RV-Match

Product Overview

What We Do

runtime verification

Runtime Verification Inc. **applies runtime verification-based techniques to improve the safety, reliability, and correctness of software systems** for aerospace, automotive, and the blockchain.

formal design

dynamic analysis

blockchain

formal analysis framework

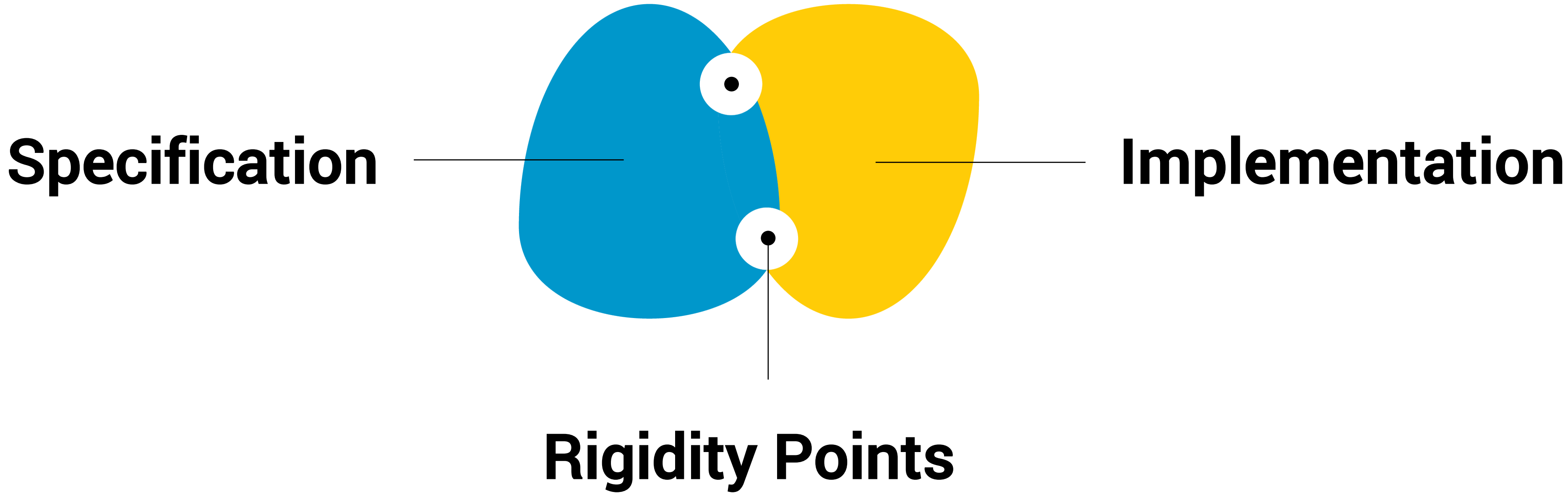
Story

runtime verification

The **runtime verification** term was coined by Professor Grigore Rosu (UIUC) and his colleague Dr. Klaus Havelund (NASA) in three papers they published in 2001 and 2002. The papers received the **Most Influential Paper award** at the ACM/IEEE Automated Software Engineering Conference in 2016, the **Test of Time award** at the Runtime Verification Conference in 2018, and respectively the **Best Software Science Paper award** at ETAPS 2002.

The company was founded in 2010.

Symbol



During **runtime verification** we prove that the specification and the implementation are tightly connected, hence two rigidity points.

What is runtime verification?

runtime verification

A subfield of program analysis and verification – just like static analysis – aimed at verifying computing systems as they execute: with good scalability, rigor, and **no false alarms**.

Runtime Verification **complements** **Static Analysis**

Runtime verification is **different** from static analysis because: it **executes** programs to analyze, **observes** execution traces, **builds** models from the execution trace, and **analyzes** the model.

