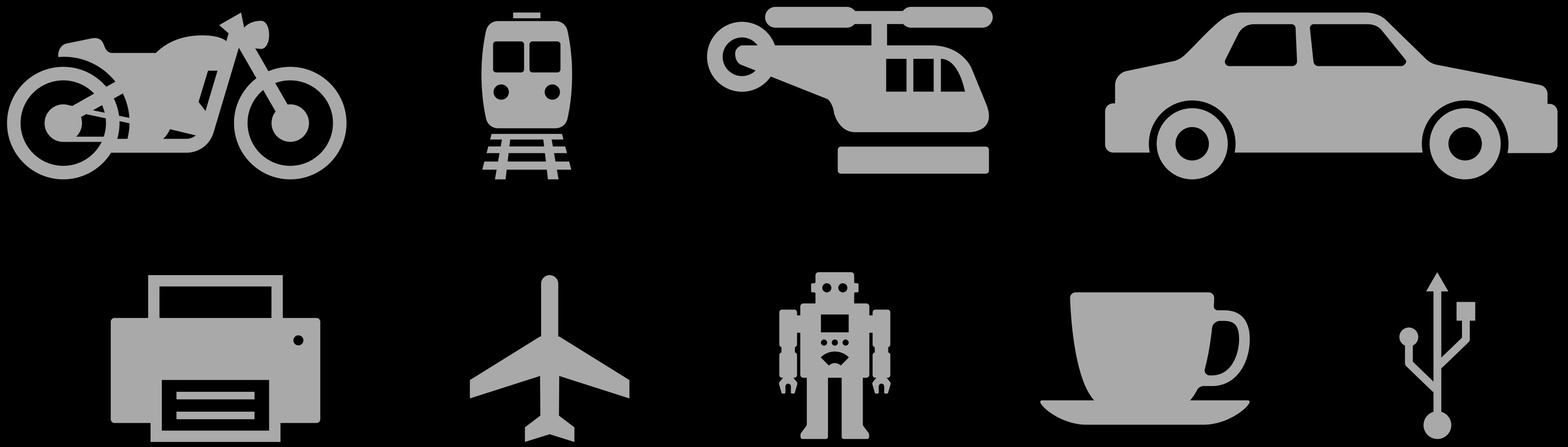
Pipit on the post(-condition)

Amos Robinson, FP-Syd, March 2024

photo: Australian pipit, Hexham swamp, NSW

Reactive systems



Pipit

Streaming computations with Pipit

```
let count_when (max: nat) (inc: stream bool): stream nat =
  let rec pre_count = 0 `fby` count
      and after_inc = pre_count + (if inc then 1 else 0)
      and count     = minimum after_inc max
  in count
```

Streaming computations with Pipit

```
let count_when (max: nat) (inc: stream bool): stream nat =
  let rec pre_count = 0 `fby` count
      and after_inc = pre_count + (if inc then 1 else 0)
      and count     = minimum after_inc max
  in count
```

---

max	3	-	-
inc	true	false	true
pre_count	0	1	1
after_inc	1	1	2
count	1	1	2

Streaming computations with Pipit

```
let count_when (max: nat) (inc: stream bool): stream nat =
  let rec pre_count = 0 `fby` count
      and after_inc = pre_count + (if inc then 1 else 0)
      and count     = minimum after_inc max
  in count
```

Intrinsic proofs

```
let count_when (max: nat) (inc: stream bool)
                  : stream nat =
  let rec pre_count: stream nat
                  = 0 `fby` count

      and after_inc: stream nat
                  = pre_count + (if inc then 1 else 0)

      and count:     stream nat
                  = minimum after_inc max

  in count
```

Intrinsic proofs

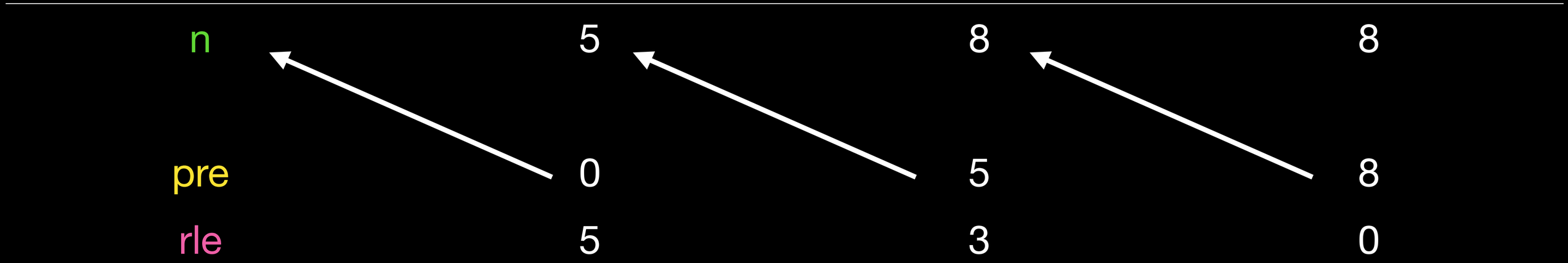
```
let count_when (max: nat) (inc: stream bool)
                  : stream nat {  c. 0 <=  c <= max } =
  let rec pre_count: stream nat { pc. 0 <= pc <= max }
                  = 0 `fby` count

      and after_inc: stream nat { ai. 0 <= ai <= max + 1 }
                  = pre_count + (if inc then 1 else 0)

      and count:     stream nat {  c. 0 <=  c <= max }
                  = minimum after_inc max

  in count
```

