Intuitive robotic programming using string diagrams

Blake S. Pollard
NIST

Ongoing work with: Angeline Aguinaldo (UMD), Spencer Breiner, and Eswaran Subrahmanian
Motivation

- An intuitive high-level robot behavior specification language
- Interoperability among multiple robot types and manufacturers
Motivation

- An intuitive high-level robot behavior specification language
- Interoperability among multiple robot types and manufacturers

Elements enabling existing approaches:
Motivation

- An intuitive high-level robot behavior specification language
- Interoperability among multiple robot types and manufacturers

Elements enabling existing approaches:

- Canonical Robot Command Language (CRCL)
- Core Ontology for Robotics and Automation (CORA)
- Robot Operating System (ROS)
Motivation

- An intuitive high-level robot behavior specification language
- Interoperability among multiple robot types and manufacturers

Elements enabling existing approaches:

- Canonical Robot Command Language (CRCL)
- Core Ontology for Robotics and Automation (CORA)
- Robot Operating System (ROS)

The role of category theory?
Motivation

- An intuitive high-level robot behavior specification language
- Interoperability among multiple robot types and manufacturers

Elements enabling existing approaches:

- Canonical Robot Command Language (CRCL)
- Core Ontology for Robotics and Automation (CORA)
- Robot Operating System (ROS)

The role of category theory?

- String diagrams for specifying robot behaviors
- Compilers to different low-level languages
- Connecting components of an architecture
One architecture

Planning Domain Description Language (PDDL)

domain
objects: things in the world
predicates: boolean-valued statements specifying state of the world
actions: state updates

problem
initial state
goal state

solution
chain of actions from initial state to goal state
Planning Domain Description Language (PDDL)

(define (domain gripper-strips)
  (:predicates (room ?r) (ball ?b) (gripper ?g) (at-robby ?r)
   (at ?b ?r) (free ?g) (carry ?o ?g))

(:action move
  :parameters (?from ?to)
  :precondition (and (room ?from)
                      (room ?to)
                      (at-robby ?from))
  :effect (and (at-robby ?to)
               (not (at-robby ?from))))

(:action pick
  :parameters (?obj ?room ?gripper)
  :precondition (and (ball ?obj)
                      (room ?room)
                      (gripper ?gripper)
                      (at ?obj ?room)
                      (at-robby ?room)
                      (free ?gripper))
  :effect (and (carry ?obj ?gripper)
               (not (at ?obj ?room))
               (not (free ?gripper))))
... and a similar ‘drop’ action.
One architecture

Problem:

(define (problem strips-gripper2)
  (:domain gripper-strips)
  (:objects rooma roomb ball1 ball2 left right)
  (:init (room rooma)
    (room roomb)
    (ball ball1)
    (ball ball2)
    (gripper left)
    (gripper right)
    (at-robby rooma)
    (free left)
    (free right)
    (at ball1 rooma)
    (at ball2 rooma))
  (:goal (at ball1 roomb)))
Initial state:
\[
\begin{align*}
&at_{-robb}(\text{roomba}) = T \\
&at(ball1, \text{roomba}) = T \\
&at(ball2, \text{roomba}) = T 
\end{align*}
\]

Goal state:
\[
\begin{align*}
&at(ball1, \text{roomb}) = T 
\end{align*}
\]

Enabled actions:
\[
\begin{align*}
pick(ball1/2, \text{rooma}, \text{left}/\text{right}) \\
pick(ball1, \text{rooma}, \text{left}) \\
move(\text{rooma}, \text{roomb}) 
\end{align*}
\]
Initial state:

\[ at\_robb\_1 (rooma) = T \]
\[ at(ball1, rooma) = T \]
\[ at(ball2, rooma) = T \]

Goal state:

\[ at(ball1, roomb) = T \]

Enabled actions:

\[ pick(ball1/2, rooma, left/right) \]
\[ pick(ball1, rooma, left) \]
\[ move(rooma, roomb) \]

