

KMIR

**Formal verification
of Rust programs
using Stable MIR and
the K Framework**



Symbolic Execution & MIR

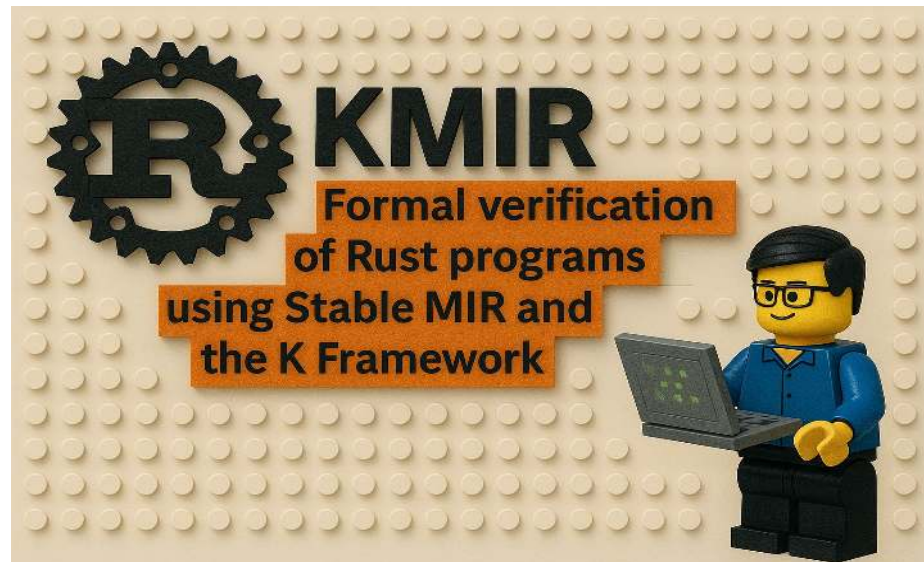
Rust Sydney, 2025 04 09

Jost Berthold

Runtime Verification Inc.

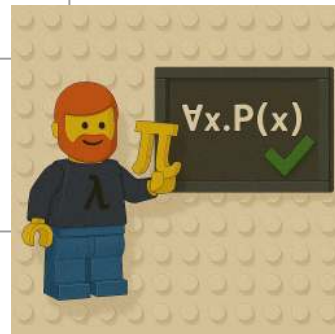
Presentation Overview

- Intro & Motivation
- KMIR: Symbolic Execution for Rust
- How it Works
 - Stable MIR and stable-mir-json
 - Semantics of (Stable) MIR in K
 - Verification of Properties in KMIR
- KMIR: Status, Limitations, Summary



Why Formal Verification Matters

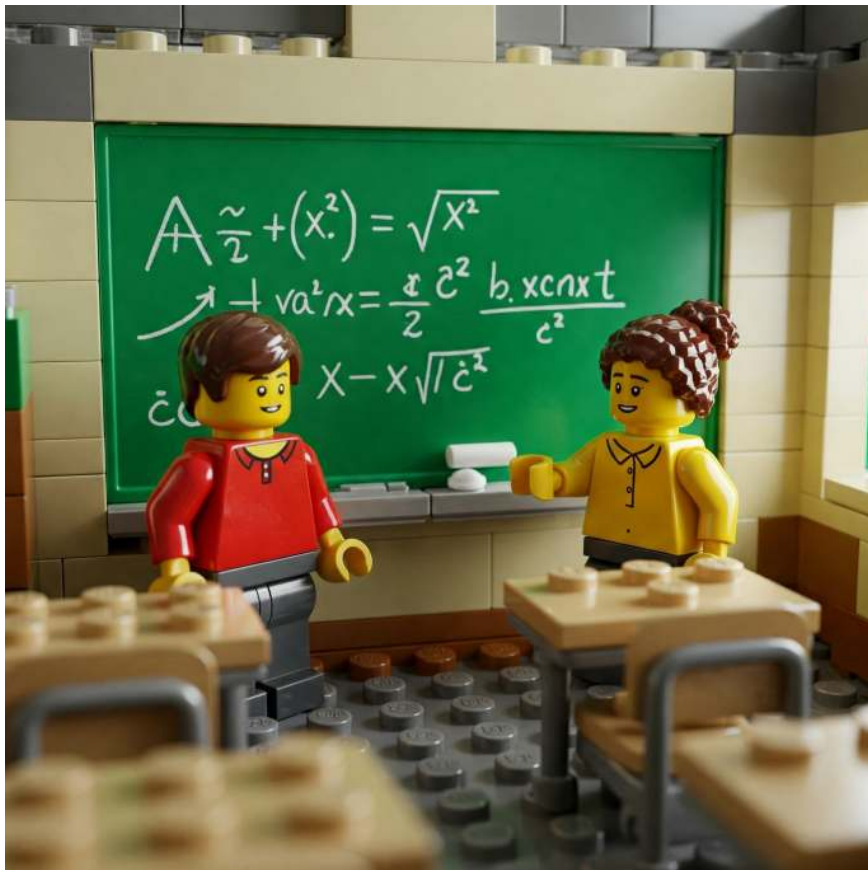
Method	Confidence	Problem
Unit tests	I checked my internal functions	What about my public endpoints?
Integration tests	I checked my external functions	What about the inputs I haven't checked?
Property tests / fuzzing	I checked my functions for random inputs	What about the inputs I haven't checked?
Formal Verification	I checked my program (exhaustively) against a mathematical specification	What if my specification is incomplete?



Formal Verification for Rust

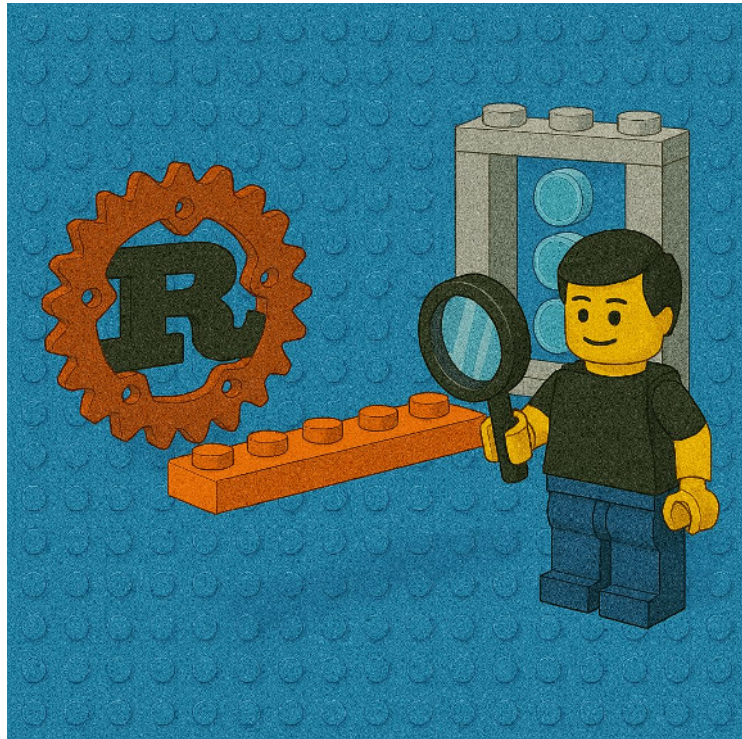
- Rust has great safety guarantees!
 - Thread and memory safety via ownership
However, there is `unsafe`...
 - Various other checks (accessing uninitialised values, incomplete matches, etc.)
- Rust's safety guarantees are not absolute.
 - Testing has limited scope
 - `Safe` code can still do the wrong thing.
- Beyond Testing: **Formal Verification**
 - Formal verification provides guarantees (through *exhaustive* exploration)

Goal: Verification tool for Rust programs including `unsafe` code.



