A Graphical Model-Based Representation for PDDL Plans using Category Theory

Angeline Aguinaldo[1], William Regli[1]

[1] *Computer Science, University of Maryland, College Park*

“Why is this action in the plan?” Chakraborti 2020

PDDL Solution

```
(move-and-eat-spawn pos4-0 pos4-1 pos2-0 pos1-4)
(move-and-eat-spawn pos4-1 pos3-1 pos1-4 pos1-1)
(move pos3-1 pos2-1 pos3-0 pos4-0)
(move-and-eat-spawn pos2-1 pos2-0 pos1-1 pos0-1)
(move pos2-0 pos1-0 pos4-0 pos4-1)
(move-and-eat-spawn pos1-0 pos1-1 pos0-1 pos3-3)
(move-and-eat-spawn pos1-1 pos0-1 pos3-3 pos4-2)
(move pos0-1 pos0-2 pos4-1 pos3-1)
(move pos0-2 pos0-3 pos3-1 pos2-1)
(move-and-eat-spawn pos0-3 pos1-3 pos4-2 pos3-4)
(move-and-eat-spawn pos1-3 pos1-4 pos3-4 pos0-0)
(move-and-eat-spawn pos1-4 pos2-4 pos0-0 pos1-2)
(move-and-eat-spawn pos2-4 pos3-4 pos1-2 pos1-0)
(move-and-eat-spawn pos3-4 pos3-3 pos1-0 dummypoint)
(move pos3-3 pos3-2 pos2-1 pos2-0)
(move-and-eat-no-spawn pos3-2 pos4-2)
(move pos4-2 pos4-1 pos2-0 pos1-0)
(move pos4-1 pos3-1 pos1-0 pos1-1)
(move pos3-1 pos2-1 pos1-1 pos0-1)
(move pos2-1 pos1-1 pos0-1 pos0-2)
(move-and-eat-no-spawn pos1-1 pos1-0)
(move-and-eat-no-spawn pos1-0 pos0-0)
(move pos0-0 pos0-1 pos0-2 pos0-3)
(move pos0-1 pos0-2 pos0-3 pos1-3)
(move-and-eat-no-spawn pos0-2 pos1-2)
(move pos1-2 pos2-2 pos1-3 pos1-4)
(move pos2-2 pos2-3 pos1-4 pos2-4)
(move pos2-3 pos1-3 pos2-4 pos3-4)
(move pos1-3 pos0-3 pos3-4 pos3-3)
(move-and-eat-no-spawn pos0-3 pos0-4)%
```

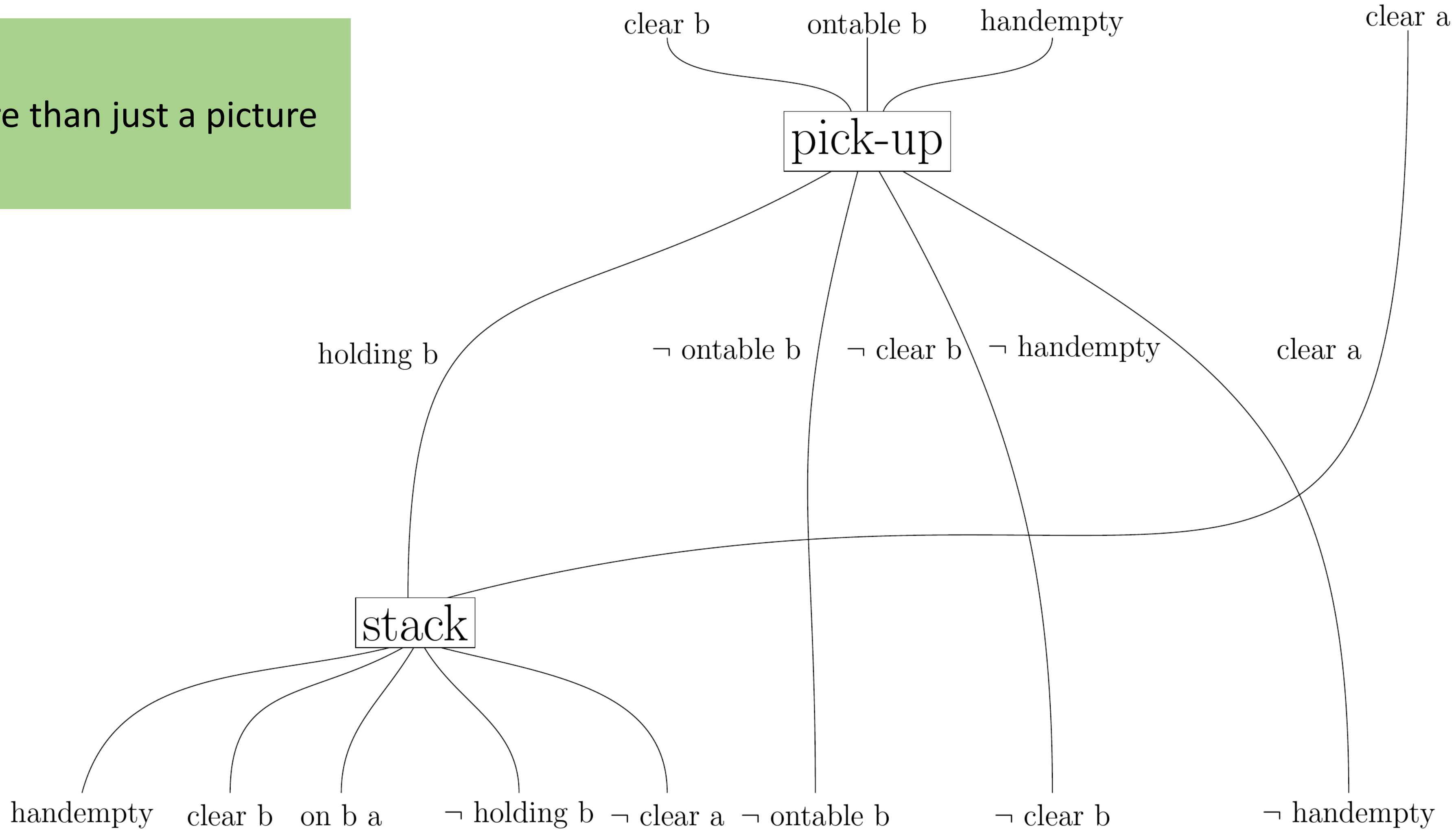

```
(:action move-and-eat-spawn
(:action move-and-eat-spawn
(:action move
    :parameters (pos3-1 pos2-1 pos3-0 pos4-0)
    :precondition
      (and
        (headsnake pos3-1)
        (isadjacent pos3-1 pos2-1)
        (tailsnake pos3-0)
        (nextsnake pos4-0 pos3-0)
        (not
          (blocked pos2-1)
        )
        (not
          (ispoint pos2-1)
        )
      )
    :effect
      (and
        (blocked pos2-1)
        (headsnake pos2-1)
        (nextsnake pos2-1 pos3-1)
        (not
          (headsnake pos3-1)
        )
        (not
          (blocked pos3-0)
        )
        (not
          (tailsnake pos3-0)
        )
        (not
          (nextsnake pos4-0 pos3-0)
        )
        (tailsnake pos4-0)
      )
  )
```