RV-Match

RV-Match is a semantics based automatic debugger for common and subtle C errors, and 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

Case study – Toyota ITC benchmark

runtime verification

Toyota ITC benchmark

In a Toyota ITC benchmark evaluation, comparing RV-Match with various static analysis solutions, our product received the **best** score by finding more bugs than the static analysis tools and achieving a perfect false positive rate of zero false positives.

runtime verification match

Case study – NASA cFE

NASA core Flight Executive

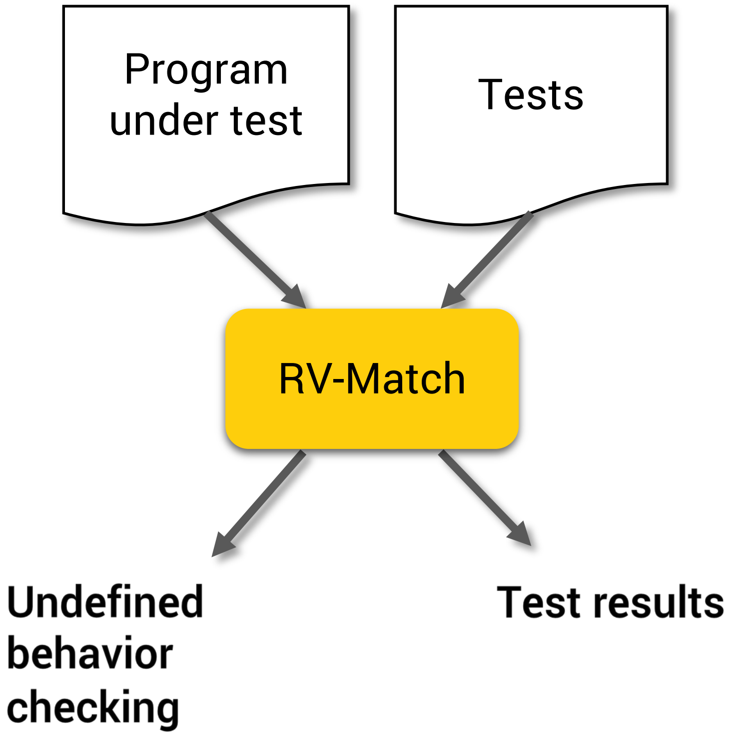
NASA core Flight Executive (cFE) is a development and run-time environment for enabling cross-platform embedded systems.

RV-Match detected:

- 15 undefined behaviors
- 1036 implementation-defined behaviors

Unit testing with RV-Match

RV-Match can replace GCC or Clang in unit-testing infrastructure to detect undefined behavior while executing the tests.



Analysis with RV-Match – the kcc tool

undef.c

```c
int main() {
    int a;
    &a + 2;
}
```

kcc detects and reports undefined behavior with ISO C11 citation.

```
$ kcc bounds.c
$ ./a.out
A pointer (or array subscript) outside the bounds of an object:
      > in main at undef.c:3:7

    Undefined behavior (UB-CEA1):
        see C11 section 6.5.6:8 http://rvdoc.org/C11/6.5.6
        see C11 section J.2:1 item 46 http://rvdoc.org/C11/J.2
        see CERT-C section ARR30-C http://rvdoc.org/CERT-C/ARR30-C
        see CERT-C section ARR37-C http://rvdoc.org/CERT-C/ARR37-C
        see CERT-C section STR31-C http://rvdoc.org/CERT-C/STR31-C
        see MISRA-C section 8.18:1 http://rvdoc.org/MISRA-C/8.18
        see MISRA-C section 8.1:3 http://rvdoc.org/MISRA-C/8.1
```

runtime verification

runtime verification match

Analysis with RV-Match – the kcc tool

runtime verification match

bounds.c

```
#include <stdio.h>
#include <string.h>

int main() {
    struct { int a; int b; } s = {0, 1};
    int * p = &s.a;
    printf("%d\n", *(p + 1));
}
```

```
$ kcc bounds.c
$ ./a.out
Dereferencing a pointer past the end of an array:
      > in main at bounds.c:9:7

    Undefined behavior (UB-CER4):
        see C11 section 6.5.6:8 http://rvdoc.org/C11/6.5.6
        see C11 section J.2:1 items 47 and 49 http://rvdoc.org/C11/J.2
        see CERT-C section ARR30-C http://rvdoc.org/CERT-C/ARR30-C
        see CERT-C section ARR37-C http://rvdoc.org/CERT-C/ARR37-C
        see CERT-C section STR31-C http://rvdoc.org/CERT-C/STR31-C
        see MISRA-C section 8.18:1 http://rvdoc.org/MISRA-C/8.18
        see MISRA-C section 8.1:3 http://rvdoc.org/MISRA-C/8.11
```