KMIR: Symbolic Execution for Rust

- Execution and verification of Rust programs using **Stable MIR** and the **K Framework**.
- Focus: Symbolic execution for program verification.
- Enables verification of both safe and unsafe Rust code (by using intermediate code).
- Bridges the gap between Rust's compiler internals and formal verification tools.
- Open Source (BSD3):
<https://github.com/runtimeverification/mir-semantics>



Who are we?

The work on KMIR is part of my job at [Runtime Verification Inc.](#)

[Runtime Verification Inc.](#) is a software quality assurance company aimed at using formal methods to perform security audits on virtual machines and smart contracts on public blockchains.

It is dedicated to improving the safety, reliability, and correctness of software systems in the blockchain field (and other fields, too!)



- Several blockchains using Rust smart contracts (Stellar, Solana)
- Open Source development at Runtime Verification

How does it work?



kmir's Workflow and Pipeline

- Modified `rustc` to extract MIR

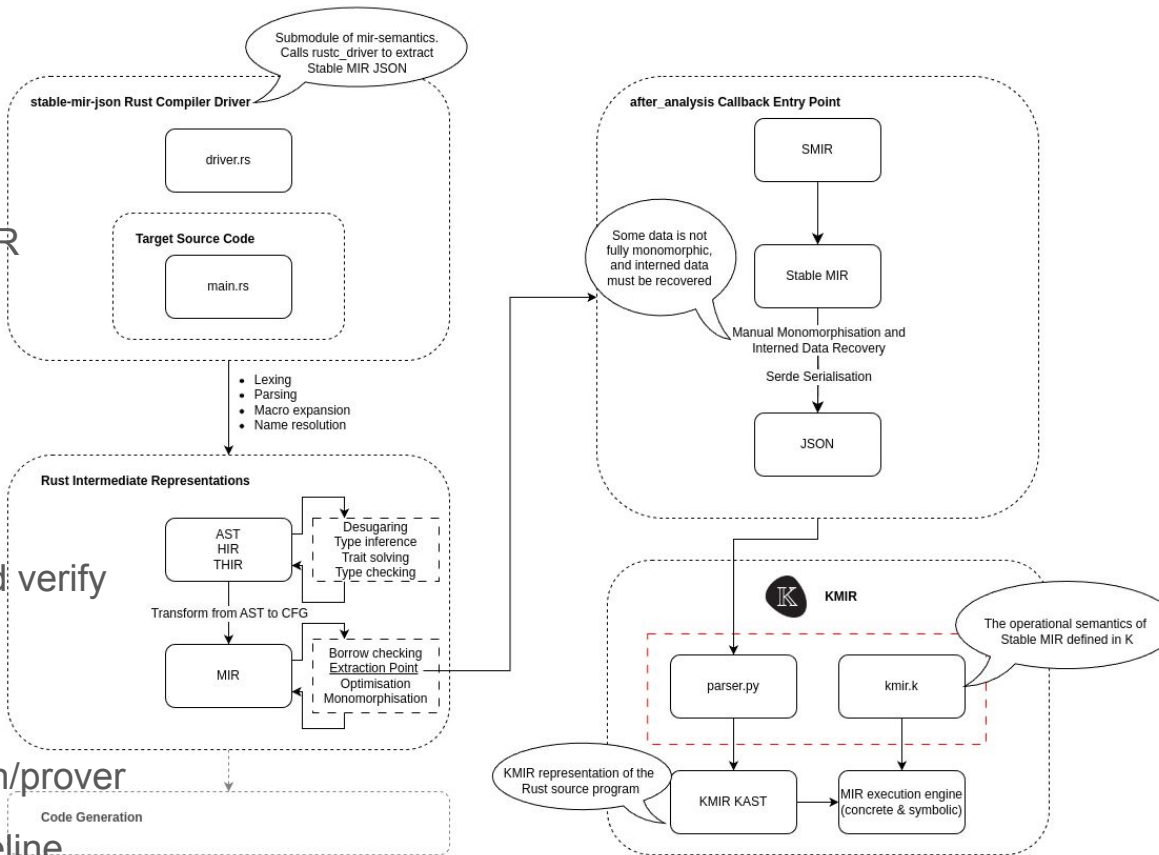
- Call-back `after_analysis`
- Externalises MIR to Stable MIR
- Adds additional context data and several lookup tables

- Semantics of Stable MIR (in K)

- Meaning of extracted code
- K Framework can execute and verify properties.

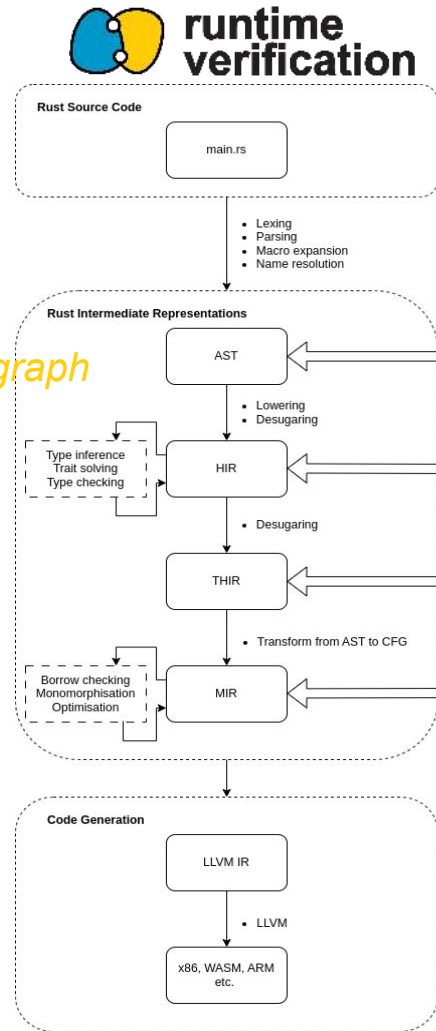
- `kmir` tool (in Python)

