

ROSY

An elegant language to teach the pure reactive nature of robot programming

Hugo Pacheco (FCUP & INESC TEC) and Nuno Macedo (FEUP & INESC TEC)
IRC'20

Motivation

Robotics for novice programmers

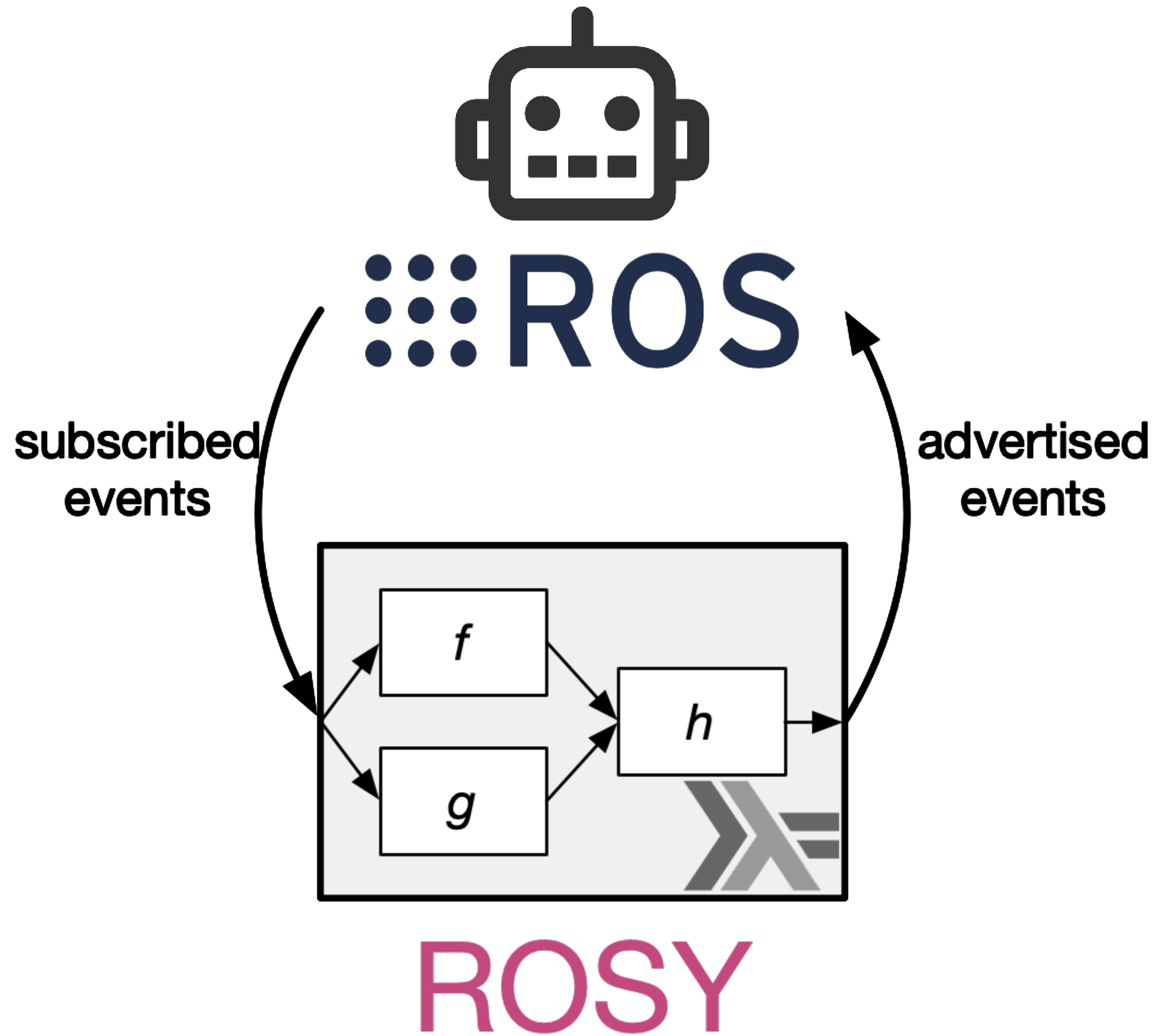
- Robotic software
 - appeals to programming novices
 - requires reasoning about relevant concepts (modularization, communication, ...)
- But this richness is accompanied by similar complexity
 - clocks, sampling rates, synchronization, ...

