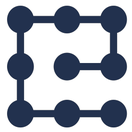


ROL

Robotics Language Tutorial

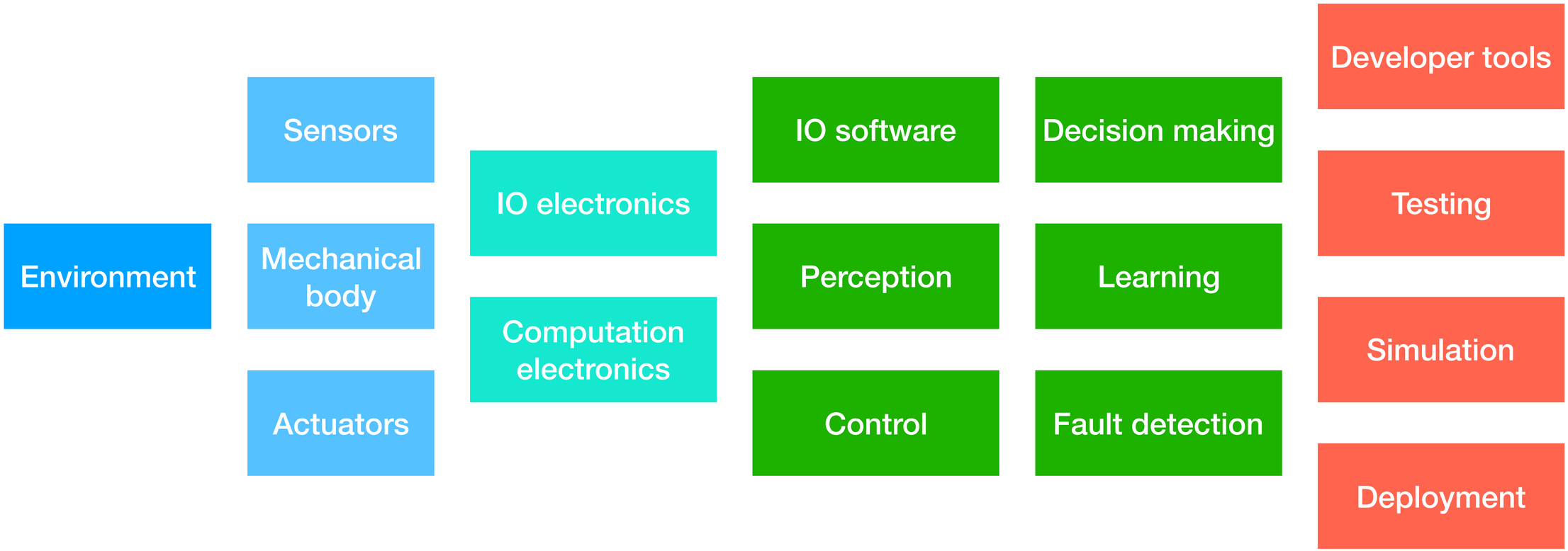
Gabriel A.D. Lopes
Floris Gaisser
Dimitrios Chronopolous

RRC Robotics, the Netherlands
g.lopes@rrcrobotics.com

IEEE IRC 2019

Robotics Language

Languages to address the complexity
of designing robotics software



Robotics Language

Languages to address the complexity
of designing robotics software

Lea robot:
- 70 sensors
- Autonomous Navigation
- User interface
- ROS

WePods:
- 8 cameras
- 6 lidar
- 3 radars
- Self driving bus
- ROS

Robotics Language

Languages to address the complexity
of designing robotics software

Robotics Language

Domain Specific Languages are computer languages specialised to a particular application domain.

Abstraction languages are programming languages with the purpose of focusing on the required information to solve a problem. They “transcompile” into standard programming languages.

Robotics Language

What is this tutorial about?

1. A general purpose programming language for robotics
   Robotics Language

2. A general purpose compiler
   rol

Robotics Language

Abstractions
Automation
Simplicity
Open

Problems in Robotics development

1. Big leap from behaviour to code
2. Reinventing the wheel
3. Developer tools

1. Big leap

Conceptual idea in **estimation**

"If observation is good, **update** estimate.
If observation is less good, **predict**."

Abstract mathematical realisation (Bayesian estimation)

$$p(x_k|z_k) = \sigma p(\underbrace{z_k|x_k}) \int p(\underbrace{x_k|x_{k-1}}) p(x_{k-1}|z_{k-1}) dx_{k-1}$$

update Predict

1. Big leap

Numerical realisation (Kalman filter)

predict

$$\begin{cases} \hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k \mathbf{u}_k \\ \mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^\mathsf{T} + \mathbf{Q}_k \end{cases}$$

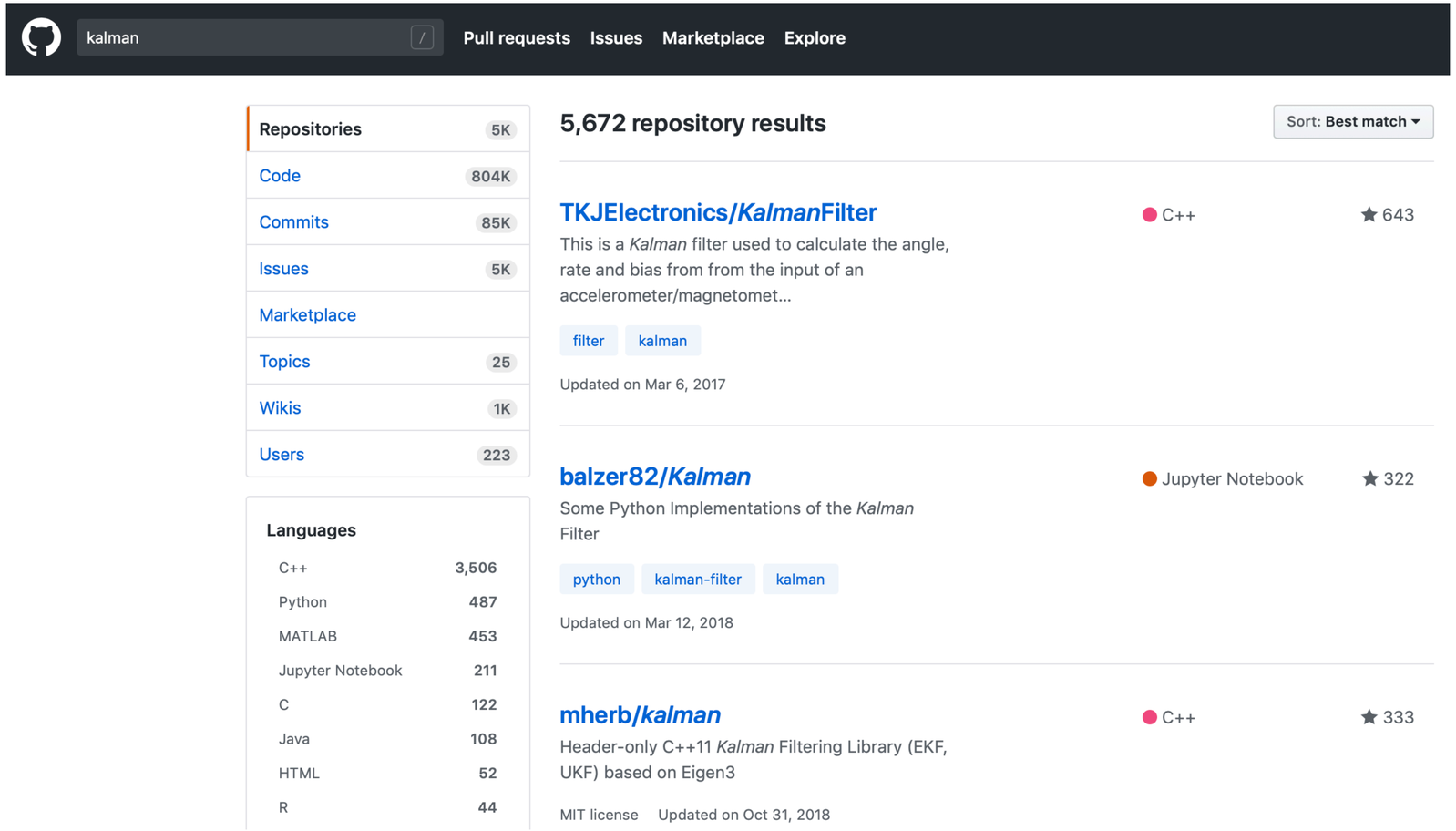
$$\begin{aligned} \tilde{\mathbf{y}}_k &= \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1} \\ \mathbf{S}_k &= \mathbf{R}_k + \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^\mathsf{T} \\ \mathbf{K}_k &= \mathbf{P}_{k|k-1} \mathbf{H}_k^\mathsf{T} \mathbf{S}_k^{-1} \end{aligned}$$