- Drives K Framework execution/prover
- Will orchestrate the entire pipeline



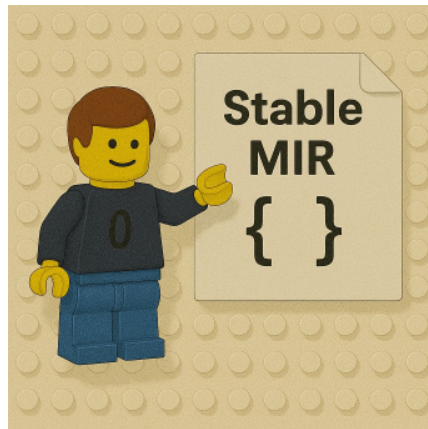
MIR: The Core of Rust's Compiler

- Rust's Mid-Level Intermediate Representation (MIR)
 - An internal representation of Rust code within the compiler
 - Follows source AST, HIR (high-level IR) and type-checking (THIR)
 - Function bodies represented with their *basic blocks* in a *control flow graph*
 - Target for borrow-checking, monomorphisation, and optimisations
- Why MIR for Verification?
 - Simpler structure (easier to analyse) than Rust source code
 - Already type-checked and borrow-checked
 - Last target-independent stage in the Rust compilation



stable-mir-json: MIR Serialization

- Why *Stable* MIR?
 - Textual MIR incomplete, not intended for programmatic consumption
 - Internal MIR changes quickly with the compiler
 - Project `stable_mir` to provide a stable API for MIR <https://github.com/rust-lang/project-stable-mir>
- `stable-mir-json`
 - A Rust compiler plugin that extracts Stable MIR into a structured JSON format
 - External format requires extracting some additional data (beyond internal Stable MIR)
 - Provides a data format for MIR-based tools that run outside the compilation pipeline



Demo 1:

stable-mir-json

- Steps:
 - Write a few simple Rust functions calling stdlib code and each other
 - `rustc --emit mir` for textual MIR
 - `stable-mir-json` for MIR JSON
 - `stable-mir-json --dot some.rs -Zno-codegen`
 - `stable-mir-json --dot some.rs -Zno-codegen`

Demo 1: stable-mir-json



- Textual MIR vs Stable MIR JSON

```
main-max-with-ll-mir x
sample-programs > maximum-example-proof > @ main-max-with-ll-mir > main
1 // WARNING: This output format is intended for human consumers only
2 // and is subject to change without notice. Knock yourself out.
3 fn main() -> () {
4   let mut _0: ();
5   let _1: usize;
6   let mut _5: bool;
7   let mut _6: bool;
8   let mut _7: bool;
9   let mut _8: bool;
10  let mut _9: bool;
11  let mut _10: bool;
12  let mut _11: !;
13  scope 1 {
14    debug a => _1;
15    let _2: usize;
16  }
17  scope 2 {
18    bb0: {
19      _1 = const 42 usize;
20      _2 = const 22 usize;
21      _3 = const 0 usize;
22      _4 = maximum(copy _1, copy _2, copy _3) -> [return: bb1, unwind continue];
23    }
24    bb1: {
25      _5 = Ge(copy _4, copy _1);
26      switchInt(move _5) -> [0: bb7, otherwise: bb2];
27    }
28    bb2: {
29      _6 = Ge(copy _4, copy _2);
30      switchInt(move _6) -> [0: bb7, otherwise: bb3];
31    }
32    bb3: {
33      _7 = Ge(copy _4, copy _3);
34    }
35  }
36  }
37 }
```

```
main-max-with-lt.mir (1) main-max-with-lt.smir.json x
sample-programs > maximum-example-proof > (1) main-max-with-lt.smir.json > ...
1  {
2    "name": "main_max_with_lt",
3    "crate_id": 537393554796547206,
4    "allocs": [
5      ],
6    "functions": [
7      {
8        "name": "main_max_with_lt",
9        "entry": true,
10       "body": {
11         "instructions": [
12           {
13             "op": "const",
14             "val": 0,
15             "type": "i32",
16             "dest": "r1",
17             "srcs": []
18           },
19           {
20             "op": "const",
21             "val": 0,
22             "type": "i32",
23             "dest": "r2",
24             "srcs": []
25           },
26           {
27             "op": "const",
28             "val": 0,
29             "type": "i32",
30             "dest": "r3",
31             "srcs": []
32           },
33           {
34             "op": "const",
35             "val": 0,
36             "type": "i32",
37             "dest": "r4",
38             "srcs": []
39           },
40           {
41             "op": "const",
42             "val": 0,
43             "type": "i32",
44             "dest": "r5",
45             "srcs": []
46           },
47           {
48             "op": "const",
49             "val": 0,
50             "type": "i32",
51             "dest": "r6",
52             "srcs": []
53           },
54           {
55             "op": "const",
56             "val": 0,
57             "type": "i32",
58             "dest": "r7",
59             "srcs": []
60           },
61           {
62             "op": "const",
63             "val": 0,
64             "type": "i32",
65             "dest": "r8",
66             "srcs": []
67           },
68           {
69             "op": "const",
70             "val": 0,
71             "type": "i32",
72             "dest": "r9",
73             "srcs": []
74           },
75           {
76             "op": "const",
77             "val": 0,
78             "type": "i32",
79             "dest": "r10",
80             "srcs": []
81           },
82           {
83             "op": "const",
84             "val": 0,
85             "type": "i32",
86             "dest": "r11",
87             "srcs": []
88           },
89           {
90             "op": "const",
91             "val": 0,
92             "type": "i32",
93             "dest": "r12",
94             "srcs": []
95           },
96           {
97             "op": "const",
98             "val": 0,
99             "type": "i32",
100            "dest": "r13",
101            "srcs": []
102          },
103          {
104            "op": "const",
105            "val": 0,
106            "type": "i32",
107            "dest": "r14",
108            "srcs": []
109          },
110          {
111            "op": "const",
112            "val": 0,
113            "type": "i32",
114            "dest": "r15",
115            "srcs": []
116          },
117          {
118            "op": "const",
119            "val": 0,
120            "type": "i32",
121            "dest": "r16",
122            "srcs": []
123          },
124          {
125            "op": "const",
126            "val": 0,
127            "type": "i32",
128            "dest": "r17",
129            "srcs": []
130          },
131          {
132            "op": "const",
133            "val": 0,
134            "type": "i32",
135            "dest": "r18",
136            "srcs": []
137          },
138          {
139            "op": "const",
140            "val": 0,
141            "type": "i32",
142            "dest": "r19",
143            "srcs": []
144          },
145          {
146            "op": "const",
147            "val": 0,
148            "type": "i32",
149            "dest": "r20",
150            "srcs": []
151          },
152          {
153            "op": "const",
154            "val": 0,
155            "type": "i32",
156            "dest": "r21",
157            "srcs": []
158          },
159          {
160            "op": "const",
161            "val": 0,
162            "type": "i32",
163            "dest": "r22",
164            "srcs": []
165          },
166          {
167            "op": "const",
168            "val": 0,
169            "type": "i32",
170            "dest": "r23",
171            "srcs": []
172          },
173          {
174            "op": "const",
175            "val": 0,
176            "type": "i32",
177            "dest": "r24",
178            "srcs": []
179          },
180          {
181            "op": "const",
182            "val": 0,
183            "type": "i32",
184            "dest": "r25",
185            "srcs": []
186          },
187          {
188            "op": "const",
189            "val": 0,
190            "type": "i32",
191            "dest": "r26",
192            "srcs": []
193          },
194          {
195            "op": "const",
196            "val": 0,
197            "type": "i32",
198            "dest": "r27",
199            "srcs": []
200          },
201          {
202            "op": "const",
203            "val": 0,
204            "type": "i32",
205            "dest": "r28",
206            "srcs": []
207          },
208          {
209            "op": "const",
210            "val": 0,
211            "type": "i32",
212            "dest": "r29",
213            "srcs": []
214          },
215          {
216            "op": "const",
217            "val": 0,
218            "type": "i32",
219            "dest": "r30",
220            "srcs": []
221          },
222          {
223            "op": "const",
224            "val": 0,
225            "type": "i32",
226            "dest": "r31",
227            "srcs": []
228          },
229          {
230            "op": "const",
231            "val": 0,
232            "type": "i32",
233            "dest": "r32",
234            "srcs": []
235          },
236          {
237            "op": "const",
238            "val": 0,
239            "type": "i32",
240            "dest": "r33",
241            "srcs": []
242          },
243          {
244            "op": "const",
245            "val": 0,
246            "type": "i32",
247            "dest": "r34",
248            "srcs": []
249          },
250          {
251            "op": "const",
252            "val": 0,
253            "type": "i32",
254            "dest": "r35",
255            "srcs": []
256          },
257          {
258            "op": "const",
259            "val": 0,
260            "type": "i32",
261            "dest": "r36",
262            "srcs": []
263          },
264          {
265            "op": "const",
266            "val": 0,
267            "type": "i32",
268            "dest": "r37",
269            "srcs": []
270          },
271          {
272            "op": "const",
273            "val": 0,
274            "type": "i32",
275            "dest": "r38",
276            "srcs": []
277          },
278          {
279            "op": "const",
280            "val": 0,
281            "type": "i32",
282            "dest": "r39",
283            "srcs": []
284          },
285          {
286            "op": "const",
287            "val": 0,
288            "type": "i32",
289            "dest": "r40",
290            "srcs": []
291          },
292          {
293            "op": "const",
294            "val": 0,
295            "type": "i32",
296            "dest": "r41",
297            "srcs": []
298          },
299          {
300            "op": "const",
301            "val": 0,
302            "type": "i32",
303            "dest": "r42",
304            "srcs": []
305          },
306          {
307            "op": "const",
308            "val": 0,
309            "type": "i32",
310            "dest": "r43",
311            "srcs": []
312          },
313          {
314            "op": "const",
315            "val": 0,
316            "type": "i32",
317            "dest": "r44",
318            "srcs": []
319          },
320          {
321            "op": "const",
322            "val": 0,
323            "type": "i32",
324            "dest": "r45",
325            "srcs": []
326          },
327          {
328            "op": "const",
329            "val": 0,
330            "type": "i32",
331            "dest": "r46",
332            "srcs": []
333          },
334          {
335            "op": "const",
336            "val": 0,
337            "type": "i32",
338            "dest": "r47",
339            "srcs": []
340          },
341          {
342            "op": "const",
343            "val": 0,
344            "type": "i32",
345            "dest": "r48",
346            "srcs": []
347          },
348          {
349            "op": "const",
350            "val": 0,
351            "type": "i32",
352            "dest": "r49",
353            "srcs": []
354          },
355          {
356            "op": "const",
357            "val": 0,
358            "type": "i32",
359            "dest": "r50",
360            "srcs": []
361          },
362          {
363            "op": "const",
364            "val": 0,
365            "type": "i32",
366            "dest": "r51",
367            "srcs": []
368          },
369          {
370            "op": "const",
371            "val": 0,
372            "type": "i32",
373            "dest": "r52",
374            "srcs": []
375          },
376          {
377            "op": "const",
378            "val": 0,
379            "type": "i32",
380            "dest": "r53",
381            "srcs": []
382          },
383          {
384            "op": "const",
385            "val": 0,
386            "type": "i32",
387            "dest": "r54",
388            "srcs": []
389          },
390          {
391            "op": "const",
392            "val": 0,
393            "type": "i32",
394            "dest": "r55",
395            "srcs": []
396          },
397          {
398            "op": "const",
399            "val": 0,
400            "type": "i32",
401            "dest": "r56",
402            "srcs": []
403          },
404          {
405            "op": "const",
406            "val": 0,
407            "type": "i32",
408            "dest": "r57",
409            "srcs": []
410          },
411          {
412            "op": "const",
413            "val": 0,
414            "type": "i32",
415            "dest": "r5
```

```

    "NormalSym": "_ZN16main_max_with_lt7maximum17h5e37",
  },
  {
    27,
    {
      "NormalSym": "_ZN4core9panicking5panic17h941160ead",
    }
  },
  {
    -
  },
],
"uneval_consts": [],
"items": [
  {
    "symbol_name": "ZN16main_max_with_lt4main17h96bac61ef",
    "mono_item_kind": {
      "MonoItemFn": {
        "name": "main",
        "id": 6,
        "body": {
          "blocks": [
            {
              "statements": [
                {
                  "kind": {
                    "Assign": {
                      "local": 1,
                      "projection":
                    },
                    "Use": { -
                  }
                }
              ],
              "span": 52
            },
          ],
        },
        "terminator": {
          "kind": {
            "call": {
              "func": {
                "Constant": {
                  "span": 50,