PDDL Domain and Problem Files

Model-based representation relies only on the solution, domain, and problem model. It is agnostic to the solver.

String Diagrams from Category Theory

! More than just a picture

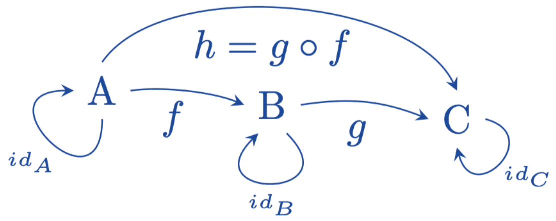


What is category theory?

Category theory is a branch of mathematics under abstract algebra that provides mathematical structures whose properties are attentive to **composition of maps**.

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 *unitors*

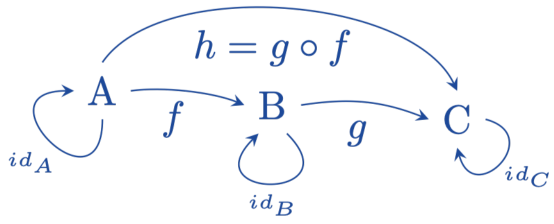


What is category theory?

Category theory is a branch of mathematics under abstract algebra that provides mathematical structures whose properties are attentive to **composition of maps**.

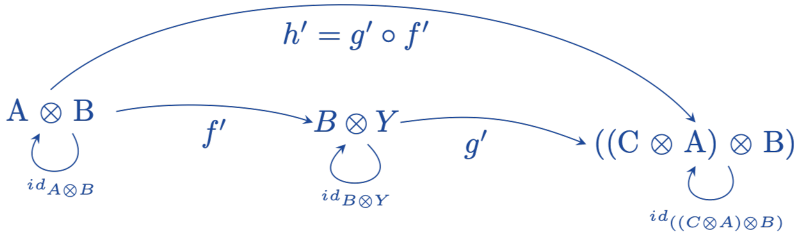
A category ($\mathbb{C}$) is:

- 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 *unitors*

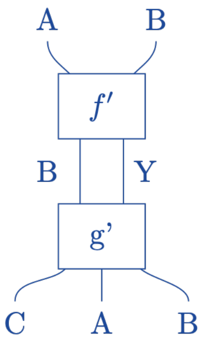


A symmetric monoidal category ($\mathbb{M}$), adds:

+ **Tensor product**, $\otimes$, which is the product of $\mathbb{M}$ (objects and morphisms) with itself that is *associative* and has *unitor isomorphisms*
+ Braiding isomorphism where $B_{\{X,Y\}}: X \otimes Y \rightarrow Y \otimes X$

