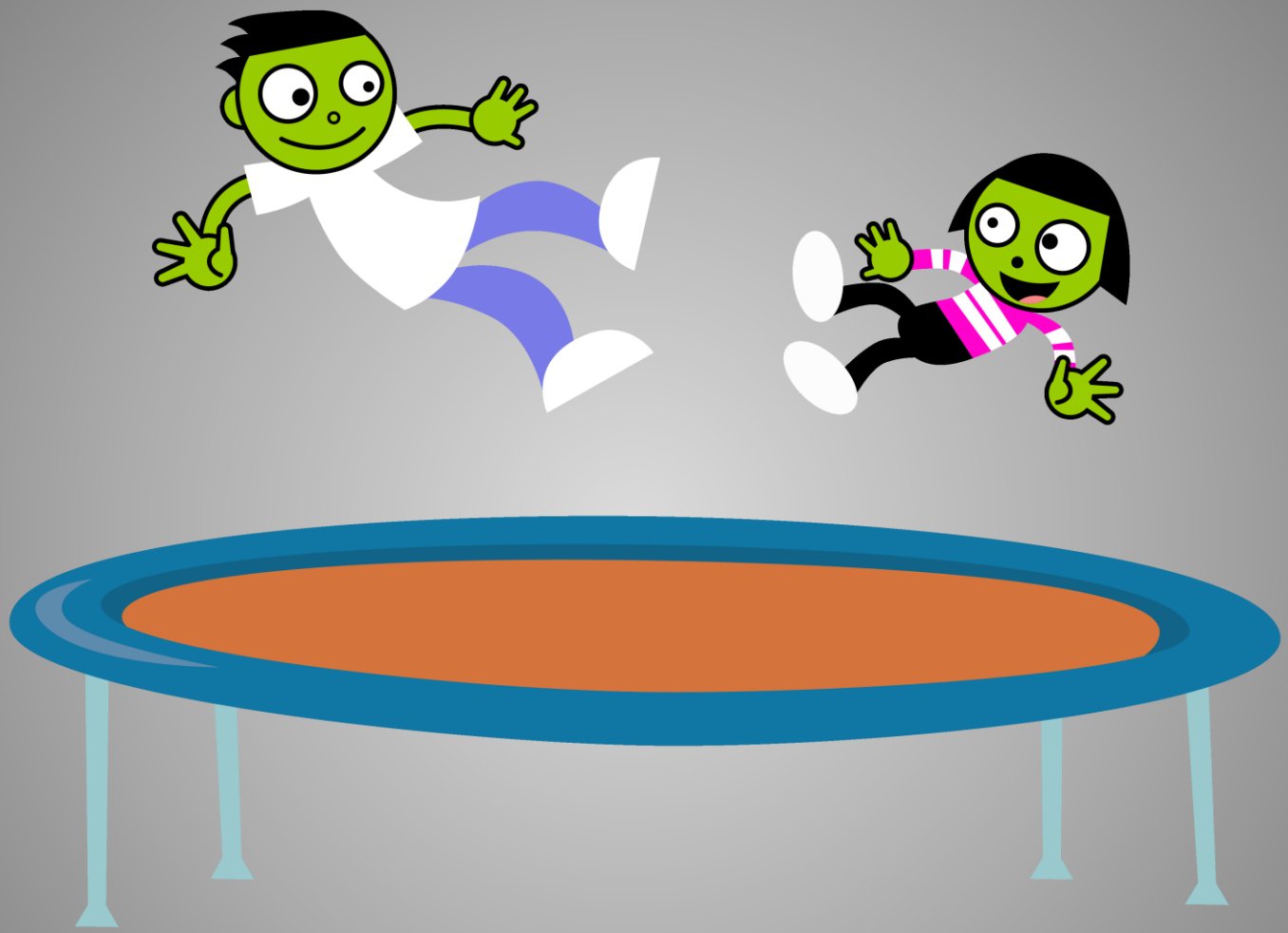
CPS – Trampolining

T. METIN SEZGIN



Source: https://www.deviantart.com/luxoveggiedude9302/art/PBS-Kids-GIF-Trampoline-808697584

