NanoWasm Specification

The SpecTec Team

Feb 05, 2025

CONTENTS

NanoWasm is a small language with simple types and instructions.

CHAPTER

ONE

ABSTRACT SYNTAX

The *abstract syntax* of types is as follows:

$$\begin{array}{rcl} \mathit{mut} & ::= & \mathsf{mut} \\ \mathit{valtype} & ::= & \mathsf{i32} \mid \mathsf{i64} \mid \mathsf{f32} \mid \mathsf{f64} \\ \mathit{functype} & ::= & \mathit{valtype}^* \rightarrow \mathit{valtype}^* \\ \mathit{globaltype} & ::= & \mathit{mut}^? ~ \mathit{valtype} \end{array}$$

Instructions take the following form:

$$\begin{array}{rcl} \mathit{const} & ::= & 0 \mid 1 \mid 2 \mid \dots \\ \mathit{instr} & ::= & \mathsf{nop} \\ & \mid & \mathsf{drop} \\ & \mid & \mathsf{select} \\ & \mid & \mathit{valtype}.\mathsf{const} ~ \mathit{const} \\ & \mid & \mathsf{local.get} ~ \mathit{localidx} \\ & \mid & \mathsf{local.set} ~ \mathit{localidx} \\ & \mid & \mathsf{global.get} ~ \mathit{globalidx} \\ & \mid & \mathsf{global.set} ~ \mathit{globalidx} \end{array}$$

The instruction nop does nothing, drop removes an operand from the stack, select picks one of two operands depending on a condition value. The instruction $t.\mathsf{const}~c$ pushed the constant c to the stack. The remaining instructions access local and global variables.

CHAPTER

TWO

VALIDATION

NanoWasm instructions are *type-checked* under a context that assigns types to indices:

$$context \quad ::= \quad \{\mathsf{globals}\ globaltype^*, \mathsf{locals}\ valtype^*\}$$

2.1 nop

nop is valid with $\epsilon \ \rightarrow \ \epsilon$.

$$\frac{}{C \vdash \mathsf{nop} : \epsilon \rightarrow \epsilon}$$

2.2 drop

drop is valid with $t \ \rightarrow \ \epsilon$.

$$\frac{}{C \vdash \mathsf{drop} : t \rightarrow \epsilon}$$

2.3 select

select is valid with $t\ t\ \mathsf{i32} \ \rightarrow \ t$.

$$\frac{}{C \vdash \mathsf{select} : t\ t\ \mathsf{i32} \rightarrow t}$$

2.4 const

$(t.\mathsf{const}\ c)$ is valid with $\epsilon \ \rightarrow \ t$.

$$\frac{}{C \vdash t.\mathsf{const}\ c : \epsilon \rightarrow t}$$

2.5 local.get

$(\mathsf{local.get}\ x)$ is valid with $\epsilon \ \rightarrow \ t$ if:

- $C.\mathsf{locals}[x]$ exists.
- $C.\mathsf{locals}[x]$ is equal to t.

$$\frac{C.\mathsf{locals}[x] = t}{C \vdash \mathsf{local.get}\ x : \epsilon \rightarrow t}$$

2.6 local.set

$(\mathsf{local.set}\ x)$ is valid with $t \ \rightarrow\ \epsilon$ if:

- $C.\mathsf{locals}[x]$ exists.
- $C.\mathsf{locals}[x]$ is equal to t.

$$\frac{C.\mathsf{locals}[x] = t}{C \vdash \mathsf{local.set}\ x : t \rightarrow \epsilon}$$

2.7 global.get

$(\mathsf{global.get}\ x)$ is valid with $\epsilon \ \rightarrow\ t$ if:

- $C.\mathsf{globals}[x]$ exists.
- $C.\mathsf{globals}[x]$ is equal to $(\mathsf{mut}^?\ t)$.

$$\frac{C.\mathsf{globals}[x] = \mathsf{mut}^?\ t}{C \vdash \mathsf{global.get}\ x : \epsilon \rightarrow t}$$

2.8 global.set

$(\mathsf{global.get}\ x)$ is valid with $t \ \rightarrow\ \epsilon$ if:

- $C.\mathsf{globals}[x]$ exists.
- $C.\mathsf{globals}[x]$ is equal to $(\mathsf{mut}\ t)$.