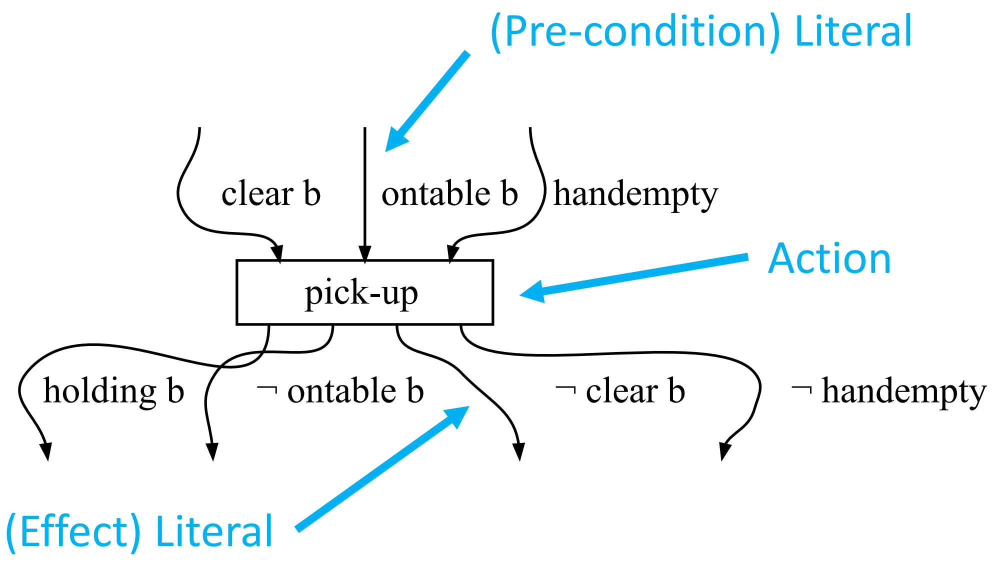


A string diagram is the graphical syntax for symmetric monoidal categories, where **boxes are morphisms** and **strings are objects**.