update

$$\begin{cases} \hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \\ \mathbf{P}_{k|k} = \left(\mathbf{I} - \mathbf{K}_k \mathbf{H}_k\right) \mathbf{P}_{k|k-1} \left(\mathbf{I} - \mathbf{K}_k \mathbf{H}_k\right)^\mathsf{T} + \mathbf{K}_k \mathbf{R}_k \mathbf{K}_k^\mathsf{T} \end{cases}$$

1. Big leap

Computer implementation

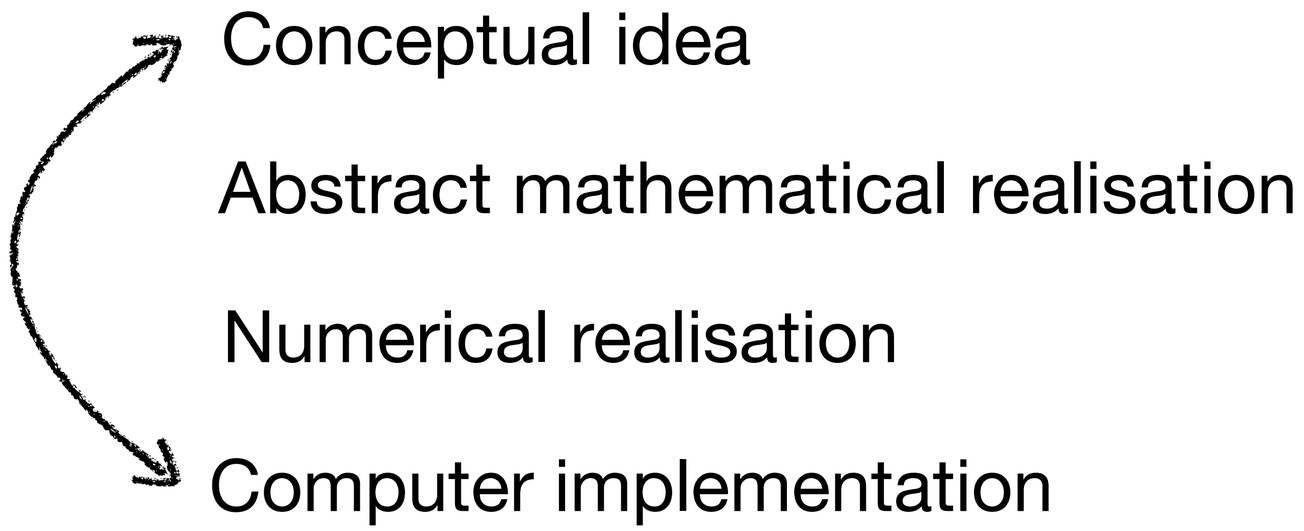


1. Big leap

Computer implementation

```
namespace Kalman {

    /**
     * @brief Abstract base class for all Kalman Filters
     *
     * @param StateType The vector-type of the system state (usually some type derived from Kalman::Vector)
     */
    template<class StateType>
    class KalmanFilterBase
    {
    public:
        static_assert(/*StateType::RowsAtCompileTime == Dynamic ||*/StateType::RowsAtCompileTime > 0,
                      "State vector must contain at least 1 element" /* or be dynamic */);
        static_assert(StateType::ColsAtCompileTime == 1, "State type must be a column vector");

        //! Numeric scalar type
        typedef typename StateType::Scalar T;

        //! Type of the state vector
        typedef StateType State;

    protected:
        //! Estimated state
        State x;
```

