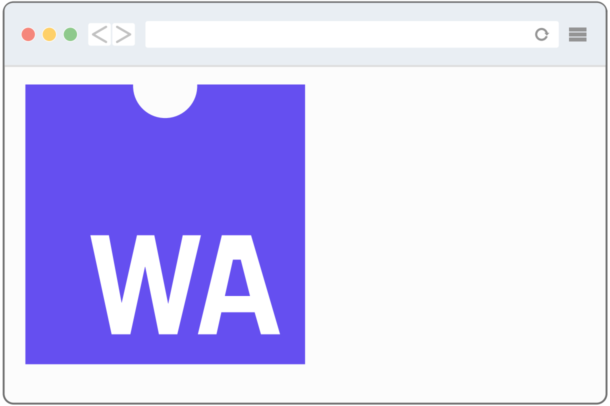
Wappler:
Sound Reachability Analysis for WebAssembly

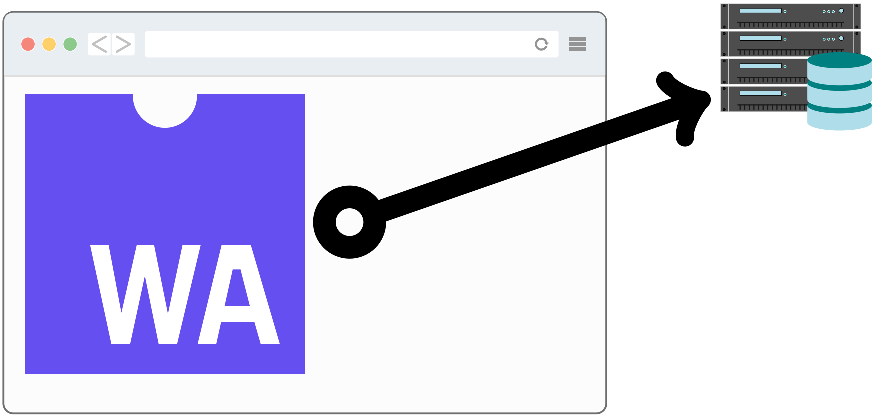
Markus Scherer · Jeppe Fredsgaard Blaabjerg ·
Magdalena Solitro · Alexander Sjösten ·
Matteo Maffei