Our vision

Robotics for novice programmers

Introductory robot programming languages should:

- be compatible with robotic standards
- adopt a simple declarative programming style with a cause-and-effect interface
- support compositional component design
- rely on general-purpose programming languages



ROSY by example

Basic reactive control

The screenshot displays the ROSY interface, which is split into two main sections. On the left is a code editor with the following code:

```
1 accelerate :: Velocity -> Velocity
2 accelerate (Velocity vl va) = Velocity (vl+0.01) va
3
4 main = simulate accelerate
5
```

On the right is a simulation environment. At the top, it shows the robot's current state: PositionX: 0.00 m, PositionY: 0.00 m, and Orientation: 0.00 rads. The environment is a 2D grid with a grey floor and red walls. A black robot icon is positioned in the center. There are blue rectangular obstacles on the grid. At the bottom of the simulation, it shows the robot's current velocities: Linear Velocity: 0.01 m/s and Angular Velocity: 0.00 rads/s. At the bottom of the interface, there are several buttons: Guide, Report a Bug, Share, Inspect, Stop, and Run.

Pure functions reacting to and producing events, may be wrappers to ROS topics

ROSY by example

Basic proactive control

The screenshot displays the ROSY simulation interface. On the left, a code editor contains the following code:

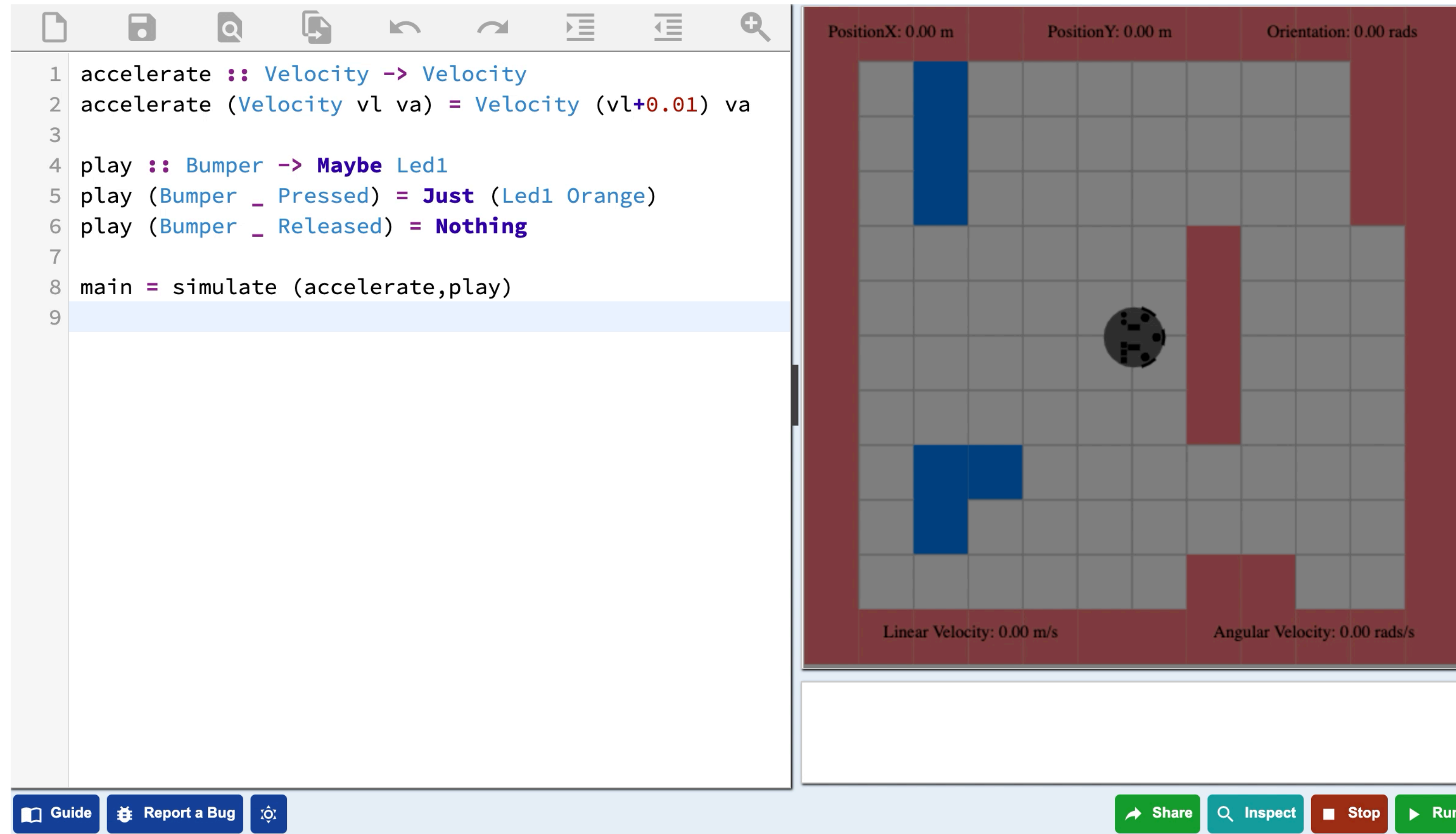
```
1 move :: Velocity
2 move = Velocity 0.1 0
3
4 main = simulate move
5
```

