A Verified, Analysis-Focused Interpreter for WebAssembly

Florian Märkl
Technical University of Munich

October 6, 2021

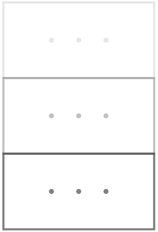
WebAssembly

WebAssembly

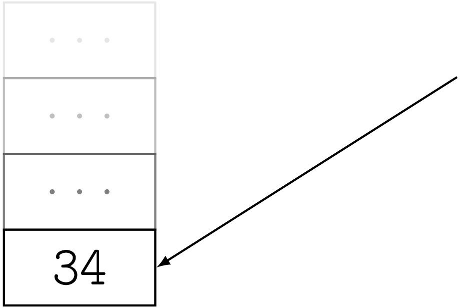
ARM,
RISC-V,
PowerPC,
x86, ...

JVM/Dalvik
Bytecode

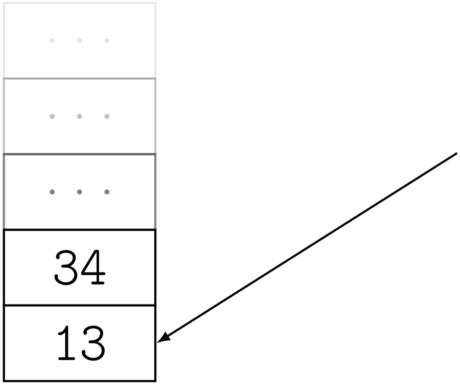
Low-level → High-level



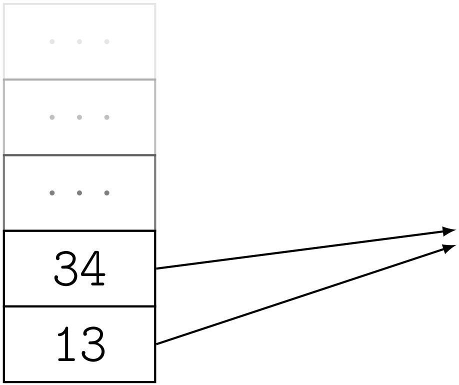
```
i32.const 34
i32.const 13
i32.sub
i32.const 2
i32.mul
```



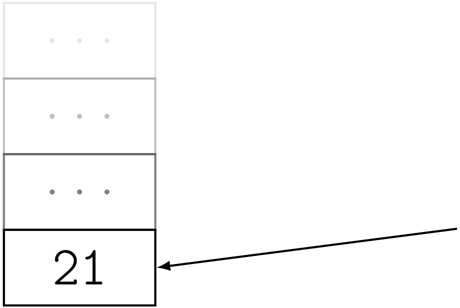

```
i32.const 34
i32.const 13
i32.sub
i32.const 2
i32.mul
```



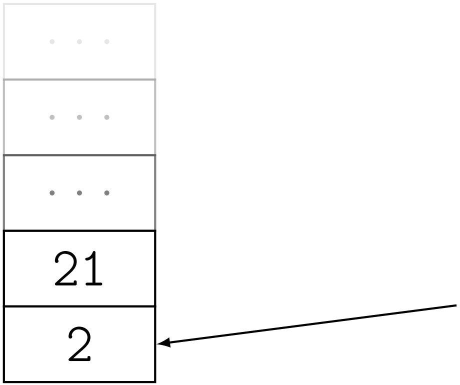

```
i32.const 34
i32.const 13
i32.sub
i32.const 2
i32.mul
```



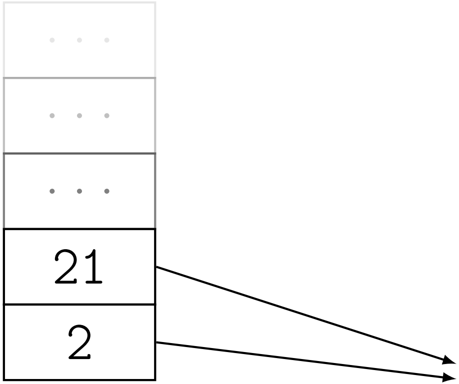

```
i32.const 34
i32.const 13
i32.sub
i32.const 2
i32.mul
```




```
i32.const 34
i32.const 13
i32.sub
i32.const 2
i32.mul
```