CSF 2024
July 10, 2024

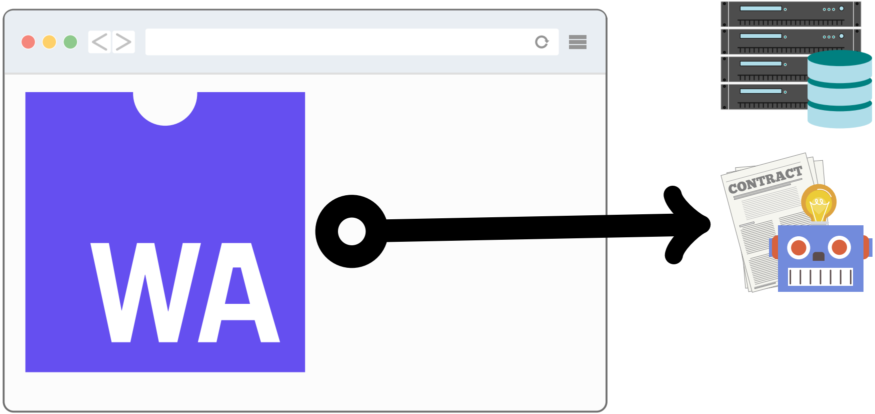
WebAssembly



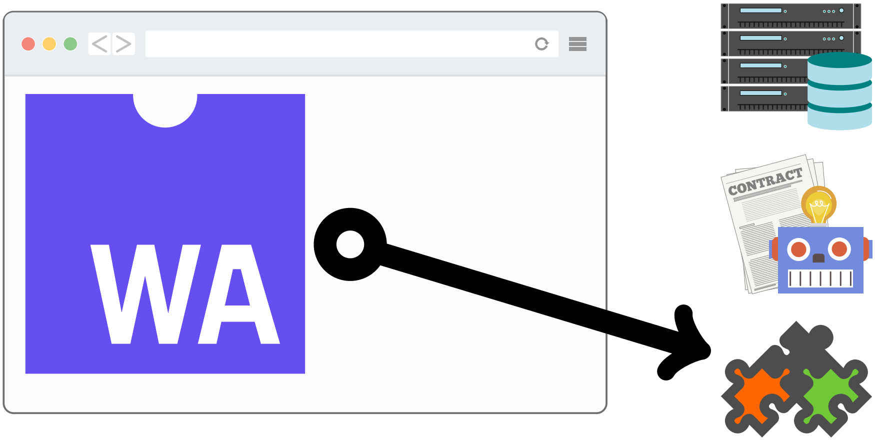
WebAssembly



WebAssembly



WebAssembly



Motivation

```c
static const char* trusted = "TRUSTED";
char game_state[64] = ...;
extern void eval(const char*);

void update_game_state(int x, int y, char c) {
  if (x < 8 && y < 8) {
    game_state[y * 8 + x] = c;
  }
  eval(TRUSTED);
}
```

Motivation

```c
static const char* trusted = "TRUSTED";
char game_state[64] = ...;
extern void eval(const char*);

void update_game_state(int x, int y, char c) {
  if (x < 8 && y < 8) {
    game_state[y * 8 + x] = c;
  }
  eval(TRUSTED);
}
```

not meant to be modified

Motivation

```
static const char* trusted = "TRUSTED";
char game_state[64] = ...;
extern void eval(const char*);

void update_game_state(int x, int y, char c) {
  if (x < 8 && y < 8) {
    game_state[y * 8 + x] = c;
  }
  eval(TRUSTED);
}
```

assumes trusted input

Motivation

```
static const char* trusted = "TRUSTED";
char game_state[64] = ...;
extern void eval(const char*);

void update_game_state(int x, int y, char c) {
  if (x < 8 && y < 8) {
    game_state[y * 8 + x] = c;
  }
  eval(TRUSTED);
}
```

insufficient constraints

Motivation

```c
static const char* trusted = "TRUSTED";
char game_state[64] = ...;
extern void eval(const char*);

void update_game_state(int x, int y, char c) {
  if (x < 8 && y < 8) {
    game_state[y * 8 + x] = c;
  }
  eval(TRUSTED);
}
```

can overflow

Motivation

```
static const char* trusted = "TRUSTED";
char game_state[64] = ...;
extern void eval(const char*);

void update_game_state(int x, int y, char c) {
  if (x < 8 && y < 8) {
    game_state[y * 8 + x] = c;
  }
  eval(TRUSTED);
}
```

can overwrite trusted

Motivation

```
static const char* trusted = "TRUSTED";
char game_state[64] = ...;
extern void eval(const char*);

void update_game_state(int x, int y, char c) {
  if (x < 8 && y < 8) {
    game_state[y * 8 + x] = c;
  }
  eval(TRUSTED);   !!!
}
```

Motivation

```c
static const char* trusted = "TRUSTED";
char game_state[64] = ...;
extern void eval(const char*);

void update_game_state(int x, int y, char c) {
  if (x < 8 && y < 8) {
    game_state[y * 8 + x] = c;
  }
  eval(TRUSTED);
}
```

Everything Old is New Again: Binary Security of WebAssembly

Daniel Lehmann, *University of Stuttgart*
Johannes Kinder, *Bundeswehr University Munich*
Michael Pradel, *University of Stuttgart*

Abstract

WebAssembly is an increasingly popular compilation target designed to run code in browsers and on other platforms safely and securely, by strictly separating code and data, enforcing

both based on LLVM. Originally devised for client-side computation in browsers, WebAssembly's simplicity and generality has sparked interest to use it as a platform for many other domains, e.g., on the server side in conjunction with Node.js, for "serverless" cloud computing [33–35, 64]. Internet of

Safety Properties

```
static const char* trusted = "TRUSTED";
char game_state[64] = ...;
extern void eval(const char*);

void update_game_state(int x, int y, char c) {
  if (x < 8 && y < 8) {
    game_state[y * 8 + x] = c;
  }
  eval(TRUSTED);
}
```

Safety Properties

```
static const char* trusted = "TRUSTED";
char game_state[64] = ...;
extern void eval(const char*);

void update_game_state(int x, int y, char c) {
  if (x < 8 && y < 8) {
    game_state[y * 8 + x] = c;
  }
  eval(TRUSTED);
}
```