Run-length encoding

```
let run_length (n: stream nat)
  : stream nat =
  let rec pre = 0 `fby` n
      and rle = n - pre
  in rle
```

Run-length encoding

```
let run_length (n: stream nat)
  : stream nat =
  let rec pre = 0 `fby` n
      and rle = n - pre
  in rle
```



Run-length encoding

```
let run_length (n: stream nat { n. nondecreasing n })
  : stream nat =
  let rec pre = 0 `fby` n
      and rle = n - pre
  in rle
```

Run-length encoding

```
let count = count_when max inc in
let rle   = run_length ~~count~~ in
                       ^^^^^
        expected type: stream nat { c. nondecreasing c }
        actual type:   stream nat { c. 0 <= c <= max }
```

Intrinsic proofs - more properties?

```
let count_when (max: nat) (inc: stream bool)
  : stream nat {  c.
        0 <=  c <= max
    /\ nondecreasing c
    /\ (not inc ==> c == 0 `fby` c)
    /\ (    inc ==> c <= (0 `fby` c) + 1)
}
```

Intrinsic proofs - more properties?

```
let count_when (max: nat) (inc: stream bool)
  : stream nat {  c.
   let rec pre_count = 0 `fby` count
       and after_inc = pre_count + (if inc then 1 else 0)
       and count     = minimum after_inc max
    in c == count
    /\ 0 <=  c <= max
    /\ nondecreasing c
}
```

Extrinsic proofs

```
let count_when (max: nat) (inc: stream bool): stream nat =
  let rec pre_count = 0 `fby` count
      and after_inc = pre_count + (if inc then 1 else 0)
      and count     = minimum after_inc max
  in count


lemma count_when_in_range:
  forall (max: nat) (inc: stream bool) (⏰: time).
    0 <= (count_when max inc) @ ⏰ <= max
```

Mixtrinsic proofs

```
let count_when? (max: nat) (inc: stream bool): stream nat =
  let rec pre_count = 0 `fby` count
      and after_inc = pre_count + (if inc then 1 else 0)
      and count     = minimum after_inc max
  in
  check? (0 <= count <= max);
  count
```

Mixtrinsic proofs

```
let count_when? (max: nat) (inc: stream bool): stream nat =
  let rec pre_count = 0 `fby` count
      and after_inc = pre_count + (if inc then 1 else 0)
      and count     = minimum after_inc max
  in
  check? (0 <= count <= max);
  count

let count_when: nat -> stream nat -> stream nat =
  _ by (pipit_verify `count_when?)
```

Mixtrinsic proofs - preconditions

```
let run_length? (n: stream nat): stream nat =
  let rec pre = 0 `fby` n
      and rle = n - pre
  in
  check? (nondecreasing n);
  rle
```

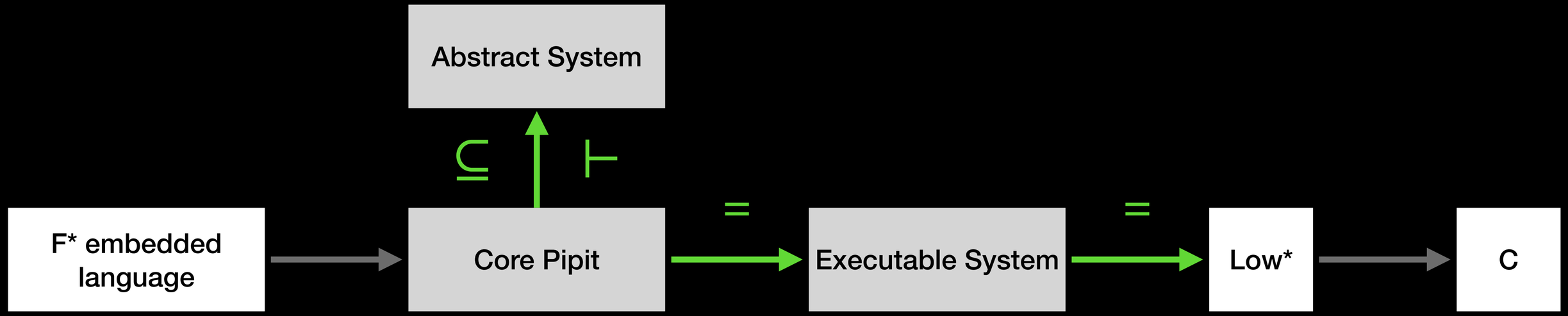
Mixtrinsic proofs - preconditions

```
let run_length? (n: stream nat): stream nat =
  let rec pre = 0 `fby` n
      and rle = n - pre
  in
  check? (nondecreasing n);
  rle

let run_length_count? (max: nat) (inc: stream bool)
  : stream nat =
  run_length (count_when max inc)
let run_length_count = _ by (pipit_verify `run_length_count?)
```

Mixtrinsic proofs - abstraction

```
let run_length? (n: stream nat): stream nat =
  contract?
    { nondecreasing n }
    (n - (0 `fby` n))
    { rle. rle >= 0 }
```

Pipit overview



verified

trusted (unverified)

Pipit

photo: Australian pipit, Hunter Wetlands National Park, NSW