```

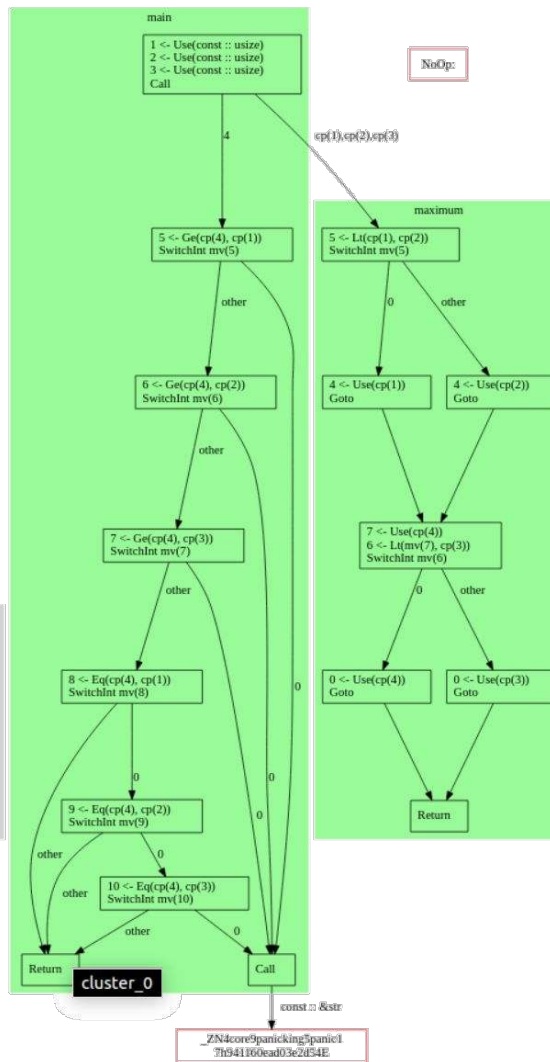
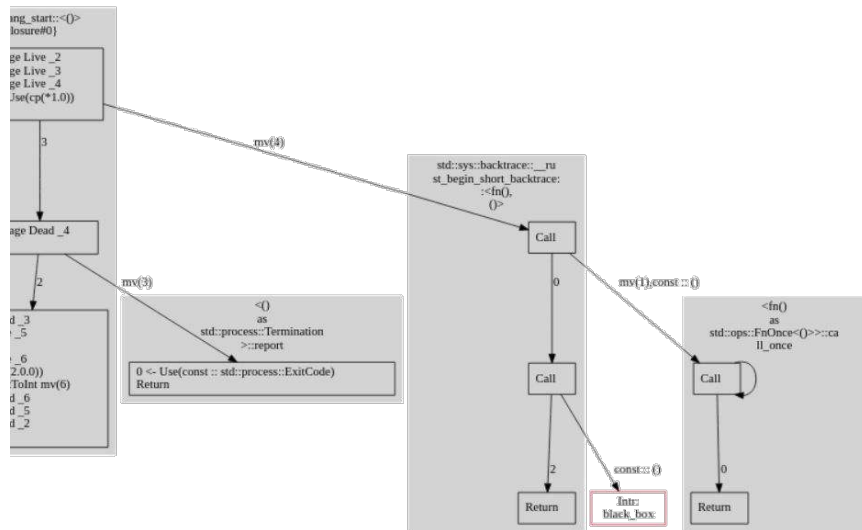
```

    },
    "types": [
      {
        2,
        {
          "PrimitiveType": {
            "Int": "I8"
          }
        }
      ],
      {
        6,
        {
          "PrimitiveType": {
            "Int": "Isize"
          }
        }
      ],
      {
        9,
        {
          "PrimitiveType": {
            "UInt": "UB8"
          }
        }
      ],
      {
        10,
        {
          "EnumType": {
            "name": "std::result::Result<T, E/#1>",
            "adt_def": 14,
            "discriminants": [
              {
                0,
                0
              },
              {
                1,
                1
              }
            ]
          }
        }
      ],
      {
        15,
        {
          "StructType": {
            "name": "std::sys::pal::unix::process::process_common::Ex",
            "adt_def": 10
          }
        }
      ]
    ]
  },