On the right, a 2D environment is shown with a grid. A black robot is positioned in the center. The environment features several obstacles: a blue vertical bar at the top left, a blue L-shaped bar at the bottom left, and a red vertical bar on the right. The status bar at the top right shows: PositionX: 0.00 m, PositionY: 0.00 m, Orientation: 0.00 rads. The status bar at the bottom right shows: Linear Velocity: 0.00 m/s, Angular Velocity: 0.00 rads/s. At the bottom of the interface, there are buttons for Guide, Report a Bug, Share, Inspect, Stop, and Run.

Constant functions for proactive nodes, abstract spin rate

ROSY by example

Modularity



The screenshot displays the ROSY simulation interface. On the left, a code editor shows the following code:

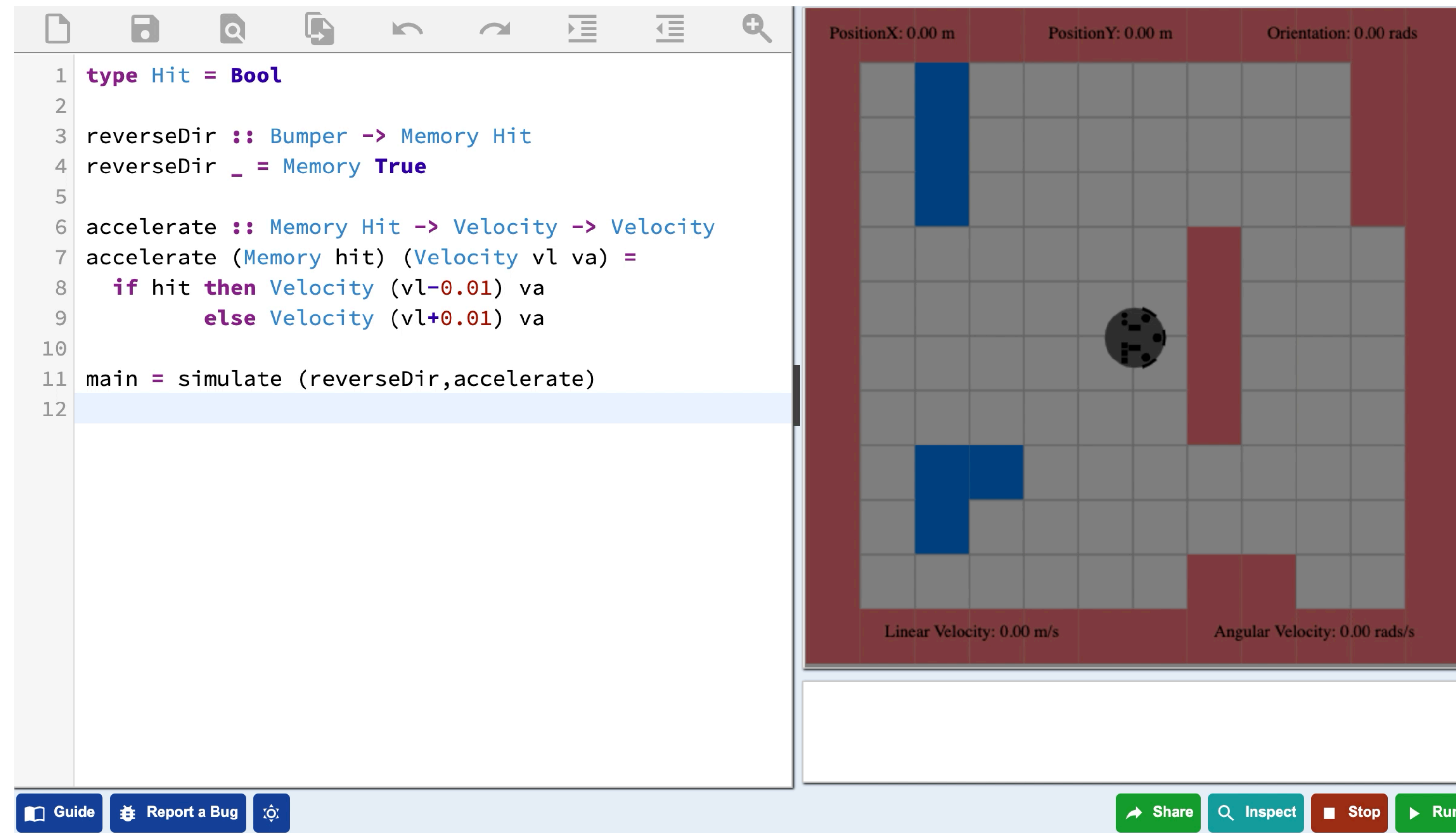
```
1 accelerate :: Velocity -> Velocity
2 accelerate (Velocity vl va) = Velocity (vl+0.01) va
3
4 play :: Bumper -> Maybe Led1
5 play (Bumper _ Pressed) = Just (Led1 Orange)
6 play (Bumper _ Released) = Nothing
7
8 main = simulate (accelerate,play)
9
```