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

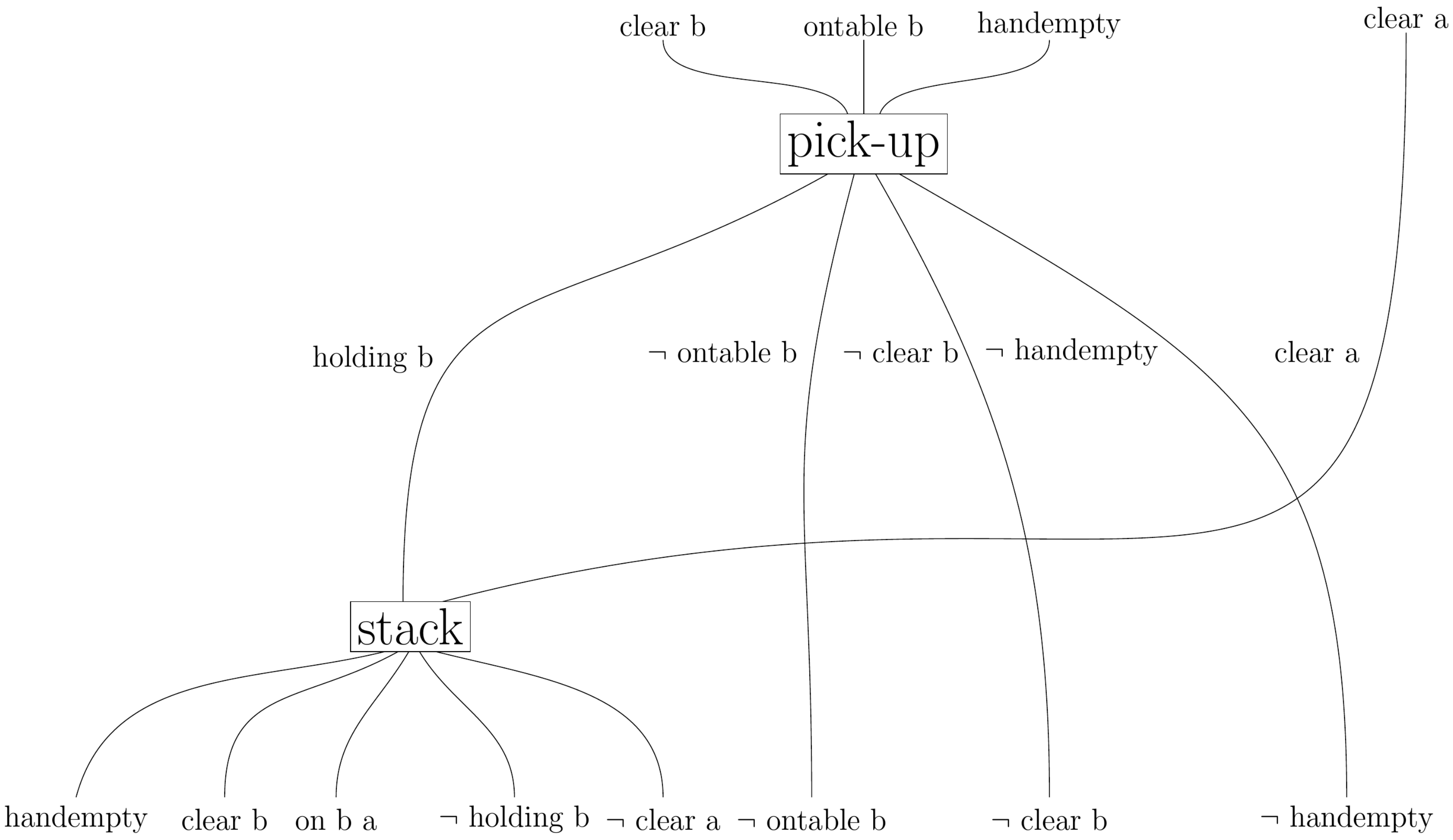
String Diagrams for PDDL



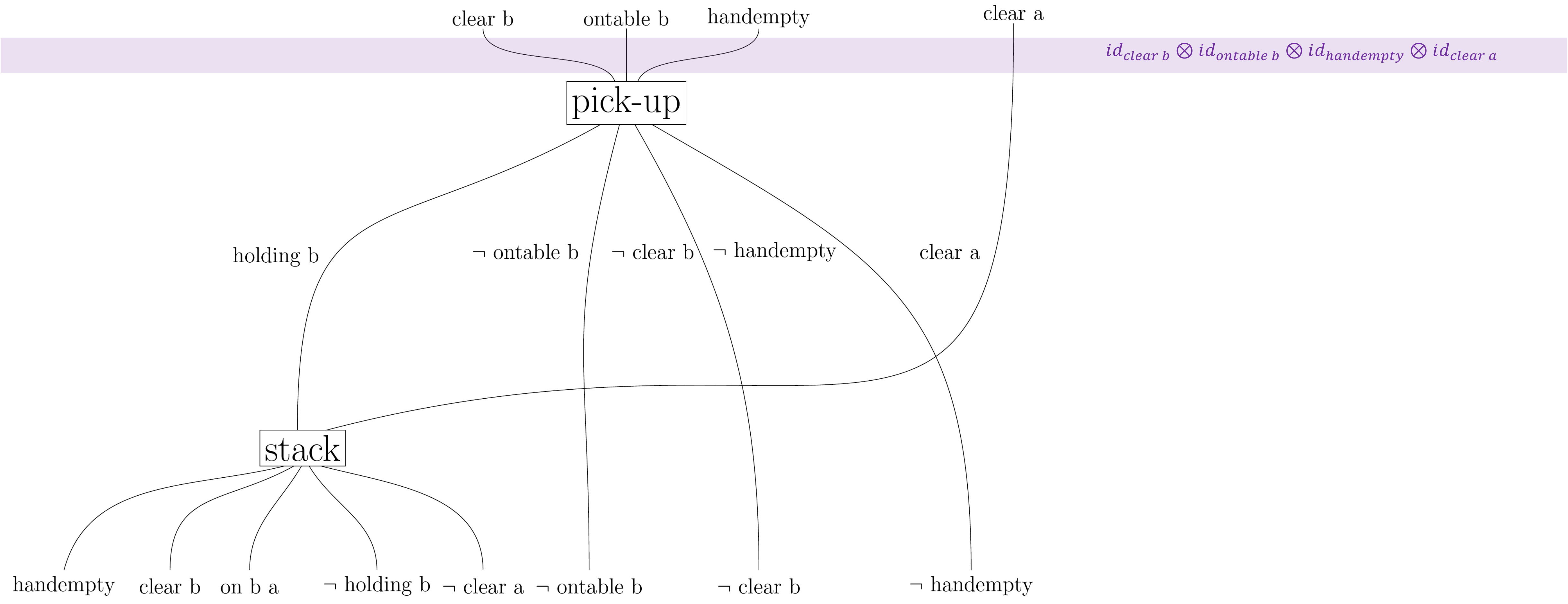
Solution

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

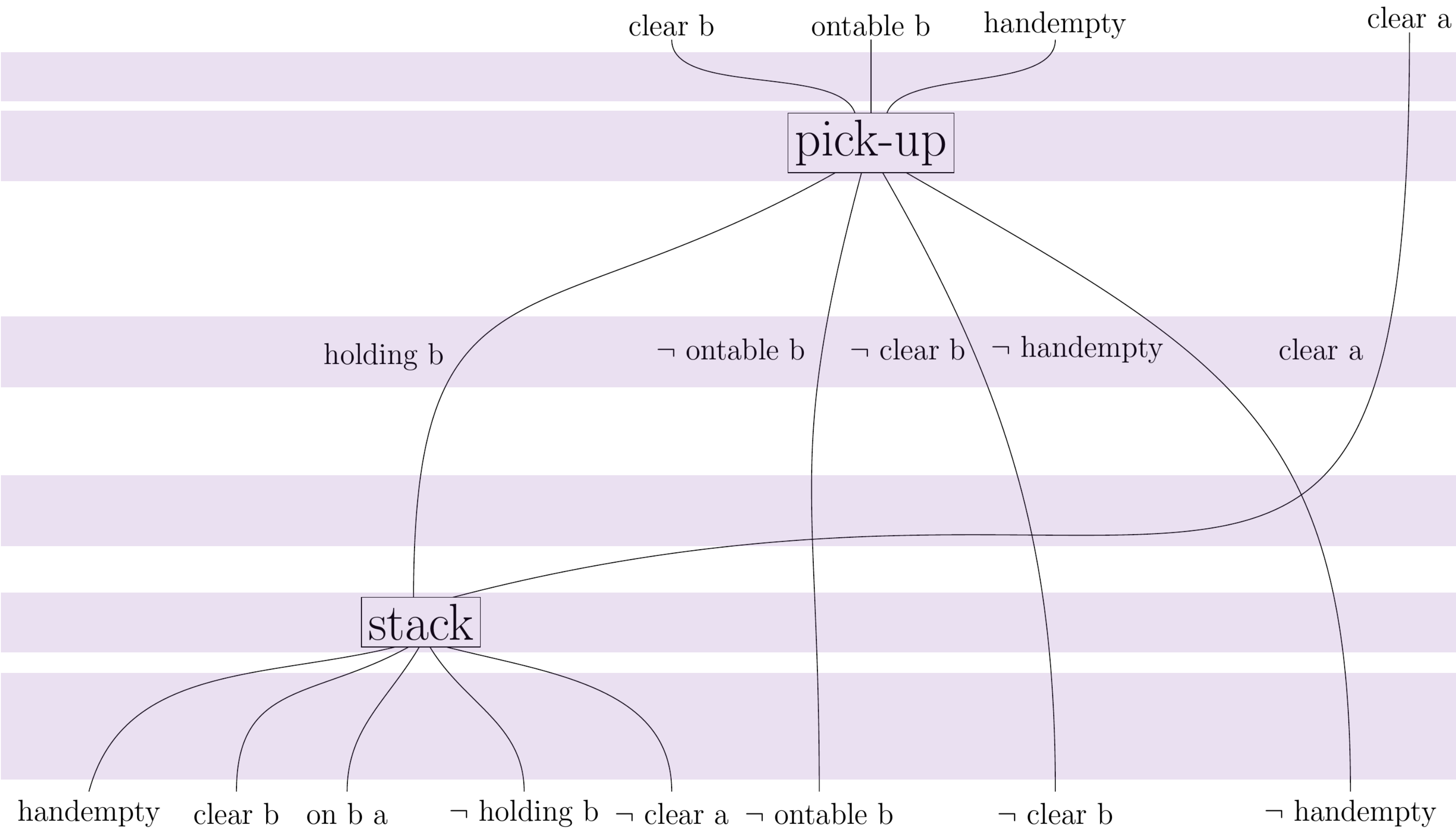
Deriving linear syntax



Deriving linear syntax