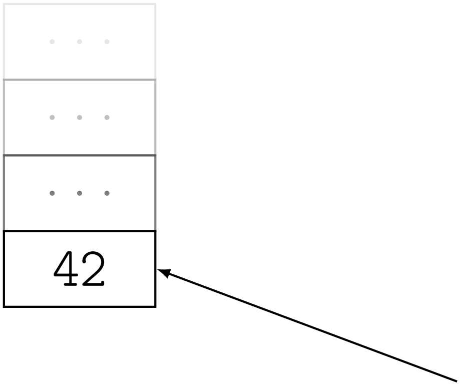

```
i32.const 34
i32.const 13
i32.sub
i32.const 2
i32.mul
```




```
i32.const 34
i32.const 13
i32.sub
i32.const 2
i32.mul
```

WebAssembly

Stack Machine



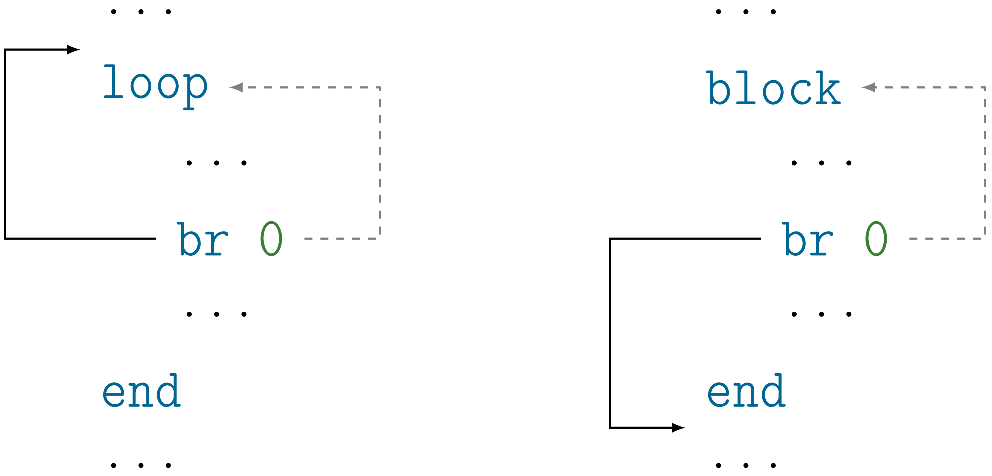

```
i32.const 34
i32.const 13
i32.sub
i32.const 2
i32.mul
```

```
...
loop

    ...

end
...
```

```
...
loop
    ...
    br 0
    ...
end
...
```

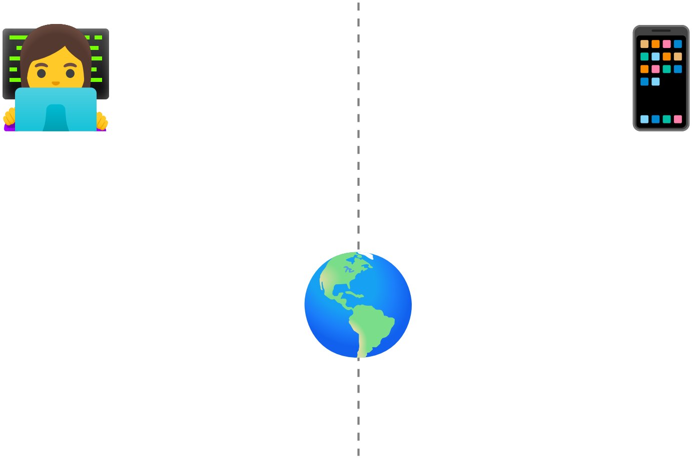
```
...
loop
    ...
    br 0
    ...
end
...
```

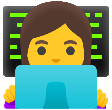
```
...
loop
   ...
   br 0
   ...
end
...
```


...
loop
...
br 0
...
end
...
...
block
...
br 0
...
end
...