```

Demo 1: stable-mir-json

- Stable MIR JSON Graph Display



runtime
verification

Semantics of Stable MIR

- Executing (Stable) MIR statements and terminators
- Computing results of basic primitive operations
 - Representing (primitive and aggregate) data at a high level
- Execution as *rewrite operations* to a *global execution state*
 - Program input is Stable MIR AST
 - Execution of given function as entry point (default: `main`)
- Semantics modelled using the *K Framework*



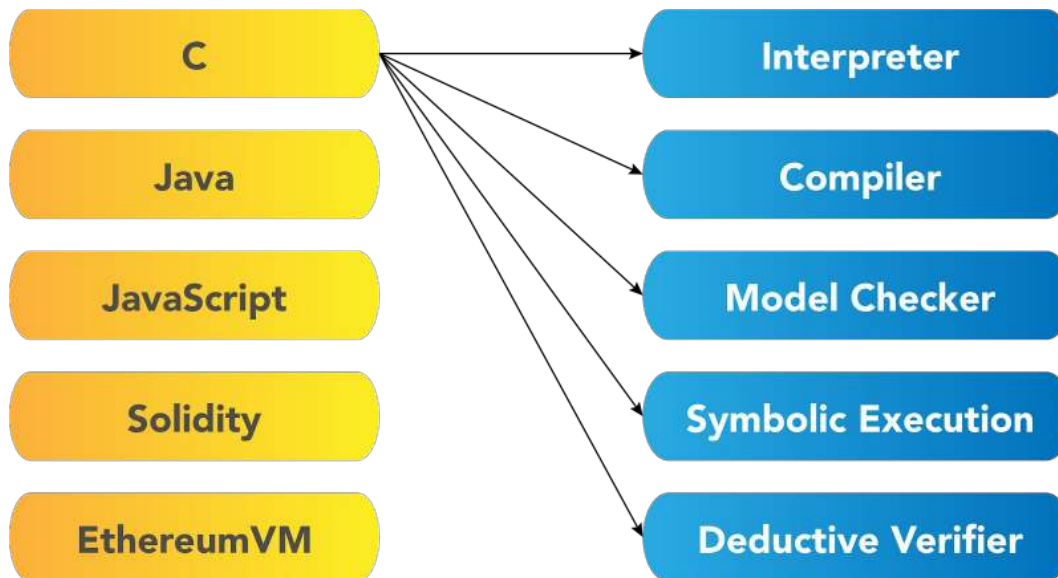
What is K Framework?



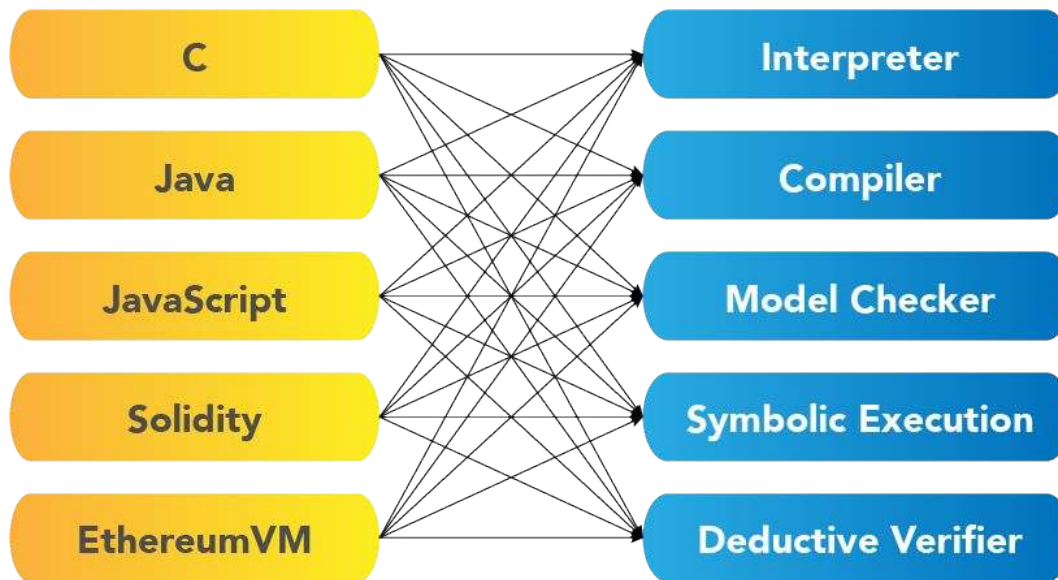
- <https://kframework.org> maintained by Runtime Verification / Pi²
- K is an *operational semantics framework* based on rewriting.
 - Specify your language or system as a K definition.
 - The K compiler derives a number of tools (parser, printer, interpreter, prover)
- Project started >15 years ago, building on earlier rewriting systems
- K's logical foundation is [Matching Logic](#)
- Given a K specification, there are two main backends you can use:
 - LLVM backend is for *concrete execution*, generating a fast interpreter.
 - Haskell backend is for *symbolic execution*: verification engine and model checker



