Minimini-Wasm Tutorial

Introduction

In this page, we will practice how to write a spec. Here, instead of a full Wasm, a very simplified version of Wasm (which we call Minimini-Wasm) is used as our goal. Abstract syntax of Minimini-Wasm is as follows:

$$
\begin{aligned}
module &::= function^* \ start^? \\
function &::= type_{func} \ code \ export^* \\
start &::= idx_{func} \\
export &::= \text{“name”} \\
code &::= (\text{local } type_{val})^* \ instr^* \\
init &::= instr^*
\end{aligned}
$$

$$
\begin{aligned}
instr &::= \text{nop} \mid \text{drop} \mid \text{select} \mid type_{val}\text{.const } value\_(type_{val}) \mid binary \mid (\text{set} \mid \text{get} \mid \text{tee})\text{\_local } idx_{local} \\
&\quad \mid \text{call } idx_{func} \mid \text{return} \mid \text{block} \mid \text{loop} \mid \text{if} \mid \text{br } idx_{label} \mid \text{br\_if } idx_{label} \mid \\
binary &::= \text{i32.add} \mid \text{i32.sub} \mid \text{i32.mul} \mid \text{i64.add} \mid \text{i64.sub} \mid \text{i64.mul}
\end{aligned}
$$

$$
\begin{aligned}
type_{val} &::= \text{i32} \mid \text{i64} \\
type_{func} &::= type_{val}^* \to type_{val}^* \\
idx_{func \mid local \mid label} &\in [0,\ 2^{32} - 1] \\
value\_(i32) &\in [0,\ 2^{32} - 1] \\
value\_(i64) &\in [0,\ 2^{64} - 1]
\end{aligned}
$$

Directory of specs

Each folder in the directory `spectec/spec` indicates different versions of Wasm spec. Among them, directory `wasm-0.0` is where we will work on.

Syntax

Now, we will start from writing the syntax of Minimini-Wasm. Make a new file `1-syntax.wastup` in directory `wasm-0.0`.

Declaring syntax in Wasm-DSL is basically done like this:

```
syntax <name_of_syntax> = | <case> | <case> | <case> |
```

Use keyword `syntax`, and simply list the possible cases with the seperator `|`.

We'll declare each of the syntax one by one.

`valtype`

$$type_{val} ::= \text{i32} \mid \text{i64}$$

This syntax is simply written in Wasm-DSL like this:

```
syntax valtype = | I32 | I64 |
```

This means the syntax `valtype` is either `I32` or `I64`.

Similarly, let's declare each of the syntax one by one.

`idx`

$$idx_{func\,|\,local\,|\,label} \in [0,\ 2^{32} - 1]$$

You can use `...` to describe a range. The Wasm-DSL version of upper syntax will be:

```
syntax idx = 0 | ... | 2^32-1
```

Since we have three types of index (which are semantically same, but syntactically different), write like this:

```
syntax funcidx = idx
syntax labelidx = idx
syntax localidx = idx
```

`value`

$$value\_(i32) \in [0,\ 2^{32} - 1]$$
$$value\_(i64) \in [0,\ 2^{64} - 1]$$

Here, the syntax `value` is declared in regard with parameter `valtype`. This can be done like this:

```
syntax val(valtype)
syntax val(I32) = 0 | ... | 2^32-1
syntax val(I64) = 0 | ... | 2^64-1
```

Here we can declare a general range, by using parameter again.

```
syntax iN(N) = 0 | ... | 2^N-1
```

Here, `N` is a pre-defined syntax, which indicates any natural number.

Now, we can write `iN(32)` and `iN(64)` instead of `0 | ... | 2^32-1` and `0 | ... | 2^64-1`. New declaration of `idx` and `val` is as follows:

```
syntax idx = iN(32)

syntax val(valtype)
```

```
syntax val(I32) = iN(32)
syntax val(I64) = iN(64)
```

`char`

```
syntax char = U+0000 | ... | U+D7FF | U+E000 | ... | U+10FFFF
```

From the syntax `char`, we can declare `name`, which is an iteration of `char`s. Use `*` to represent a sequence:

```
syntax name = char*
```

`binop`

```
syntax binop_(valtype) = | ADD | SUB | MUL
```

`instr`

```
syntax instr =
  | NOP
  | DROP
  | SELECT
  | CONST valtype val_(valtype)
  | BINOP valtype binop_(valtype)
  | LOCAL.GET idx
  | LOCAL.SET idx
  | LOCAL.TEE idx
  | CALL idx
  | RETURN
  | BLOCK blocktype instr*
  | LOOP blocktype instr*
  | IF blocktype instr* ELSE instr*
  | BR labelidx
  | BR_IF labelidx
```