We create algorithms, software and hardware to solve problems. Before we can use our computerised solution to actually solve problem, we need to test it. By Testing, most of us imagine simple, most popular way - by giving some input to the program we expect a solution to be correct!   
However, there might be and usually are bugs, which are noticed only after some period of time, and only applying some specific data. The bug in a software means that something in the logic of the algorithm went wrong than expected. But what? Usually we go back and we spend a lot of time (depending on the size of the program) while we find that this behaviour of the software is wrong, or this not closed file causing a lot of problems later on.

There is another way of testing software. It is a run time verification. In the simplest way, it is when you put algorithm in a monitored system and watch every step your code does. Every step of the code you compare with the Model and if something does not match you flag it and you know at which point something went wrong. Runtime Verification is a model checking – automated process to check if program meets given specific requirement.

?

RuleR is a rule-based runtime veriﬁcation tool. It consists of a speciﬁcation language and an algorithm. The RuleR language allows the user to deﬁne properties in terms of parameterised conditional rules. The language is very powerful and complex. Howard Barringer , David Rydeheard, and Klaus Havelund created this language instead of trying to improve EAGLE language.

The core of a RULER rule system is a collection of named rules. A rule is formed from a condition part (antecedent) and a body part (consequent). A rule gets activated for the next evaluation step, then gets used and automatically deactivated. Rules have modifiers and extra modifiers. Always – states that this rule will be active all the time, State – tells system that this rule will be active only one time and will be deleted after it was fired, while Step – is a rule which will be deleted after any step in program no matter if this was fired or not.  
Start - ? , Assert - ?, Forbidden - ?

Algorithm.

Example

Problems.

Goal. **Interfacing with the outside world.** A monitoring tool needs to take as input both a specification and some observations about a monitored system. For the specification it needs to provide usable parsing for RuleR specifications. For the observations it will need to interface with instrumentation techniques (such as AspectJ) and parse log files for offline monitoring. **Making it go fast.**The algorithm needs to identify which rules to fire. Naively this can be implemented as a linear search but it is likely that more efficient indexing solutions exist. Additionally, the fact database can be stored in a simple set or list data structure, however this may also be non-optimal for traversal and fact deletion operations. There are then further possible optimisations related to garbage collection and other certain special cases that could be explored. **Making it more powerful.**There are possible extensions to the RuleR language that could be supported by extensions to the monitoring algorithm. For example, using arithmetic or data structures within rules. This part would look at the (theoretical) language extension and how to extend the monitoring algorithm.

Plan

Expected Results