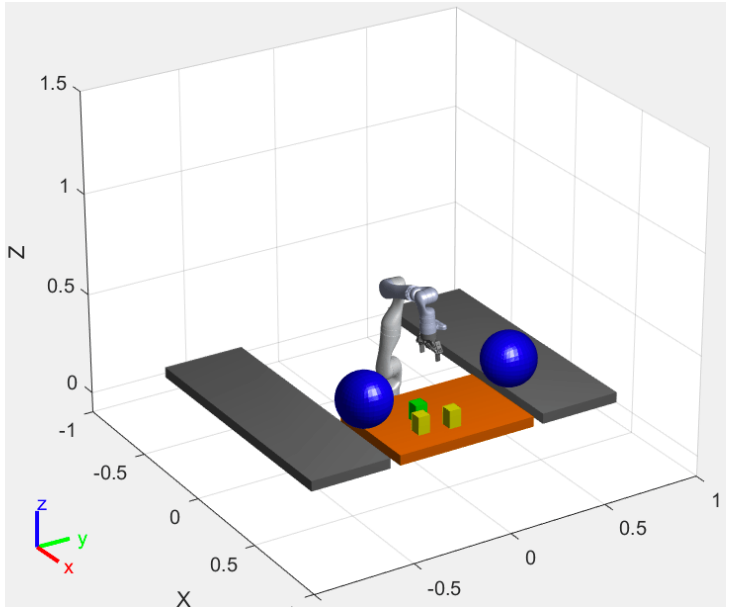
How much did it cost?

$\Delta_{\text{Knowledge}}$ = (World A + Goal) – (World A' + Goal)

Δ_{Plan} = Plan A – Plan A'

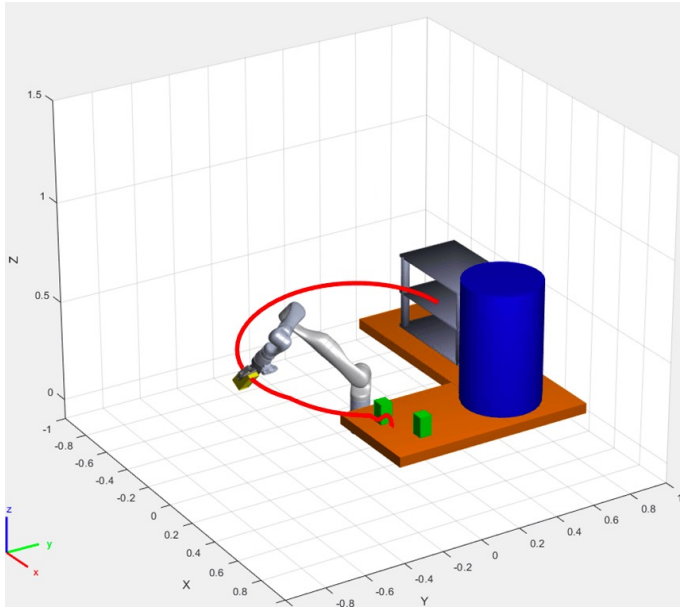
Δ_{Control} = Control A – Control A'

World A + Goal



Plan A **Control A**

Pick-and-Place Workflow Using Stateflow for MATLAB.
https://www.mathworks.com/help/robotics/ug/pick-and-place-workflow-using-stateflow.html