The Problem: Too Many Tools

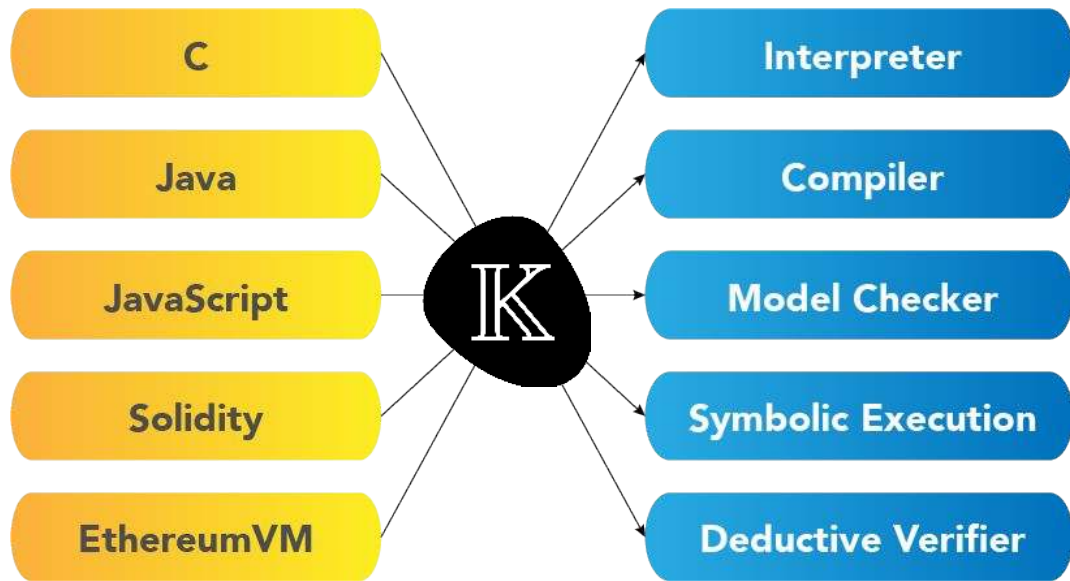


The Problem: Too Many Tools



The K Approach

- Develop each language definition and each tool once:



- Updates to tools benefit *all* the languages

Semantics for Stable MIR in K

- MIR AST as K data structures
 - 1:1 with `stable_mir` data structures (and additional context data)
 - Annotations drive a JSON parser
- Global configuration of running program
 - Control cell (`<k>`), call stack, local data
 - all program items (for call/return)
 - type metadata (for computation)
- Rewrite rules to model MIR execution
 - Driven by `<k>` cell (containing MIR)
 - Other cells manipulated as effects
 - Unchanged irrelevant cells omitted

```
module KMIR-CONFIGURATION
  imports KMIR-AST
  imports INT-SYNTAX
  imports BOOL-SYNTAX
  imports RT-VALUE-SYNTAX

  syntax RetVal ::= return( Value )
                  | "noReturn"

  syntax StackFrame ::= StackFrame( caller:Ty,           // index of caller function
                                     dest:Place,          // place to store return value
                                     target:MaybeBasicBlockIdx, // basic block to return to
                                     UnwindAction,         // action to perform on panic
                                     localsList )          // return val, args, local variables

  configuration <kmir>
    <k> #init($PGM:Pgm) </k>
    <retVal> noReturn </retVal>
    <currentFunc> ty(-1) </currentFunc> // to retrieve caller
    // unpacking the top frame to avoid frequent stack read/write operations
    <currentFrame>
      <currentBody> .List </currentBody>
```

```
rule <k> #execTerminator(terminator(terminatorKindReturn, _SPAN)) ~> _
  => #setLocalValue(DEST, #decrementRef(LOCAL0)) ~> #execBlockIdx(TARGET)
</k>
<currentFunc> _ => CALLER </currentFunc>
//<currentFrame>
  <currentBody> _ => #getBlocks(FUNCS, CALLER) </currentBody>
  <caller> CALLER => NEWCALLER </caller>
  <dest> DEST => NEWDEST </dest>
  <target> someBasicBlockIdx(TARGET) => NEWTARGET </target>
  <unwind> _ => UNWIND </unwind>
  <locals> ListItem(LOCAL0:TypedValue) _ => NEWLOCALS </locals>
//</currentFrame>
// remaining call stack (without top frame)
<stack> ListItem(StackFrame(NEWCALLER, NEWDEST, NEWTARGET, UNWIND, NEWLOCALS)) STACK => STACK </stack>
<functions> FUNCS </functions>
requires CALLER in_keys(FUNCS)
```



Demo 2: Executing MIR

Steps:

- Using programs from Demo 1
- `stable-mir-json my-program.rs`
- `kmir run my-program.smir.json --depth <N>`

Demo 2:

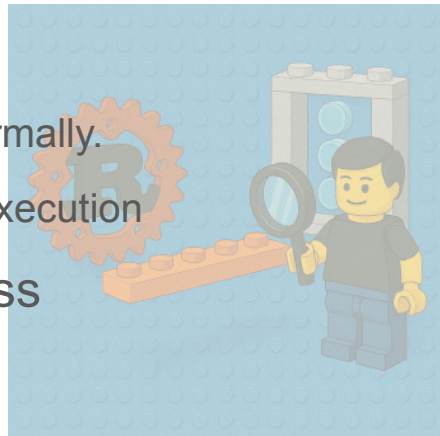
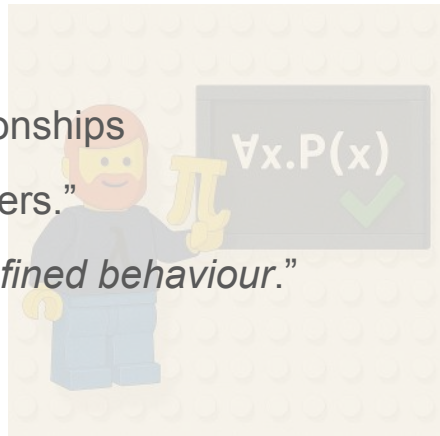
Executing MIR

- Example
 - SwitchInt branch
 - on a BoolVal
 - in local _5
- Note the data model
 - High level values for primitive types (u64 values here)
- Not shown:
 - Type mappings and metadata

```
<kmir>
<k>
  #execTerminator ( terminator (... kind: terminatorKindSwitchInt (... discr: operandMove ( place (... local: local ( 5 ) , projec>
</k>
<retVal>
  noReturn
</retVal>
<currentFunc>
  ty ( 25 )
</currentFunc>
<currentFrame>
  <currentBody>
    ListItem ( basicBlock (... statements: statement (... kind: statementKindAssign (... place: place (... local: local ( 5 ) , pro>
    ListItem ( basicBlock (... statements: statement (... kind: statementKindAssign (... place: place (... local: local ( 4 ) , pro>
    ListItem ( basicBlock (... statements: statement (... kind: statementKindAssign (... place: place (... local: local ( 4 ) , pro>
    ...
  </currentBody>
  <caller>
    ty ( -1 )
  </caller>
  <dest>
    place (... local: local ( 4 ) , projection: .ProjectionElems )
  </dest>
  <target>
    someBasicBlockIdx ( basicBlockIdx ( 1 ) )
  </target>
  <unwind>
    unwindActionContinue
  </unwind>
  <locals>
    ListItem ( newLocal ( ty ( 26 ) , mutabilityMut ) )
    ListItem ( typedValue ( Integer ( 42 , 64 , false ) , ty ( 26 ) , mutabilityNot ) )
    ListItem ( typedValue ( Integer ( 22 , 64 , false ) , ty ( 26 ) , mutabilityNot ) )
    ListItem ( typedValue ( Integer ( 0 , 64 , false ) , ty ( 26 ) , mutabilityNot ) )
    ListItem ( newLocal ( ty ( 26 ) , mutabilityNot ) )
    ListItem ( typedValue ( BoolVal ( false ) , ty ( 29 ) , mutabilityMut ) )
    ListItem ( newLocal ( ty ( 29 ) , mutabilityMut ) )
    ListItem ( newLocal ( ty ( 26 ) , mutabilityMut ) )
  </locals>
</currentFrame>
<stack>
  ListItem ( StackFrame ( ty ( -1 ) , place (... local: local ( -1 ) , projection: .ProjectionElems ) , noBasicBlockIdx , unwindActi>
  ...
</stack>
</kmir>
```

