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 ∈ Naturals    # or ℕ
y ∈ Integers    # or ℤ
z ∈ Reals       # or ℝ
w ∈ Strings
p ∈ Booleans

q ∈ Signals(field)

print(x)
```

Specific

```
x ∈ RosType('std_msgs/UInt16')
y ∈ RosType('std_msgs/Int32')
z ∈ RosType('geometry/Pose')
z ∈ RosType('ROS msg')

q ∈ Signals(rosType:field,
            rosTopic:topic)

print(x.data)
```

Types, variables, functions

Implicit conversion

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

print(q)


p ∈ Signals(rosType:'std_msgs/strings',
            rosTopic:'/test')

print(p.data)
```

Types, variables, functions

- Math inspired notation
- Return multiple parameters
- Default values

```
define reverse(x ∈ ℤ, y ∈ ℤ) -> (ℤ, ℤ):
    return(y,x)

define test(x ∈ ℤ = 1):
    block(
        x = x + 1
        return(x)
    )
```

Abstraction languages

```
node(
    definitions: language<{
```

Mini-abstraction language

```
    }>
)
```

Special bracket operator: <{ }>

Abstraction languages

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

```
<node>
  <option name="definitions">
    <language>
    </language>
  </option>
</node>
```

Mini-language parser

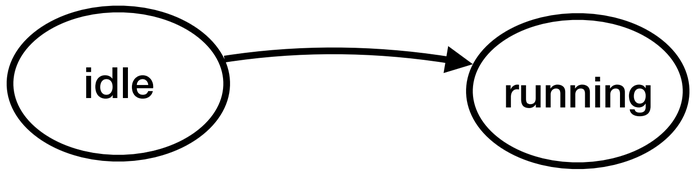
```
<node>
 <option name="definitions">
   <print>
     <string>hello world</string>
   </print>
 </option>
</node>
```

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<{
```

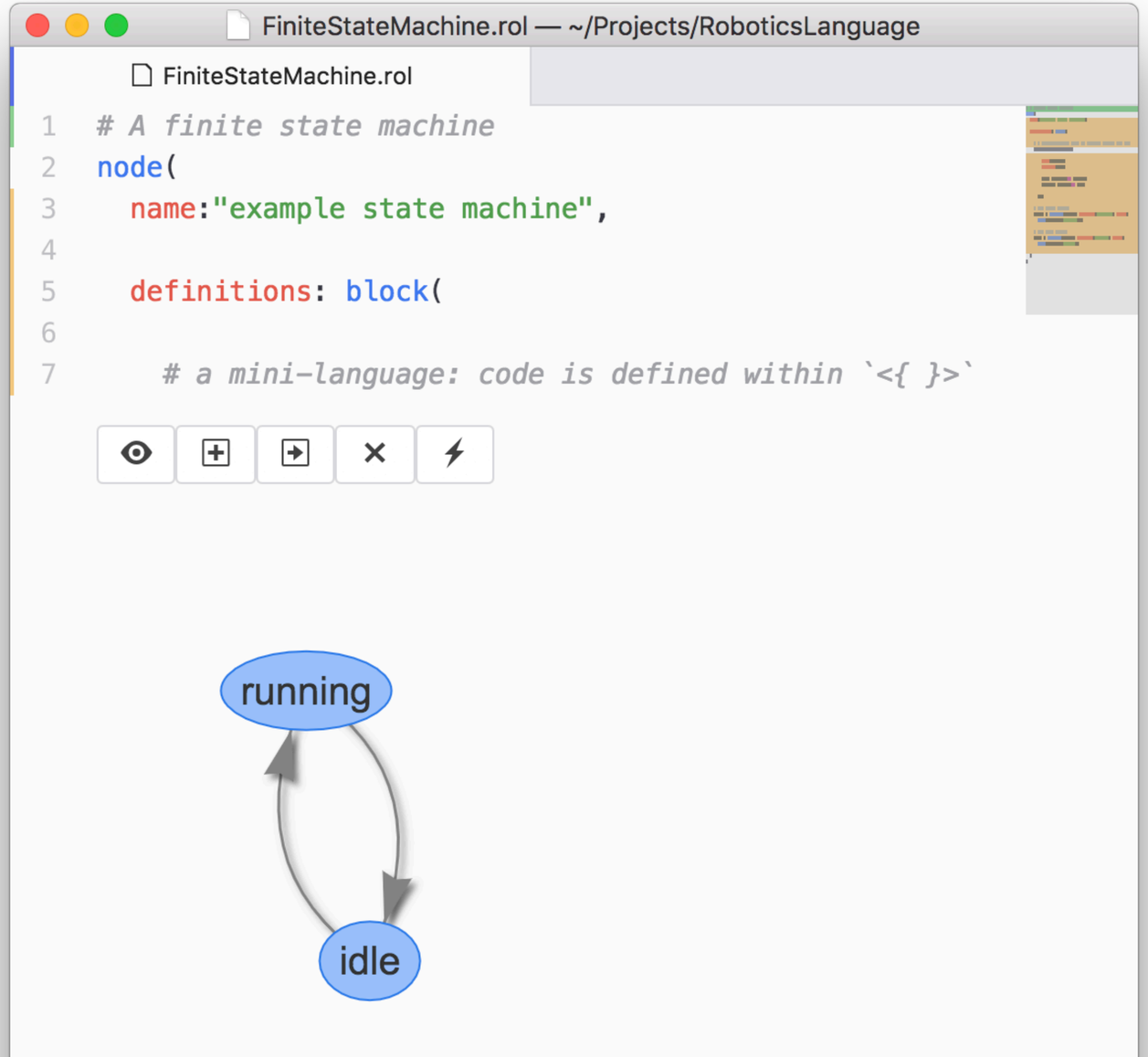



