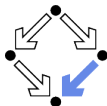
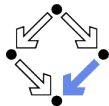


# The LogicGuard Project

Wolfgang Schreiner  
Wolfgang.Schreiner@risc.jku.at

Research Institute for Symbolic Computation (RISC)  
Johannes Kepler University, Linz, Austria  
<http://www.risc.jku.at>



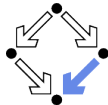


## *The Efficient Checking of Time-Quantified Logic Formulas with Applications in Computer Security.*

- FFG BRIDGE Program  
January 2012 – December 2013
- Partners
  - RISC Institute (JKU Linz/Hagenberg)  
Wolfgang Schreiner, Temur Kutsia
  - RISC Software GmbH (Hagenberg)  
Michael Krieger, Stephan Leitner
  - SecureGUARD GmbH (Linz)  
Helmut Otto, Martin Rummerstorfer.
  - Associated: George Rahonis (Thessaloniki)



<http://www.risc.jku.at/projects/LogicGuard>

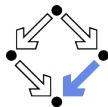


A special application of “runtime verification”.

- Monitor network traffic for security breaches.
  - Traffic is an infinite stream of TCP/IP packets.
- Specify safety property in a high-level declarative form.
  - A predicate logic formula interpreted over infinite streams with quantification over stream positions.
- Automatically translate specification into a monitor.
  - A program that surveils the traffic for violations of the property.
- Advantage: no manual low-level coding of monitors required.
  - Tedious and error-prone, difficult to maintain.
- Problem: time and space complexity of the monitor.
  - Must operate with limited time and memory resources.

Use predicate logic as the specification formalism for a runtime monitor.

# Network Traffic



HTTP\_SingleZIP\_MultipleConnections.pcap [Wireshark 1.6.5 (SVN Rev Unknown from unknown)]

Filter: tcp.stream eq 0 Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.32.0.148	85.13.136.241	TCP	66	49646 > ww [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERFECT
2	0.028743	85.13.136.241	10.32.0.148	TCP	66	ww > 49646 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SACK_PERFECT
3	0.028785	10.32.0.148	85.13.136.241	TCP	54	49646 > ww [ACK] Seq=1 Ack=1 Win=65700 Len=0
4	0.029004	10.32.0.148	85.13.136.241	HTTP	483	GET /LogicGuard/books.zip HTTP/1.1
5	0.057946	85.13.136.241	10.32.0.148	TCP	60	ww > 49646 [ACK] Seq=1 Ack=430 Win=6912 Len=0
6	0.266909	85.13.136.241	10.32.0.148	TCP	1254	[TCP segment of a reassembled PDU]
7	0.469611	10.32.0.148	85.13.136.241	TCP	54	49646 > ww [ACK] Seq=430 Ack=1201 Win=64500 Len=0
8	0.500859	85.13.136.241	10.32.0.148	TCP	1254	[TCP segment of a reassembled PDU]
9	0.703636	10.32.0.148	85.13.136.241	TCP	54	49646 > ww [ACK] Seq=430 Ack=2401 Win=65700 Len=0
10	0.734844	85.13.136.241	10.32.0.148	TCP	1254	[TCP segment of a reassembled PDU]

Frame 4: 483 bytes on wire (3864 bits), 483 bytes captured (3864 bits) on interface 0

Ethernet II, Src: 78:2b:cb:ac:06:b9 (78:2b:cb:ac:06:b9), Dst: 00:11:6b:98:85:52 (00:11:6b:98:85:52)

Internet Protocol Version 4, Src: 10.32.0.148 (10.32.0.148), Dst: 85.13.136.241 (85.13.136.241)

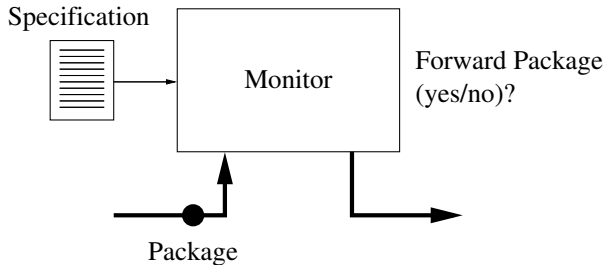
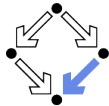
Transmission Control Protocol, Src Port: 49646 (49646), Dst Port: ww (80), Seq: 1, Ack: 1, Len: 429