Analysis with RV-Match – the kcc tool

runtime verification match

overflow.c

```
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>

void process_something(int size) {
    size += 1; // check for overflow
    if (size < 0) return;
    char *string = malloc(size);
    string[0] = 'x';
    string[1] = '\000';
    puts(string);
}

int main(int argc, char** argv) {
    process_something(2);
    process_something(INT_MAX);
}
```

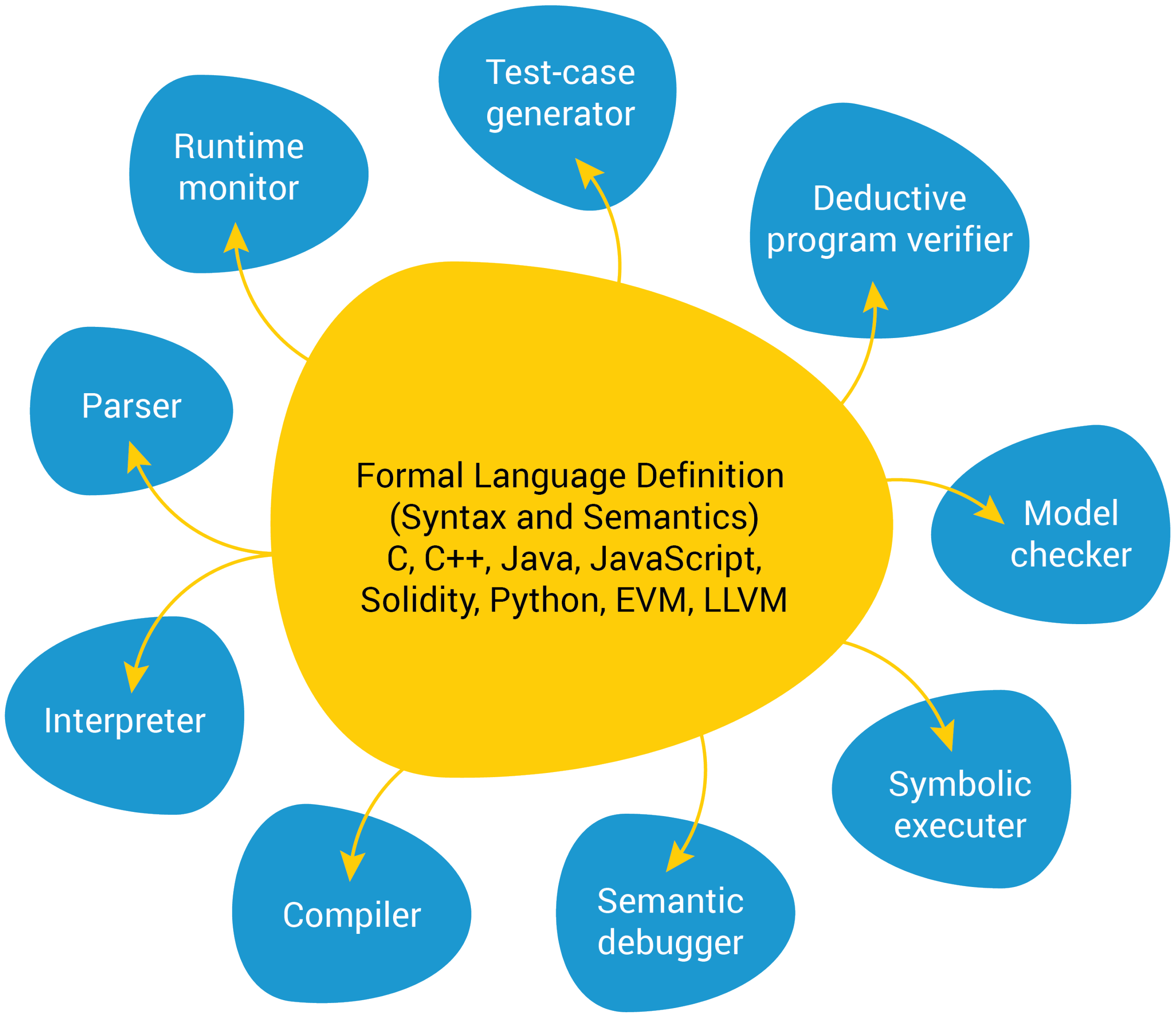
```
$ kcc overflow.c
$ ./a.out
x
Signed integer overflow:
       > in process_something at overflow.c:6:7
         in main at overflow.c:18:7

    Undefined behavior (UB-CCV1):
        see C11 section 6.5:5 http://rvdoc.org/C11/6.5
        see C11 section J.2:1 item 36 http://rvdoc.org/C11/J.2
        see CERT-C section INT32-C http://rvdoc.org/CERT-C/INT32-C
        see MISRA-C section 8.1:3 http://rvdoc.org/MISRA-C/8.1
```