$$\frac{C.\mathsf{globals}[x] = \mathsf{mut}\ t}{C \vdash \mathsf{global.get}\ x : t \rightarrow \epsilon}$$

CHAPTER

THREE

EXECUTION

NanoWasm execution requires a suitable definition of state and configuration:

$$
\begin{array}{lcl}
\mathit{addr} & ::= & 0 \mid 1 \mid 2 \mid \ldots \\
\mathit{moduleinst} & ::= & \{\mathsf{globals}\ \mathit{addr}^*\} \\
\mathit{val} & ::= & \mathsf{const}\ \mathit{valtype}\ \mathit{const} \\
\mathit{store} & ::= & \{\mathsf{globals}\ \mathit{val}^*\} \\
\mathit{frame} & ::= & \{\mathsf{locals}\ \mathit{val}^*, \mathsf{module}\ \mathit{moduleinst}\} \\
\mathit{state} & ::= & \mathit{store}; \mathit{frame} \\
\mathit{config} & ::= & \mathit{state}; \mathit{instr}^*
\end{array}
$$

We define the following auxiliary functions for accessing and updating the state:

$$
\begin{array}{lcl}
\mathrm{local}((s; f), x) & = & f.\mathsf{locals}[x] \\
\mathrm{global}((s; f), x) & = & s.\mathsf{globals}[f.\mathsf{module}.\mathsf{globals}[x]] \\
\mathrm{update}_{local}((s; f), x, v) & = & s; f[.\mathsf{locals}[x] = v] \\
\mathrm{update}_{global}((s; f), x, v) & = & s[.\mathsf{globals}[f.\mathsf{module}.\mathsf{globals}[x]] = v]; f
\end{array}
$$

With that, execution is defined as follows:

3.1 nop

1. Do nothing.

$$\mathsf{nop} \quad \hookrightarrow \quad \epsilon$$

3.2 drop

1. Assert: Due to validation, a value is on the top of the stack.
2. Pop the value val from the stack.
3. Do nothing.

$$\mathit{val}\ \mathsf{drop} \quad \hookrightarrow \quad \epsilon$$

3.3 select

1. Assert: Due to validation, a value type is on the top of the stack.
2. Pop the value $(\mathsf{i32.const}\ c)$ from the stack.
3. Assert: Due to validation, a value is on the top of the stack.
4. Pop the value val_2 from the stack.
5. Assert: Due to validation, a value is on the top of the stack.

6. Pop the value val_1 from the stack.
7. If $c \neq 0$, then:
   a. Push the value val_1 to the stack.
8. Else:
   a. Push the value val_2 to the stack.

$$\begin{array}{lcll} val_1\ val_2\ (\mathsf{i32.const}\ c)\ \mathsf{select} & \hookrightarrow & val_1 & \text{if } c \neq 0 \\ val_1\ val_2\ (\mathsf{i32.const}\ c)\ \mathsf{select} & \hookrightarrow & val_2 & \text{otherwise} \end{array}$$

3.4 local.get x

1. Let z be the current state.
2. Let val be $\mathrm{local}(z, x)$.
3. Push the value val to the stack.

$$z; (\mathsf{local.get}\ x) \quad \hookrightarrow \quad z; val \quad \text{if } val = \mathrm{local}(z, x)$$

3.5 local.set x

1. Assert: Due to validation, a value is on the top of the stack.
2. Pop the value val from the stack.
3. Do nothing.

$$z; val\ (\mathsf{local.set}\ x) \quad \hookrightarrow \quad z'; \epsilon \quad \text{if } z' = \mathrm{update}_{local}(z, x, val)$$

3.6 global.get x

1. Let z be the current state.
2. Let val be $\mathrm{global}(z, x)$.
3. Push the value val to the stack.

$$z; (\mathsf{global.get}\ x) \quad \hookrightarrow \quad z; val \quad \text{if } val = \mathrm{global}(z, x)$$

3.7 global.set x

1. Assert: Due to validation, a value is on the top of the stack.
2. Pop the value val from the stack.
3. Do nothing.

$$z; val\ (\mathsf{global.set}\ x) \quad \hookrightarrow \quad z'; \epsilon \quad \text{if } z' = \mathrm{update}_{global}(z, x, val)$$

CHAPTER FOUR

BINARY FORMAT

The following grammars define the binary representation of NanoWasm programs.

