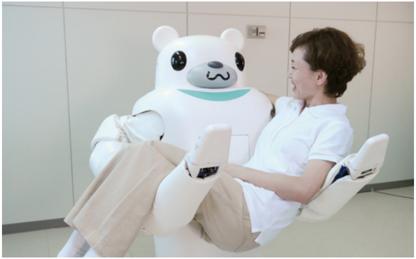
Improving ROS packages code quality with a temporal extension of first-order logic

David Come, Julien Brunel and David Doose

July 6, 2018

Robots, Robots everywhere



Entreprise 101

Industrial goal

Entreprise 101

Industrial goal

create value.

Entreprise 101

Industrial goal

create value.

Value

features that people are willing to pay for

Entreprise 101

Industrial goal

create value.

Value

features that people are willing to pay for

The features must

- fit the users' needs
- be defect-free
- cost as little as possible

Validation and Verification

Validation

Did we build the *right product* ?

Verification

Did we build the product *right* ?

Validation and Verification

Validation

Did we build the *right product* ?

Verification

Did we build the product *right* ?

Methods

- Tests
- Code generation
- Static analysis
- Code review

Style matters

Beyond correctness

Software should be *correct* and *well-written*

Style matters

Beyond correctness

Software should be *correct* and *well-written*

Well-written means

- Following idioms from the programming language
- Domain guidelines
- Project coding guide
- Library/Application specific patterns

Our goal

goal

Finding user-provided code patterns in robotics software

Our goal

goal

Finding user-provided code patterns in robotics software

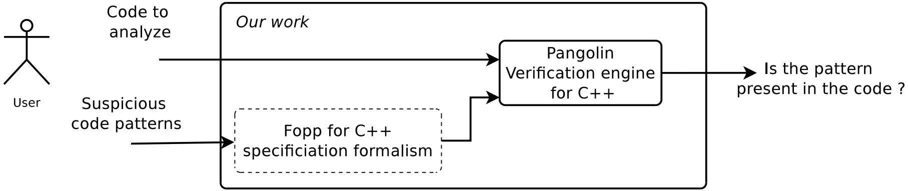
Patterns

- are not (necessarily) bugs
- just suspicious code that hinder quality / does not respect good programming practices.

Main aspects of our proposition

Main aspects

- Need to let the user specify
- Formal approach based on logic
  - unambiguous meaning to the specification
  - Complete code exploration



Running Example

Callbacks in a ROS package

All callbacks are private member functions

```
void cb(const Msg& msg){/*...*/}
int main(int argc,char* argv[]){
    ros::init(argc,argv);
    NodeHandle n;
    //...
    n.subscribe("topic",10,&cb);
}
```

FO^{++}: Overview

- A temporal extension of first-order logic, extension similar to *parametrization*
- It has well-defined semantics and is independent of any programming language

Used as a specification formalism for Pangolin

Part of the logic	FO	Temporal
Use	reasoning about the structure of the code	express properties over execution paths in functions CFG.

First-order part of FO^{++}

Definition

First-order logic = connectives, quantification and predicates

First-order part of FO^{++}

Definition

First-order logic = connectives, quantification and predicates

Example

There is a free function in which there is a locally declared variable whose type is NodeHandle

First-order part of FO^{++}

Definition

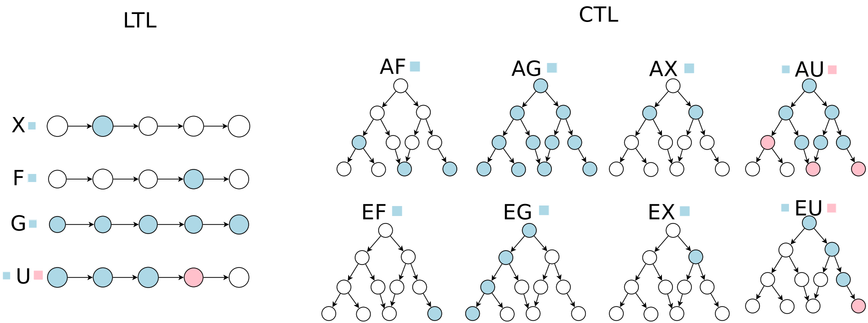
First-order logic = connectives, quantification and predicates

Example

There is a free function in which there is a locally declared variable whose type is NodeHandle

$$\exists m(\text{isFreeFunction}(m) \wedge \\ \exists n(\text{locallyDeclared}(n, m) \wedge \text{hasType}(n, \mathit{NodeHandle})))$$