Analysis with RV-Match – the kcc tool

Error	Message	ISO C11 Reference
UB-CB1	Types of function call arguments aren't compatible with declared types after promotions.	6.5.2.2:6, J.2:1 #39
UB-CB2	Function call has fewer arguments than parameters in function definition.	6.5.2.2:6, J.2:1 #38
UB-CB3	Function call has more arguments than parameters in function definition.	6.5.2.2:6, J.2:1 #38
UB-CB4	Function defined with no parameters called with arguments.	6.5.2.2:6, J.2:1 #38
UB-CCV1	Signed integer overflow.	6.5:5, J.2:1 #36
UB-CCV3	Conversion to integer from float outside the range that can be represented.	6.3.1.4:1, J.2:1 #17
UB-CCV4	Floating-point value outside the range of values that can be represented after conversion.	6.3.1.5:1, J.2:1 #18
UB-CCV5	Casting empty value to type other than void.	6.3.2.2:1, J.2:1 #23
UB-CCV6	Casting void type to non-void type.	6.3.2.2:1, J.2:1 #23
UB-CCV7	Conversion from pointer to integer of a value possibly unrepresentable in the integer type.	6.3.2.3:6, J.2:1 #24
UB-CCV11	Conversion to a pointer type with a stricter alignment requirement (possibly undefined).	6.3.2.3:7, J.2:1 #25
UB-CCV12	Floating-point overflow.	6.5:5, J.2:1 #36
UB-CEA1	A pointer (or array subscript) outside the bounds of an object.	6.5.6:8, J.2:1 #46
UB-CEA2	Pointer difference outside the range that can be represented by object of type ptrdiff_t.	6.5.6:9, J.2:1 #50
UB-CEA5	Computing pointer difference between two different objects.	6.5.6:9, J.2:1 #48
UB-CEB2	The right operand in a bitwise shift is negative.	6.5.7:3, J.2:1 #51
UB-CEB3	The right operand in a bitwise shift is greater than or equal to the bit width of the left operand.	6.5.7:3, J.2:1 #51
UB-CEB4	The left operand in a bitwise left-shift is negative.	6.5.7:4, J.2:1 #52
UB-CEB6	The right operand in a bitwise shift is negative.	6.5.7:3, J.2:1 #51
UB-CEB7	The right operand in a bitwise shift is greater than or … eft operand.	6.5.7:3, J.2:1 #51