Property Specification in K

- What are Properties?
 - Formal statements about program behavior and input/output relationships
 - “This function computes the maximum of three integral numbers.”
 - “For all *valid* inputs, function *f* executes *without* causing *undefined behaviour*.”
- How to specify properties in KMIR
 - We **execute** a program (fragment) **with symbolic data**
 - ... and formulate **expectations** about the result, in a **K claim**
 - **kmir gen-spec** generates a claim that a program terminates normally.
 - Other properties can use configuration fragments extracted from execution
- **Currently a manual process** – automation work in progress



Demo 3:

Verification of a Property in KMIR

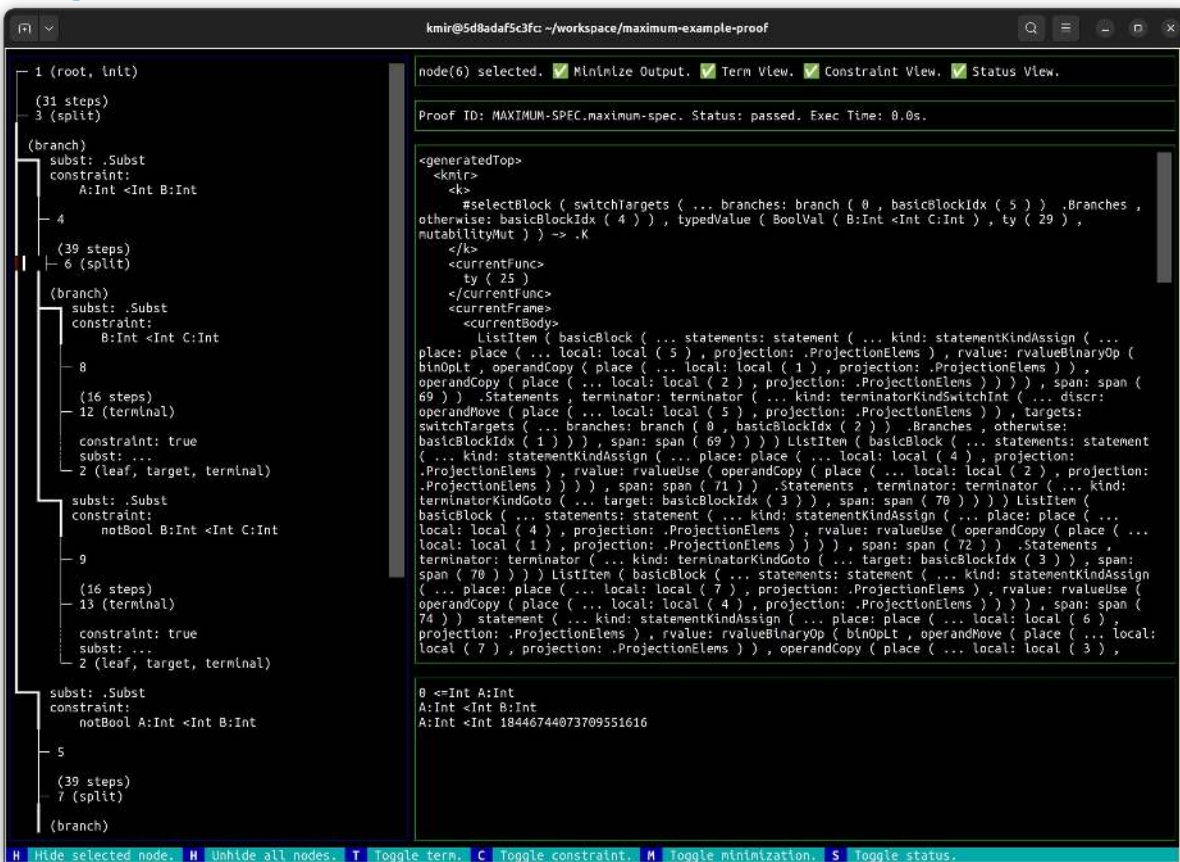
(semi-manual at the moment)

- Steps
 - Use the JSON output from Demo 1.
 - Generate claim using `gen-spec`, or/and `maximum-spec.k`
 - `kmir prove run` to verify property
 - `kmir prove view` to view proof

Demo 3:

Verification of a Property in KMIR

- Execution branches depending on the symbolic values
 - `kmir prove view` to inspect
 - Each branch carries its own *path conditions* (bottom)
- A successful proof means:
 - All branches reach a configuration consistent with the target state in the claim (“all-path reachability”).
 - The configurations *unify*,
 - accumulated *path conditions* imply target conditions.



The screenshot shows the KMIR verification tool interface. The left pane displays a control flow graph (CFG) with nodes and edges, including constraints like `notBool B:Int <Int C:Int`. The right pane shows the generated KMIR code, which includes a `selectBlock` for handling branches. The bottom status bar indicates the proof is successful: "Proof ID: MAXIMUM-SPEC.maximun-spec. Status: passed. Exec Time: 0.0s." The bottom of the interface has a row of toggle buttons: Hide selected node, Unhide all nodes, Toggle term, Toggle constraint, Toggle minimization, and Toggle status.

```
1 (root, {init})
  (31 steps)
  3 (split)
    (branch)
      subst: .Subst
      constraint:
        A:Int <Int B:Int
      4
        (39 steps)
        6 (split)
          (branch)
            subst: .Subst
            constraint:
              B:Int <Int C:Int
            8
              (16 steps)
              12 (terminal)
                constraint: true
                subst: ...
                2 (leaf, target, terminal)
            subst: .Subst
            constraint:
              notBool B:Int <Int C:Int
            9
              (16 steps)
              13 (terminal)
                constraint: true
                subst: ...
                2 (leaf, target, terminal)
            subst: .Subst
            constraint:
              notBool A:Int <Int B:Int
            5
              (39 steps)
              7 (split)
                (branch)
```

node(6) selected. ☒ Minimize Output. ☒ Term View. ☒ Constraint View. ☒ Status View.

Proof ID: MAXIMUM-SPEC.maximun-spec. Status: passed. Exec Time: 0.0s.

