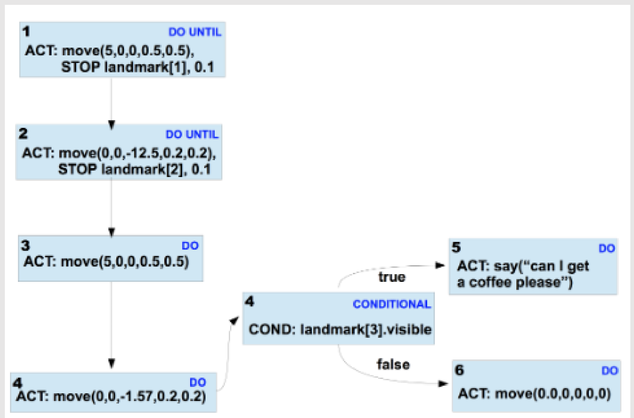
A Domain-Specific Programming Language for Robots Based on Instruction Graphs

Andrew Benson

Advisor: Jonathan Aldrich

Introduction

Instruction graphs are a recently proposed data structure that encodes a sequence of pre-determined actions for a robot to perform [1]. For example, in order for a robot to purchase coffee, a robot might move forward, looking for a barista every 5 meters, then ask for a coffee once a barista is found. The corresponding instruction graph for this group of tasks can be seen below.



Each individual task or condition is considered a vertex in the instruction graph, and the possible tasks a robot could perform after a task are its neighbors in the graph.

References

[1] Cetin Mericli, Steven Klee, Jack Paparian, and Manuela Veloso. An Interactive Approach for Situated Task Specification through Verbal Instructions. In *Proc. of AAMAS*, 2014.

The Instruction Graph Language

We contribute a formalization of instruction graphs as a domain-specific programming language. A program in this language can be viewed as a list of vertices, each with its corresponding task and its possible tasks to follow (which are edges in the instruction graph). The instruction graph program that describes the instruction graph to the left can be seen below.

```
P(V(1, do Move(5, 0, 0, 0.5, 0.5) until Stop(0.1, "landmark[1]") then 2),
  V(2, do Move(0, 0, -12.5, 0.2, 0.2) until Stop(0.1, "landmark[2]") then 3)
  ::V(3, do Move(5, 0, 0, 0.5, 0.5) then 4)
  ::V(4, do Move(0, 0, -1.57, 0.2, 0.2) then 5)
  ::V(5, if Visible("landmark[3]") then 6 else 7)
  ::V(6, do Say("Can I get a coffee please?") then 8)
  ::V(7, do Move(0, 0, 0, 0, 0) then 8)
  ::V(8, end)
  ::nil)
```

Instruction graphs can be expressed succinctly in this language. The grammar's production rules are below.

Program	p	::=	$\mathbf{P}(v,\ vs)$	programs
Vertices	vs	::=	nil	empty
			$v :: vs$	cons
Vertex	v	::=	$\mathbf{V}(n,\ c)$	vertex
Content	c	::=	**do** a **then** n	single action
			do a **until** cnd **then** n	open loop action
			if cnd **then** n **else** n	conditional
			goto n	goto
			end	termination

a and *c* are respectively actions and conditions that are specific to a particular robot. In the example above, Move and Say are actions, and Stop and Visible are conditions.

Type Safety for Instruction Graphs

We showed type safety for the instruction graph language by first defining statics and dynamics for this language, then proving statements of progress and preservation.

We define these in terms of a configuration *cfg* which represents a program midway through execution. A *cfg* is a tuple *(n, vs, I, O)* representing respectively the current task being executed, the list of vertices in the instruction graph, the input sensory data to the robot, and the output actions performed by the robot.

Theorem 1: Progress

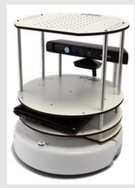
If *cfg* **cfgvalid**, then either

1. *cfg* **terminated**
2. *cfg* **waiting**
3. There exists *cfg'* such that *cfg* **->** *cfg'.*

Theorem 2: Preservation

If *cfg* **cfgvalid** and *cfg* **->** *cfg'* then *cfg'* **cfgvalid**.

Instruction Graph Interpreter



To show the feasibility of this language, we wrote an interpreter in Python for the instruction graph language. The interpreter simulates the tasks of a robot using the TurtleBot simulator, an open source mobile robot simulator.

The code for the interpreter can be found at https://github.com/anbenson/instructiongraphs.