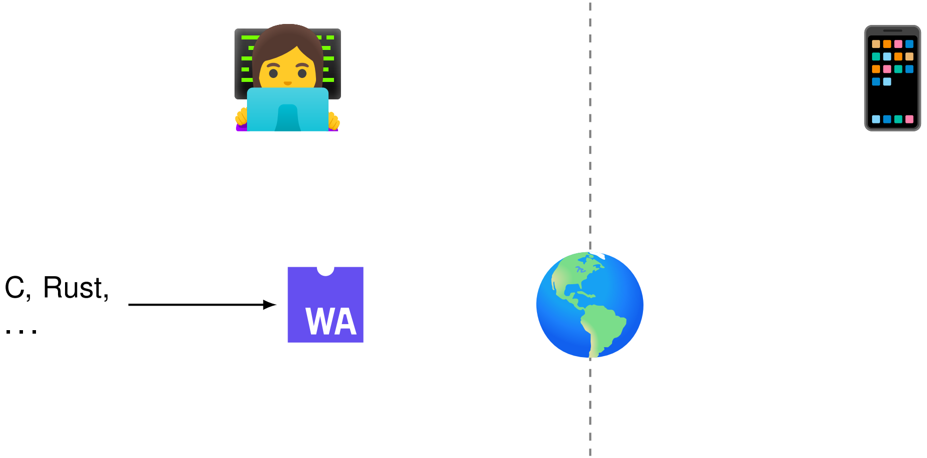
Motivation

Optimization





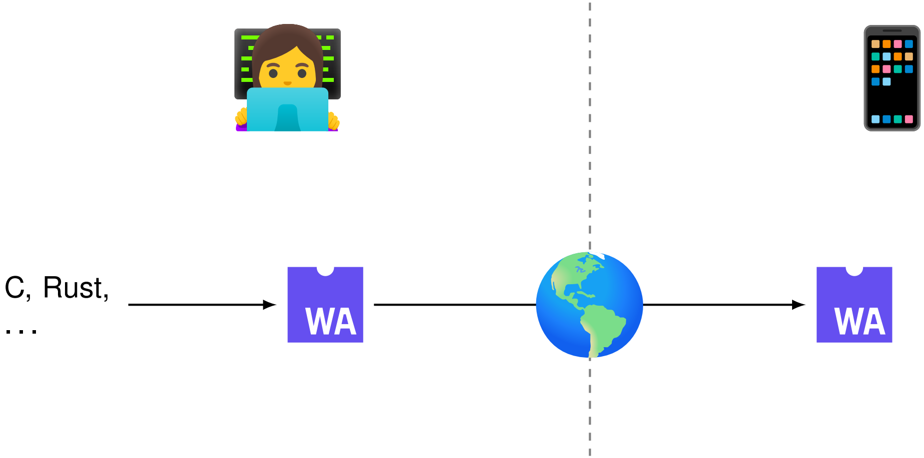
C, Rust,
...



