https://runtimeverification.com

runtime verification

Brief Overview

Grigore Rosu
Founder, President and CEO
Professor of Computer Science, University of Illinois

Description and Mission

Runtime Verification, Inc. (RV): startup company aimed at bringing the best ideas and technology developed by the runtime verification community to the real world as mature and competitive products; licensed by the University of Illinois at Urbana-Champaign (UIUC), USA.

Mission: To offer the best possible solutions for reliable software development and analysis.

Computer Science @ UIUC

Ranked **top-5 in the USA** (US News)

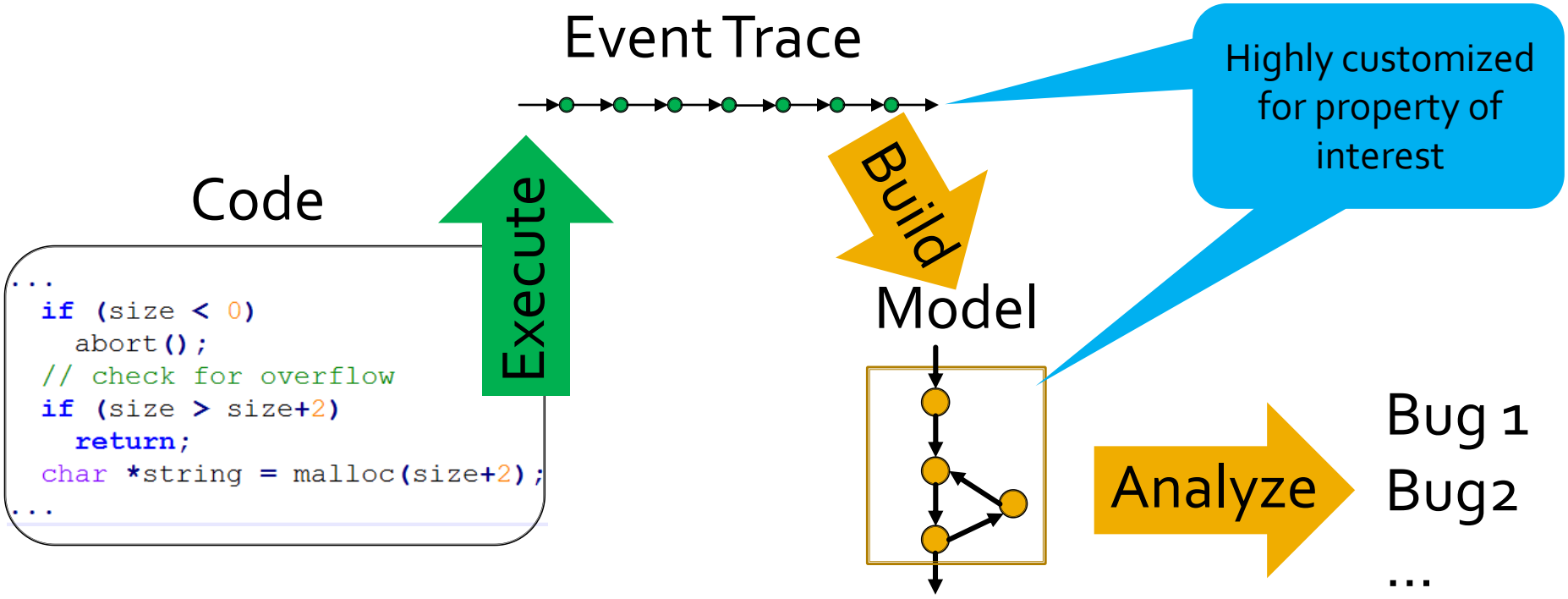
RV technology is licensed by UIUC

RV employees are former UIUC students

Technology

- *Runtime verification* is a new field aimed at verifying computing systems as they execute
  - Good scalability, rigorous, *no false alarms*
- We are leaders in the field
  - Coined the term "runtime verification"
    - As a NASA research scientist, back in 2001
  - Founded the Runtime Verification conference (RV)
  - 100+ publications
  - Raised $11.5M+ funding to develop technology

Runtime Verification Approach



Advantages:

+ precise (no false alarms)
+ good scalability and rigor
+ recovery possible

Limitations:

- code must be executable
- less code coverage, hence use with existing unit tests

