Vrije Universiteit Brussel

Meta-tracing JIT compilation

Maarten Vandercammen

Merging

```
(define (loop)
  (if (= (random 2) 0)
      (display 0)
      (display 1))
  (if (= (random 2) 0)
      (if bool
          ; do something
          ; do something else
          )
      (if otherbool
          ; do something again
          ; do something else again
          ))
  ; do some other things
  (loop))
```

Merging

```
(define (loop)
  (if (= (random 2) 0)
      (display 0)
      (display 1))
  (if (= (random 2) 0)
      (if bool
          ; do something
          ; do something else
          )
      (if otherbool
          ; do something again
          ; do something else again
          ))
  ; do some other things
  (loop))
```

Merging annotations

```
(splits-control-flow)

(merges-control-flow)
```

Merging annotations

```
(define (evaluate-if predicate consequent . alternate)
 (let* ((cond (evaluate predicate)))
   (splits-control-flow)
   (if cond
       (return-from-control-flow-split (thunkify
           consequent))
       (if (null? alternate)
           (return-from-control-flow-split '())
           (return-from-control-flow-split (thunkify (car
               alternate)))))))

(define (return-from-control-flow-split value)
   (merges-control-flow)
   value)
```

Handling annotations

```
(define (step* state)
  (match state
    ((ko (haltk) _)
     v)
    ; evaluate annotations in step* instead of step
    ; annotations might not lead to recursive call to step*
    ((ev `(splits-control-flow) (cons φ κ))
     (handle-splits-cf-annotation (ko φ κ)))
    ((ev `(merges-control-flow) (cons φ κ))
     (handle-merges-cf-annotation (ko φ κ)))
    ((ko (can-close-loopk) (cons φ κ))
     (handle-can-close-loop-annotation v (ko φ κ)))
    ((ko (can-start-loopk '() debug-info) (cons φ κ))
     (handle-can-start-loop-annotation v debug-info (ko φ
         κ)))
    (_
     (let ((new-state (step state)))
       (step* new-state)))))
```

splits-cf-id stack

```
(struct tracer-context (...
                        splits-cf-id-stack
                        ...))

(define (handle-splits-cf-annotation state)
  (execute/trace `(pop-continuation)
                 `(push-splits-cf-id! ,(inc-splits-cf-id!)))
  (step* state))

(define (handle-merges-cf-annotation continuation)
  (let ((mp-id (pop-splits-cf-id!)))
    ...))
```

Use common trace for merging

```
(define (loop)
  (if (= (random 2) 0)
      (display 0)
      (display 1))
  (if (= (random 2) 0)
      (if bool
          ; do something
          ; do something else
          )
      (if otherbool
          ; do something again
          ; do something else again
          ))
  ; do some other things
  (loop))
```

Jumping to MP-tail-traces

```
(letrec ((non-loop (lambda ()
             ; trace instructions
             (execute-mp-tail-trace id)))
   (non-loop))

(struct tracer-context (...
                        mp-tails-dictionary
                        ...)
```

Starting tracing

```
(define (handle-merges-cf-annotation continuation)
  (let ((mp-id (top-splits-cf-id)))
    (execute/trace `(pop-continuation)
                   `(pop-splits-cf-id!))
    (if (is-tracing?)
        (begin
          (append-trace! `((execute-mp-tail-trace ,mp-id ,continuation)))
          ; similar to previous closing functions
          ((tracer-context-merges-cf-function GLOBAL_TRACER_CONTEXT) (reverse
               τ))
          (if (mp-tail-trace-exists? mp-id)
              ; execute mp tail trace
              (begin (stop-tracing-normal!)
                     ; Use eval instead of execute/trace!
                          ; Redundant...
                     (let ((new-state (eval `(execute-mp-tail-trace ,mp-id
                          ,continuation))))
                       (step* new-state)))
              ; start tracing mp tail
              (begin (start-tracing-mp-tail! mp-id)
                     (step* continuation))))
        (step* continuation))))
```

Executing MP-tail-traces

```
(define (execute-mp-tail-trace mp-id state)
    (let* ((mp-tails-dictionary
        (tracer-context-mp-tails-dictionary
        GLOBAL_TRACER_CONTEXT))
           (mp-tail-trace (get-mp-tail-trace mp-id)))
      (if mp-tail-trace
          (begin (add-execution! mp-tail-trace)
                 (let ((mp-value (execute-trace
                     (trace-node-trace mp-tail-trace))))
                   ; Incorrect...
                   (bootstrap-to-evaluator mp-value)))
          ; Trace was discarded!
          (bootstrap-to-evaluator state))))
```