C, Rust,
. . .
WA

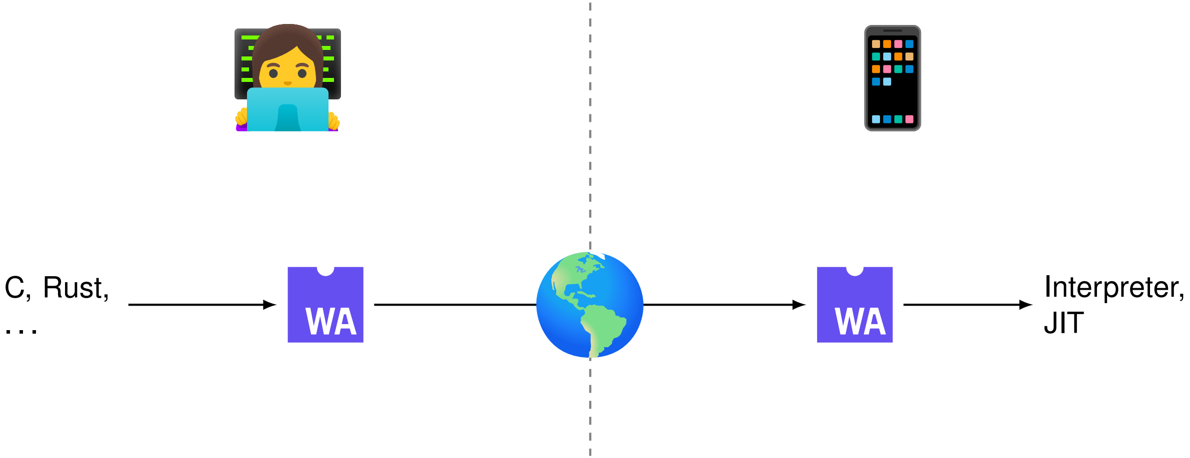
Motivation

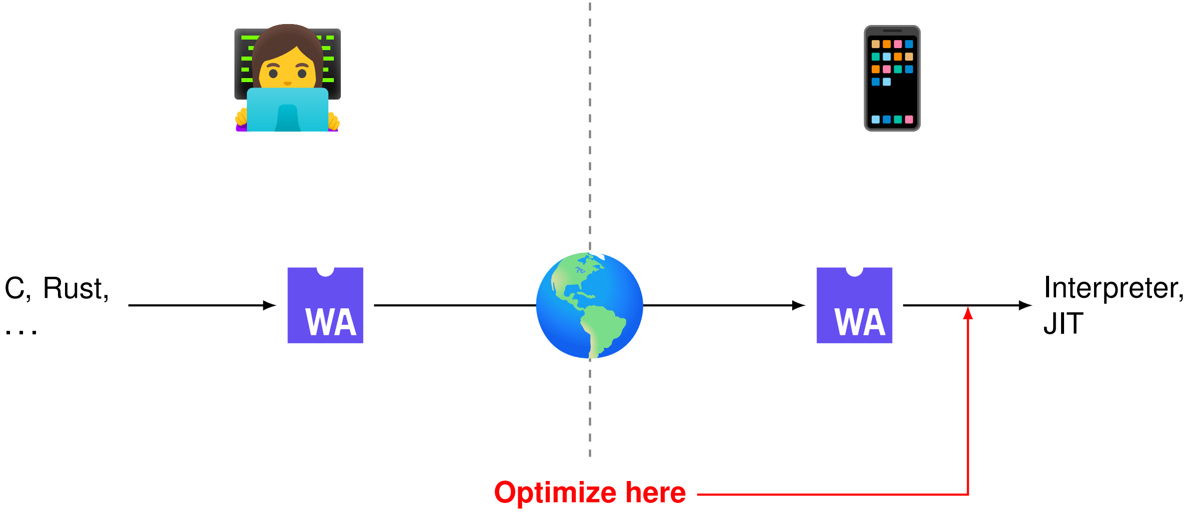
Optimization



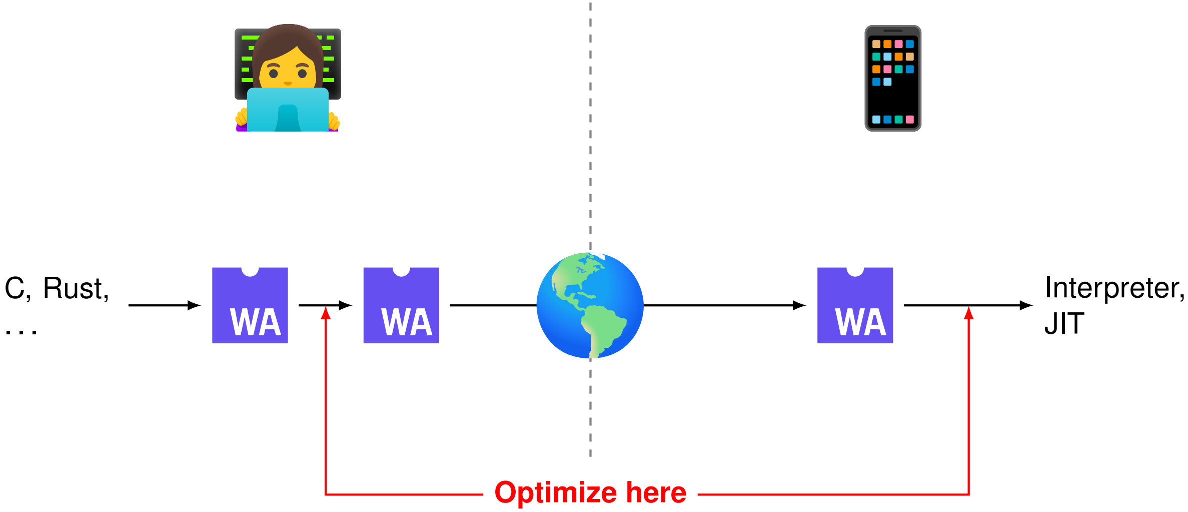
Motivation

Optimization





C, Rust, ...
WA
WA
Interpreter, JIT
Optimize here



C, Rust,
...
WA
WA
WA
Interpreter,
JIT
Optimize here

Example: Dead Code Elimination

Example: Dead Code Elimination

- What is dead code?

Example: Dead Code Elimination

- What is dead code?
- e.g. if-branch with condition that is always false

Example: Dead Code Elimination

- What is dead code?
- e.g. if-branch with condition that is always false
- Abstract Interpretation
  - **overapproximates** a property (e.g. value sets) for all possible executions of a program

Example: Dead Code Elimination

- What is dead code?
- e.g. if-branch with condition that is always false
- Abstract Interpretation
  - **overapproximates** a property (e.g. value sets) for all possible executions of a program
  - by attaching information to each **program point**

WebAssembly Specification

webassembly.github.io

t.*binop*

1. Assert: due to validation, two values of value type t are on the top of the stack.
2. Pop the value $t.\mathsf{const}\ c_2$ from the stack.
3. Pop the value $t.\mathsf{const}\ c_1$ from the stack.
4. If $binop_t(c_1, c_2)$ is defined, then:
   a. Let c be a possible result of computing $binop_t(c_1, c_2)$.
   b. Push the value $t.\mathsf{const}\ c$ to the stack.
5. Else:
   a. Trap.

$$
\begin{array}{llll}
(t.\mathsf{const}\ c_1)\ (t.\mathsf{const}\ c_2)\ t.binop & \hookrightarrow & (t.\mathsf{const}\ c) & (\text{if } c \in binop_t(c_1, c_2)) \\
(t.\mathsf{const}\ c_1)\ (t.\mathsf{const}\ c_2)\ t.binop & \hookrightarrow & \mathsf{trap} & (\text{if } binop_t(c_1, c2) = \{\})