Hypertext Transfer Protocol

```
0000 80 11 6b 98 85 52 78 2b cb ac 06 b9 08 00 45 00 ..k.Rx+ .....E.
0010 81 d5 4f 5f 40 00 80 06 00 00 0a 20 00 94 55 0d .._@_... ..U.
0020 08 f1 c1 ee 00 50 44 a1 8f ab 50 68 32 d8 50 18 ...PD...Ph2.P.
0030 40 29 aa 79 08 00 47 45 54 20 2f 4c 6f 67 69 63 @).GE T /Logic
0040 47 75 61 72 64 2f 62 6f 6f 6b 73 2e 7a 69 78 28 Guard/bk oks.zip
0050 48 54 54 50 2f 31 2e 31 0d 8a 55 73 65 72 2d 41 HTTP/1.1 .User-A
0060 67 65 6e 74 3a 20 4d 6f 7a 69 6c 6c 61 2f 35 2e gent: Mozilla/5.
0070 30 20 28 58 31 31 3b 20 55 3b 20 4c 69 6a 75 78 0 (X11; U; Linux
0080 20 69 36 38 36 3b 20 65 6e 2d 55 53 3b 28 72 76 ;686; en-US; rv
0090 3a 31 2e 39 2e 30 2e 31 30 29 20 47 65 63 6b 6f :1.9.0.10) Gecko
00a0 2f 32 30 38 39 30 34 32 35 32 33 20 55 62 75 6e /2009042 523 Ubu
00b0 74 75 2f 39 2e 30 34 20 28 6a 61 75 6e 74 79 29 tu/9.04 (Jaunt)
00c0 20 46 69 72 65 66 6f 78 2f 33 2e 30 2e 31 38 0d Firefox/3.0.10.
00d0 0a 41 63 63 65 70 74 3a 20 74 65 78 74 2f 68 74 .Accept: text/ht
```

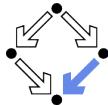
File: "HTTP\_SingleZIP\_MultipleConn... Packets: 5520 Displayed: 27 Marked: 0 Load time: 0:00:178 Profile: Default

# A Monitor



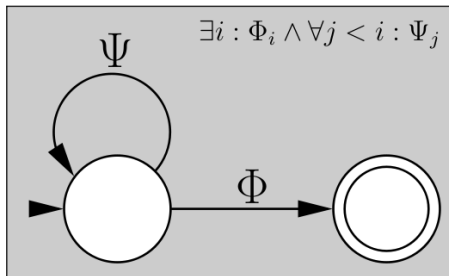
Block (or just report) every package that triggers a violation of the specified safety property.

# Core Idea

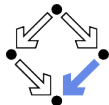


$$\Psi_0 \rightarrow \Psi_1 \rightarrow \Psi_2 \rightarrow \Psi_3 \rightarrow \Phi_4 \rightarrow \dots$$

$$\exists i : \Phi_i \wedge \forall j < i : \Psi_j$$



Monadic second order logic (MSO): the size of the automaton is non-elementary in the size of the formula.



## ■ Abstract Language

- Syntax  $F$  and semantics  $\llbracket F \rrbracket$

$$\llbracket F \rrbracket : (P^\omega \times \dots) \rightarrow Bool$$

- Translation  $T \llbracket F \rrbracket$

$$T \llbracket F \rrbracket : Step[Bool]$$

$$Step[T] = (P \times \dots) \rightarrow Answer[T]$$

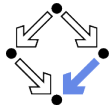
$$Answer[T] = T + Step[T]$$

## ■ Concrete Language and System

- Parsing and type checking.
- Translation.
- Runtime system.

## ■ Application Scenarios

- Modeling of “interesting” properties.
- Validation of language design.



# Application Scenario

---

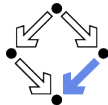
Assume that stream parts consists of downloaded parts of files.

```
predicate files <=>
  forall var now
    let part = parts@now
    with startFile(now, part)
    let file = getFile(now, part) :
      NOVIRUS(file)
```

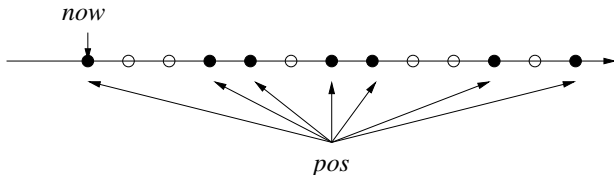
```
files ⇔
  ∀now :
    let part = parts@now :
      startFile(now, part) ⇒
        let file = getFile(now, part):
          NOVIRUS(file)
```

The combined files must not contain a virus.