On the right, a 2D grid world simulation is shown. The robot is a black circle with 'ROSY' written on it, positioned in the center of the grid. The grid has a grey floor and red walls. There are blue blocks representing obstacles. The status bar at the top right shows: PositionX: 0.00 m, PositionY: 0.00 m, Orientation: 0.00 rads. The status bar at the bottom right shows: Linear Velocity: 0.00 m/s, Angular Velocity: 0.00 rads/s. At the bottom of the interface, there are buttons for Guide, Report a Bug, Share, Inspect, Stop, and Run.

Compositionality achieved through multiple functions, event generation may be optional

ROSY by example

Stateful controllers



The screenshot displays the ROSY simulation interface. On the left, a code editor shows the following Haskell code:

```
1 type Hit = Bool
2
3 reverseDir :: Bumper -> Memory Hit
4 reverseDir _ = Memory True
5
6 accelerate :: Memory Hit -> Velocity -> Velocity
7 accelerate (Memory hit) (Velocity vl va) =
8   if hit then Velocity (vl-0.01) va
9     else Velocity (vl+0.01) va
10
11 main = simulate (reverseDir,accelerate)
12
```


On the right, a 2D grid world simulation is shown. A black robot is positioned in the center of the grid. The grid contains several obstacles: a vertical blue bar on the left, a horizontal blue bar at the bottom left, and a vertical red bar on the right. The robot's current state is displayed at the top: PositionX: 0.00 m, PositionY: 0.00 m, Orientation: 0.00 rads. At the bottom, the Linear Velocity is 0.00 m/s and the Angular Velocity is 0.00 rads/s. The interface includes a toolbar at the bottom with buttons for Guide, Report a Bug, Share, Inspect, Stop, and Run.

Global state explicitly modelled as events (reading and updating)

ROSY by example

A random walker

```
1 data Mode = Go | Stop | Turn Double Seconds
2 data ChgDir = ChgDir
3
4 bumper :: Bumper -> (Led1,Maybe ChgDir)
5 bumper (Bumper _ st) = case st of
6   Pressed -> (Led1 Orange,Just ChgDir)
7   Released -> (Led1 Black,Nothing)
8 cliff :: Cliff -> (Led2,Maybe ChgDir)
9 {- ... -}
10 wheel :: Wheel -> (Led1,Led2,Memory Mode)
11 {- ... -}
12
13 chgdir :: ChgDir -> StdGen -> Seconds -> Memory Mode
14 chgdir _ r now = Memory (Turn dir time) where
15   (b,r') = random r
16   (ang,_) = randomR (0,pi) r'
17   dir = if b then 1 else -1
18   time = now + doubleToSeconds (ang/vel_ang)
19
20 spin :: Memory Mode -> Seconds -> (Velocity,Memory Mode)
21 spin m@(Memory Stop) _ = (Velocity 0 0,m)
22 spin m@(Memory (Turn dir t)) now | t > now =
23   (Velocity 0 (dir*vel_ang),m)
24 spin m _ = (Velocity vel_lin 0,Memory Go)
```