First, constants are represented in LEB format:

$$
\begin{array}{rcll}
\mathtt{byte} & ::= & b{:}\mathtt{0x00} \mid \ldots \mid b{:}\mathtt{0xFF} & \Rightarrow\ b \\
\mathtt{u}(N) & ::= & n{:}\mathtt{byte} & \Rightarrow\ n \qquad\qquad\quad \text{if } n < 2^7 \wedge n < 2^N \\
 & \mid & n{:}\mathtt{byte}\ \ m{:}\mathtt{u}(N-7) & \Rightarrow\ 2^7 \cdot m + (n - 2^7) \quad \text{if } n \geq 2^7 \wedge N > 7 \\
\mathtt{u32} & ::= & n{:}\mathtt{u}(32) & \Rightarrow\ n \\
\mathtt{u64} & ::= & n{:}\mathtt{u}(64) & \Rightarrow\ n \\
\mathtt{f}(N) & ::= & b^*{:}\mathtt{byte}^{N/8} & \Rightarrow\ \mathrm{float}(N, b^*) \\
\mathtt{f32} & ::= & p{:}\mathtt{f}(32) & \Rightarrow\ p \\
\mathtt{f64} & ::= & p{:}\mathtt{f}(64) & \Rightarrow\ p
\end{array}
$$

Types are encoded as follows:

$$
\begin{array}{rcll}
\mathtt{valtype} & ::= & \mathtt{0x7F} & \Rightarrow\ \mathsf{i32} \\
 & \mid & \mathtt{0x7E} & \Rightarrow\ \mathsf{i64} \\
 & \mid & \mathtt{0x7D} & \Rightarrow\ \mathsf{f32} \\
 & \mid & \mathtt{0x7C} & \Rightarrow\ \mathsf{f64} \\
\mathtt{mut} & ::= & \mathtt{0x00} & \Rightarrow\ \epsilon \\
 & \mid & \mathtt{0x01} & \Rightarrow\ \mathsf{mut} \\
\mathtt{globaltype} & ::= & t{:}\mathtt{valtype}\ \ \mathit{mut}{:}\mathtt{mut} & \Rightarrow\ \mathit{mut}\ t \\
\mathtt{resulttype} & ::= & n{:}\mathtt{u32}\ \ (t{:}\mathtt{valtype})^n & \Rightarrow\ t^n \\
\mathtt{functype} & ::= & \mathtt{0x60}\ \ t_1^*{:}\mathtt{resulttype}\ \ t_2^*{:}\mathtt{resulttype} & \Rightarrow\ t_1^* \to t_2^*
\end{array}
$$

Finally, instruction opcodes:

$$
\begin{array}{rcll}
\mathtt{globalidx} & ::= & x{:}\mathtt{u32} & \Rightarrow\ x \\
\mathtt{localidx} & ::= & x{:}\mathtt{u32} & \Rightarrow\ x \\
\mathtt{instr} & ::= & \mathtt{0x01} & \Rightarrow\ \mathsf{nop} \\
 & \mid & \mathtt{0x1A} & \Rightarrow\ \mathsf{drop} \\
 & \mid & \mathtt{0x1B} & \Rightarrow\ \mathsf{select} \\
 & \mid & \mathtt{0x20}\ \ x{:}\mathtt{localidx} & \Rightarrow\ \mathsf{local.get}\ x \\
 & \mid & \mathtt{0x21}\ \ x{:}\mathtt{localidx} & \Rightarrow\ \mathsf{local.set}\ x \\
 & \mid & \mathtt{0x23}\ \ x{:}\mathtt{globalidx} & \Rightarrow\ \mathsf{global.get}\ x \\
 & \mid & \mathtt{0x24}\ \ x{:}\mathtt{globalidx} & \Rightarrow\ \mathsf{global.set}\ x \\
 & \mid & \mathtt{0x41}\ \ n{:}\mathtt{u32} & \Rightarrow\ \mathsf{i32.const}\ n \\
 & \mid & \mathtt{0x42}\ \ n{:}\mathtt{u64} & \Rightarrow\ \mathsf{i64.const}\ n \\
 & \mid & \mathtt{0x43}\ \ p{:}\mathtt{f32} & \Rightarrow\ \mathsf{f32.const}\ p \\
 & \mid & \mathtt{0x44}\ \ p{:}\mathtt{f64} & \Rightarrow\ \mathsf{f64.const}\ p
\end{array}
$$