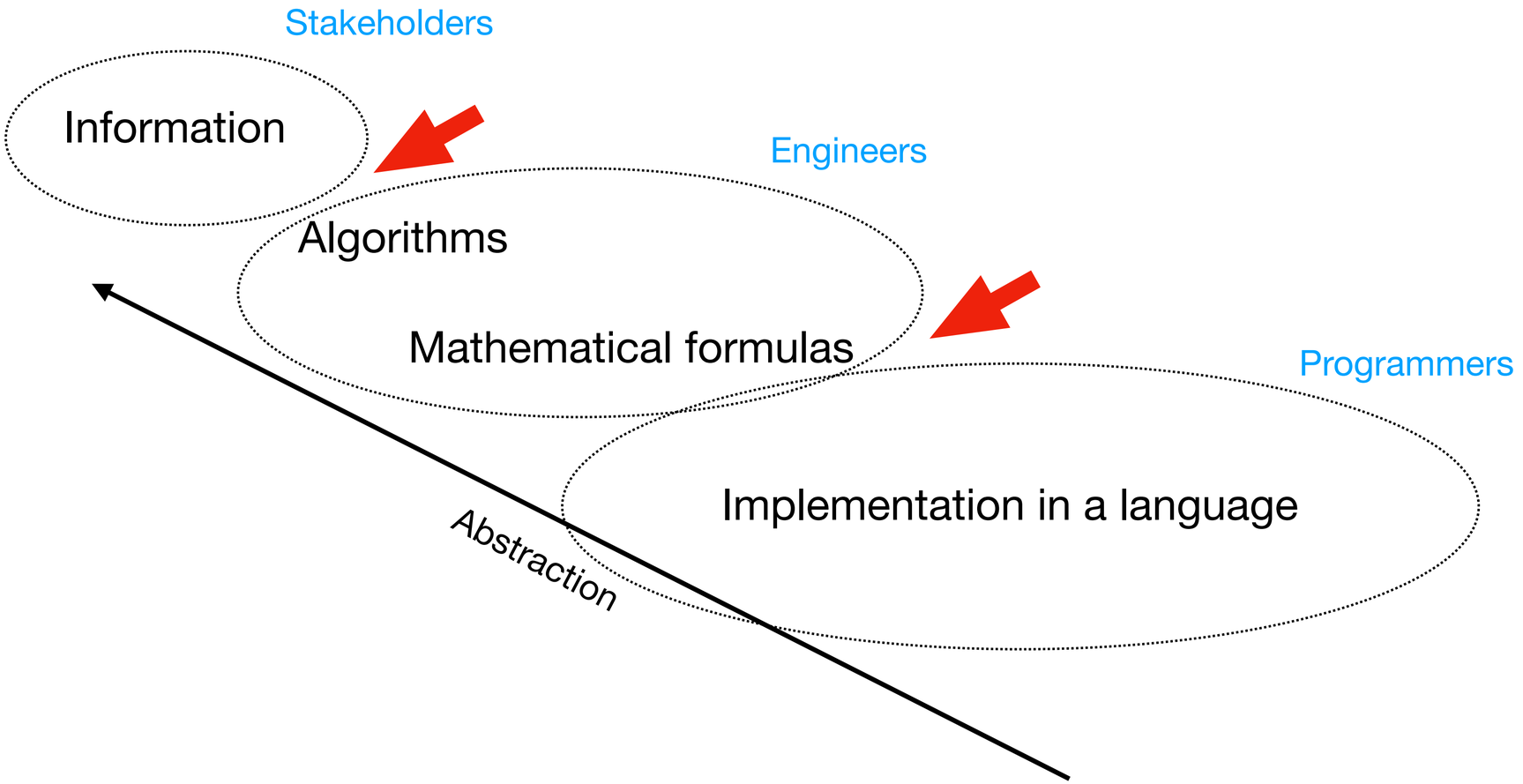
1. Big leap



Big disconnect from conceptual idea to computer implementation

1. Big leap

Development process



2. Reinventing the wheel

Automatically Proving Mathematical Theorems with Evolutionary Algorithms and Proof Assistants

Li-An Yang, Jui-Bin Liu, Chao-Hong Chen, and Ying-ping Chen*
Department of Computer Science, National Chiao Tung University, HsinChu City, TAIWAN
layang@nclab.tw, jbliu@nclab.tw, chchen@nclab.tw, ypchen@cs.nctu.edu.tw

***Abstract*—Mathematical theorems are human knowledge able to be accumulated in the form of symbolic representation, and proving theorems has been considered intelligent behavior. Based on the BHK interpretation and the Curry-Howard isomorphism, proof assistants, software capable of interacting with human for constructing formal proofs, have been developed in the past several decades. Since proofs can be considered and expressed as programs, proof assistants simplify and verify a proof by computationally evaluating the program corresponding to the proof. Thanks to the transformation from logic to computation, it is now possible to generate or search for formal proofs directly in the realm of computation. Evolutionary algorithms, known to be flexible and versatile, have been successfully applied to handle a variety of scientific and engineering problems in numerous disciplines for also several decades. Examining the feasibility of establishing the link between evolutionary algorithms, as the program generator, and proof assistants, as the proof verifier, in order to automatically find formal proofs to a given logic sentence is the primary goal of this study. In the article, we describe in detail our first, ad-hoc attempt to fully automatically prove theorems as well as the preliminary results. Ten simple theorems from various branches of mathematics were proven, and most of these theorems cannot be proven by using the tactic `auto` alone in Coq, the adopted proof assistant. The implication and potential influence of this study are discussed, and the developed source code with the obtained experimental results are released as open source.**

***Index Terms*—Evolutionary algorithm, proof assistant, Coq, automatic theorem proving.**

I. Introduction

Human knowledge has been accumulated in various forms. Among them are theorems in mathematics that are presented in a precise fashion and can be applied to innumerous domains. The importance of mathematics and its influence on science, engineering, and all kinds of technologies are undoubtedly beyond discussion. Modern mathematics are formal systems composed of four components: an alphabet, a grammar, axioms, and inference rules. Given the alphabet and grammar, in an information technology way of thinking, the development of a mathematical field or branch can be viewed as the growth of a database containing knowledge, in the form of logical sentences considered true or proven to be true. Initially, the database is empty. The first step is to put in the axioms, which are logical sentences considered true, followed by a loop: proving a new logical sentence by applying the inference rules to the database and inserting the proven logical sentence back to the database. The proven logical sentences are called theorems, propositions, lemmas, or corollaries.

Before the proposal of the link between logic and computation, the principle of Propositions as Types, logic and computation were previously considered two separate fields [1]. Based on the BHK interpretation [2], [3] and the Curry-Howard isomorphism [4], [5], (functional) programming languages, including Haskell [6], and proof assistants, such as Coq [7], HOL [8], Isabelle [9], and LEGO [10] have been developed. Due to the transformation from logic to computation, proofs to mathematical theorems can then be expressed as programs and verified computationally.

Mathematical theorem proving has been considered intelligent behavior [11]. Even for today, while a Go computer program, called *AlphaGo*, able to beat the human Go champion of Europe by five games to zero [12], [13] exists, the ability of computers to fully automatically prove mathematical theorems still seems quite limited. It is probably because most of the effort made on proof assistants aims at automated proof checking, which ensures the correctness of the proof, and at providing a proof development environment interacting with human. Moreover, there seems to be a lack of effective search mechanisms, which can "think outside of the box," especially randomized or stochastic ones, used for this purpose.

Evolutionary algorithms [14], as stochastic methods, have been known for their flexility and versatility. They have been successfully applied to handle a variety of scientific and engineering problems in numerous disciplines. Hence, in this study, we would like to make our first attempt to link evolutionary algorithms, as the program generator, and proof assistants, as the proof verifier, in a simplistic way. The primary goal of this study is to assess the feasibility of establishing frameworks capable of fully automatically proving mathematical theorems by integrating the facility of generating programs and the facility of verifying proofs as evaluating programs. Specifically, we will design a straightforward evolutionary algorithm without complicated mechanisms and simply adopt Coq [7] for fitness evaluation. The preliminary results indicate the feasibility and demonstrate that this direction of research is promising.