```
        }>,

        machine.addInit(function)
    )
)
```

Abstraction languages

Atom editor
language definition
inline graphics

language server



Special constructs: when

`if`

local check

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

only checked when this code is traversed

`when`

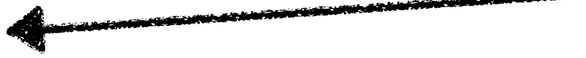
event-driven, global scope

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

globally checked, injects code into variable x assignment

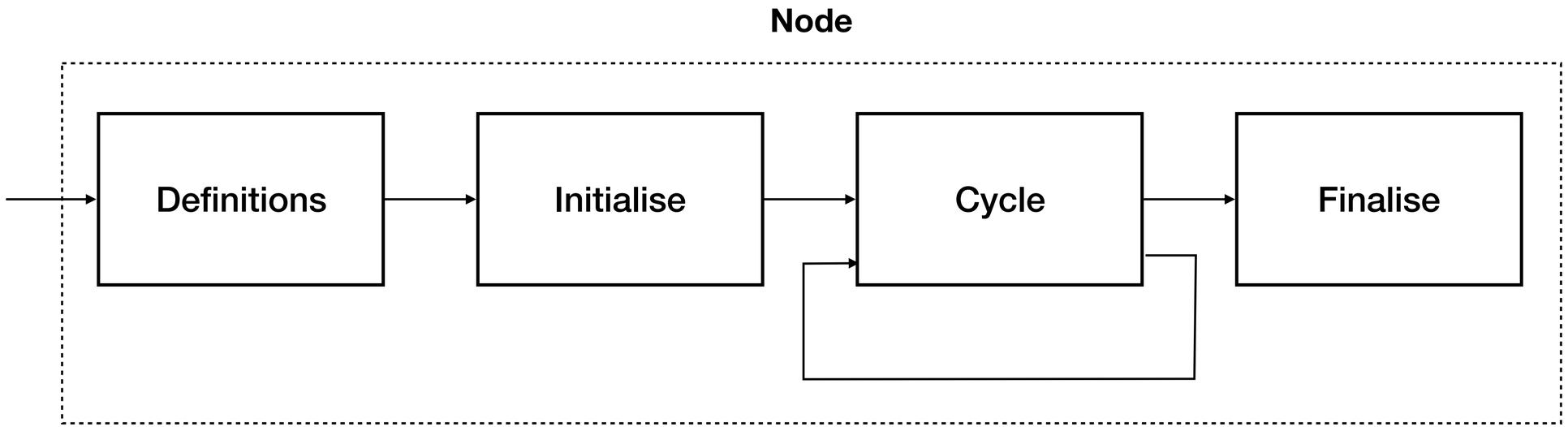
Special constructs: when

```
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!