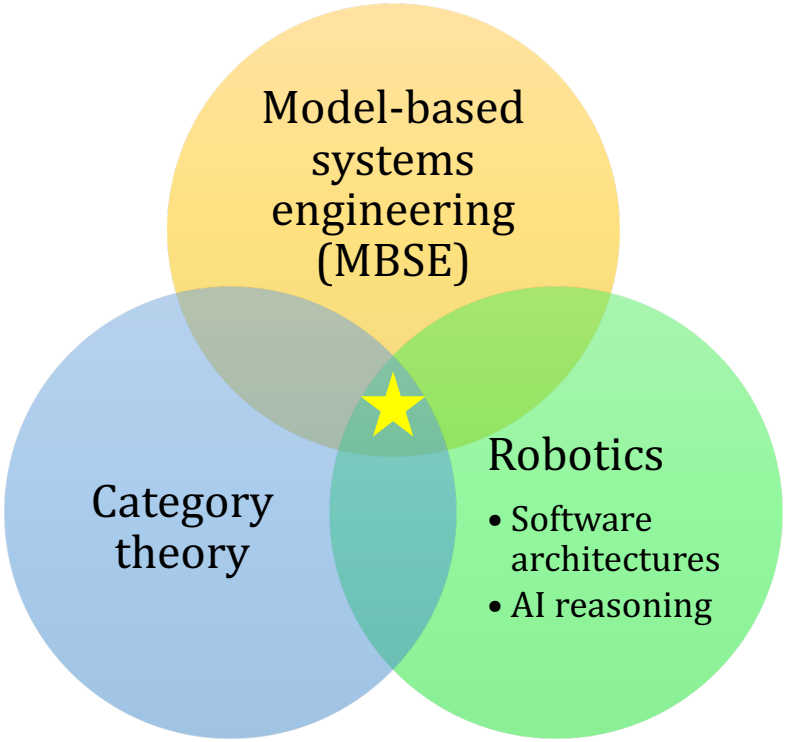
Plan A' **Control A'**

Using RRT Planner and Stateflow for MATLAB.
https://www.mathworks.com/help/robotics/ug/pick-and-place-workflow-using-stateflow-and-rrt-planner.html

World A' + Goal

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)*



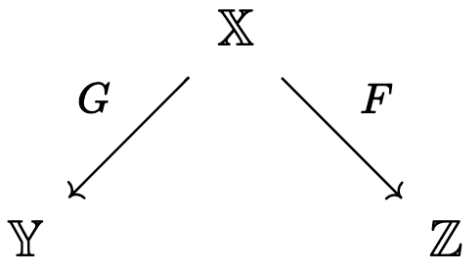
Proposed framework requirements

(a)	The ability to encode both procedural (task, motion, and control sequences) and declarative (world model) data.
(b)	The ability to track varying number of abstraction levels within knowledge, plans, and control.
(c)	The ability to encode composite (parts of a whole, decomposition, traceability) relationships and composition (merging, gluing, planning) relationships.
(d)	The ability to encode binary relations such as equivalence and similarity with order.
(e)	The ability to adhere to constraints demanded by the internal semantics of knowledge, plans, and control.

Categorical Data Model

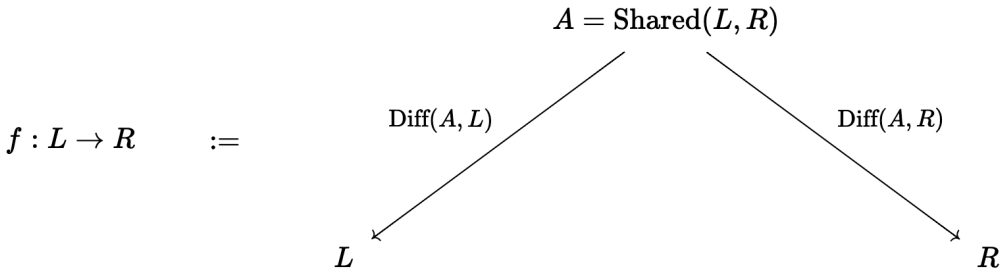
Symmetric delta lenses (Johnson 2017)
Spans in the category of small categories, **Cat**