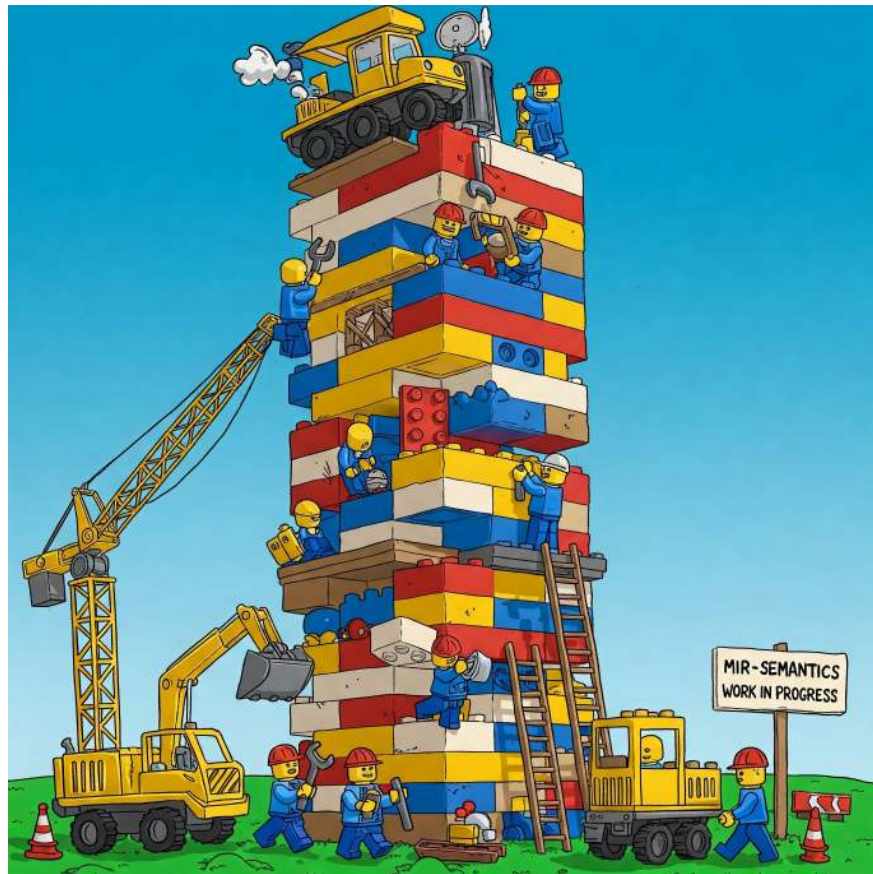
```
<generatedTop>
<kmir>
<k>
  #selectBlock ( switchTargets ( ... branches: branch ( 0 , basicBlockIdx ( 5 ) ) .Branches ,
otherwise: basicBlockIdx ( 4 ) ) , typedValue ( BoolVal ( B:Int <Int C:Int ) , ty ( 29 ) ,
mutabilityMut ) ) -> .K
</k>
<currentFunc>
  ty ( 25 )
</currentFunc>
<currentFrame>
  <currentBody>
    ListItem ( basicBlock ( ... statements: statement ( ... kind: statementKindAssign ( ...
place: place ( ... local: local ( 5 ) , projection: .ProjectionElems ) , rvalue: rvalueBinaryOp (
binOpLt , operandCopy ( place ( ... local: local ( 1 ) , projection: .ProjectionElems ) ) ,
operandCopy ( place ( ... local: local ( 2 ) , projection: .ProjectionElems ) ) ) , span: span (
69 ) ) .Statements , terminator: terminator ( ... kind: terminatorKindSwitchInt ( ... discr:
operandMove ( place ( ... local: local ( 5 ) , projection: .ProjectionElems ) ) , targets:
switchTargets ( ... branches: branch ( 0 , basicBlockIdx ( 2 ) ) .Branches , otherwise:
basicBlockIdx ( 1 ) ) ) , span: span ( 69 ) ) ) ListItem ( basicBlock ( ... statements: statement
( ... kind: statementKindAssign ( ... place: place ( ... local: local ( 4 ) , projection:
.ProjectionElems ) , rvalue: rvalueUse ( operandCopy ( place ( ... local: local ( 2 ) , projection:
.ProjectionElems ) ) ) , span: span ( 71 ) ) .Statements , terminator: terminator ( ... kind:
terminatorKindGoto ( ... target: basicBlockIdx ( 3 ) ) , span: span ( 70 ) ) ) ) ListItem (
basicBlock ( ... statements: statement ( ... kind: statementKindAssign ( ... place: place ( ...
local: local ( 4 ) , projection: .ProjectionElems ) , rvalue: rvalueUse ( operandCopy ( place ( ...
local: local ( 1 ) , projection: .ProjectionElems ) ) ) , span: span ( 72 ) ) .Statements ,
terminator: terminator ( ... kind: terminatorKindGoto ( ... target: basicBlockIdx ( 3 ) ) , span:
span ( 70 ) ) ) ) ListItem ( basicBlock ( ... statements: statement ( ... kind: statementKindAssign
( ... place: place ( ... local: local ( 7 ) , projection: .ProjectionElems ) , rvalue: rvalueUse (
operandCopy ( place ( ... local: local ( 4 ) , projection: .ProjectionElems ) ) ) , span: span (
74 ) ) statement ( ... kind: statementKindAssign ( ... place: place ( ... local: local ( 6 ) ,
projection: .ProjectionElems ) , rvalue: rvalueBinaryOp ( binOpLt , operandMove ( place ( ... local:
local ( 7 ) , projection: .ProjectionElems ) ) , operandCopy ( place ( ... local: local ( 3 ) ,
```

0 <=Int A:Int
A:Int <Int B:Int
A:Int <Int 18446744073709551616

Hide selected node. Unhide all nodes. Toggle term. Toggle constraint. Toggle minimization. Toggle status.

KMIR: Status and Limitations

- KMIR is **under active development**
 - Supports **most primitive types** and **structs**
 - Support for **enums/arrays** in progress
 - Next on the list: Support for **heap operations** and pointers/addresses
 - Automation of **claim generation** and **custom start symbols** in progress
 - No multi-crate support yet
- Limitations
 - No support for inline ASM (obvious)
 - High-level data model (difficult to support **transmute**)



Summary: The three pillars of KMIR

- **Stable MIR JSON:**
 - Extracts Stable MIR from `rustc` into a simple JSON format
- **Semantics of Stable MIR** in K
 - Models Stable MIR operational semantics with **K Framework**
 - Program execution as rewrites to a global configuration
 - Symbolic execution of Rust and property proofs (reachability logic)
- **KMIR frontend:**
 - Drives the Stable MIR execution and property proofs
 - Uses K's Python bindings

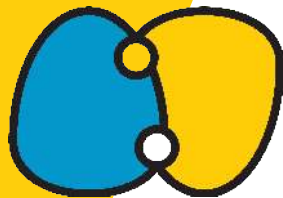


Learn More



- Source code on Github
 - <https://github.com/runtimeverification/mir-semantics>
 - <https://github.com/runtimeverification/stable-mir-json>
- Docker image:
 - <https://hub.docker.com/r/runtimeverificationinc/kmir/tags>
- K Framework Documentation
 - <https://kframework.org>

- <https://github.com/runtimeverification/mir-semantics>
- <https://github.com/runtimeverification/stable-mir-json>
- <https://hub.docker.com/r/runtimeverificationinc/kmir/tags>
- <https://kframework.org>



Questions?

 <https://runtimeverification.com/>

 @rv_inc

 <https://discord.com/invite/CurfmXNtbN>

 contact@runtimeverification.com

Picture Warehouse

