LASRE looking for a new name

Albert Author[1] and Bernard D. Researcher[2]

Control flow:

```
Stmt -> Act | Stmt; Stmt
Stmt -> repeat n times Stmt
Stmt -> foreach point in Area Stmt
```

Actions:

```
Act -> visit Loc | pick Itm | drop Itm
Act -> select Itm
Act -> strict Act
```

Locations:

```
Area -> Area containing Itm
Area -> Area and Area | Area or Area
Area -> world // every point on the map
Area -> Pnt | [Pnt, Pnt, ..., Pnt]
Pnt -> [x, y]
```

Items:

```
Itm -> item | item Fltr
Fltr -> Fltr and Fltr | Fltr or Fltr | not Fltr
Fltr -> has color C | has shape S
C -> red | blue | green | yellow
S -> triangle | square | circle
```

Fig. 1. Syntax of core language (subset)

***Abstract*— abstract...**

I. INTRODUCTION

II. CORE LANGUAGE

The user of our system instructs a single robot which moves in a world consisting of several connected rooms. These rooms contain a number of items, which can have different properties; for simplicity we consider here different colors and shapes. The robot can modify this world by moving to a location containing an item, by picking up and dropping items, possibly at different locations.

A. Syntax

While the abstract world in which our robot operates is relatively simple, the core language used to instruct the robot is sufficiently expressive to allow for interesting scenarios. Figure 1 illustrates the syntax of the core language.

(Eva) I have based this on syntaxsemantics.md. That document does not have the select statement though. Do we have this?

*This work was not supported by any organization

[1]Albert Author is with Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, 7500 AE Enschede, The Netherlands albert.author@papercept.net

[2]Bernard D. Researcheris with the Department of Electrical Engineering, Wright State University, Dayton, OH 45435, USA b.d.researcher@ieee.org

$$[\![\texttt{item Fltr}]\!] ::= \{i \in I : Filter(i) == true\}$$
$$[\![\texttt{has color c}]\!] ::= i.color == c$$
$$\ldots$$

Fig. 2. Semantics of the core language

In addition, our language also allows to express sets of areas and conditionals which we omit here for space reasons.

With this core langauage, we can express for example the following:

- Drop an item to any field that contains both a red and a circle-shaped item (possibly the same item):

```
foreach point in {world containing item has color red} and
{world containing item has shape circle } {visit point; drop item}
```

- If possible, form a horizontal line on the floor out of all items robot currently has and starting at robot's current position:

```
strict {while robot has item {drop item; move right}};
strict {while robot has item {drop item; move left}}
```

- Keep bringing circle-shaped items to room1 until there is a red item in room1:

(Eva) I have thought to not include while in the syntax, as everything we include we need to explain which takes space. This example uses it though.

```
while not { item has color red at room1}
  {visit {world containing item has shape circle} minus room1;
pick item has shape circle; visit room1; drop item has shape circle}
```

B. Semantics

The model of our world consists of a tuple (M, I, r) where M is a two-dimensional grid, I is the set of items, and r is the robot. Each item $i \in I$ is has an associated color, shape and a unique identifier. The robot has a position in M and holds a set of items. All items are either held by the robot, or can be mapped to a position in M ($pos : i \in I \rightarrow M$). The semantics of our core language is summarized in Figure 2.

(Eva) To be continued... Not sure yet how to present this.

C. User Interface

(Eva) The way users program, i.e. with visual feedback on what their command does is probably relevant here too

[1]

III. Naturalization of the Language

The task of putting all items of different colors to different rooms looks in the core language like this

> *foreach point in world containing item has color red { visit point; pick every item has color red}; visit room1; drop every item has color red; foreach point in world containing item has color green { visit point; pick every item has color green}; visit room2; drop every item has color green; foreach point in world containing item has color blue { visit point; pick every item has color blue}; visit room3; drop every item has color blue; foreach point in world containing item has color yellow { visit point; pick every item has color yellow}; visit room4; drop every item has color yellow*

The same task could be accomplished using naturalization with

> *red to room1; green to room2; blue to room3; yellow to room4*

with a single definition of *red to room1* as

> *foreach point in world containing item has color red { visit point; pick every item has color red}; visit room1; drop every item has color red*

. If the next thing would be to put all items of different shapes to different rooms, it would again be possible to do it by

> *triangle to room1; circle to room2; square to room3*

IV. Evaluation and System Description

References

[1] S. I. Wang, S. Ginn, P. Liang, and C. D. Manning, "Naturalizing a programming language via interactive learning," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pp. 929–938, 2017.