The simulation interface displays a grid world with a robot (black circle) and obstacles (blue and red rectangles). The robot's current state is shown at the top: PositionX: 0.00 m, PositionY: 0.00 m, Orientation: 0.00 rads. At the bottom, the robot's velocity is shown: Linear Velocity: 0.00 m/s, Angular Velocity: 0.00 rads/s. The interface includes a code editor on the left and a control panel at the bottom with buttons for Guide, Report a Bug, Share, Inspect, Stop, and Run.

User may define events, functions may handle multiple events (implicit event merging)

ROSY by example

Multiplexing events

```
4 bumper :: Bumper -> (Led1,Maybe ChgDir)
5 {- ... -}
6 cliff :: Cliff -> (Led2,Maybe ChgDir)
7 {- ... -}
8 wheel :: Wheel -> (Led1,Led2,Memory Mode)
9 {- ... -}
10 chgdir :: ChgDir -> StdGen -> Seconds -> Memory Mode
11 {- ... -}
12 spin :: Memory Mode -> Seconds
13     -> (M2 Velocity,Memory Mode)
14 {- ... -}
15
16 safetyControl :: Either (Either Bumper Cliff) Wheel
17                -> Maybe (M1 Velocity)
18 {- ... -}
19
20 data M = Start | Ignore Seconds
21 data M1 a = M1 a
22 data M2 b = M2 b
23
24 mux :: Seconds -> Memory M
25     -> Either (M1 Velocity) (M2 Velocity)
26     -> Maybe (Velocity,Memory M)
27 mux t _ (Left (M1 a)) = Just (a,Memory (Ignore (t+0.5)))
28 mux t (Memory (Ignore s)) (Right (M2 a)) | s>t = Nothing
29 mux t _ (Right (M2 a)) = Just (a,Memory Start)
```



PositionX: 0.01 m PositionY: 0.00 m Orientation: 0.00 rads

Linear Velocity: 0.20 m/s Angular Velocity: 0.00 rads/s

Guide Report a Bug Share Inspect Stop Run

Functions may also react to alternative events (implicit events merging)

ROSY by example

Task management

```
1 type Side = Either Degrees Degrees
2
3 turn :: Side -> Task ()
4 turn s = task (startTurn s) runTurn
5
6 startTurn :: Side -> Orientation -> Memory Orientation
7 startTurn (Left a) o = Memory (o+degreesToOrientation a)
8 startTurn (Right a) o = Memory (o-degreesToOrientation a)
9
10 errTurn = 0.01
11
12 runTurn :: Memory Orientation -> Orientation
13         -> Either (Velocity) (Done ())
14 runTurn (Memory to) from = if abs d <= errTurn
15     then Right (Done ())
16     else Left (Velocity 0 (orientation d))
17     where d = normOrientation (to-from)
18
19 main = simulate (turn (Left 90))
20
```

PositionX: 0.00 m PositionY: 0.00 m Orientation: 0.00 rads

Linear Velocity: 0.00 m/s Angular Velocity: 0.00 rads/s

Guide Report a Bug Share Inspect Stop Run

Tasks are given an initial value and a function executing until “done”

ROSY by example

Task composition

```
3 type Side = Either Degrees Degrees
4
5 turn :: Side -> Task ()
6 turn s = task (startTurn s) runTurn
7
8 startTurn :: Side -> Orientation -> Memory Orientation
9 {- ... -}
10 runTurn :: Memory Orientation -> Orientation
11         -> Either (Velocity) (Done ())
12 {- ... -}
13
14 data Direction = Forward Centimeters | Backward Centimeters
15
16 move :: Direction -> Task ()
17 move d = task (startMove d) runMove
18
19 startMove :: Direction -> Orientation -> Position
20           -> Memory Position
21 {- ... -}
22 runMove :: Memory Position -> Position
23         -> Either Velocity (Done ())
24 {- ... -}
25
26 drawSquare :: Task ()
27 drawSquare = replicateM_ 4 $
28     move (Forward 32) >> turn (Left 90)
```