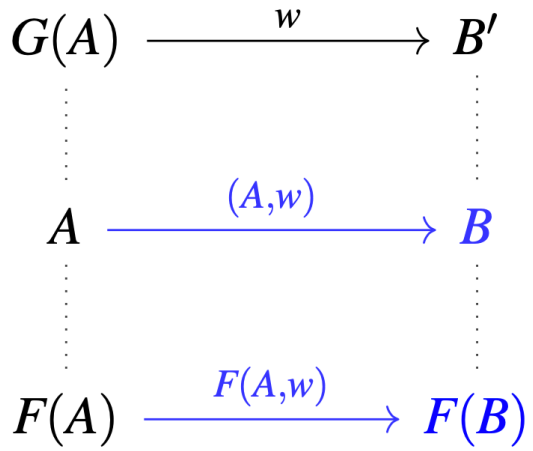
- Left leg: Discrete opfibration functor, G
- Right leg: Arbitrary functor, F

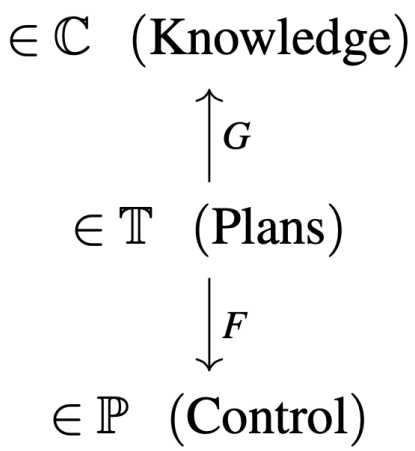


Within each category, $(\mathbb{X}, \mathbb{Y}, \mathbb{Z})$

- Objects are models
- Arrows, f, are model updates (deltas)







Modeling capabilities

- **Traceability** is defined via functors, G and F
- **Change information** is captured via the span, or delta, construction for arrows
- **Synthesis** of new implementations, namely task plans and control programs, is computed automatically using the forward and backward propagation operations

Category of Knowledge Configurations

$\mathbb{D}$ is an olog category, the syntactic category for databases, where objects are tables or types and arrows are table columns or typing arrows.

Objects

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

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

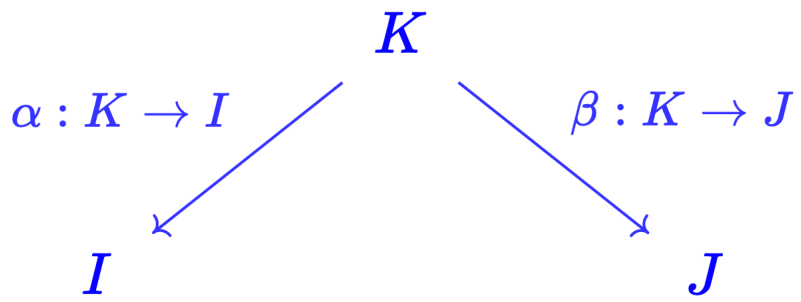
...

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

where $I, J, \ldots K \in \mathbf{D} - \mathbf{Inst}$ map to sets with the empty element

Arrows

$f : I \to J$



Category of Plans

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

Objects

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

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

...