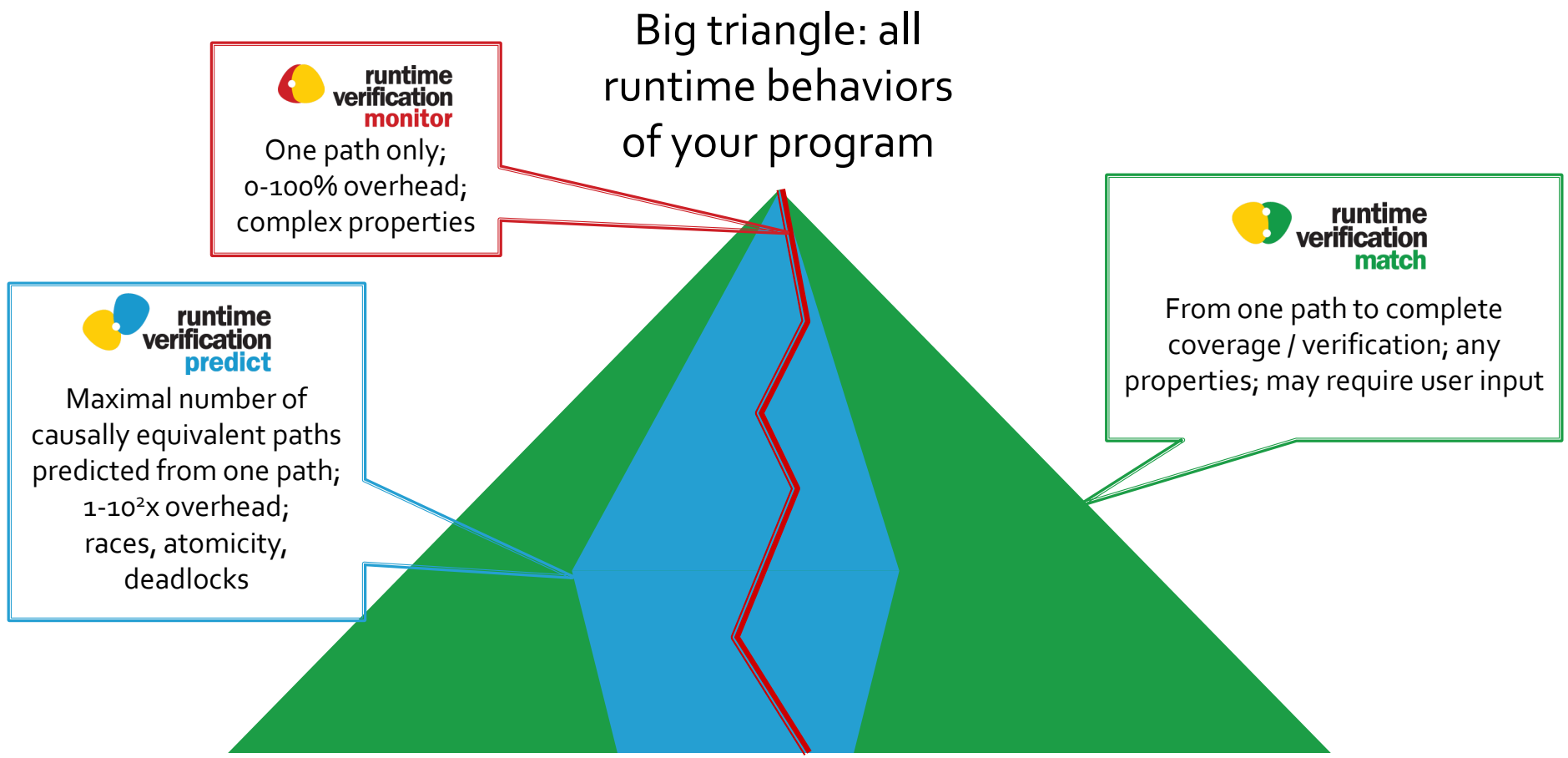
Products

RV-Match is a semantics-based automatic debugger for common and subtle C errors, and an automatic dynamic checker for all types of ISO C11 undefinedness.
- C (mature); Java and JavaScript (prototypes)

RV-Predict is an automatic dynamic data-race detector for Java, which is sound (no false positives) and maximal (no other sound dynamic tool can find more races).
- Java (mature), C/C++ with interrupts (prototype)

RV-Monitor is a runtime monitoring tool that allows for checking and enforcement of safety properties over the execution of your software.
- Java (prototype), C/C++ (prototype)

Products
Coverage vs. Performance vs. Expressiveness



RV-Match

Code (6-int-overflow.c)

...

```
int main() {
  short int a = 1;
  int i;
  for (i = 0; i < 15; i++) {
    a *= 2;
  }
  return a;
}
```

Get to market faster, increase code portability, and save on development and debugging with the most advanced and precise semantics-based bug finding tool. **RV-Match** gives you:

- an automatic debugger for subtle bugs other tools can't find, with no false positives
- seamless integration with unit tests, build infrastructure, and continuous integration
- a platform for analyzing programs, boosting standards compliance and assurance

Conventional compilers do not detect problem

```
$ gcc 6-int-overflow.c
$ ./a.out
$
$ kcc 6-int-overflow.c
$ ./a.out
Error: IMPL-CCV2
Description: Conversion to signed integer outside the range that can be represented.
Type: Implementation defined behavior.
See also: C11 sec. 6.3.1.3:3, J.3.5:1 item 4
  at main(6-int-overflow.c:29)
```

RV-Match's kcc tool precisely detects and reports error, and points to ISO C11 standard

RV-Match Approach

1. Execute program within precise mathematical model of ISO C11
2. Build abstract program state model during execution
3. Analyze each event, performing consistency checks on state

Event Trace

Are all ISO C11 rules matched?
If "no" then error

Code

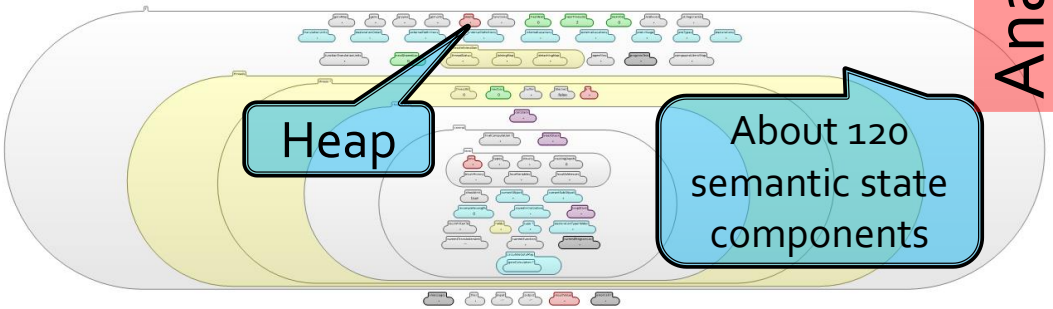
```
int main() {
  short int a = 1;
  int i;
  for (i = 0; i < 15; i++) {
    a *= 2;
  }
  return a;
}
```

Execute

Build

Analyze

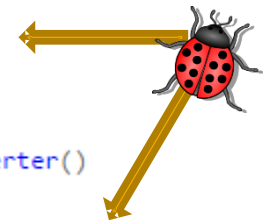
Abstract State Model



RV-Predict

Tomcat (OutputBuffer.java)

```
...
public void clearEncoders() {
    encoders.clear();
}
...
protected void setConverter()
    ...
    conv = (C2BConverter) encoders.get(enc);
    ...
```



Automatically detect the rarest and most difficult data races in your Java code, saving on development effort with the most precise race finder available. **RV-Predict** gives you:

- an automatic debugger for subtle Java data races with no false positives
- seamless integration with unit tests, build infrastructure, and continuous integration
- a maximal detection algorithm that finds more races than any sound dynamic tool

Conventional testing approaches do not detect the data-race

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Results :
Tests run: 0, Failures: 0, Errors: 0, Skipped: 0
...
...
```

RV-Predict precisely detects the data-race, and reports the relevant stack-traces

```
Data race on field java.util.HashMap.$state:
{{{ Concurrent write in thread T83 (locks held: {Monitor@67298f15})
 ----> at org.apache.catalina.connector.OutputBuffer.clearEncoders(OutputBuffer.java:255)
...
Concurrent read in thread T61 (locks held: {})
 ----> at org.apache.catalina.connector.OutputBuffer.setConverter(OutputBuffer.java:604)
...
```

RV-Predict Approach

1. Instrument program to emit event trace when executed
2. Give every observed event an order variable
3. Encode event causal ordering and data race as constraints
4. Solve constraints with SMT solver

Code

```
#include <thread>

int var = 0; // shared

void thread1() {
    var++;
}

void thread2() {
    var++;
}

int main() {
    thread t1(thread1);
    thread t2(thread2);

    t1.join();
    t2.join();

    return var;
}
```

Event Trace

Execute

Build

Model

Causal dependence as mathematical formula φ

Analyze

Is φ satisfiable?
(we use Z3 solver)
If "yes" then data race

Next Steps

- For more details see Technology and Products 
- Checkout our products 
  - First read documentation, see if tool fits your need
    - Illustrated with lots of examples
  - Then download and evaluate them
    - 90 day fully-featured evaluation versions available
- Contact us with any questions