A common theme in this class

- Problem
  - The CPS style interpreter has its problems
    - The control context keeps on growing
    - Each procedure application adds to the control context
    - This happens even with tail calls (our calls are tail calls)
- Solution
  - Break the chain
  - Every once in a while, return something even if the computation is not over yet
  - Trampolining
  - More specifically
    - Return a 0 argument procedure that when called continues the computation

The new `apply-procedure/k`

- We return with each procedure application

```
(define apply-procedure/k
  (lambda (proc1 val cont)
    (lambda ()
      (cases procedure proc1
        (... (value-of/k ...))))))
```

- The body of the procedure should be evaluated at the appropriate point

trampoline : *Bounce* → *FinalAnswer*

```
(define trampoline
  (lambda (bounce)
    (if (expval? bounce)
      bounce
      (trampoline (bounce)))))
```

The procedural representation of trampolining

- Value of program

```
value-of-program : Program → FinalAnswer
(define value-of-program
  (lambda (pgm)
    (cases program pgm
      (a-program (exp)
        (trampoline
          (value-of/k exp (init-env) (end-cont)))))))
```

The tail calls in the procedural representation

value-of-program : *Program* → *FinalAnswer*

```
(define value-of-program
  (lambda (pgm)
    (cases program pgm
      (a-program (exp)
        (trampoline
          (value-of/k exp (init-env) (end-cont)))))))
```

trampoline : *Bounce* → *FinalAnswer*

```
(define trampoline
  (lambda (bounce)
    (if (expval? bounce)
      bounce
      (trampoline (bounce)))))
```

value-of/k : *Exp* × *Env* × *Cont* → *Bounce*

```
(define value-of/k
  (lambda (exp env cont)
    (cases expression exp
      (... (value-of/k ...))
      (... (apply-cont ...))))))
```

apply-cont : *Cont* × *ExpVal* → *Bounce*

```
(define apply-cont
  (lambda (cont val)
    (cases continuation cont
      (... val)
      (... (value-of/k ...))
      (... (apply-cont ...))
      (... (apply-procedure/k ...))))))
```

apply-procedure/k : *Proc* × *ExpVal* × *Cont* → *Bounce*

```
(define apply-procedure/k
  (lambda (proc1 val cont)
    (lambda ()
      (cases procedure proc1
        (... (value-of/k ...))))))
```

The tail calls in the procedural representation

- The **Bounce** set

$$Bounce = ExpVal \cup (() \rightarrow Bounce)$$

value-of-program : $Program \rightarrow FinalAnswer$
trampoline : $Bounce \rightarrow FinalAnswer$
value-of/k : $Exp \times Env \times Cont \rightarrow Bounce$
apply-cont : $Cont \times ExpVal \rightarrow Bounce$
apply-procedure/k : $Proc \times ExpVal \times Cont \rightarrow Bounce$

trampoline : $Bounce \rightarrow FinalAnswer$

```
(define trampoline
  (lambda (bounce)
    (if (expval? bounce)
      bounce
      (trampoline (bounce)))))
```

value-of/k : $Exp \times Env \times Cont \rightarrow Bounce$

```
(define value-of/k
  (lambda (exp env cont)
    (cases expression exp
      (... (value-of/k ...))
      (... (apply-cont ...)))))
```

apply-cont : $Cont \times ExpVal \rightarrow Bounce$

```
(define apply-cont
  (lambda (cont val)
    (cases continuation cont
      (... val)
      (... (value-of/k ...))
      (... (apply-cont ...))
      (... (apply-procedure/k ...))))))
```

apply-procedure/k : $Proc \times ExpVal \times Cont \rightarrow Bounce$

```
(define apply-procedure/k
  (lambda (proc1 val cont)
    (lambda ()
      (cases procedure proc1
        (... (value-of/k ...))))))
```