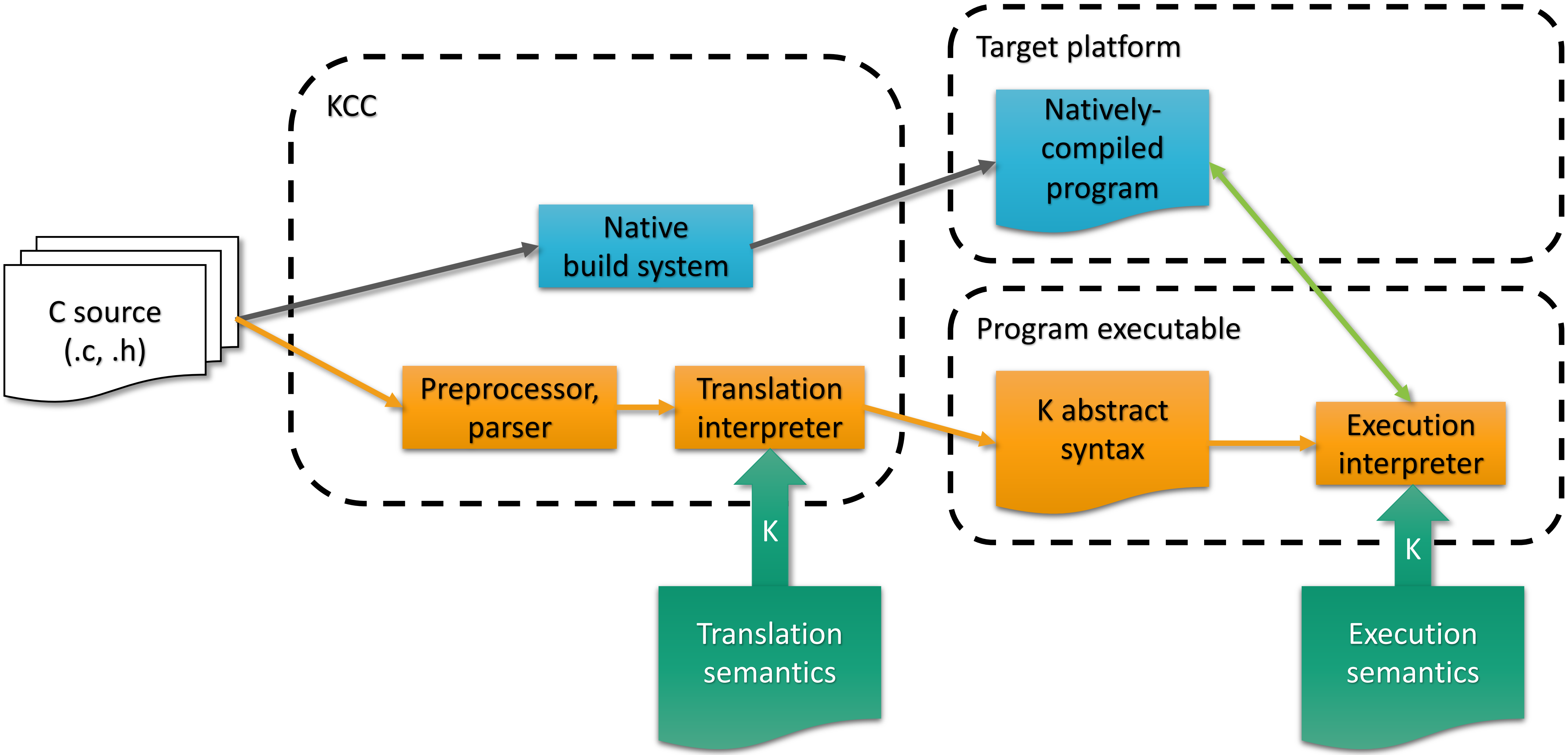
more than 200 reported issues

True semantics-based analysis



At the heart of RV-Match is a complete formal semantics of the ISO C standard powered by the K framework.

True semantics-based analysis



Partners & Customers

runtime verification

NSF

NASA

DARPA

DENSO
Crafting the Core

TOYOTA
InfoTechnology
Center Co., Ltd.

BOEING

INPUT | OUTPUT

ethereum

Algorand™

MAKER

GNOSIS

casper

Executive Team

runtime verification

Our company is fueled by people. We are **pioneers in the runtime verification community**, with hundreds of publications that shaped the field.