Deriving linear syntax



$id_{clear\ b} \otimes id_{ontable\ b} \otimes id_{handempty} \otimes id_{clear\ a}$

$pick\_up \otimes id_{clear\ a}$

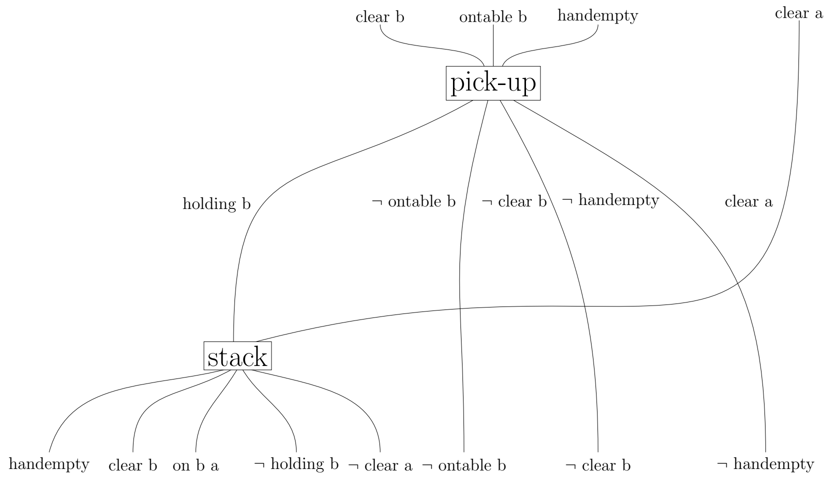
$id_{holding\ b} \otimes id_{\neg\ ontable\ b} \otimes id_{\neg clear\ b} \otimes id_{\neg handempty} \otimes id_{clear\ a}$

