Google

WebAssembly Exception Handling (Phase 1)

Heejin Ahn

People

- Champion / WebAssembly toolchain implementation: Heejin Ahn
- V8 code owner for exception handling: Clemens Backes

Agenda

- Goals / Dependencies
- Spec recap
- Spec updates
- Implementation status
- Poll to phase 2

Goals

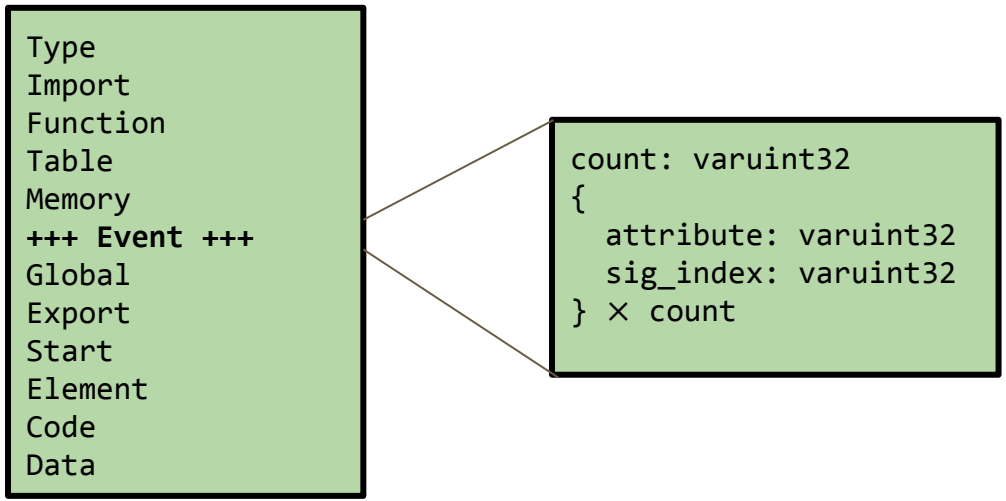
- Provide a primitive for exception handling for Wasm programs
  - Zero-cost: no runtime cost until exceptions are thrown or caught
  - Structured: composes properly with existing control flow constructs
  - Safe: fast, single-pass verification

Dependencies

- Reference Types
  - `Exnref` type is a subtype of `anyref`
- Multi-value
  - An exception can contain multiple values
  - Multivalue support not required for many cases, in which only one value is thrown (e.g. C++ exception)

Event section

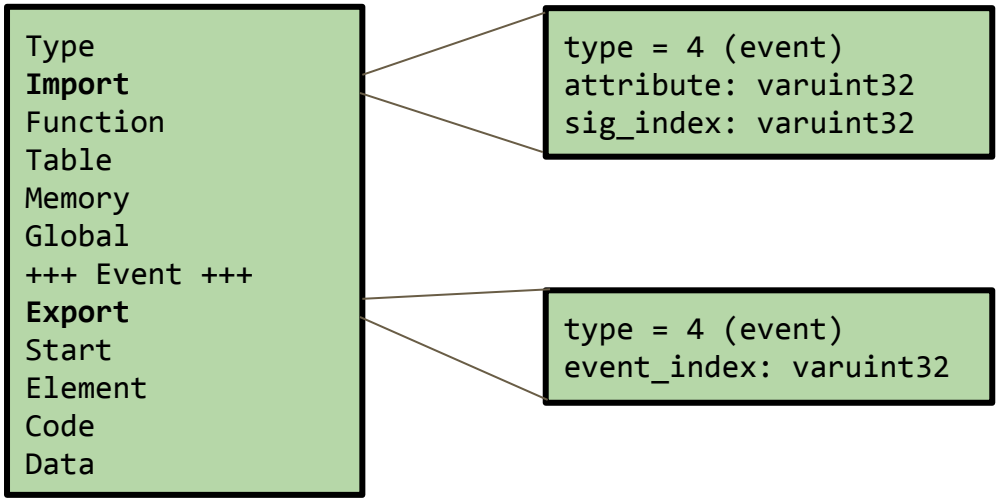
- Wasm events are features that suspend the current execution and transfer the control flow to a corresponding handle
  - Only supported kind is exceptions now



- Event section declares a list of event types
- `attribute`: application-specified number (0 for exceptions)
- `sig_index`: index into types section of function signature

Event import/export

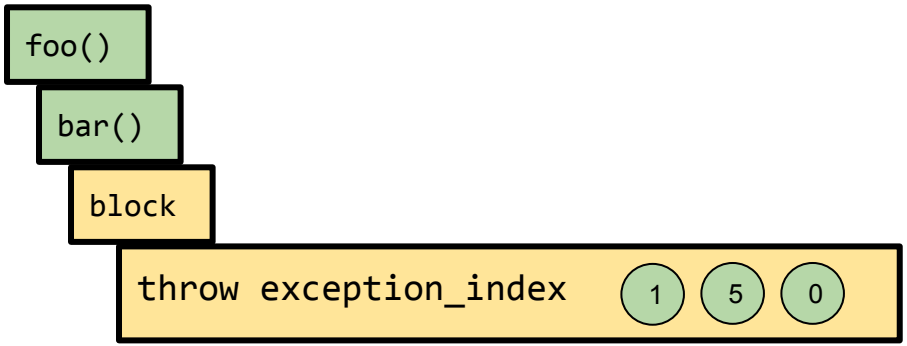
- Events can be imported and exported from a module
- Each instantiation of a module produces new event tags



