Rule-Based Runtime Verification Tool

Mantas Daraciunas

Supervised by Giles Reger

Motivation

- most common way of testing - input-output testing
- increase confidence, but not 100%
- takes lots of time to test and find a bug

Runtime Verification

- unusual way of testing
- program Model
- is program acting according my specifications?

Runtime Verification Tools

- The field started in 2000
- Early developments: JavaMAC and Eagle
- Early tools included RuleR (Manchester) and JavaMOP (US)
- MarQ (Manchester) LogFire and TraceContract (NASA)
- for financial service - Larva (Malta),
- for medice software testing - Mufin (Germany),
- E-ACSL (France), BeepBeep and RiTHM (Canada)

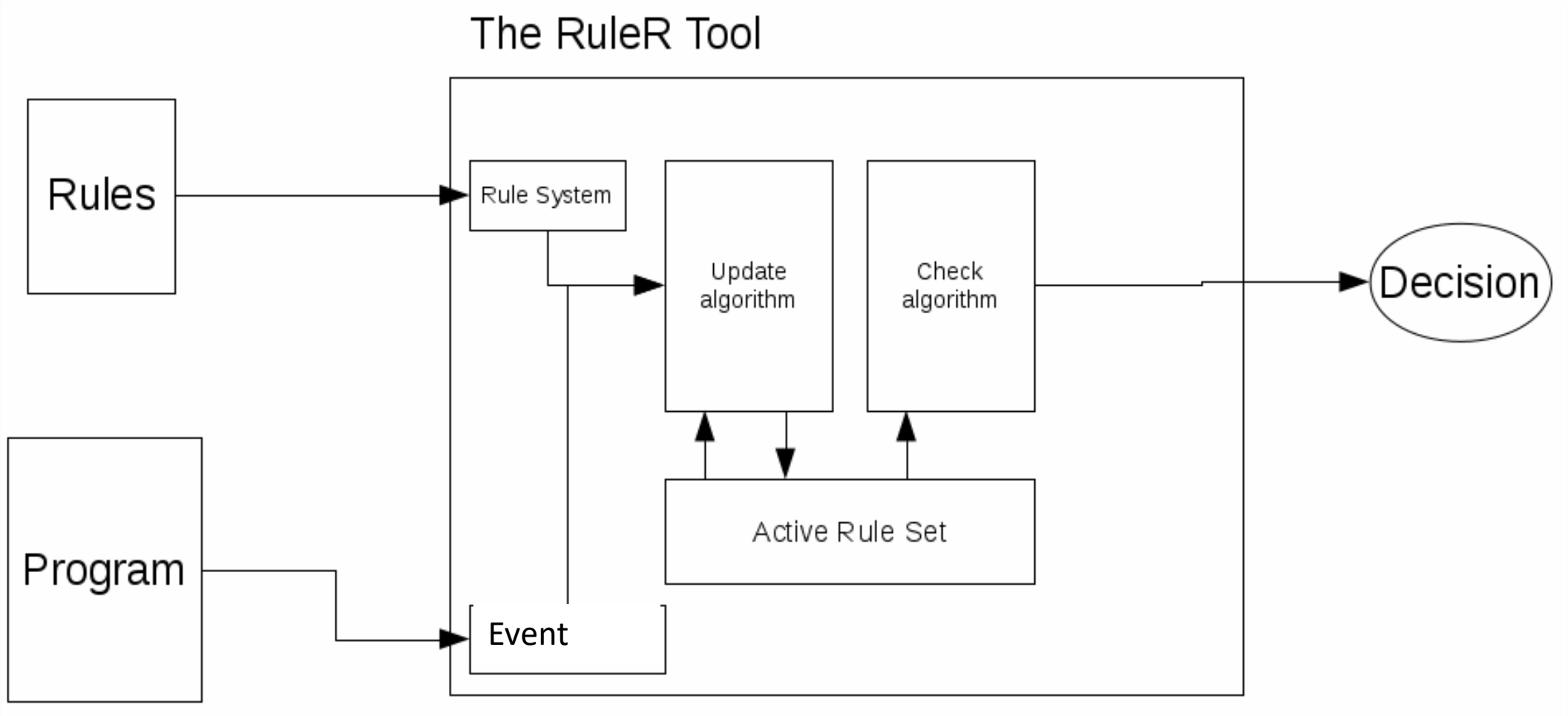
RuleR

- rule-based runtime verification tool
- consist of specification language and algorithm
- created by Howard Barringer , David Rydeheard (University of Manchester), and Klaus Havelund (NASA)

Rule

- stored in Rule System
- rule_name : { antecedent → consequent; … }
- modifier {Always, State, Step}
- extra modifier {Start, Assert, Forbidden}

The Process


The RuleR Tool
Rules
Rule System
Update algorithm
Check algorithm
Decision
Program
Active Rule Set
Event

Example

Rule System

```
Always Open {
        open(f) -> Close(f)
        close(f) -> Fail
}

State Close(f) {
        close(f) -> OK;
        open(f) -> Fail;
}
```

Programs

- open(“file.txt”).close(“file.txt”)
- open(“file.txt”).open(“file2.txt”).close(“file.txt”)
- close(“file.txt”)