\end{array}
$$

$$\frac{c \in \mathtt{binop}_t(c_1, c_2)}{[t.\mathtt{const}\ c_1,\ t.\mathtt{const}\ c_2,\ t.\mathtt{binop}] \leadsto [t.\mathtt{const}\ c]}$$

$$\frac{c \in \mathtt{binop}_t(c_1, c_2)}{[t.\mathtt{const}\ c_1,\ t.\mathtt{const}\ c_2,\ t.\mathtt{binop}] \leadsto [t.\mathtt{const}\ c]}$$

```
i32.const 34
i32.const 13
i32.sub
i32.const 2
i32.mul
```

$$\frac{c \in \mathtt{binop}_t(c_1, c_2)}{[t.\mathtt{const}\ c_1,\ t.\mathtt{const}\ c_2,\ t.\mathtt{binop}] \leadsto [t.\mathtt{const}\ c]}$$

```
i32.const 34
i32.const 13              i32.const 21
i32.sub          ⤳        i32.const 2
i32.const 2               i32.mul
i32.mul
```

$$\frac{c \in \mathtt{binop}_t(c_1, c_2)}{[t.\mathtt{const}\ c_1,\ t.\mathtt{const}\ c_2,\ t.\mathtt{binop}] \rightsquigarrow [t.\mathtt{const}\ c]}$$

i32.const 34				
i32.const 13		i32.const 21		
i32.sub	$\rightsquigarrow$	i32.const 2	$\rightsquigarrow$	i32.const 42
i32.const 2		i32.mul		
i32.mul				

```
loop
  nop
  br 0
end
```

```
loop                  label [loop, nop, br 0, end]
   nop                   nop
   br 0       ⤳         br 0
end                   end
```

```
loop                label [loop, nop, br 0, end]
  nop                 nop                            label [...]
  br 0      ⤳         br 0                      ⤳       br 0
end                 end                              end
```

```
loop                 label [loop, nop, br 0, end]                                        loop
   nop                  nop                                     label [...]                 nop
   br 0      ~>         br 0                          ~>           br 0          ~>         br 0
end                  end                                        end                      end
```