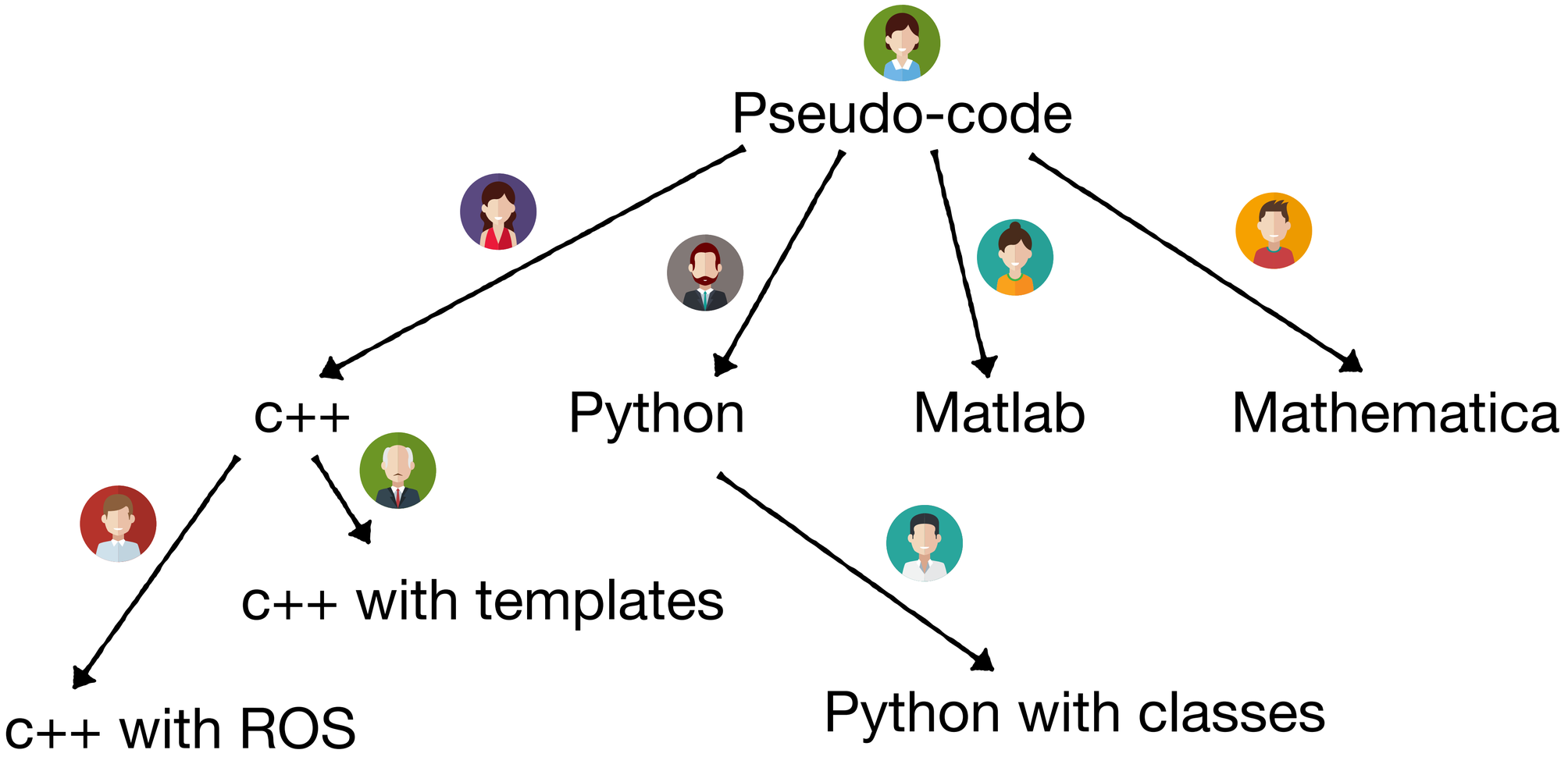
The remainder of the paper is organized as follows. Section II introduces the background regarding the primary goal of the present work. Section III describes in detail the ad-hoc attempt we adopt to investigate the feasibility of automatically proving mathematical theorems. The preliminary results obtained in our experiments are presented in section IV, followed by section V giving a broad discussion on this work and its potential influence. Finally, section VI concludes the paper.

Pseudo-code

Algorithm 1 Flow of the adopted evolutionary algorithm

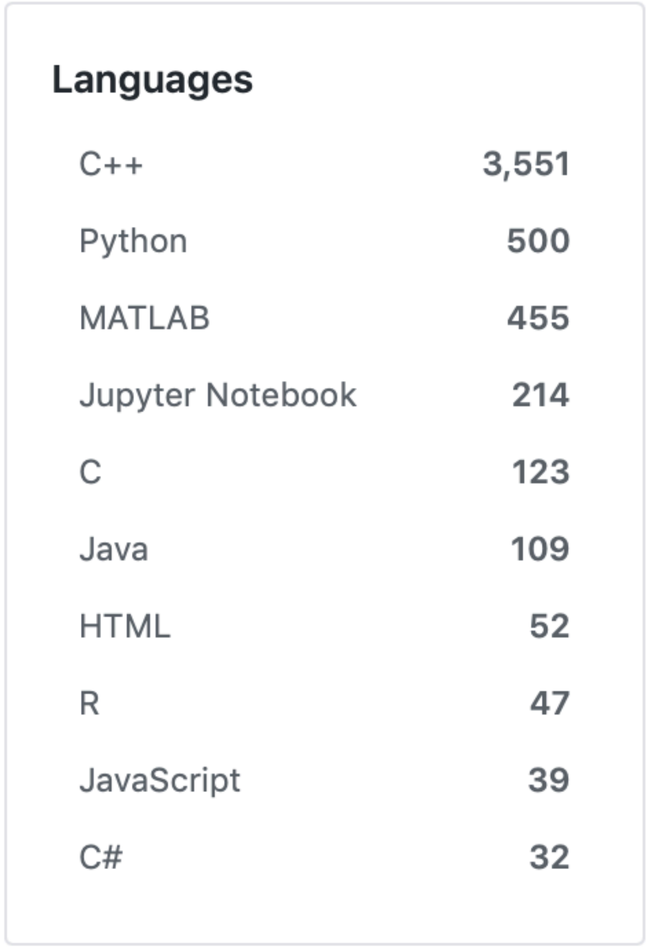
```
1:  procedure AD-HOC_EA(PopSize, MaxGen)
2:      t ← 0;
3:      Pop(t) ← Initialization(PopSize);
4:      Evaluation(Pop(t));
5:      repeat
6:          t′ ← t + 1;
7:          Pop(t′) ← ∅;
8:          i ← 0;
9:          repeat
10:             p1 ← SelectOneParent(Pop(t));
11:             p2 ← SelectOneParent(Pop(t));
12:             c ← Crossover(p1, p2);
13:             if UniformReal[0, 1] ≤ MutRat then
14:                 c ← Mutation(c);
15:             end if
16:             Pop(t′) ← Pop(t′) ∪ {c};
17:             i ← i + 1;
18:         until i ≥ PopSize;
19:         Pop(t) ← Pop(t′);            ▷ Generational model
20:         Evaluation(Pop(t));
21:         t ← t′;
22:     until t ≥ MaxGen;
23: end procedure
```

2. Reinventing the wheel



2. Reinventing the wheel

Search “Kalman Filter” in GitHub:



3. Developer tools

Used programming languages are **too general purpose**.
Not designed for robotics

Most middleware build on top of general purpose languages

c++: performance
python: prototyping

3. Developer tools

C++ 

Going strong since

1979

Abstraction languages

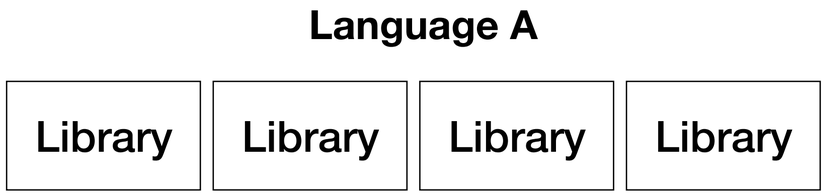
Abstraction languages are programming languages with the purpose of focusing on the required information to solve a problem.

Abstraction languages “transcompile” into standard programming languages.