But not, e.g.:

\[ pick(ball1, roomb, left) \]
Problem:

(define (problem strips-gripper2)
  (:domain gripper-strips)
  (:objects rooma roomb ball1 ball2 left right)
  (:init (room rooma)
         (room roomb)
         (ball ball1)
         (ball ball2)
         (gripper left)
         (gripper right)
         (at-robby rooma)
         (free left)
         (free right)
         (at ball1 rooma)
         (at ball2 rooma))
  (:goal (at ball1 roomb)))
Problem:

```
(define (problem strips-gripper2)
  (:domain gripper-strips)
  (:objects rooma roomb ball1 ball2 left right)
  (:init (room rooma)
    (room roomb)
    (ball ball1)
    (ball ball2)
    (gripper left)
    (gripper right)
    (at-robby rooma)
    (free left)
    (free right)
    (at ball1 rooma)
    (at ball2 rooma))
  (:goal (at ball1 roomb)))
```

Solution:

```
(pick ball1 rooma left)
(move rooma roomb)
(drop ball1 roomb left)
```
Given a PDDL domain and problem file, there is a symmetric monoidal category $\text{PDDL}$ where objects are PDDL objects and morphisms are PDDL actions.
One architecture

Canonical Robot Control Language (CRCL)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<CRCLCommandInstance
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="../xmlSchemas/CRCLCommandInstance.xsd">
    <CRCLCommand xsi:type="MoveToType">
        <CommandID>2</CommandID>
        <MoveStraight>false</MoveStraight>
        <EndPosition>
            <Point>
                <X>2.5</X> <Y>1</Y> <Z>1</Z>
            </Point>
            <XAxis>
                <I>1</I> <J>0</J> <K>0</K>
            </XAxis>
            <ZAxis>
                <I>0</I> <J>0</J> <K>-1</K>
            </ZAxis>
        </EndPosition>
    </CRCLCommand>
</CRCLCommandInstance>
```
Siemens Corporation Corporate Technology & University of Maryland, College Park - Functional Interoperable Compiler

TA8: Software Enabler: Interoperability (ARM-18-01-TA8), Advanced Robotics for Manufacturing
Hierarchical Tasking

Binding combinations of CRCL commands to PDDL actions by hand induces a functor on the syntax

\[ I: \text{PDDL} \rightarrow \text{CRCL} \]
Hierarchical Tasking

Binding combinations of CRCL commands to PDDL actions by hand induces a functor on the syntax

\[ I : \text{PDDL} \rightarrow \text{CRCL} \]

Assigning semantics leads to the following diagram:

\[
\begin{array}{ccc}
\text{PDDL} & \rightarrow & \text{Bool} \\
\downarrow & & \\
\text{CRCL} & \rightarrow & \text{Geom}
\end{array}
\]
Siemens Corporation Corporate Technology & University of Maryland, College Park - Functional Interoperable Compiler

TA8: Software Enabler: Interoperability (ARM-18-01-TA8), Advanced Robotics for Manufacturing
Closing the loop

Diagram:

Planner → Robot

World ↔ Planner
Core Ontology for Robotics and Automation (CORA)
Where are we going?

- PDDL to Catlab (and vice-versa) parser
- XSDL/OWL to CQL
- Parallelization
- Collaboration
- Gazebo testing
- Souped up string diagrams
move

move.to

move.to

=
Lemon Meringue Pie
Special thanks to Fred Proctor and the rest of the Intelligent Systems Group at NIST.

Funding for this project provided by the U.S. Department of Defense through Advanced Robotics for Manufacturing (ARM), a Manufacturing USA Institute. ARM is the nation’s leading collaborative in robotics and workforce innovation. Structured as a public-private partnership, ARM accelerates the advancement of transformative robotic technologies and education to increase U.S. global manufacturing competitiveness.

Disclaimer: Mention or use of products in the presentation does not constitute endorsement or recommendation by NIST. It is not intended to provide official guidance by NIST and NIST does not make policy.