Grigore Rosu
President and CEO

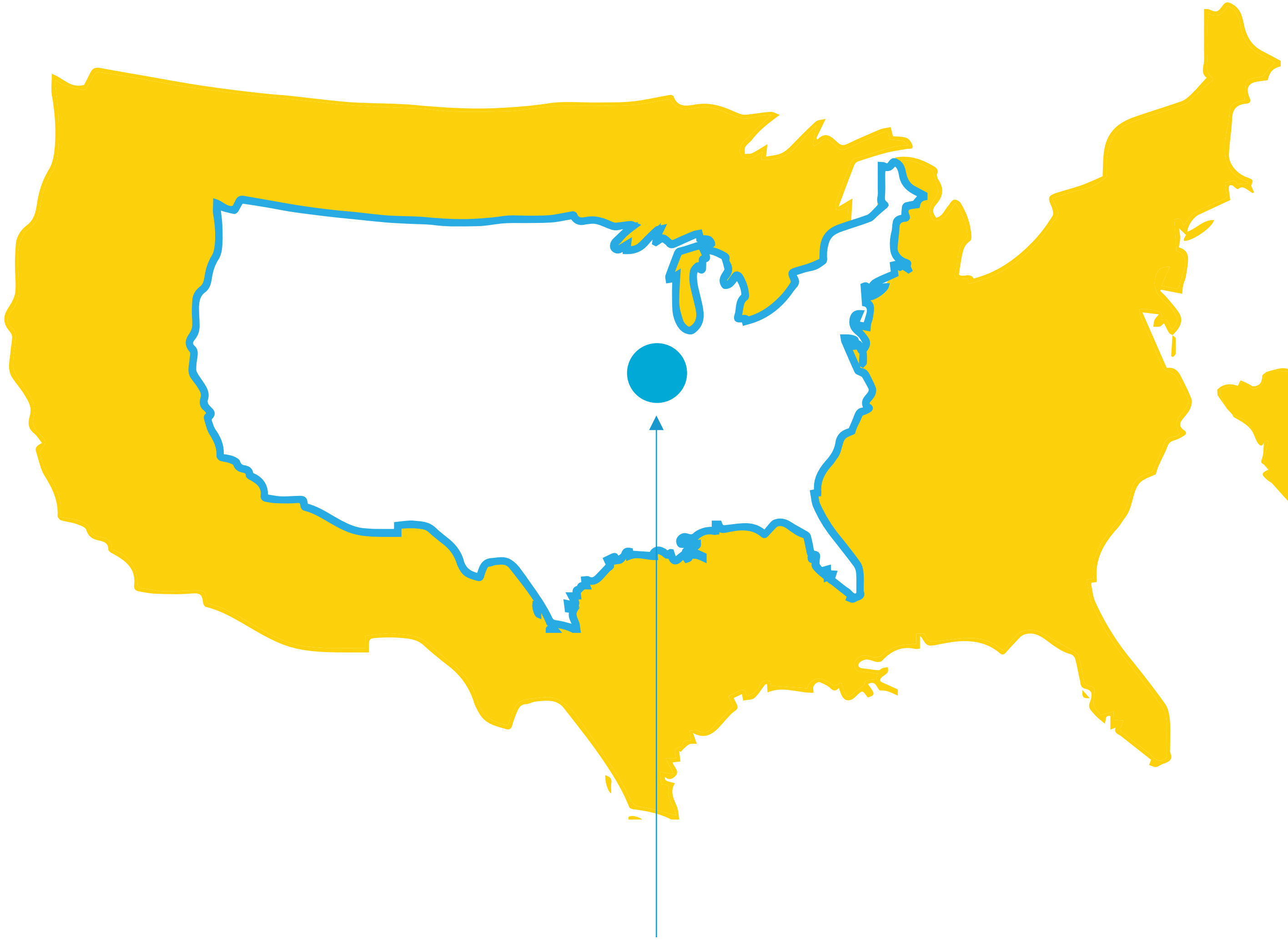
Patrick MacKay
Chief Operating Officer

Ralph Johnson
Program Management Officer

Darko Marinov
Chief Quality Officer

Main Offices

runtime verification



University of Illinois at Urbana-Champaign

Ranked #2 worldwide in Formal Methods

University of Bucharest

Ranked #1 University in Romania