Temporal logics



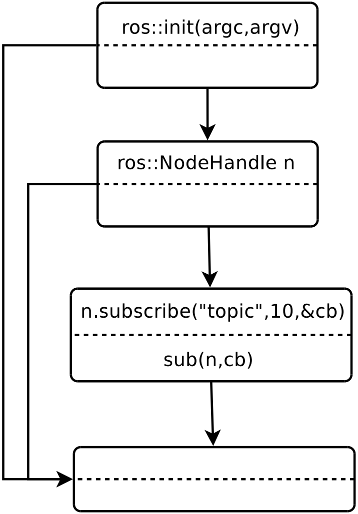
Temporal properties in FO^{++}

- Restricted to two special predicates $\mathrm{models}_{\mathsf{CTL}}(x, \psi)$ and $\mathrm{models}_{\mathsf{LTL}}(x, \psi)$

Temporal properties in FO^{++}

- Restricted to two special predicates $\mathrm{models}_{\mathsf{CTL}}(x, \psi)$ and $\mathrm{models}_{\mathsf{LTL}}(x, \psi)$

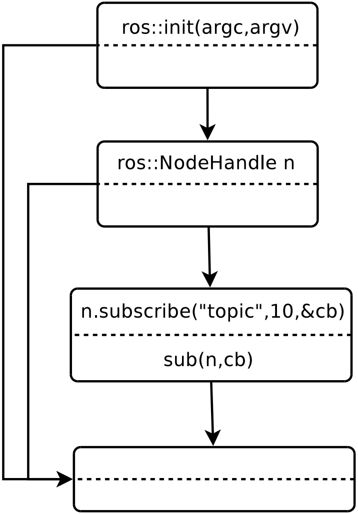
Evaluation structure: the CFG of functions



Temporal properties in FO^{++}

- Restricted to two special predicates $\mathrm{models}_{\mathsf{CTL}}(x, \psi)$ and $\mathrm{models}_{\mathsf{LTL}}(x, \psi)$

Evaluation structure: the CFG of functions



ROS callbacks formalization

Incomplete informal description

All callbacks are private member functions

ROS callbacks formalization

Complete informal description

There is a free function, in which, *there is finally a call* to subscribe on a NodeHanlde variable such as a non-private function is passed as third argument.

ROS callbacks formalization

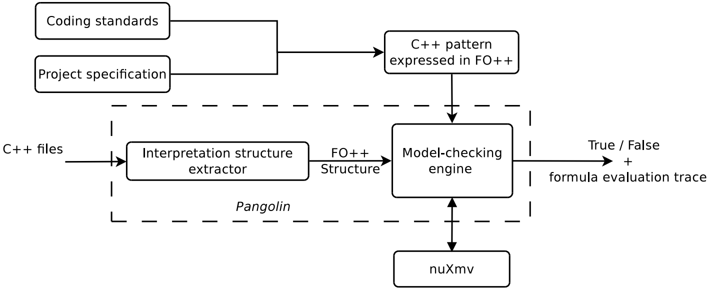
Complete informal description

There is a free function, in which, *there is finally a call* to subscribe on a NodeHanlde variable such as a non-private function is passed as third argument.

It formally express as

$$
\begin{aligned}
& \exists m(\text{isFreeFunction}(m) \\
\wedge \quad & \exists n(\text{localyDeclared}(n, m) \wedge \text{hasType}(n, \mathit{NodeHandle}) \\
\wedge \quad & \exists c(\text{allFunctions}(c) \wedge \text{models}_{\mathsf{CTL}}(m, \mathbf{EF}\text{sub}(n, c)) \\
\wedge \quad & \neg\text{isPrivate}(c)))) \qquad (1)
\end{aligned}
$$

Pangolin



Two model-checking algorithms available

- fast mode : stops at first counter example found
- complete mode : complete code exploration

Availlable at: https://gitlab.com/Davidbrcz/Pangolin

Analysis result

Pangolin evaluates a formula to

- True: the pattern is absent

Analysis result

Pangolin evaluates a formula to

- True: the pattern is absent
- False: two cases:

  False positive: a legitimate code turns out to be a counter-example for the formula because

  - unforeseen cases
  - not the intended meaning
  - Pangolin limitations

Analysis result

Pangolin evaluates a formula to

- True: the pattern is absent
- False: two cases:

  False positive: a legitimate code turns out to be a counter-example for the formula because

  - unforeseen cases
  - not the intended meaning
  - Pangolin limitations

  True positive: the code is truly suspicious.