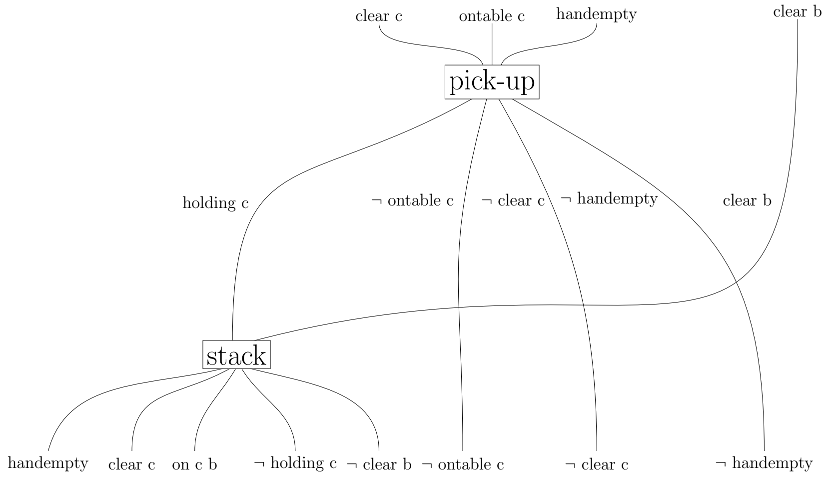
$id_{holding\ b} \otimes B_{\{\neg ontable\ b,\ \neg clear\ b,\ \neg handempty,\ clear\ a\}}$

$stack \otimes id_{\neg\ ontable\ b} \otimes id_{\neg clear\ b} \otimes id_{\neg handempty}$

$id_{handempty} \otimes id_{clear\ b} \otimes id_{on\ b\ a} \otimes id_{\neg holding\ b} \otimes id_{\neg clear\ a} \otimes id_{\neg ontable\ b} \otimes id_{\neg clear\ b} \otimes id_{\neg handempty}$