We want to:

- **Overapproximate** WebAssembly semantics
- Attach information to each **program point**

We want to:

- **Overapproximate** WebAssembly semantics ✓
- Attach information to each **program point**

We want to:

- **Overapproximate** WebAssembly semantics ✓
- Attach information to each **program point** ✗

Interpreter

```
record c_state =
```

```
record c_state =
  c_pc :: c_pc
```

```
record c_state =
  c_pc :: c_pc
  val_stack :: v list
  ...
```

```
record c_state =
  c_pc :: c_pc
  val_stack :: v list
  ...

c_step ::  c_state ⇒ c_state set
```

We now can:

- **Overapproximate** WebAssembly semantics
- Attach information to each **program point**

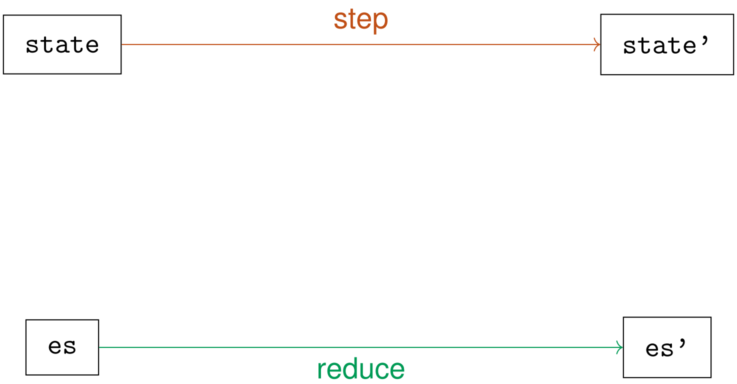
We now can:

- **Overapproximate** WebAssembly semantics
- Attach information to each **program point** ✓

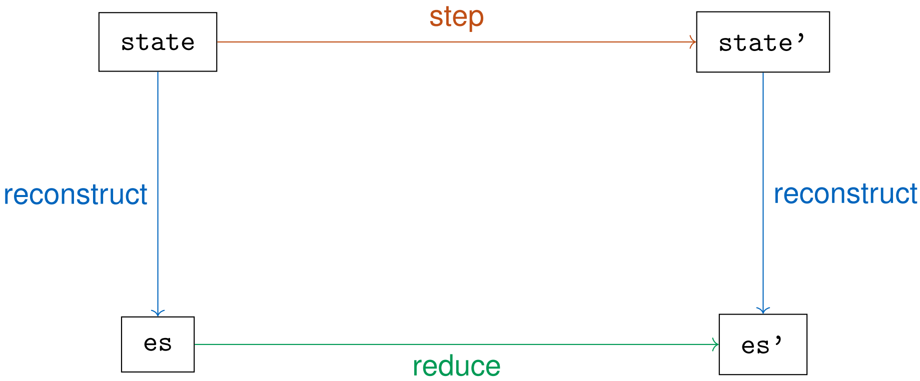
We now can:

- **Overapproximate** WebAssembly semantics ?
- Attach information to each **program point** ✓


state
step
state'



state
step
state'
es
reduce
es'


step
state
state'
reconstruct
reconstruct
es
es'
reduce

Given some `state` and es',
Let $\mathrm{es} = \texttt{reconstruct}\ \mathrm{state}$.

$$\frac{}{}\ \text{Completeness}$$

Given some $\texttt{state}$ and es',
Let $\mathrm{es} = \texttt{reconstruct}\ \mathrm{state}$.

$$\frac{}{\mathrm{es} \leadsto \mathrm{es}'}\ \text{Completeness}$$

Given some $\texttt{state}$ and es',
Let $\mathrm{es} = \texttt{reconstruct}\ \mathrm{state}$.

$$\frac{\mathrm{es} \leadsto \mathrm{es}'}{\exists \texttt{state}'\ .}\ \text{Completeness}$$

