

Using the robotics language

Robotics Language Tutorial - IEEE IRC 2019

General purpose robotics language

Why?

- Quickly create and deploy a node
- Programming language independent
- Closer to mathematical notation / pseudo-code
- Deploy in ROS1, ROS2, Python, C++

Types, variables, functions

Generic

$x \in \text{Naturals} \quad \# \text{ or } \mathbb{N}$

$y \in \text{Integers} \quad \# \text{ or } \mathbb{Z}$

$z \in \text{Reals} \quad \# \text{ or } \mathbb{R}$

$w \in \text{Strings}$

$p \in \text{Booleans}$

$q \in \text{Signals}(\textit{field})$

$\text{print}(x)$

Specific

$x \in \text{RosType}(\text{'std_msgs/UInt16'})$

$y \in \text{RosType}(\text{'std_msgs/Int32'})$

$z \in \text{RosType}(\text{'geometry/Pose'})$

$z \in \text{RosType}(\text{'ROS msg'})$

$q \in \text{Signals}(\text{rosType: } \textit{field},$
 $\text{rosTopic: } \textit{topic})$

$\text{print}(x.\text{data})$

Types, variables, functions

Implicit conversion

```
q ∈ Signals(Strings, rospy: '/test')
```

```
print(q)
```

```
p ∈ Signals(rospy: 'std_msgs/strings',  
            rospy: '/test')
```

```
print(p.data)
```


Types, variables, functions

- Math inspired notation
- Return multiple parameters
- Default values

```
define reverse( $x \in \mathbb{Z}$ ,  $y \in \mathbb{Z}$ )  $\rightarrow$  ( $\mathbb{Z}$ ,  $\mathbb{Z}$ ):  
    return(y,x)
```

```
define test( $x \in \mathbb{Z} = 1$ ):  
    block(  
         $x = x + 1$   
        return(x)  
    )
```

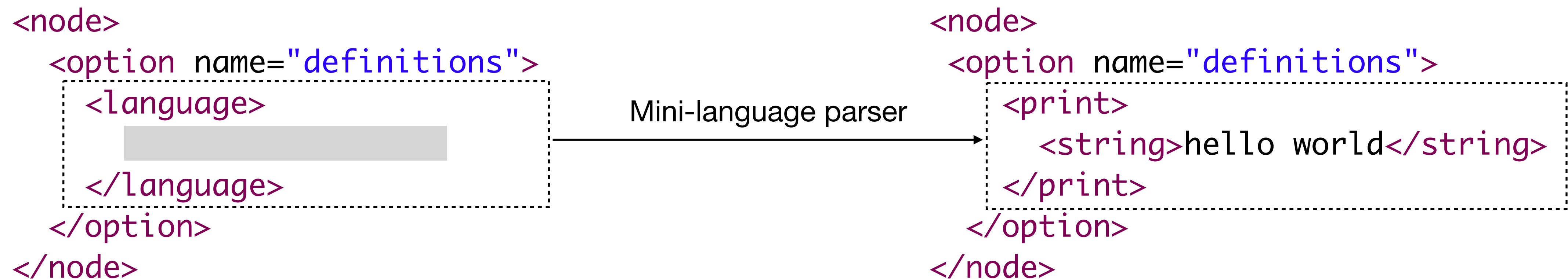
Abstraction languages

```
node(  
  definitions: language<  
      
  >  
)
```

Special bracket operator: <{ }>

Abstraction languages

```
node(  
  definitions: language{  
    Mini-abstraction language  
  }  
)
```



Abstraction languages

```
node(  
  definitions: block(  
    x ∈ Naturals,  
  
    FiniteStateMachine<  
      name: machine  
      initial: idle  
      (idle) -start-> (running) -stop-> (idle)  
    >,  
  
    machine.addInit(function)  
  )  
)
```


Abstraction languages

```
node(  
  definitions: block(  
    x ∈ Naturals,  
  
    FiniteStateMachine<{  
      idle → running  
    }>,  
  
    machine.addInit(function)  
  )  
)
```

Abstraction languages

Atom editor

language definition

inline graphics



language server

A screenshot of the Atom editor interface. The title bar shows the file name 'FiniteStateMachine.rol' and the path '~/Projects/RoboticsLanguage'. The editor displays the following code:

```
1 # A finite state machine
2 node(
3   name:"example state machine",
4
5   definitions: block(
6
7     # a mini-language: code is defined within '<{ }>'
```

The code is color-coded. Below the code, there is a toolbar with icons for view, add, run, close, and a lightning bolt. Below the toolbar, an inline state machine diagram is displayed. It consists of two blue oval nodes: 'running' at the top and 'idle' at the bottom. A curved arrow points from 'idle' to 'running', and another curved arrow points from 'running' to 'idle', forming a cycle.