Composing actions in PDDL solutions

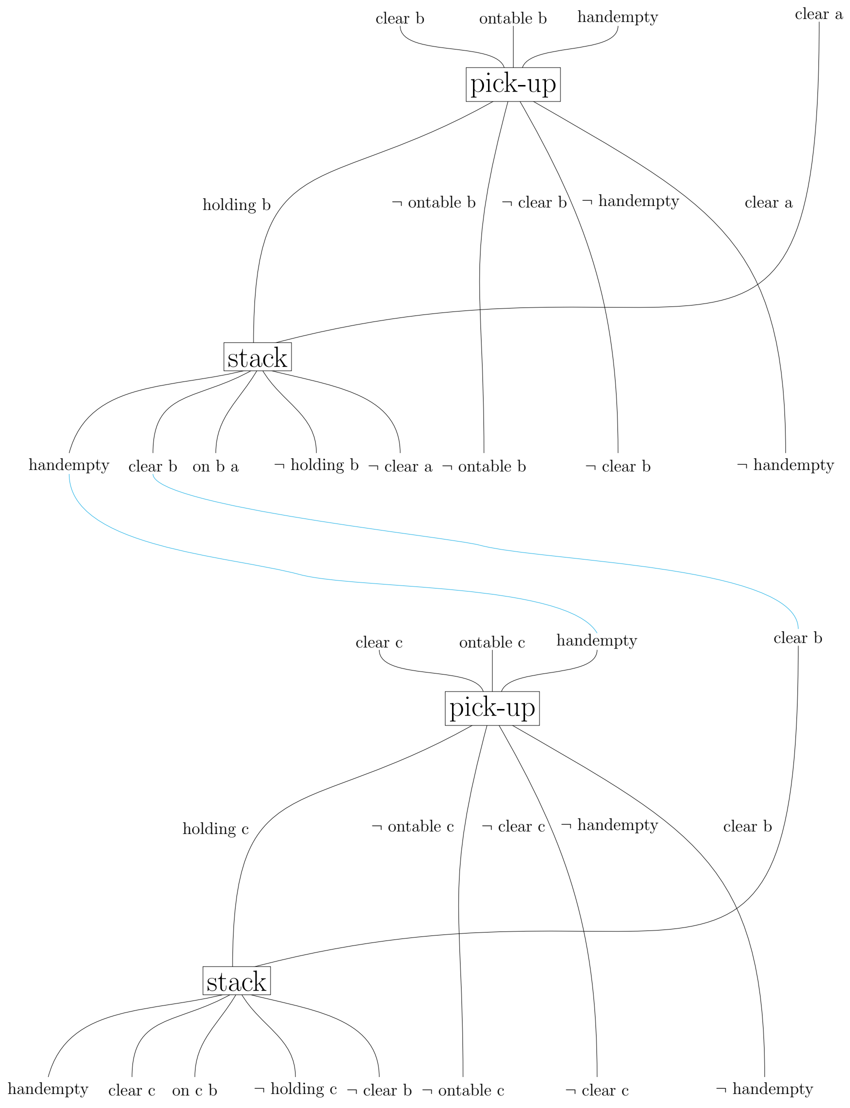




Composition Steps

1. Find common elements in domain and codomain of adjacent morphisms
2. Tensor identity morphisms of other strings

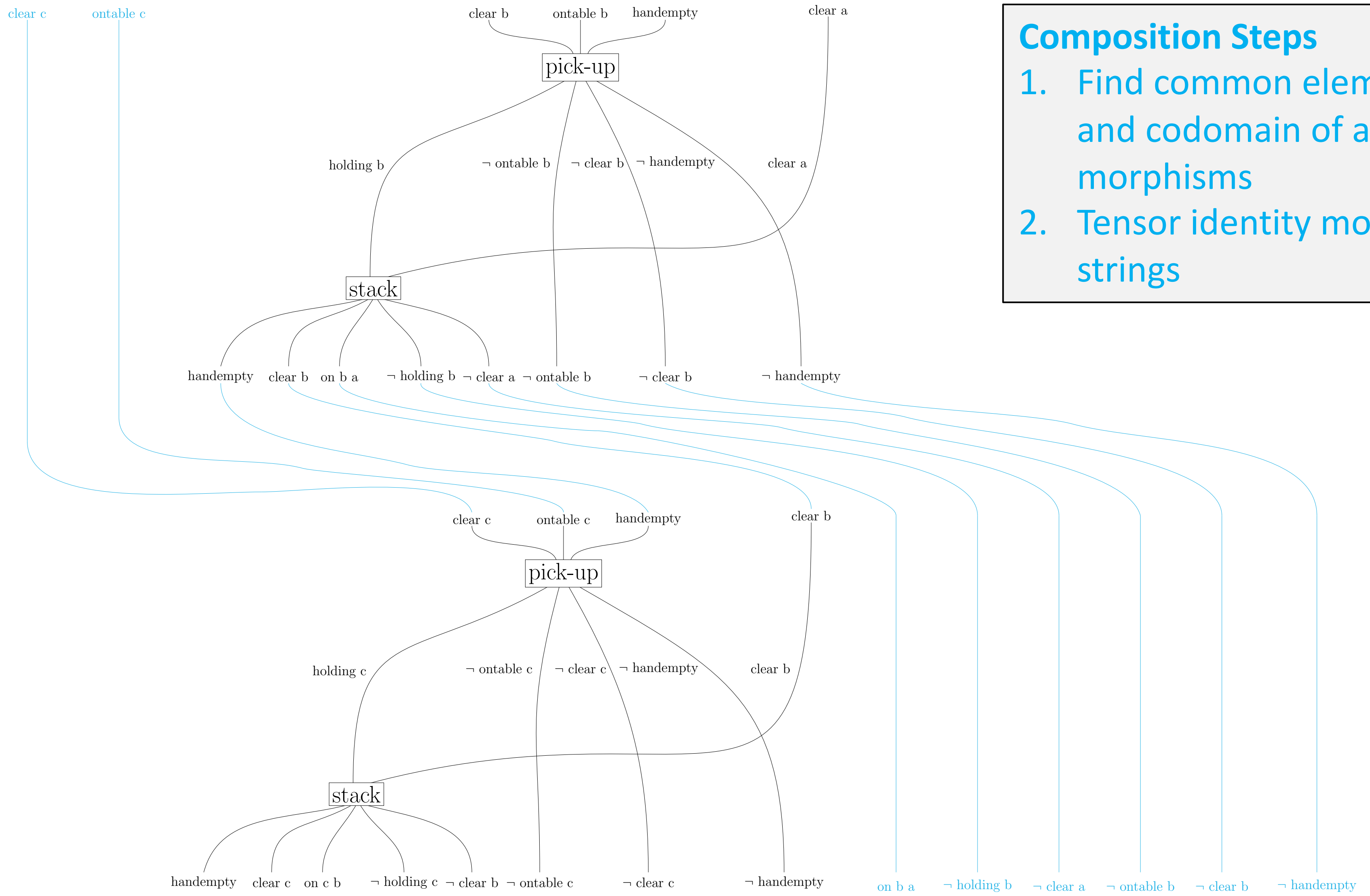
Composing actions in PDDL solutions



Composition Steps

1. Find common elements in domain and codomain of adjacent morphisms
2. Tensor identity morphisms of other strings

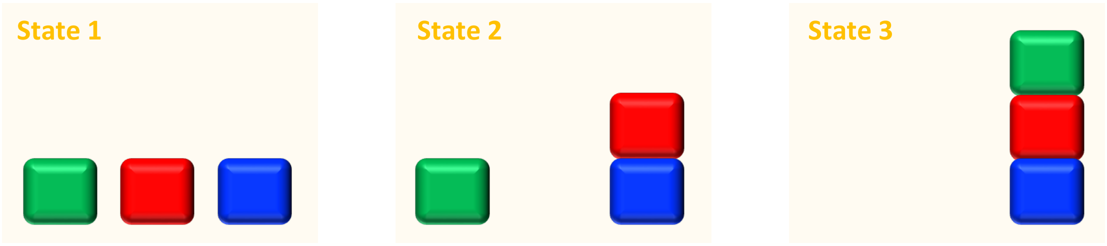
Composing actions in PDDL solutions



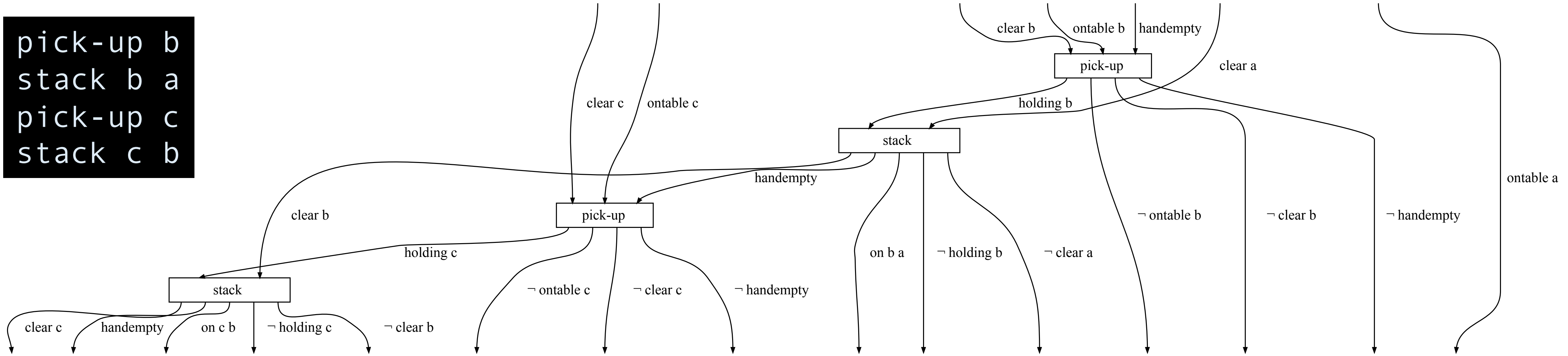
Composition Steps

1. Find common elements in domain and codomain of adjacent morphisms
2. Tensor identity morphisms of other strings

Example: Blocksworld




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



1
2
3

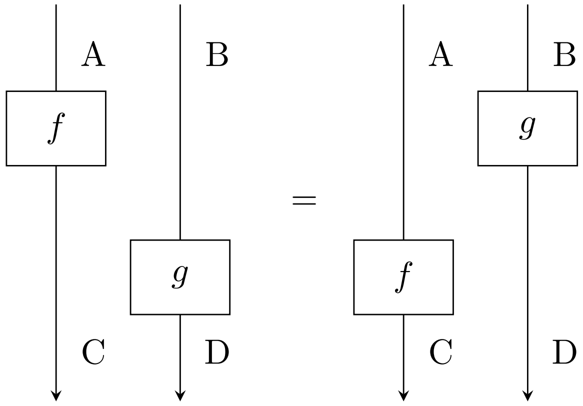
Observations

- The first *stack* relies on *(holding b)* which is an effect from *pick-up*
- The *(clear c)* and *(ontable c)* literals from the initial state do not get referenced until the second *pick-up* action
- When comparing the strings at the top to the initial state, we see there are no new assumptions introduced
- *(ontable a)* from the initial state was never used as a pre-condition to an action

Benefits and Limitations

Benefits

- Transferability of skills is a proof-by-construction
- A corresponding graphical syntax for mathematical expressions whose layout is determined by $\circ$ and $\otimes$
- Additional context for how parameters populate the predicates
- Deformation invariance property that allows you to slide boxes to find alternate plans



Limitations

- PDDL extensions supported are restricted. We have not defined string diagram encodings for quantifiers, equalities, and other extensions.