No-i32-Overflow

Safety Properties

```c
static const char* trusted = "TRUSTED";
char game_state[64] = ...;
extern void eval(const char*);

void update_game_state(int x, int y, char c) {
  if (x < 8 && y < 8) {
    game_state[y * 8 + x] = c;
  }
  eval(TRUSTED);
}
```

No-Sensitive-Overwrite

Safety Properties

```
static const char* trusted = "TRUSTED";
char game_state[64] = ...;
extern void eval(const char*);

void update_game_state(int x, int y, char c) {
  if (x < 8 && y < 8) {
    game_state[y * 8 + x] = c;
  }
  eval(TRUSTED);   assert(trusted == "TRUSTED")
}
```

Safety Properties

```
static const char* trusted = "TRUSTED";
char game_state[64] = ...;
extern void eval(const char*);

void update_game_state(int x, int y, char c) {
  if (0 <= x && x < 8 && 0 <= y && y < 8) {
    game_state[y * 8 + x] = c;
  }
  eval(TRUSTED);
}
```

The Semantics of WebAssembly

$S; F; instr^*$

global data (memory, globals, function definitions)

The Semantics of WebAssembly

$S; F; instr^*$

function local data (locals)

The Semantics of WebAssembly

$S; F; instr^*$

stack (values, control flow data, instructions)

The Semantics of WebAssembly

$S; F; instr^*$

The Semantics of WebAssembly

$S; F; instr^*$

$E[(\mathrm{i32}.\mathbf{const}\ 2)\ (\mathrm{i32}.\mathbf{const}\ 3)\ \mathrm{i32}.\mathbf{add}]$

The Semantics of WebAssembly

$S; F; instr^*$

$E[(\text{i32}.\textbf{const}\ 2)\ (\text{i32}.\textbf{const}\ 3)\ \text{i32}.\textbf{add}]$

The Semantics of WebAssembly

$S; F; instr^*$

$E[(\text{i32.}\mathbf{const}\ 2)\ (\text{i32.}\mathbf{const}\ 3)\ \text{i32.}\mathbf{add}]$

$S; F; (\text{i32.}\mathbf{const}\ c_1)\ (\text{i32.}\mathbf{const}\ c_2)\ \text{i32.}\mathbf{add} \hookrightarrow S; F; (\text{i32.}\mathbf{const}\ c_1 + c_2)$

The Semantics of WebAssembly

$S; F; instr^*$

$E[(\text{i32}.\textbf{const}\ 5)]$

$S; F; (\text{i32}.\textbf{const}\ c_1)\ (\text{i32}.\textbf{const}\ c_2)\ \text{i32}.\textbf{add} \hookrightarrow S; F; (\text{i32}.\textbf{const}\ c_1 + c_2)$

The Semantics of WebAssembly

$$S; F; instr^* \quad \hookrightarrow \quad S'; F'; instr'^*$$

The Semantics of WebAssembly

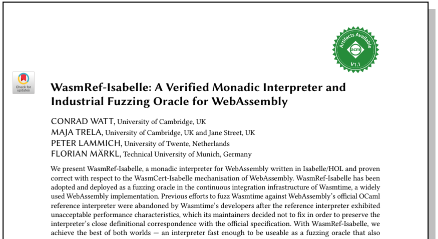
$$S; F; instr^* \hookrightarrow S'; F'; instr'^*$$

The Semantics of WebAssembly

$S; F; instr^* \hookrightarrow S'; F'; instr'^*$

The Semantics of WebAssembly

$S; F; instr^* \hookrightarrow S'; F'; instr'^*$



WasmRef-Isabelle: A Verified Monadic Interpreter and Industrial Fuzzing Oracle for WebAssembly

CONRAD WATT, University of Cambridge, UK
MAJA TRELA, University of Cambridge, UK and Jane Street, UK
PETER LAMMICH, University of Twente, Netherlands
FLORIAN MÄRKL, Technical University of Munich, Germany

We present WasmRef-Isabelle, a monadic interpreter for WebAssembly written in Isabelle/HOL and proven correct with respect to the WasmCert-Isabelle mechanisation of WebAssembly. WasmRef-Isabelle has been adopted and deployed as a fuzzing oracle in the continuous integration infrastructure of Wasmtime, a widely used WebAssembly implementation. Previous efforts to fuzz Wasmtime against WebAssembly's official OCaml reference interpreter were abandoned by Wasmtime's developers after the reference interpreter exhibited unacceptable performance characteristics, which its maintainers decided not to fix in order to preserve the interpreter's close definitional correspondence with the official specification. With WasmRef-Isabelle, we achieve the best of both worlds — an interpreter fast enough to be useable as a fuzzing oracle that also