PositionX: 0.00 m PositionY: 0.00 m Orientation: 0.00 rads

Linear Velocity: 0.00 m/s Angular Velocity: 0.00 rads/s

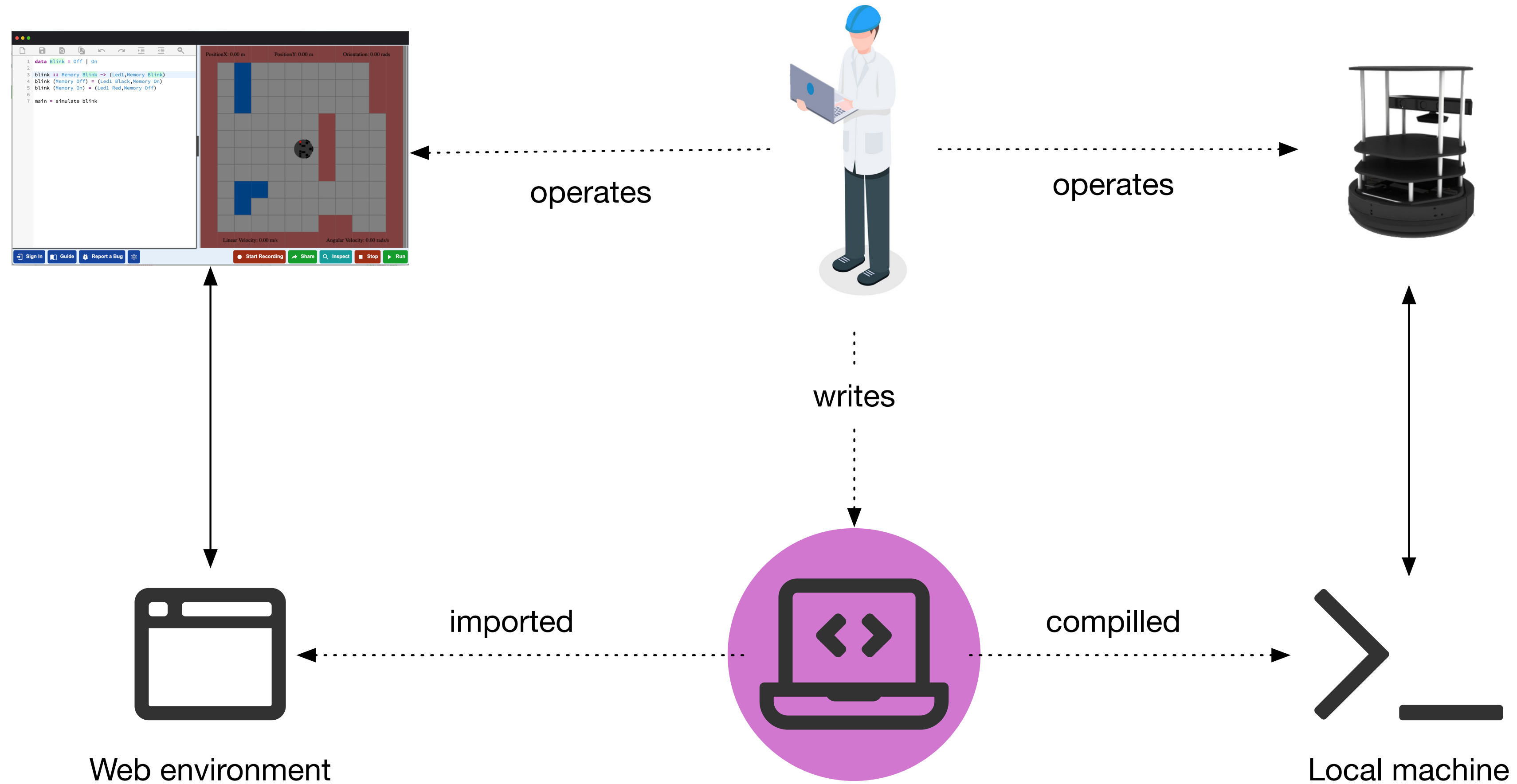
Processing request...

