Effect handlers for a low-level stack machine

Andreas Rossberg

(joint work with Daan, Daniel, Jonathan, KC, Matija, Sam, Stephen)

Shonan Meeting on Effects 2019
WebAssembly
a.k.a. Wasm

a virtual instruction set architecture

a universal low-level virtual machine

near-native performance

can be embedded anywhere
strategic roadmap

v1 (2017): support low-level languages

v2 (2019+): support high-level languages
big new features

tail calls
exceptions
garbage collection
continuations
(threads – stalled by Spectre)
design goals & constraints

language-independent
platform-independent
hardware-independent
fast to execute
safe to execute
deterministic
easy to reason about
compact
easy to generate
fast to decode
fast to validate
fast to compile
streamable
parallelisable
stack machine

(local.get $x) : \varepsilon \rightarrow \Gamma($x$)
(i32.const 42) : \varepsilon \rightarrow i32
(i32.add) : i32 i32 \rightarrow i32

}: \varepsilon \rightarrow i32
structured control flow

(block $l$ ($t_1^* \rightarrow t_2^*$))

... ($\text{br } l$) : $t_2^* \rightarrow \bot$

... )

(loop $l$ ($t_1^* \rightarrow t_2^*$))

... ($\text{br } l$) : $t_1^* \rightarrow \bot$

... )
switch(x) {
    case 0: A; break;
    case 1: B; break;
}

(block $switch (ε → ε)
    (block $case0 (ε → ε)
        (block $case1 (ε → ε)
            (local.get $x)
            (br_table $case0 $case1 $switch)
        )
    )
    (A) (br $switch)
)

(B) (br $switch)
)
exception proposal

nominal exceptions, im/exportable

throw, try-catch, br_on_exn

type of exn packages
(exception $e t^*)

(throw $e) : t^* \rightarrow \bot

(try (t_1^* \rightarrow t_2^*) ...
      ...
      catch
      ...
      ... )

(br_on_exn $l $e) : exn \rightarrow exn

(if $l : t^*$)
(exception $e1 i32)
(exception $e2 i32 i32)

(i32.const 1) (i32.const 1) (throw $e2)

(try $l (i32 → i32)
   (call $f) ;; $f : i32 → i32

catch
   ...
   ...
   ...
   ...
   ...
   ...
   ...
   ...
   ...
   ...
   ...

)
(exception $e1 i32)
(exception $e2 i32 i32)

(i32.const 1) (i32.const 1) (throw $e2)

(try $l (i32 → i32)
  (call $f) ;; $f : i32 → i32
catch
  (block $l1 (exn → i32)
    (block $l2 (exn → i32 i32)
      (br_on_exn $l1 $e1)
      (br_on_exn $l2 $e2)
    ...
  )
  ...
  ;; handle $e2
)
  ...
  ;; handle $e1
)
(exception $e1 i32)
(exception $e2 i32 i32)

(i32.const 1) (i32.const 1) (throw $e2)

(try $l (i32 → i32)
  (call $f) ;; $f : i32 → i32
catch
  (block $l1 (exn → i32)
    (block $l2 (exn → i32 i32)
      (br_on_exn $l1 $e1)
      (br_on_exn $l2 $e2)
      ...
    )
  )
  (i32.add) (br $l) ;; handle $e2
)  

(i32.neg) (br $l) ;; handle $e1
)
(try $l (i32 → i32)
  (call $f) ;; $f : i32 → i32
catch
  (block $l1 (exn → i32)
    (block $l2 (exn → i32 i32)
      (br_on_exn $l1 $e1)
      (br_on_exn $l2 $e2)
      (rethrow) ;; propagate
    )
  )
  (i32.add) (br $l) ;; handle $e2
)

(i32.neg) (br $l) ;; handle $e1
)
effect handlers

enable compilation of control abstractions
sell as a generalisation of exceptions
that provides efficient stack switching
don’t mention “algebraic” :)


(exception $e \ t^*)

(throw $e) : t^* \rightarrow \bot

(try (t_1^* \rightarrow t_2^*)
    ...
    catch
    ...
)

(br_on_exn $l \ $e) : exn \rightarrow exn
    (iff $l : t^*)
(exception $e (t^* \rightarrow t'^*))

(throw $e) : t^* \rightarrow \perp

(try (t_1^* \rightarrow t_2^*) ...
      : t_1^* \rightarrow t_2^*
    )

  catch ...
    : exn \rightarrow t_2^*

) (br_on_exn $l $e) : exn \rightarrow exn
  (iff $l : t^*)
(exception $e (t* → t'*))

(throw $e) : t* → t'*

(try (t_1* → t_2*))

... : t_1* → t_2*

catch ...

... : exn → t_2*

)

(br_on_exn $l $e) : exn → exn

(iff $l : t*)
(exception $e (t* → t’*))

(throw $e) : t* → t’*

(try (t₁* → t₂*)
    ...
  ) : t₁* → t₂*

  catch

  ...
  ) : exn → t₂*

(br_on_exn $l $e) : exn → exn
  (iff $l : t*)

(resume)
(exception $e (t^* → t'^*))

(throw $e) : t^* → t'^*

(try (t_1^* → t_2^*))

... : t_1^* → t_2^*

catch

... : exn → t_2^*

)

(br_on_exn $l $e) : exn → exn

(iff $l : t^*)

(resume) : (cont (t'^* → t_2^*)) t'^* → t_2^*
(exception $e (t^* \rightarrow t'^*))

(throw $e) : t^* \rightarrow t'^*

(try (t_1^* \rightarrow t_2^*)

... : t_1^* \rightarrow t_2^*

catch

... : exn \rightarrow t_2^*

)

(br_on_exn $l $e) : exn \rightarrow exn

(iff $l : t^* (cont (t'^* \rightarrow t_2^*))

(resume) : (cont (t'^* \rightarrow t_2^*)) t'^* \rightarrow t_2^*
(exception $e (t* → t'*))

(throw $e) : t* → t'*

(try (t₁* → t₂*))

... : t₁* → t₂*

catch

... : (exn t₂*) → t₂*

)

(br_on_exn $l $e) : (exn t₂*) → (exn t₂*)

(iff $l : t* (cont (t'* → t₂*)))

(resume) : (cont (t'* → t₂*)) t'* → t₂*
operational semantics

we have defined an operational semantics

handlers are shallow
(already have recursion/loops)

continuations are affine
(cheaper, engines cannot always copy stacks)
open design choices

lacks return clause, not properly algebraic 
(how important is it in this setting?)

catch clause is catch-all 
(should probably add a filter list)
implementation & performance

**try** needs to create new stack upon entry to enable delayed resumption

want to pay only when necessary

additional annotations
(exception resumable $e (t* → t’*))

(throw resumable $e)

(try resumable (t1* → t2*)

... catch ... : (exn resumable t2*) → t2*

)

(br_on_exn $l $e) : (exn resumable t2*) → (exn resumable t2*)

(resume) : (cont (t’* → t2*)) t’* → t2*
implementation & performance

at this point, effects are almost entirely a separate from exceptions…
(effect $e (t* → t'*))

(perform $e)

(run (t1* → t2*)
  ...
  handle
  ... : (eff t2*) → t2*
)

(br_on_eff $l $e) : (eff t2*) → (eff t2*)
  (iff $l : t* (cont (t'* → t2*)))

(resume) : (cont (t'* → t2*)) t'* → t2*