Horn-Clause-Based Abstractions

$S; F; instr^*$

Horn-Clause-Based Abstractions

$$S; F; instr^* \xrightarrow{\qquad\qquad\alpha\qquad\qquad} \{MState(\ldots), Table(\ldots), \ldots\}$$

Horn-Clause-Based Abstractions

$$S; F; instr^* \xrightarrow{\quad\quad\quad\alpha\quad\quad\quad} \{MState(\ldots), Table(\ldots), \ldots\}$$

$$S; F; (\text{i32}.\textbf{const}\ c_1)\ (\text{i32}.\textbf{const}\ c_2)\ \text{i32}.\textbf{add}$$

$$\hookrightarrow S; F; (\text{i32}.\textbf{const}\ c_1 + c_2)$$

Horn-Clause-Based Abstractions

$$S; F; instr^* \xrightarrow{\qquad\qquad\qquad\alpha\qquad\qquad\qquad} \{MState(\ldots), Table(\ldots), \ldots\}$$

$$S; F; (\text{i32.}\textbf{const}\ c_1)\ (\text{i32.}\textbf{const}\ c_2)\ \text{i32.}\textbf{add} \hookrightarrow S; F; (\text{i32.}\textbf{const}\ c_1 + c_2) \xrightarrow{\alpha} \{MState(c_1 : c_2 : st \ldots) \implies MState((c_1 + c_2) : st \ldots)\}$$

Horn-Clause-Based Abstractions

$$S; F_{fid}; instr^* \xrightarrow{\qquad\qquad\qquad \alpha \qquad\qquad\qquad} \{MState(\ldots), Table(\ldots), \ldots\}$$

$$S; F_{fid}; (\text{i32.}\textbf{const}\ c_1)\ (\text{i32.}\textbf{const}\ c_2)\ \text{i32.}\textbf{add}_{pc} \xrightarrow{\ \alpha\ } \{MState(c_1 : c_2 : st\ldots) \implies$$

$$\hookrightarrow S; F_{fid}; (\text{i32.}\textbf{const}\ c_1 + c_2) \qquad\qquad MState((c_1 + c_2) : st\ldots)\}$$

Horn-Clause-Based Abstractions

$$S; F_{fid}; instr^* \xrightarrow{\qquad\qquad\qquad\alpha\qquad\qquad\qquad} \{MState_{fid,pc}(\ldots), Table(\ldots), \ldots\}$$

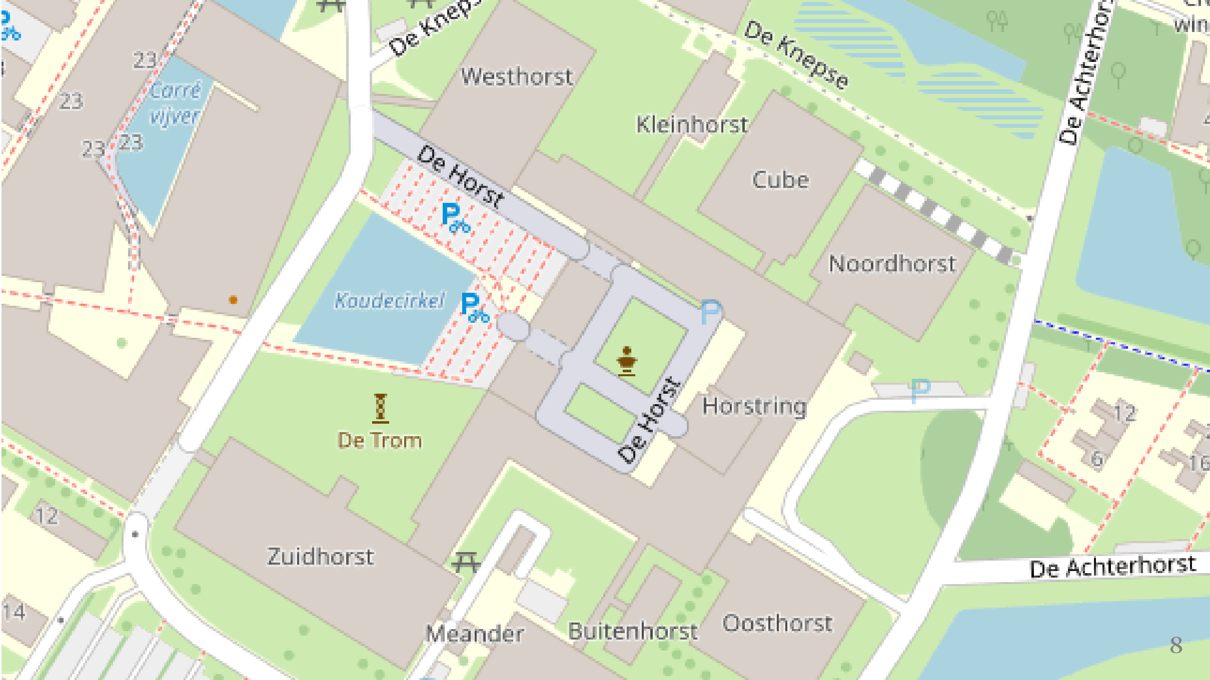
$$\begin{array}{lcl} S; F_{fid}; (\text{i32}.\textbf{const}\ c_1)\ (\text{i32}.\textbf{const}\ c_2)\ \text{i32}.\textbf{add}_{pc} & \xrightarrow{\alpha} & \{MState_{fid,pc}(c_1 : c_2 : st\ldots) \implies \\ \qquad \hookrightarrow S; F_{fid}; (\text{i32}.\textbf{const}\ c_1 + c_2) & & \qquad MState_{fid,pc+1}((c_1 + c_2) : st\ldots)\} \end{array}$$