Share Inspect Stop Run

Tasks can be composed using Haskell's imperative notation

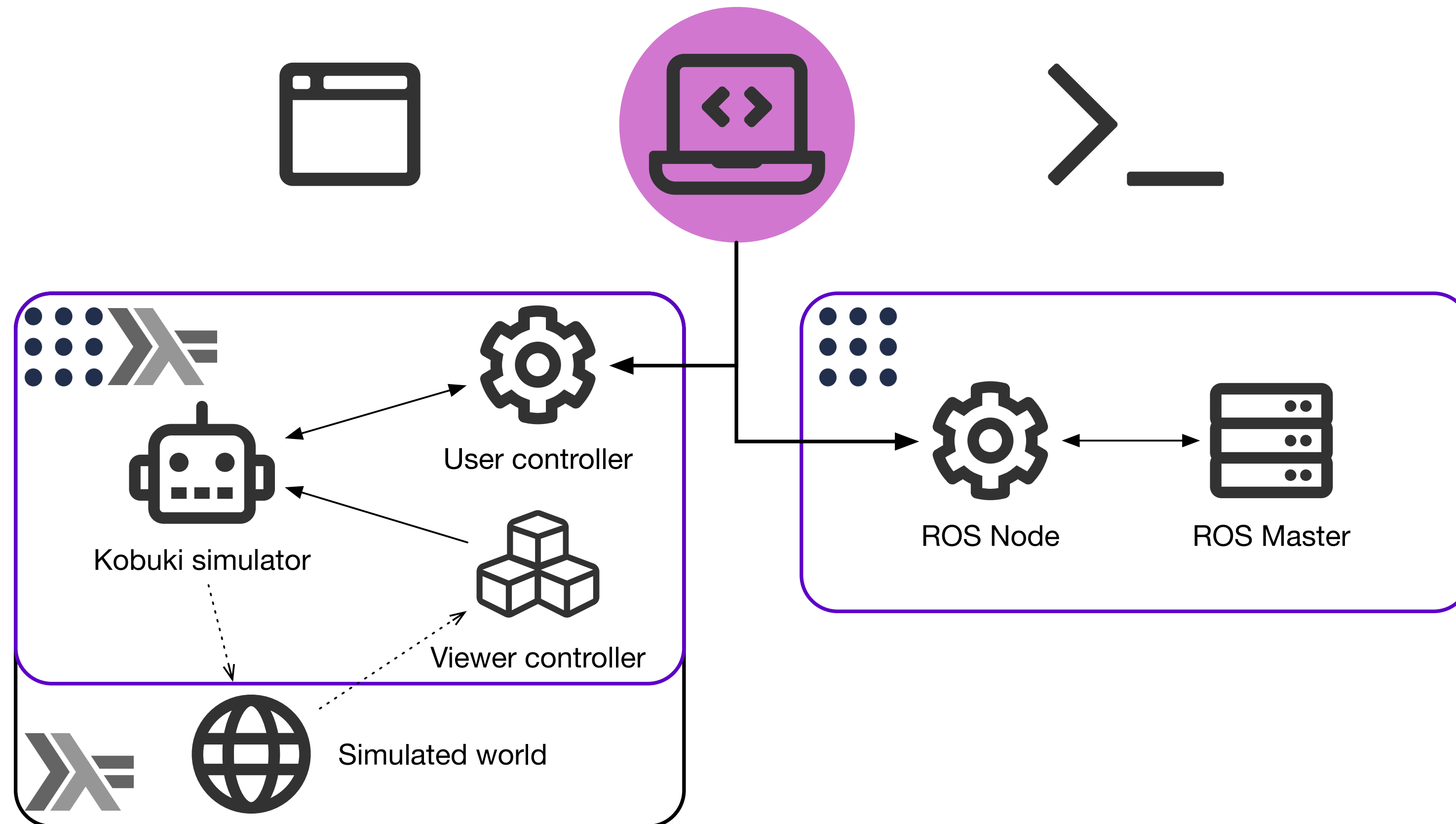
ROSY environment

Overview



ROSY environment

A peek under the hood



Final remarks

- Pedagogical robot programming language
- Sweet-spot between relevant computational concepts and simplicity
- Used in a summer camp for K12 students with no programming background
- Future work
 - Empirical evaluation
 - More advanced simulation environment
 - Explore blending with block-based languages

ROSY

An elegant language to teach the pure reactive nature of robot programming

Hugo Pacheco (FCUP & INESC TEC) and Nuno Macedo (FEUP & INESC TEC)
IRC'20