- The visualization does not scale effectively to long plans with many actions.

```
(define (domain jug-pouring)
   (:requirements :typing :fluents)
   (:types jug)
   (:functions
        (amount ?j - jug)
        (capacity ?j - jug))

   (:action pour
       :parameters (?jug1 ?jug2 - jug)
       :precondition (>= (- (capacity ?jug2) (amount ?jug2)) (amount ?jug1))
       :effect (and (assign (amount ?jug1) 0)
                    (increase (amount ?jug2) (amount ?jug1)))
)
```

Implemented using Catlab.jl from Algebraic Julia

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`.

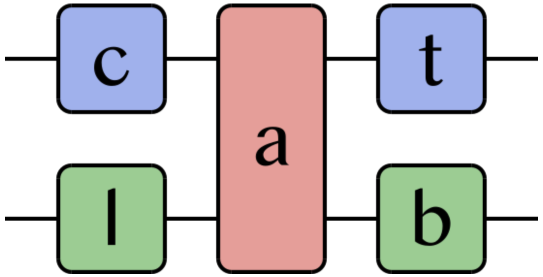
About files

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 locbox2 grippera
drop boxa locbox2 grippera
```

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



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

Next Steps

- **Functors (maps between categories)**
  - Relate composition of actions in domain specific language (e.g. PDDL) to conceptual models of plans
  - Relate symbolic plans to geometric plans

- **Visualization**
  - Sliding boxes along strings to view alternative plans.
  - Scale the length of the strings or the height of the boxes according to some solver metadata, such as cost, or a real-world parameter, such as time to execute.
  - Interactions such as highlighting the strings of a particular literal in order to witness its path through the plan

A Graphical Model-Based Representation for PDDL Plans using Category Theory

Explainable AI Planning (XAIP) Workshop – Paper 9

Thank you for listening!

Angeline Aguinaldo
Computer Science, University of Maryland, College Park
aaguinal@cs.umd.edu