Special constructs: when

if

local check

```
if( x > 1,  
    print(x)  
)
```

only checked when this
code is traversed

when

event-driven, global scope

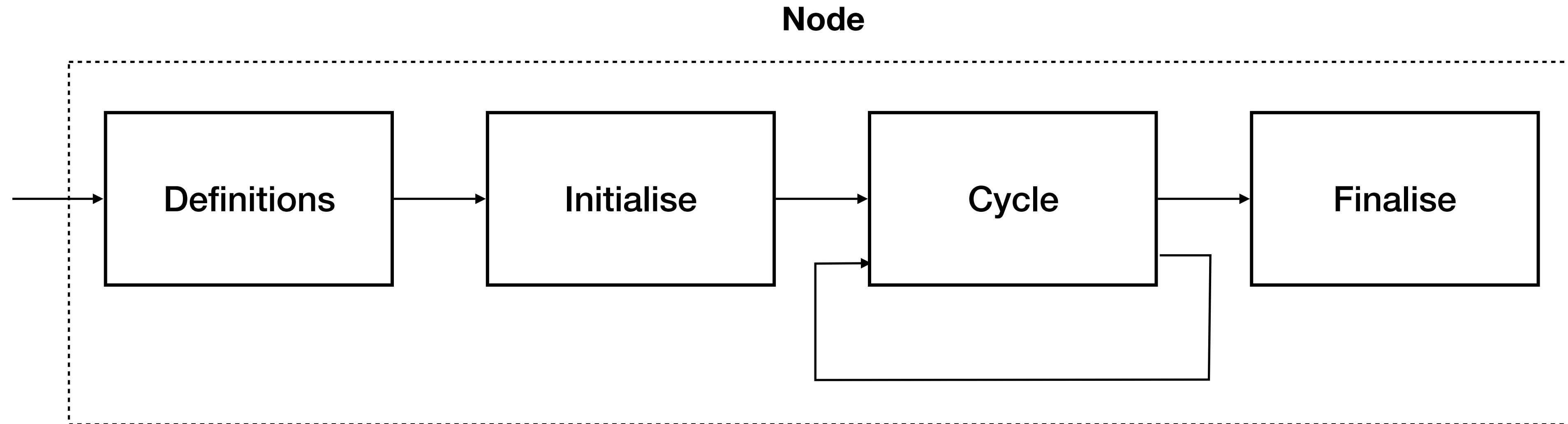
```
when( x > 1,  
      print(x)  
)
```

globally checked, injects code
into variable x assignment

Special constructs: when

```
node(  
  definitions: block(  
    x ∈ Signals(Reals),  
    when( x > 1,  
      print(x)  
    )  
  )  
  
  finalize: x = 42, ← checked  
)
```

Node lifecycle



Definitions: callbacks, event-driven behaviour

Cycle: repetitive behaviour

Node lifecycle

```
node(  
  rate:5,  
  
  definitions: x ∈ Signals(Reals, onNew: print(x)),  
  
  initialise: x = 0,  
  
  cycle:      print(x),  
  
  finalise:   x = 0  
)
```

Event-based



Synchronous



Extending the language

type checking

```
'print': {  
  'definition': {  
    'arguments': arguments('(string | real)+'),  
    'optional': {'level': optional('string', 'info')},  
    'returns': returns('none')  
  },  
  'output': {  
    'Cpp': 'std::cout << {{children|join(" << ")}} << std::endl',  
    'Python': 'print(str({{children|join(" " + str(")"))})',  
    'RoLXML': "<{{tag}}>{{children|join(' ')}}</{{tag}}>",  
    'Ros2Cpp': 'std::cout << {{children|join(" << ")}} << std::endl',  
    'RosCpp': 'ROS_INFO_STREAM({{children|join(" << ")}})',  
    'RosPy': 'rospy.loginfo(str({{children|join(" " + str(")"))})',  
  }  
}
```

code snippets for various languages

Open questions

- variable scope
- lambda calculus
- object oriented
- model checkers
- formal semantics
- compiler error handling

What is the “best” syntax for a
general purpose robotics language ?

we want your opinion!