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

where,

$X_i \in$ Set of possible predicates in the world

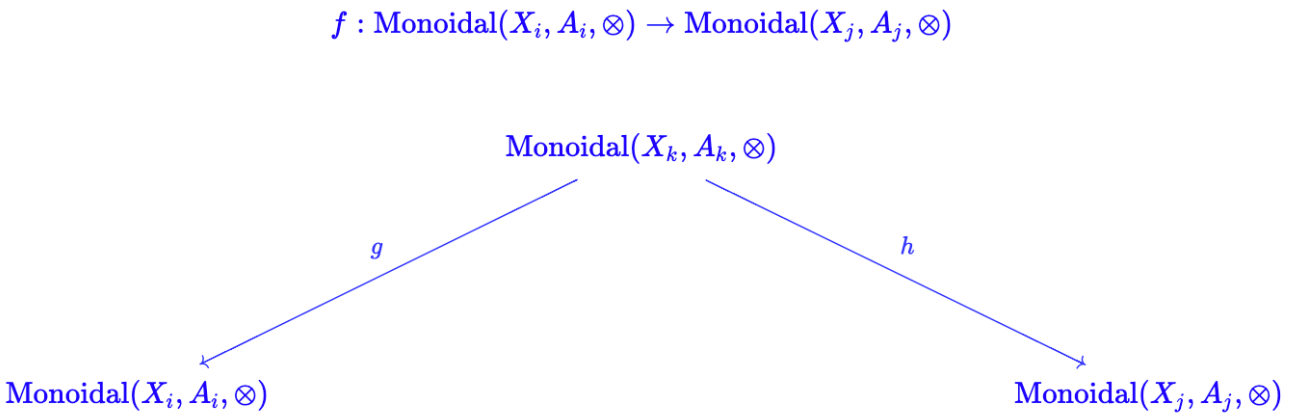
and

$A_i \in$ Set of possible actions in the world that transition states in the world

and

$\otimes$ is the conjunction of predicates and actions

Arrows



Grph: Category of labelled directed multigraphs
$\mathbb{M}$: Category of generalized sketches (metamodel)

Category of Control Programs

Given

$$L\colon \mathbf{Grph} \to \mathbb{M},$$

the category of control programs is $\int L$ ($\int$ is the Grothendieck construction)

Objects

(A_1, M)

(A_2, M)

(A_3, M)

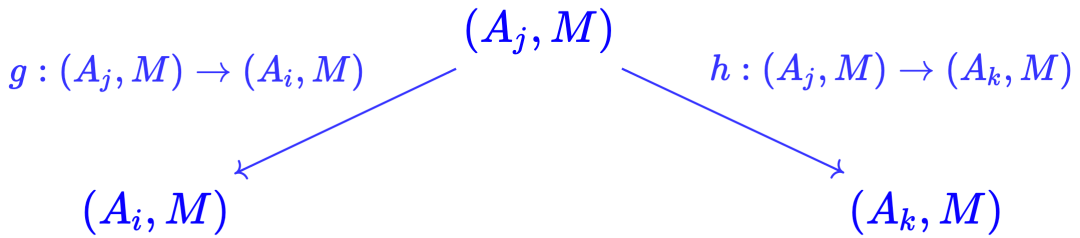
...

(A_n, M)

where $A_i \in \mathbf{Grph}$ and $M \in L(\mathbf{Grph})$

Arrows

$$f : (A_i, M) \to (A_k, M)$$



Summary

- Re-usability in engineering
- Tracking changes in implementation with respect to changes in requirements
- Autonomous system architecture components: knowledge, plans, and control
- Symmetric delta lens as a semantic framework
- Propose syntax categories based on engineering data formats

Future Work

- ❑ What **functors, G and F,** can be defined between the proposed categories?
  - Do G and F meet the requirements of symmetric delta lenses?
- ❑ Are all items in the **proposed framework requirements (a)-(e)** met in this framework?
- ❑ What **other properties** does this framework afford us?
  - Can we make a statement about whether an architecture is more re-usable than another using this framework?
- ❑ How might we **implement this framework on a computer**? What is the computational complexity of these queries?

Thank you for listening!