Horn-Clause-Based Abstractions

$$S; F_{fid}; instr^* \xrightarrow{\qquad\qquad\alpha\qquad\qquad} \{MState_{fid,pc}(\ldots), Table(\ldots), \ldots\}$$

$$S; F_{fid}; (\text{i32}.\textbf{const}\ c_1)\ (\text{i32}.\textbf{const}\ c_2)\ \text{i32}.\textbf{add}_{pc} \xrightarrow{\alpha} \{MState_{fid,pc}(c_1 : c_2 : st\ldots) \Longrightarrow$$

$$\hookrightarrow S; F_{fid}; (\text{i32}.\textbf{const}\ c_1 + c_2) \qquad MState_{fid,pc+1}((c_1 + c_2) : st\ldots)\}$$



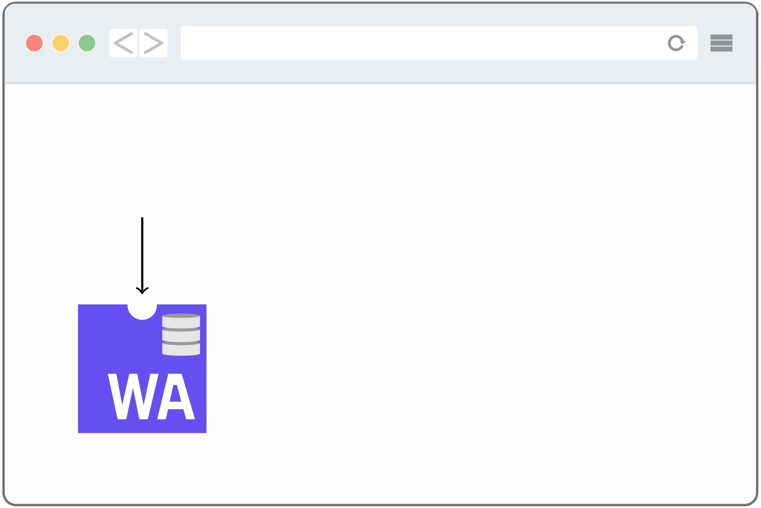
Westhorst
Kleinhorst
Cube
Noordhorst
De Knepse
De Achterhorst
De Horst
Carré vijver
Koudecirkel
De Trom
Horstring
Zuidhorst
Meander
Buitenhorst
Oosthorst