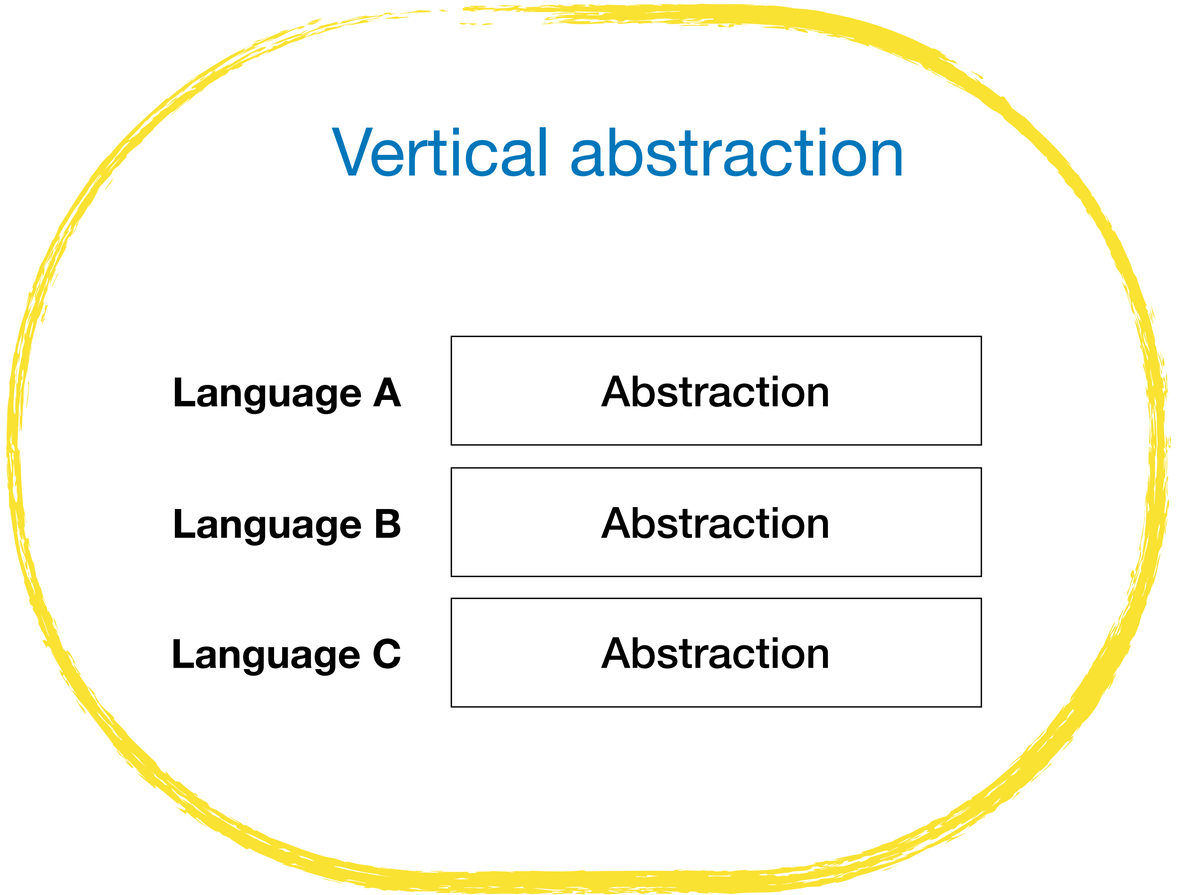
e.g.: TypeScript, CoffeeScript

Abstraction languages

Horizontal abstraction

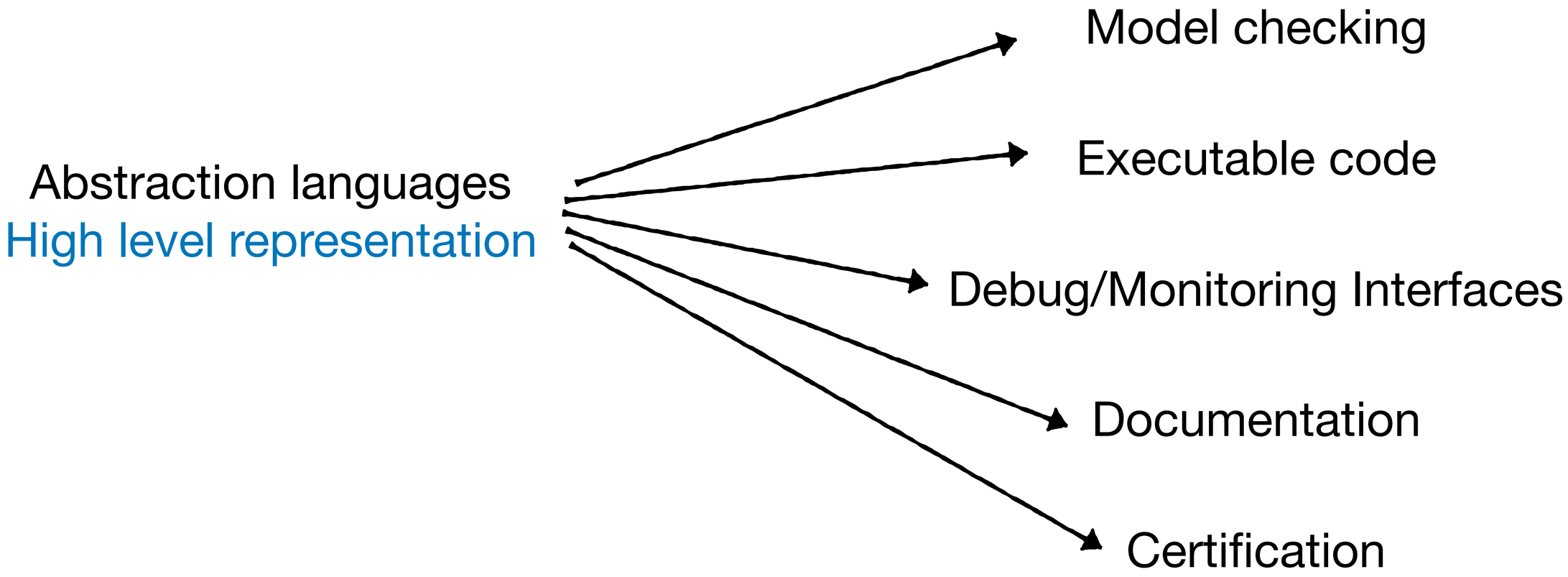


Vertical abstraction

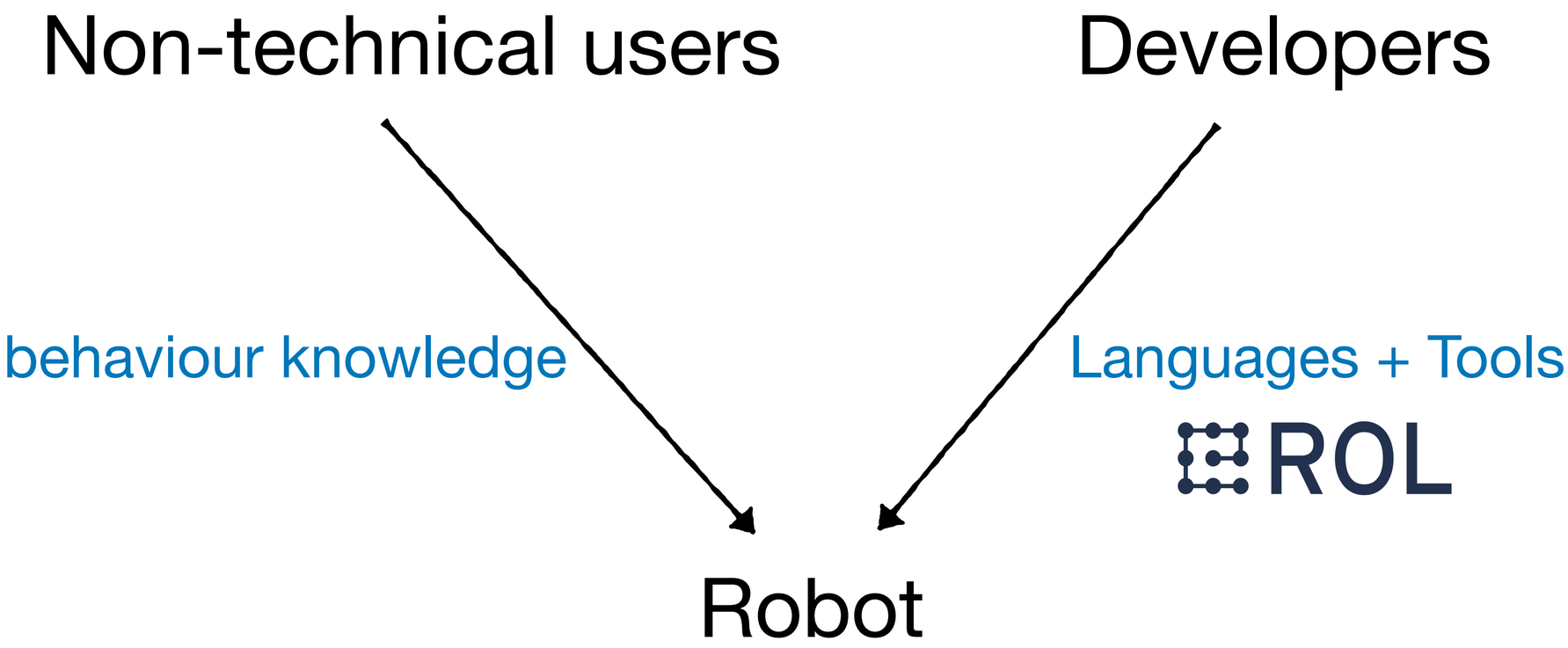


Abstraction languages

Correct by design



Abstraction languages



Program

Part I, The philosophy of the Robotics Language and the compiler

- Background
- The ROSin project
- The language of robotics
- Compiler architecture

Part II, Using the Robotics Language

- Types, variables, functions
- Abstraction languages
- Event-based vs synchronous execution
- Special constructs
- Extending the language

Program

Part III, Tools

- Temporal Logic
- Finite State Machines
- Deep learning
- Fault Detection

Part IV, Developing abstraction languages

- Compiler architecture
- A Parser for a finite state machine abstraction language
- Transformer and code generation

ROS industrial

Robotics Language Tutorial - IEEE IRC 2019

ROS-INDUSTRIAL QUALITY-ASSURED ROBOT SOFTWARE COMPONENTS

Carlos Hernandez Corbato c.h.corbato@tudelft.nl

ROSIN Coordinator TU Delft - Robotics Institute

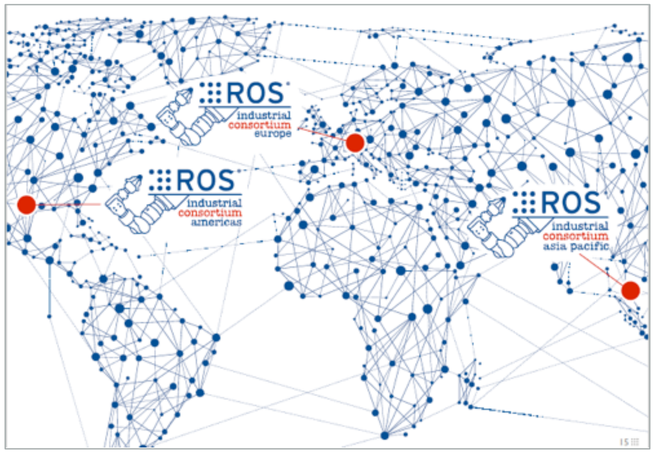
rosin-project.eu

IEEE IRC 2019, February 27, Genoa

This project is funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 732287

ROSIN PROJECT

- ROSIN: 4 years, ~8 million EUR IA H2020-ICT-2016-1
  - Speed-up the **industrial** uptake of advanced **robotics** applications.