More complex example

```
Always Spin {
        getNumber(n), <Compare(n),Take(a),Pay(a)> -> Fail;
        compare(account,n) <!Compare(n)> -> FAIL;
        pay(account,a) <!Win(acc,a)> -> FAIL;
        take(account,a) <!Lose(acc,a)> -> FAIL;
        getNumber(n)  -> Compare(n);
}
State Compare(n) {
        compare(account,m), < m > n > -> Win(account,m+1);
        compare(account,m), < m <= n > -> Lose(account,n+1);
}
State Lose(acc,a){
        take(account, amount) <a = amount, account = acc> -> OK;
        take(account, amount) <a != amount, account != acc> -> FAIL;
}
State Win(acc,a){
        pay(account, amount) <a = amount, account = acc> -> OK;
        pay(account, amount) <a != amount, account != acc> -> FAIL;
}
```

```
getNumber(1).compare(JohnAccount,2).pay(JohnAccount,3)

getNumber(2).compare(JohnAccount,2).take(JohnAccount,3)

getNumber(10)

getNumber(5).pay(MantasAccount,10)

compare(MantasAccount, 4)

getNumber(1).compare(JohnAccount,2).pay(MantasAccount,6)
```

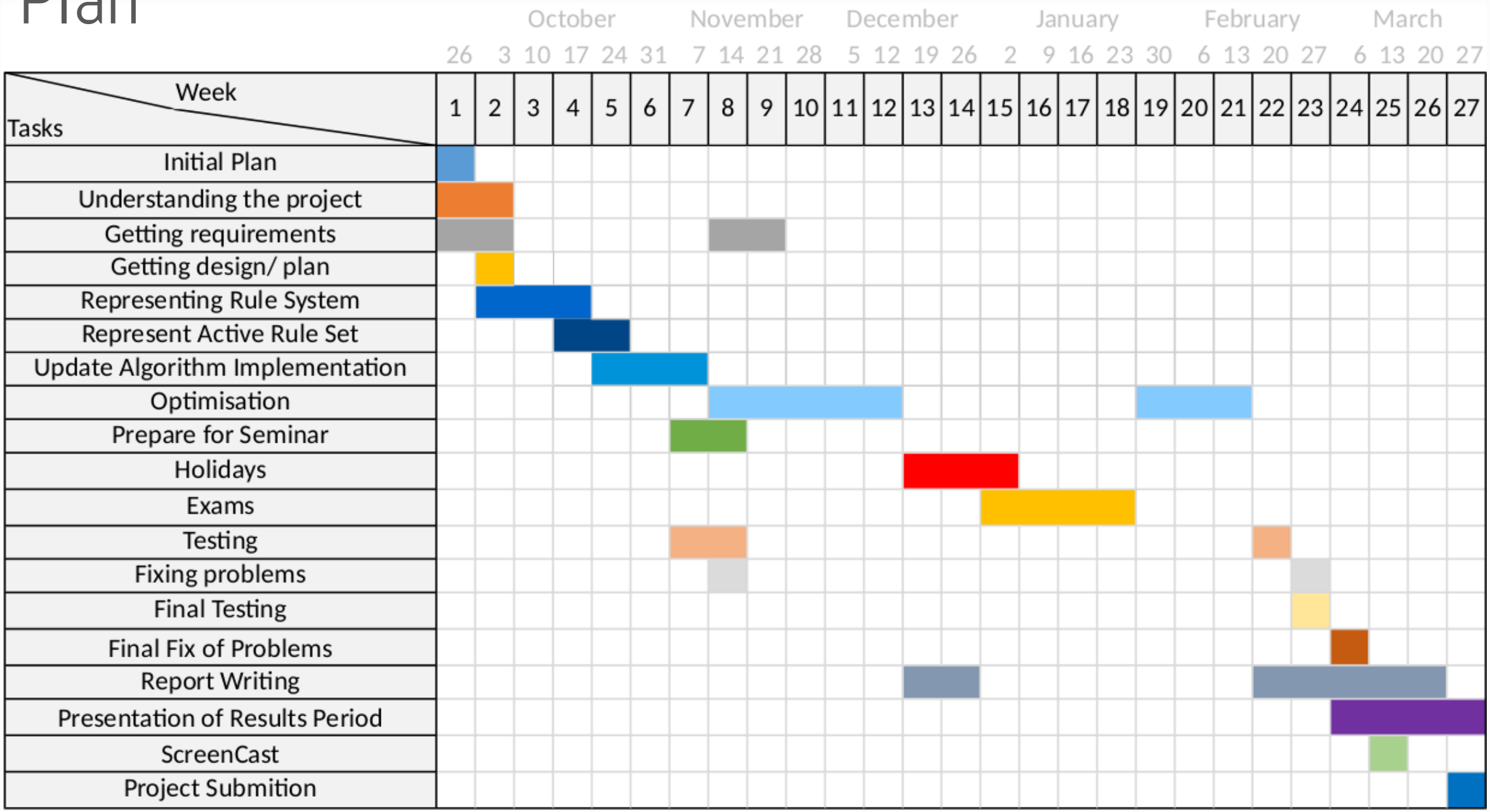
Problems

- Scalability because of Linear Search
- Redundancy of no longer needed Rule Activations

Goal

- interface with the outside world
- make it go fast
- bring RuleR to the community

Plan


October
November
December
January
February
March
26 3 10 17 24 31 7 14 21 28 5 12 19 26 2 9 16 23 30 6 13 20 27 6 13 20 27
Week
Tasks
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
Initial Plan
Understanding the project
Getting requirements
Getting design/ plan
Representing Rule System
Represent Active Rule Set
Update Algorithm Implementation
Optimisation
Prepare for Seminar
Holidays
Exams
Testing
Fixing problems
Final Testing
Final Fix of Problems
Report Writing
Presentation of Results Period
ScreenCast
Project Submition

Measuring Success

- create working product
- basic implementation versus optimised implementation
- benchmark against other tools in competition