Implementation

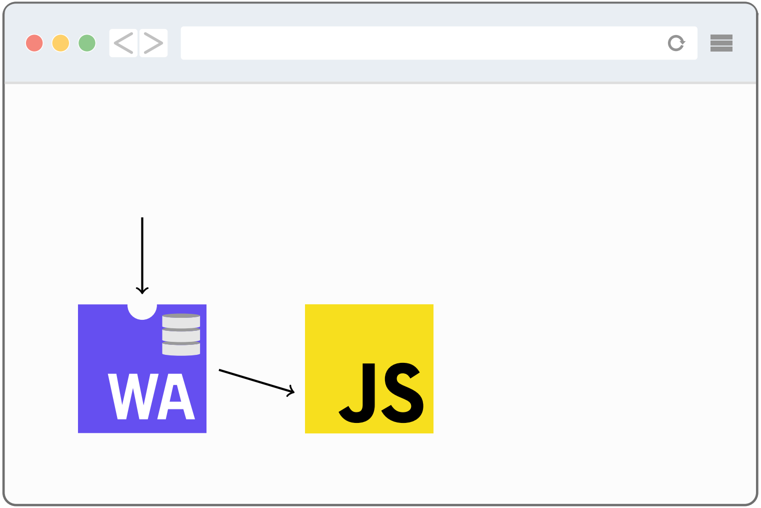
```
rule binOpRule := for
  (!fid: int) in functionIds(),
  (!op: int) in binOps(),
  (!pc: int) in pcsForFunctionIdAndOpcode(!fid, !op)

  clause [?x: Value, ?y: Value, ?st: tuple<Value; ss{!fid,!pc}()-2>,
          ?gt: tuple<Value; gs()>, ?lt: tuple<Value; ls{!fid}()>,
          ?mem: Memory, ?at0: tuple<Value; as{!fid}()>,
          ?gt0: tuple<Value; gs()>, ?mem0: Memory]

    MState{!fid, !pc}(?x :: ?y :: ?st, ?gt, ?lt, ?mem, ?at0, ?gt0, ?mem0)
      => MState{!fid, !pc + 1}(binOp{!op}(?y, ?x) :: ?st, ?gt, ?lt, ?mem, ?at0, ?gt0, ?mem0)
  ;
```

Interaction With Embedders

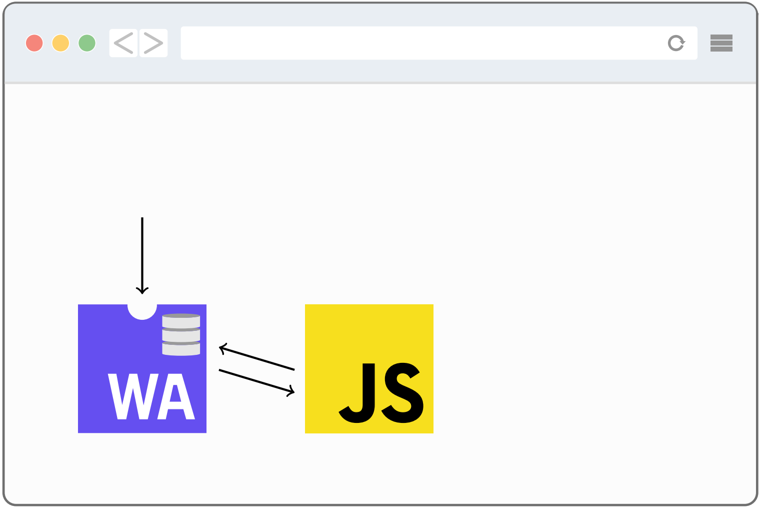
Interaction With Embedders



Interaction With Embedders



Interaction With Embedders



Interaction With Embedders

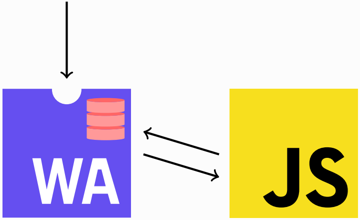
$$S; F; val^n\ (\mathbf{call}\ a) \hookrightarrow S'; F; result$$