  - Aims to **consolidate** the (many) EU-based ROS activities.
  - Builds upon the **ROS-Industrial Europe** community, to make it sustainable and leading worldwide.



ROSIN Project

www.rosin-project.eu

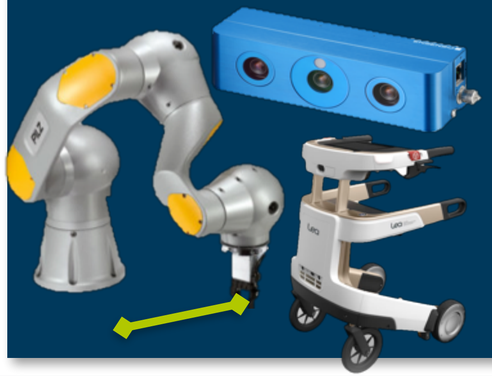
4 years, ~8 million EUR

- Speed-up the industrial uptake of advanced robotics applications in EU
- Robot Operating System (**ROS**) for an open-source **EU Digital Industrial Platform for Robotics**
- ROS-Industrial Europe community: self-sustaining and **leading** world-wide

3+ Million EUR funding

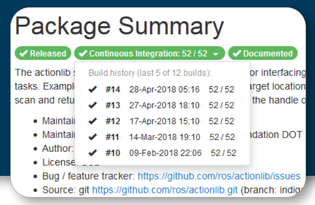
- For ROS-I devel. and education.
- 4 calls a year:

Nov 16



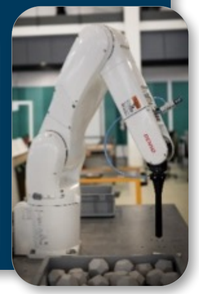
Software Quality Assurance

- Community involvement
- Continuous Integration
- Code scanning
- Model-in-the loop



ROS Education

- Academy for professionals
- School for students



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 732287.

Grants for robot software development: **Focused Technical Projects**

What service?

- **Finance ROS open source development**
  - **Concrete industry need**: driver, algorithm, application template, license or code audits…
  - We fund 1/3 of the development efforts
  - Up to **EUR 100K** ~ 1 year duration
  - ☛ also **ROS education** actions

Who can benefit?

- Robot software **developers** and **users**: companies, research centers…
  - EU H2020 program eligible entities (small consortiums)

How to apply?