Analysis result

Pangolin evaluates a formula to

- True: the pattern is absent
- False: two cases:

  False positive: a legitimate code turns out to be a counter-example for the formula because

  - unforeseen cases
  - not the intended meaning
  - Pangolin limitations

  True positive: the code is truly suspicious.

The user has to review the code

Rules

1. *All user-provided global variables must be constant*
2. *There should be no local non-constant variable passed to a function and never used again*
3. *There should be not call to* `std::cout<<`, `std::cerr<<` *in any function. No std::ofstream variables should be created*
4.
   a. *If the publisher is local to a function, then there is a call to publish within that function*
   b. *If the publisher is an attribute, then there is a member function in which there is a call to publish on it.*
5. *All callbacks are private member functions*.

Experiments results

Corpus:

- 25 common ROS packages (172 files)
- 3 categories : Navigation, Perception, LIDAR

Results overview

- 218 defects found:
  - 179 global variables
  - 4 variables with a scope too wide
  - 4 uses of standard streams
  - 9 member ROS publishers not used as specified
  - 22 public callbacks
- 11 false positives, False positive rate of 5%

ROSApplication pattern

```
struct ROSApplication{
  ROSApplication():rate(10){init();}
  void run(){
    while(ros::ok()){
      ros::spinOnce();
      computation();
      rate.sleep();
    }}
private:
  void init(){
    pub = nh.advertise<Msg>("pub_topic",10);
    sub = nh.subscribe("sub_topic",10,
            &ROSApplication::callback,this);
  }
  void callback(Msg const& m){/*... */ }
```

```
   void computation(){
    //...
    Msg m;
    pub.publish(m);
  }
  ros::NodeHandle nh ;
  ros::Publisher pub ;
  ros::Subscriber sub;
  ros::Rate rate     ;
};
int main(int argc, char *argv[]) {
  ros::init(argc,argv);
  ROSApplication app;
  app.run();
}
```

Pattern formalization

Consitant ROS communication

To centralize topics related operation, *there is an* `init` *method in which each publisher and subscriber is affected*. Also, *all constructors should call* `init` to ensure the publishers/subscribers are always affected.

Pattern formalization

Consitant ROS communication

To centralize topics related operation, *there is an* `init` *method in which each publisher and subscriber is affected*. Also, *all constructors should call* `init` to ensure the publishers/subscribers are always affected.

$$
\begin{aligned}
&\exists c\ (\mathrm{isClass}(c) \wedge \mathrm{name}(c, \mathit{ROSApplication}) \wedge \\
&\exists i(\ \mathrm{isMemFctOf}(i, c) \wedge \mathrm{name}(i, \mathit{init}) \wedge \\
&(\forall d\ (\mathrm{isConstructorOf}(d, c) \Rightarrow \mathrm{models}_{\mathsf{CTL}}(d, \mathbf{AF}\mathrm{call}(i)))) \wedge \\
&\forall p(\mathrm{isAttributeOf}(p, c) \wedge \mathrm{hasType}(p, \mathit{Publisher}) \Rightarrow \\
&(\exists n(\mathrm{isAttributeOf}(n, c) \wedge \mathrm{hasType}(n, \mathit{NodeHandle}) \wedge \\
&\mathrm{models}_{\mathsf{CTL}}(i,\ \mathbf{AF}(\mathrm{aPub}(p, n)) \wedge \\
&\qquad\qquad \mathbf{AG}(\mathrm{aPub}(p, n)) \Rightarrow \mathbf{AX}\ \mathbf{AG} \neg \mathrm{aPub}(p, n)))))))
\end{aligned}
$$

Conclusion and future work

Improving ROS packages code quality

- Looking for suspicious patterns in a code base
- A specification formalism: FO^{++}
- A verification engine: Pangolin
- Analyzed 25 packages, ROSApplication pattern for future packages

Conclusion and future work

Improving ROS packages code quality

- Looking for suspicious patterns in a code base
- A specification formalism: FO^{++}
- A verification engine: Pangolin
- Analyzed 25 packages, ROSApplication pattern for future packages

Future work

- Improved user input language
- Interprocedural and multi-file analysis