where a is a host function

Interaction With Embedders

$$S; F; val^n\ (\mathbf{call}\ a) \hookrightarrow S'; F; result$$

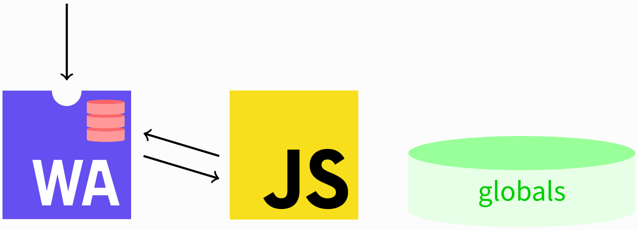
where a is a host function



Interaction With Embedders

$$S; F; val^n\ (\mathbf{call}\ a) \hookrightarrow S'; F; result$$

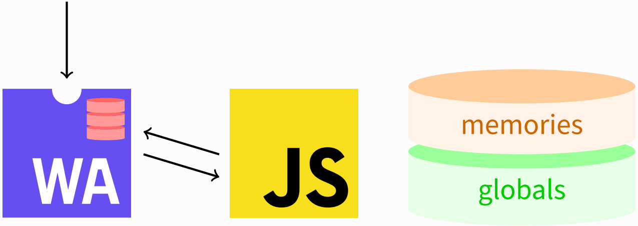
where a is a host function



Interaction With Embedders

$$S; F; val^n\ (\mathbf{call}\ a) \hookrightarrow S'; F; result$$

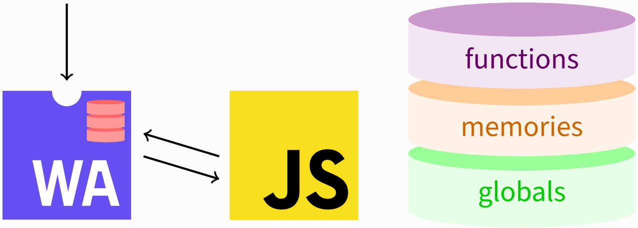
where a is a host function



Interaction With Embedders

$S; F; val^n\ (\mathbf{call}\ a) \hookrightarrow S'; F; result$

where a is a host function



Evaluation

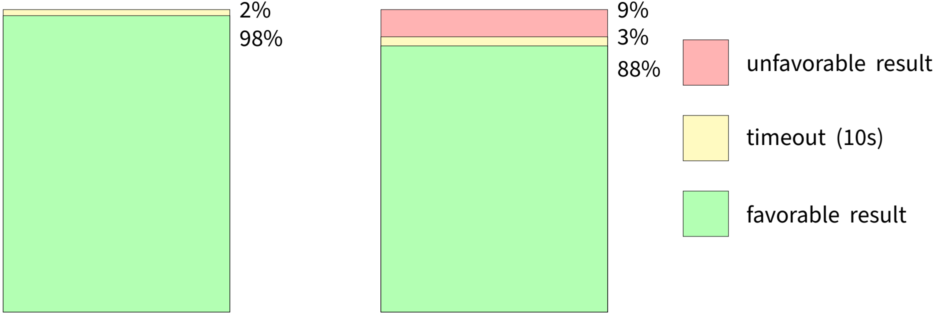
Evaluation

Is the required state reachable?

Evaluation

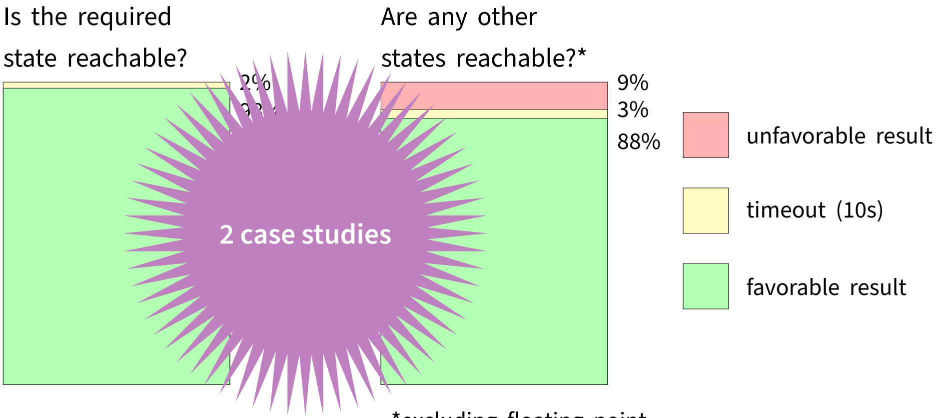
Is the required state reachable?

Are any other states reachable?*



*excluding floating point

Evaluation



*excluding floating point

Conclusions / Future Work

- Wappler is the first *sound* reachability analysis approach for WebAssembly
- possible improvements
  - support multiple modules
  - support floating points
  - increase efficiency of memory handling
  - analysis-specific abstractions
  - increase composability

https://secpriv.wien/wappler