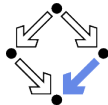




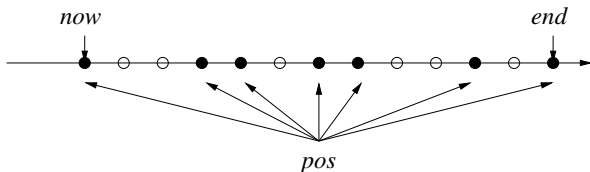
# Application Scenario (Contd)



```
function getFile(now, part) =  
  let set = combine[EMPTYSET, ADDPART, FILETIMEOUT]  
    var pos with now <= pos  
    let part0 = parts@pos  
    with SAMEFILE(part, part0)  
    resettimer  
    with COMPLETEPART(part0)  
    until COMPLETESET(this) :  
    part0 :  
  
  FILE(set)
```



## Application Scenario (Contd)



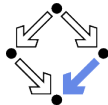
*getFile(now, part) =*

**let** set = *C(now, part)* :  
*FILE(set)*

*C(now, part) =*

**let** end =  $\min p \geq \text{now}$  such that *COMPLETESET(C(now, part, p))* :  
*C(now, part, end)*

*C(now, part, p) = combine[EMPTYSET, ADDPART] {part0 |  
now ≤ pos ≤ p ∧ part0 = parts@pos ∧  
SAMEFILE(part, part0) ∧ COMPLETEPART(part0)}*



# Stream Transformations

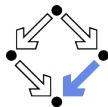
Actually, the input stream consists of TCP/IP packets, not file parts.

```
tcpip -> connections ---> http ---+---> downloads -> parts -> files
      |      |                ^
      |      +-----+        |
      |                |      |
      v                v      |
ftprequests ----> ftp ----+
```

```
let connections = ... tcpip ... :
let http = ... connections ... :
let ftprequests = ... connections ... :
let ftp = ... ftprequests ... connections ... :
let downloads = ... http ... ftp ... :
let parts = ... downloads ... :
files
```

The stream has to be transformed to appropriate layers of abstraction.

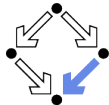
# The Connection Layer



```
Follow TCP Stream
Stream Content
GET /LogicGuard/books.zip HTTP/1.1
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.10)
Gecko/2009042523 Ubuntu/9.04 (jaunty) Firefox/3.0.10
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de, en-gb;q=0.9, en;q=0.8
Accept-Encoding:
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Cache-Control: no-cache
Pragma: no-cache
Connection: close
Host: stephan-leitner.at

HTTP/1.1 200 OK
Date: Fri, 27 Jan 2012 15:31:00 GMT
Server: Apache
Last-Modified: Fri, 27 Jan 2012 14:23:42 GMT
ETag: "7c017a-3675d5-4b7833e63dcb5"
Accept-Ranges: bytes
Content-Length: 3569109
Connection: close
Content-Type: application/zip

Entire conversation (15349 bytes)
Find Save As Print ASCII EBCDIC Hex Dump C Arrays Raw
Help Filter Out This Stream Close
```

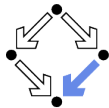


# Abstraction Layers

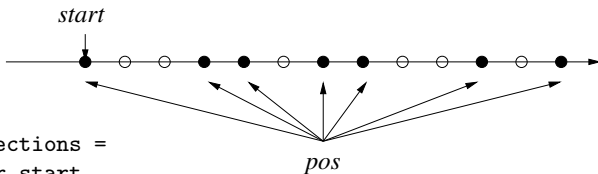
---

- Packages are merged to *connections*.
  - A connection is a sequence of bytes flowing between two hosts.
  - Content is formed according to some protocol (FTP, HTTP, ...).
- From connections *downloads* are extracted.
  - A download is a range of bytes ( $XXX-YYY$ ) from a file part.
  - Different downloads may use different protocols.
  - Some protocols (FTP) involve multiple connections for a download.
- Downloads are combined to *file parts*.
  - A file part (*file.zip.001*) is part of a file located on a host.
  - Different parts of the same file may be on different hosts.
- File parts are combined to *files*.
  - The content of a file (*file.zip*) can be monitored for a virus.

All these layers are described in predicate logic.

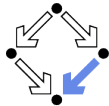


# Application Scenario (Contd)



```
stream connections =  
  stream var start  
    let packet0 = tcpip@start  
    with startConnection(packet0) :  
      partial combine[EMPTYCONNECTION, ADDPACKET, CONNECTIONTIMEOUT]  
        var pos with start <= pos  
          let packet = tcpip@pos  
          with sameConnection(packet0, packet)  
            resettimer  
            with ~SKIP(packet)  
              until endConnection(packet0, packet) :  
                packet
```

Combine all packets from the start of a TCP/IP connection till its end;  
the result is a stream of (partial) connections.



- Start with a simple core language.
  - Abstract syntax, denotational semantics, translation.
  - Not yet adequate to cover desired scenarios.
    - Basis: 4-valued logic (true, false,  $\perp$ , ?).
- Modeling application scenarios in a revised and extended language.
  - Iterative process until language seems adequate.
  - Syntax, semantics, translation still to be defined.
  - Major issues: semantics of stream transformations and timeouts.
- Prototype implementation.
  - Runtime system (C#): read stream from network or file.
  - Parser/type checker (C#): process specification and construct AST.
  - Monitor (F#): translate AST to monitor.
- Theoretical analysis.
  - Time/space complexity of monitor depending on specification.

Still at an early/exploratory stage.