Given some $\texttt{state}$ and es',
Let $\mathrm{es} = \texttt{reconstruct}\ \mathrm{state}$.

$$\frac{\mathrm{es} \leadsto \mathrm{es}'}{\exists \texttt{state}'\texttt{ . }\ \texttt{state}' \in \texttt{c\_step state} \land} \text{Completeness}$$

Given some `state` and es',
Let $\mathrm{es} = \texttt{reconstruct}\ \mathrm{state}$.

$$\frac{\mathrm{es} \leadsto \mathrm{es}'}{\exists \texttt{state}'\ .\quad \texttt{state}' \in \texttt{c\_step}\ \ \texttt{state} \wedge \texttt{reconstruct}\ \ \texttt{state}' = \mathrm{es}'}\ \text{Completeness}$$

We now can:

- **Overapproximate** WebAssembly semantics ✓
- Attach information to each **program point** ✓

Demo

Summary

Goals reached

Summary

Goals reached

- Interpreter
  - Value and call stack
  - Explicit program counter
  - Complete wrt. specification

Summary

Goals reached

- Interpreter
  - Value and call stack
  - Explicit program counter
  - Complete wrt. specification
- Integer Types and Operations
  - Official specification translated to Isabelle/HOL
  - Executable implementation using Word Library

Summary

Goals reached

- Interpreter
  - Value and call stack
  - Explicit program counter
  - Complete wrt. specification
- Integer Types and Operations
  - Official specification translated to Isabelle/HOL
  - Executable implementation using Word Library

Challenges

- Interpreter by default does more steps than reduction (constants)

Summary

Goals reached

- Interpreter
  - Value and call stack
  - Explicit program counter
  - Complete wrt. specification
- Integer Types and Operations
  - Official specification translated to Isabelle/HOL
  - Executable implementation using Word Library

Challenges

- Interpreter by default does more steps than reduction (constants)
- Distribution of value stack over labels in reconstruction

Summary

Goals reached

- Interpreter
  - Value and call stack
  - Explicit program counter
  - Complete wrt. specification
- Integer Types and Operations
  - Official specification translated to Isabelle/HOL
  - Executable implementation using Word Library

Challenges

- Interpreter by default does more steps than reduction (constants)
- Distribution of value stack over labels in reconstruction
- Rule inversion of reduction relation in assumption

Summary

Goals reached

- Interpreter
  - Value and call stack
  - Explicit program counter
  - Complete wrt. specification
- Integer Types and Operations
  - Official specification translated to Isabelle/HOL
  - Executable implementation using Word Library

Challenges

- Interpreter by default does more steps than reduction (constants)
- Distribution of value stack over labels in reconstruction
- Rule inversion of reduction relation in assumption

Future work

- Static analysis and optimization

Summary

Goals reached

- Interpreter
  - Value and call stack
  - Explicit program counter
  - Complete wrt. specification
- Integer Types and Operations
  - Official specification translated to Isabelle/HOL
  - Executable implementation using Word Library

Challenges

- Interpreter by default does more steps than reduction (constants)
- Distribution of value stack over labels in reconstruction
- Rule inversion of reduction relation in assumption

Future work

- Static analysis and optimization
- Soundness proof

Summary

Goals reached

- Interpreter
  - Value and call stack
  - Explicit program counter
  - Complete wrt. specification
- Integer Types and Operations
  - Official specification translated to Isabelle/HOL
  - Executable implementation using Word Library

Challenges

- Interpreter by default does more steps than reduction (constants)
- Distribution of value stack over labels in reconstruction
- Rule inversion of reduction relation in assumption

Future work

- Static analysis and optimization
- Soundness proof
- Executable, deterministic and sound interpreter

Summary

Goals reached

- Interpreter
  - Value and call stack
  - Explicit program counter
  - Complete wrt. specification
- Integer Types and Operations
  - Official specification translated to Isabelle/HOL
  - Executable implementation using Word Library

Challenges

- Interpreter by default does more steps than reduction (constants)
- Distribution of value stack over labels in reconstruction
- Rule inversion of reduction relation in assumption

Future work

- Static analysis and optimization
- Soundness proof
- Executable, deterministic and sound interpreter
- Float