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

Create value by providing features that people are willing to pay for. These features must fit the users' needs, be defect-free, and cost as little as possible.
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
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

Finding user-provided code patterns in robotics software
Our 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
Callbacks in a ROS package

All callbacks are private member functions

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

<table>
<thead>
<tr>
<th>Part of the logic</th>
<th>FO</th>
<th>Temporal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use</td>
<td>reasoning about the structure of the code</td>
<td>express properties over execution paths in functions CFG.</td>
</tr>
</tbody>
</table>
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
### 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

\[
\exists m (\text{isFreeFunction}(m) \land \\
\exists n (\text{locallyDeclared}(n, m) \land \text{hasType}(n, \text{NodeHandle})))
\]
Temporal logics

LTL

- $X$
- $F$
- $G$
- $U$

CTL

- $AF$
- $AG$
- $AX$
- $AU$

- $EF$
- $EG$
- $EX$
- $EU$
Temporal properties in $FO^{++}$

- Restricted to two special predicates $\text{models}_{\text{CTL}}(x, \psi)$ and $\text{models}_{\text{LTL}}(x, \psi)$
Temporal properties in $FO^{++}$

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

Evaluation structure: the CFG of functions
Temporal properties in $FO^{++}$

- Restricted to two special predicates $models_{CTL}(x, \psi)$ and $models_{LTL}(x, \psi)$

Evaluation structure: the CFG of functions
Incomplete informal description

All callbacks are private member functions
Complete informal description

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

It formally express as

\[
\exists m \text{isFreeFunction}(m) \\
\land \exists n \text{localyDeclared}(n, m) \land \text{hasType}(n, \text{NodeHandle}) \\
\land \exists c \text{allFunctions}(c) \land \text{models}_{\text{CTL}}(m, \text{EFsub}(n, c)) \\
\land \neg \text{isPrivate}(c))))
\]

(1)
Two model-checking algorithms available

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

Available at: https://gitlab.com/Davidbrcz/Pangolin
Analysis result

Pangolin evaluates a formula to
  - True: the pattern is absent
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


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.
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%
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);
}

int main(int argc, char *argv[]){
    ros::init(argc,argv);
    ROSApplication app;
    app.run();
}
To centralize topics related operation, there is an \textit{init} method in which each publisher and subscriber is affected. Also, all constructors should call \textit{init} to ensure the publishers/subscribers are always affected.
To centralize topics related operation, \textit{there is an init method in which each publisher and subscriber is affected.} Also, \textit{all constructors should call init} to ensure the publishers/subscribers are always affected.

\begin{align*}
\exists c \ (\text{isClass}(c) \land \text{name}(c, \text{ROSApplication}) \land \\
\exists i \ (\text{isMemFctOf}(i, c) \land \text{name}(i, \text{init}) \land \\
(\forall d \ (\text{isConstructorOf}(d, c) \Rightarrow \text{models}_{\text{CTL}}(d, \text{AFcall}(i)))) \land \\
\forall p \ (\text{isAttributeOf}(p, c) \land \text{hasType}(p, \text{Publisher}) \Rightarrow \\
(\exists n \ (\text{isAttributeOf}(n, c) \land \text{hasType}(n, \text{NodeHandle}) \land \\
\text{models}_{\text{CTL}}(i, \text{AF}(a\text{Pub}(p, n)) \land \\
\text{AG}(a\text{Pub}(p, n) \Rightarrow \text{AX AG} \neg a\text{Pub}(p, n))))))))
\end{align*}
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