- **Open call till 2020** at: http://rosin-project.eu/ftps
- **Simple** application template (~5 pages):
  - What / How / Proof of commitment

> next cut-off **April 5th**

MORE SERVICES

Quality Assurance

Working with the community to have better tools:

https://discourse.ros.org/c/quality

- continuous integration
- code scanning
- model-in-the loop
- automated test generation

Andrzej Wasowski
ROSIN Quality Assurance
ITU *University of Copenhagen*
wasowski@itu.dk

Education

Training professionals in ROS to meet industry needs:

- ROS curriculum
- ROS-I Academy prof. trainings
- ROS-I Schools for students
- ROS MOOC on edX
- Train ROS trainers

Alexander Ferrein
ROSIN Education Activities
FH Aachen
ferrein@fh-aachen.de

More information

Carlos Hernandez Corbato
Delft University of Technology
c.h.corbato@tudelft.nl

Mirko Bordignon
Fraunhofer IPA
mirko.bordignon@ipa.fraunhofer.de

Thilo Zimmermann
Fraunhofer IPA
thilo.zimmermann@ipa.fraunhofer.de

http://rosin-project.eu/ftps

info@rosin-project.eu

http://rosindustrial.org

Supported by ROSIN – ROS-Industrial Quality-Assured Robot Software Components.
More information: http://rosin-project.eu/
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 732287.

ROSIN Consortium

FH AACHEN UNIVERSITY OF APPLIED SCIENCES

ABB

Fraunhofer IPA

tecnalia Inspiring Business

IT UNIVERSITY OF COPENHAGEN

TU Delft

General purpose robotics language

Robotics Language Tutorial - IEEE IRC 2019

What is the language of robotics?

Deep question

- What is the language of mathematics?
- What is the language of … life?

What is the language of robotics?

- Physical systems modelling
- Decision making
- Learning
- Perception/Control

➡ How to describe a system and its behaviour?

➡ May need many languages

Existing tools and languages

- Hardware Description Languages (VHDL)
- Industrial scripting languages (Kuka:KRL, ABB:Rapid, Fanuc:Karel)
- Domain Specific Languages (SQL, Matlab, Excel)
- Tools for DSL (JetBrains MPS, Xtext)
- General purpose (c++, python)
- Middleware (ROS, Orocos)

Proposal

General-purpose abstraction language for robotics

- Behaviour ➙ Algorithms ➙ Mathematics:
  As close to mathematics notation as possible
- High level algorithmic/language constructs
- Multiple abstraction languages
- Portability & reuse

Proposal

General-purpose abstraction language for robotics

```
# A finite state machine
node(
  definitions: block(

      FiniteStateMachine<{
          name:machine
          initial:idle
          (idle) -start-> (running) -stop-> (idle)
        }>,

      # a topic to fire transitions
      fire ∈ Signals(Strings, rosTopic:'/fire', onNew: machine.fire(fire))
  ),
  events:
    when(□[3,0](machine.state('running'),
       machine.fire('stop')
    )
)
```

“Mini-abstraction language”

Mathematical symbols

High level statements

Manifesto

- Mathematical notation (unicode)

```
x ∈ ℤ   is equivalent to: element(x, integers)
```

- Time/state abstractions

```
x ∈ signal(ℤ,topic:”/some/signal”) // a ROS topic
w ∈ signal(ℤ,device:”/dev/ttyUSB0”) // a serial stream
y ∈ signal(ℤ) // a signal is an element of a function space
z ∈ ℤ  // a variable contains elements of a set
```

Manifesto

- Abstract objects v.s. representation

  Using 16 bits to represent integer number

```
X ∈ ℤ // Abstract mathematical object. Compiler needs to decide how to
represent the object
```

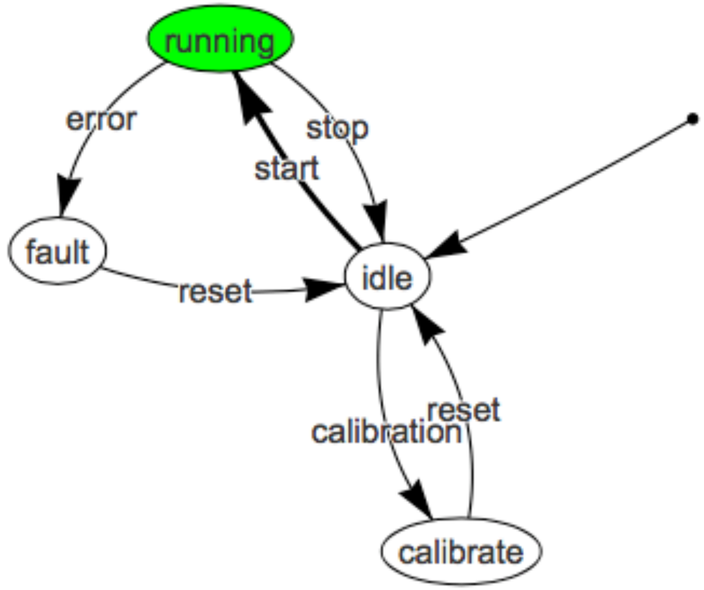
- Event driven / synchronous

```
when(□[5,0](◇[1,0](x) ∧ ◇[1,0](¬x)),
      print('oscillating faster then 1Hz for at least 5 seconds'))
```

Manifesto

Mini-abstraction languages

```
FiniteStateMachine<{

        name:machine
        initial:idle

        (running) -error-> (fault) -reset-> (idle)
        (idle) -start-> (running) -stop-> (idle)
        (idle) -calibration-> (calibrate) -reset-> (idle)

    }>,
```



Manifesto

High-level algorithmic library

Domain	Algorithm	Author	Pseudo code
Filtering	Linear Kalman	Kalman	…
Filtering	Extended Kalman	Smith et al.	…
Filtering	Unsented Kalman	Juliet et al.	…
….			

Manifesto

Maximise portability

Computation nodes


Graphical interfaces

Documentation

Syntax definition

Function composition

```
function(arguments, optional:value)
```

Special pre/post/infix operators

```
x + y
¬x
{a, b, c}
```

```
plus(x, y)
not(x)
set(a, b, c)
```

Alternatives

```
x ∈ Reals
```

```
x in Reals
```

```
element(x,Reals)
```

Syntax definition

Minimal syntax, **avoid ambiguity!**

```
node(
      name:’hello world’,
      initialise: print(’hello world’)
    )
```

Function composition

Textual code

one-to-one

Abstract syntax tree

```
node(
      name:’hello world’,
      initialise: print(’hello world’)
     )
```

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

Complete node example

```
node(
  name:"example Fibonacci",
  definitions: block(

    # incoming and outgoing signals
    question ∈ Signals(ℕ, rosTopic: ‘/fibonacci/question',
                          onNew: answer = Fibonacci(question)),

    answer ∈ Signals(ℕ, rosTopic: '/fibonacci/answer'),

    # Definition of a function
    define Fibonacci(n ∈ ℕ) -> ℕ:
      if(n ≡ 0 ∨ n ≡ 1,
          return(n),
          return(Fibonacci(n-1)+Fibonacci(n-2))
        )
  )
)
```