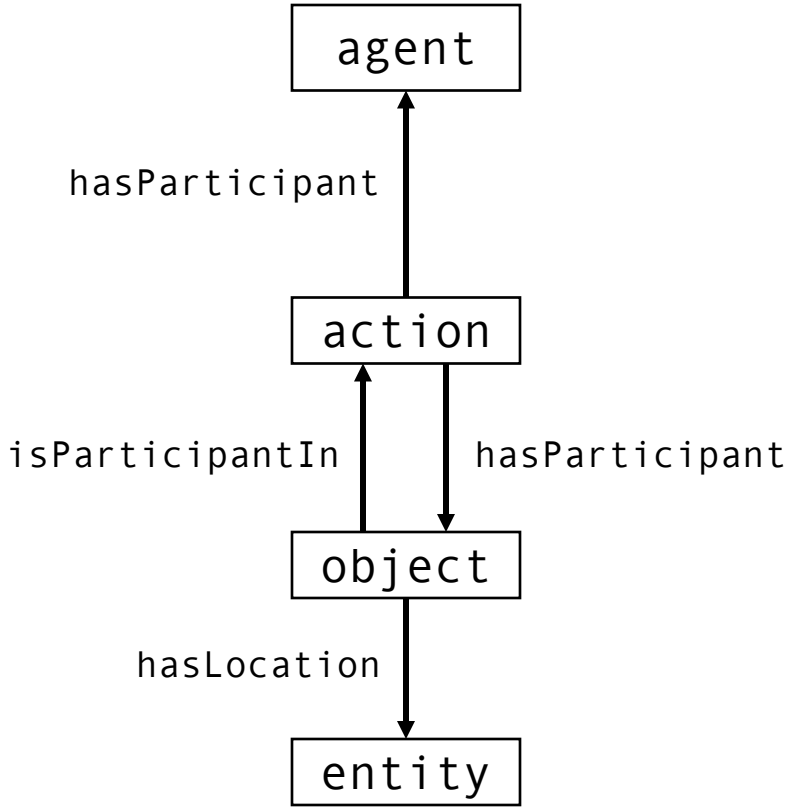
Angeline Aguinaldo

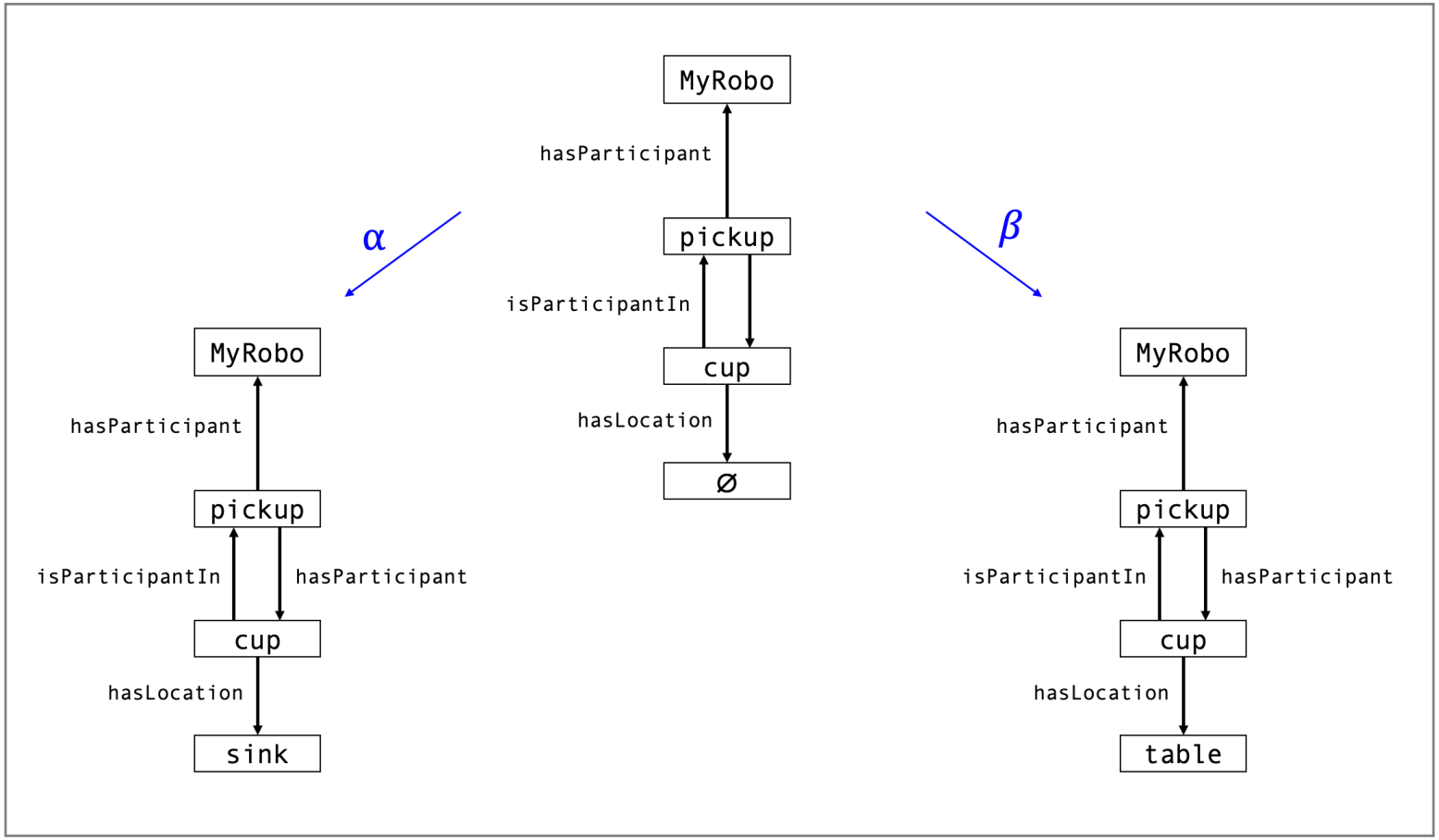
aaguinal@cs.umd.edu

Special acknowledgement to *Zinovy Diskin* for mentoring me on symmetric