# Processing and analysis of robotic arm control language

Radim Luža

xluzar00@stud.fit.vutbr.cz

This presentation is divided into two parts. The first part describes a design of a simple programming language preprocessor and an interpreter of a robotic arm control language with use of formalisms of the theoretical computer science. The language for programming of the the robotic arm is composed of two languages. First of them is a LUA programming language and the second of them is a MELFA BASIC language. The LUA is a high level universal scripting language widely used by game developers to program an AI. The MELFA BASIC is a low level language designed by MELFA (the robotic arm manufacturer) to program their products. Use of the MELFA BASIC is necessary for an advanced configuration of the robotic arm, but MELFA BASIC commands are too low level and complex for an unexperienced user. This is the reason why the LUA scripting language is being used. Combining these two languages together places special demands to language preprocessing. There is also need of an independent interpretation of the MELFA BASIC because its interpreter is built-in part of the robotic arm control unit and it's not possible to use it for other purposes such as controlling of a simulator of the robotic arm.

Second part of the presentation will focus on a static and dynamic code analysis of robotic the arm control language. Programs for robotic arm are not allowed to contain any dangerous operations. To protect the robotic arm there is a simulator that simulates the the program and decides if it is safe to run it on the real hardware. Problem with the simulator is that it is very resource demanding. Complexity of the simulation is the reason why it is very difficult to estimate the time limit for the simulation. The time limit must be long enough to let correct programs run to the end but it cannot be infinite because of a never ending loops. This presentation will present the results of my research of using the code analysis to detect a potentially infinite loops. Of course it is not possible to decide if a program will end in finite time but it could be possible to detect a potentially dangerous constructions in a code or a suspicious behavior of the running program. This part is experimental.

## Formalisms I intend to use

-finite state machines and transducers

-formal grammars

-data flow and control flow analysis

-abstract interpretation

-heuristic analysis of running program