Complete node example

Brief command-line examples

The *rol* compiler

Robotics Language Tutorial - IEEE IRC 2019

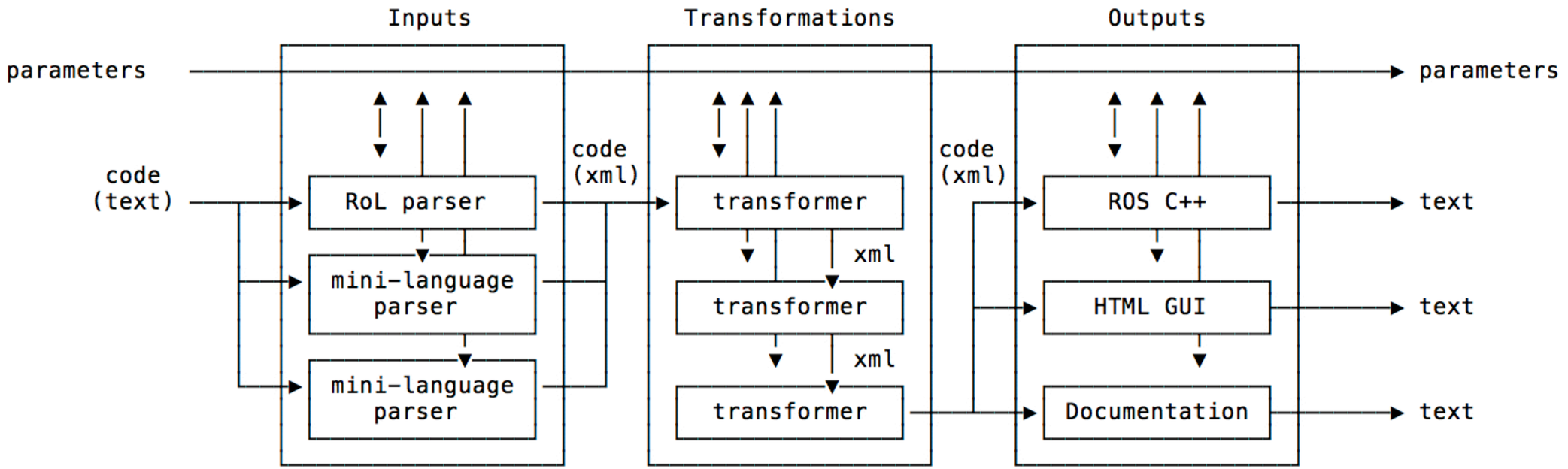
rol compiler

`rol` processes two types of information:

- **code**
- **parameters**

in three steps:

- **input**
- **transformations**
- **outputs**



rol compiler

Code: textual or abstract syntax tree representation of a program

Parameters: code-independent information that changes the behaviour of the compiler

rol compiler

Inputs:

Language parsers

Transformations:

Annotations on abstract syntax tree,
computations, decision making, file copying/creating

Outputs:

Serialisation, code generators

rol compiler

Brief command-line examples

Remarks

Robotics Language Tutorial - IEEE IRC 2019

Abstraction languages

Roadmap

- **Decision making**
  - Temporal logic, Finite state machines
  - Petri nets, behaviour trees
- **Physical systems modelling and control**
  - Lagrangian mechanics
  - Control/observer design
  - Filtering and estimation
- **Learning**
  - Deep learning
  - Reinforcement learning
- **Industry applications**
  - Fault tolerance
  - Model checking
  - Deployment
  - Developer tools

Robotics Language

The team behind






- Combined 40+ years robotics
- Faculty, PhDs, MSc
- RoboCup
- Industry
- ROS contributors

Robotics Language

Sponsors

Robot Care Systems
RRC Robotics

Industrial Quality-Assured
Robot Software Components

Horizon 2020 framework

Robotics Language

Reception

“The Robotics Language has the potential to improve the way we program robots. We look forward to its future development.”

Roger Barga, General Manager, AWS Robotics and Autonomous Services at Amazon Web Services

Q&A

1. Do I need to learn a new language?

2. How about performance?

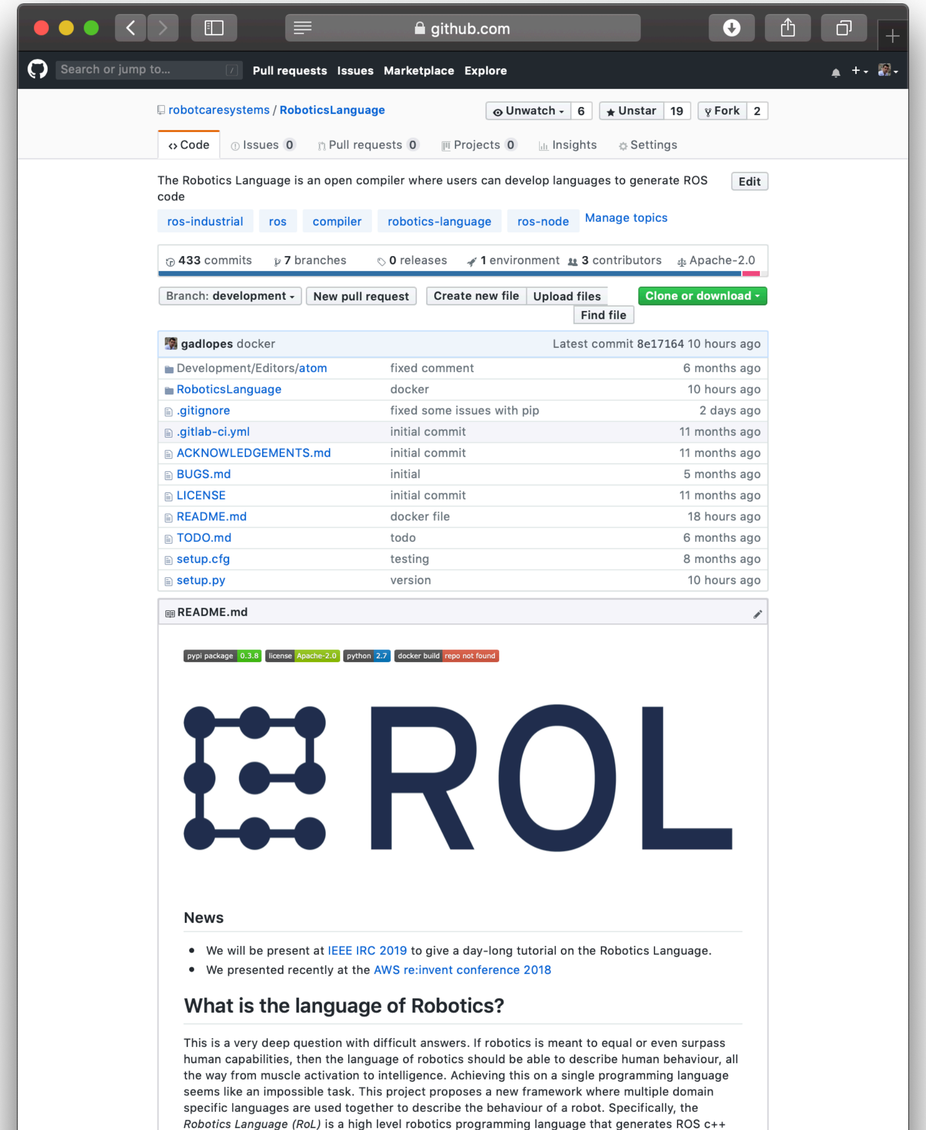
3. How many users?

Robotics Language

Looking for contributions!

- Improve the compiler engine
- Define the Robotics Language
- Create plugins

https://github.com/robotcaresystems/RoboticsLanguage



Abstraction languages

c++, python are **development tools**

rol **makes** the development tool **you need**

3D printer

Preparations

Linux/mac with ROS installed:

```
pip install RoboticsLanguage
```

Docker installed

```
docker pull roboticslanguage/rol
```

Otherwise:

https://github.com/robotcaresystems/RoboticsLanguage