delta lenses and generalized sketches and their applications in model-based systems engineering.

Appendix

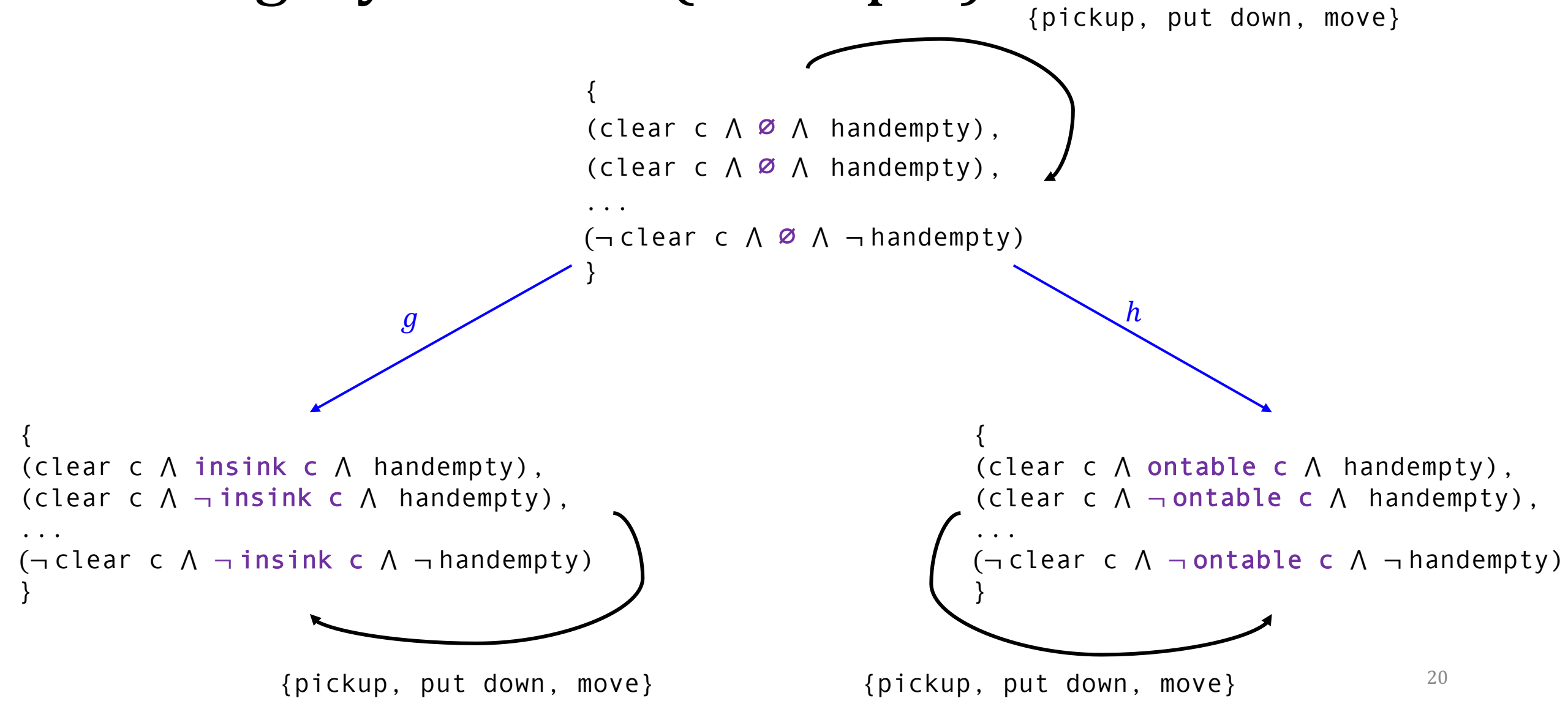
Category of Knowledge Configuration (Example)