- Imported events specify an attribute and an expected signature
- Exported events specify an exported event index

Throwing an exception

- A **`throw`** accepts its arguments on the stack and creates an **`except_ref`** value out of them
- and begins searching the *control* and *call* stacks for a handler



The exception reference data type

- EH proposal requires the reference types proposal as a prerequisite
- `except_ref` type is a subtype of `anyref`
- `except_ref` type contains values thrown
- `except_ref` type possibly can contain more information, such as a stack trace

`try` and `catch` blocks

- **`try` … `catch` … `end`** introduces a new block kind
  - `try` can have a label for branches too
- If any instruction between a **`try`** and a **`catch`** throws, the VM unwinds the wasm execution stack and resumes execution from the **`catch`**
- A **`catch`** pushes an **`except_ref`** value onto the stack

```
try [block_type]
  ...
catch
  ...
end
```

Rethrowing an exception

- A **`rethrow`** takes an **except_ref** value from the stack (produced by a **`catch`**) and continue unwinding the execution stack with the exception
- A **`rethrow`** can occur anywhere (not necessarily between **`catch`** and **`end`**)

Exception data extraction

- A **`br_on_exn`** checks the exception tag of an **`except_ref`** on top of the stack (without popping it) if it matches the given exception index
  - If they match, it
    - branches out to the label referenced
    - pops the **`except_ref`** from the stack
    - extracts the **`except_ref`** and pushes the exception's values onto the stack
  - If they don't match, it does nothing, and the **`except_ref`** remains on the stack
- Format: `br_on_exn label except_index`

Exception data extraction

- We can also test an **`except_ref`** against multiple tags

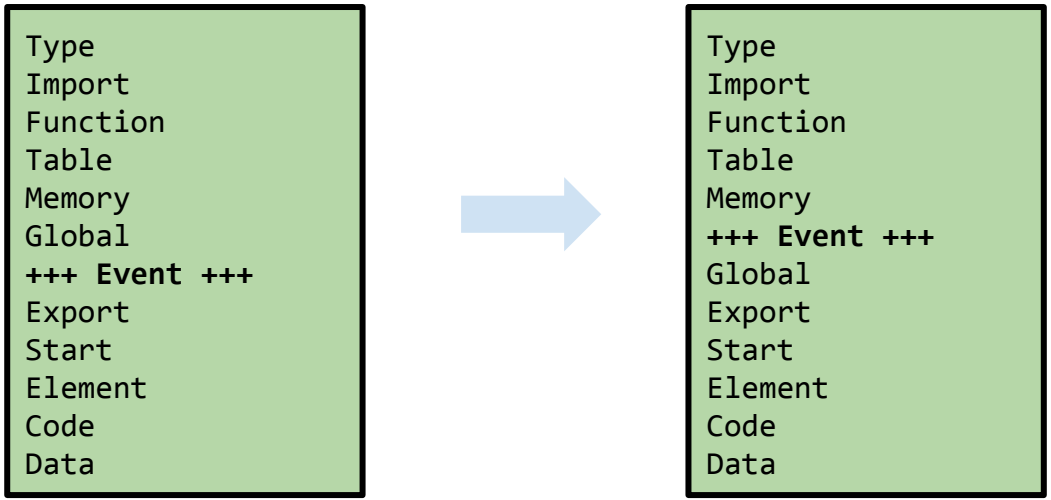
```
block $l0 (result i32, i64)
  block $l1 (result i32)
    ...
    ;; except_ref $e is on the stack at this point
    br_on_exn $l1 e(i32)      ;; branch to $l1 with $e's arguments
    ;; except_ref $e is left on the stack if br_on_exn is not taken
    br_on_exn $l0 e(i32, i64) ;; branch to $l0 with $e's arguments
    rethrow
  end
  ;; handler for $l1
end
;; handler for $l0
```

Spec Updates: Traps and JS API

- `catch` instruction does not catch traps
  - Rationale:
    - In general, traps are not locally recoverable and not needed to be handled in local scopes like try-catch
    - Catchable traps will increase code size of every cleanup pad
- JS API: `catch` instruction catches foreign exceptions, with a few exceptions:
  - `catch` does not catch exceptions generated from traps
  - `catch` does not catch JS exceptions generated from stack overflow and OOM
- JS API: Filtering exceptions should be based on an internal predicate

Other Misc. Spec Updates

- **`rethrow`** and **`br_on_exn`** trap when the value on the top of stack is null
- The order of event section and global section was swapped



Implementation Status

- Toolchain
  - Done w/ basic implementation of LLVM, Binaryen, Emscripten, and libc++abi & libunwind
    - Required implementation of reference types proposal
  - Small tests work end-to-end w/o optimizations
  - Missing parts
    - Exception specification (`throw(...)`) support
    - Some optimizations in Binaryen
    - Testing against sizable applications
- V8
  - Feature complete, modulo recent spec changes on traps

Poll to Phase 2

- Entry requirements
  - Full proposed English spec text available in a forked repo around which a reasonably high level of consensus exists.