Based on KNOWROB ontology (Tenorth 2015)

Category of Plans (Example)

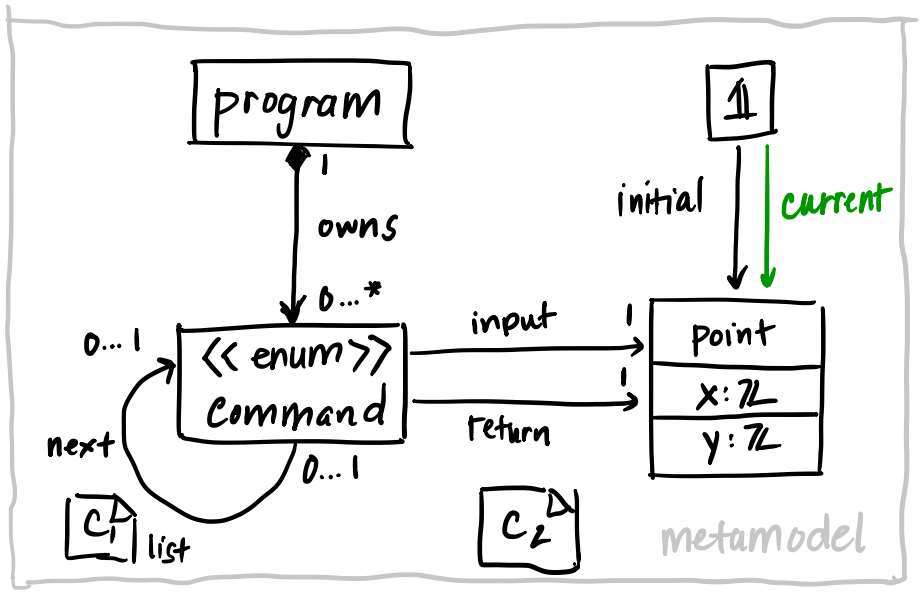


Category of Control Programs (Example)

Language metamodel (Sun 2008)

```
P := begin C end
C := left | right | up | down | C C

C(left)(x,y)     = (x-1)(y+0)
C(right)(x,y)    = (x+1)(y+0)
C(up)(x,y)       = (x+0)(y+1)
C(down)(x,y)     = (x+0)(y-1)
```



(Program instance, Language metamodel)

```
begin
    left
    down
end
```