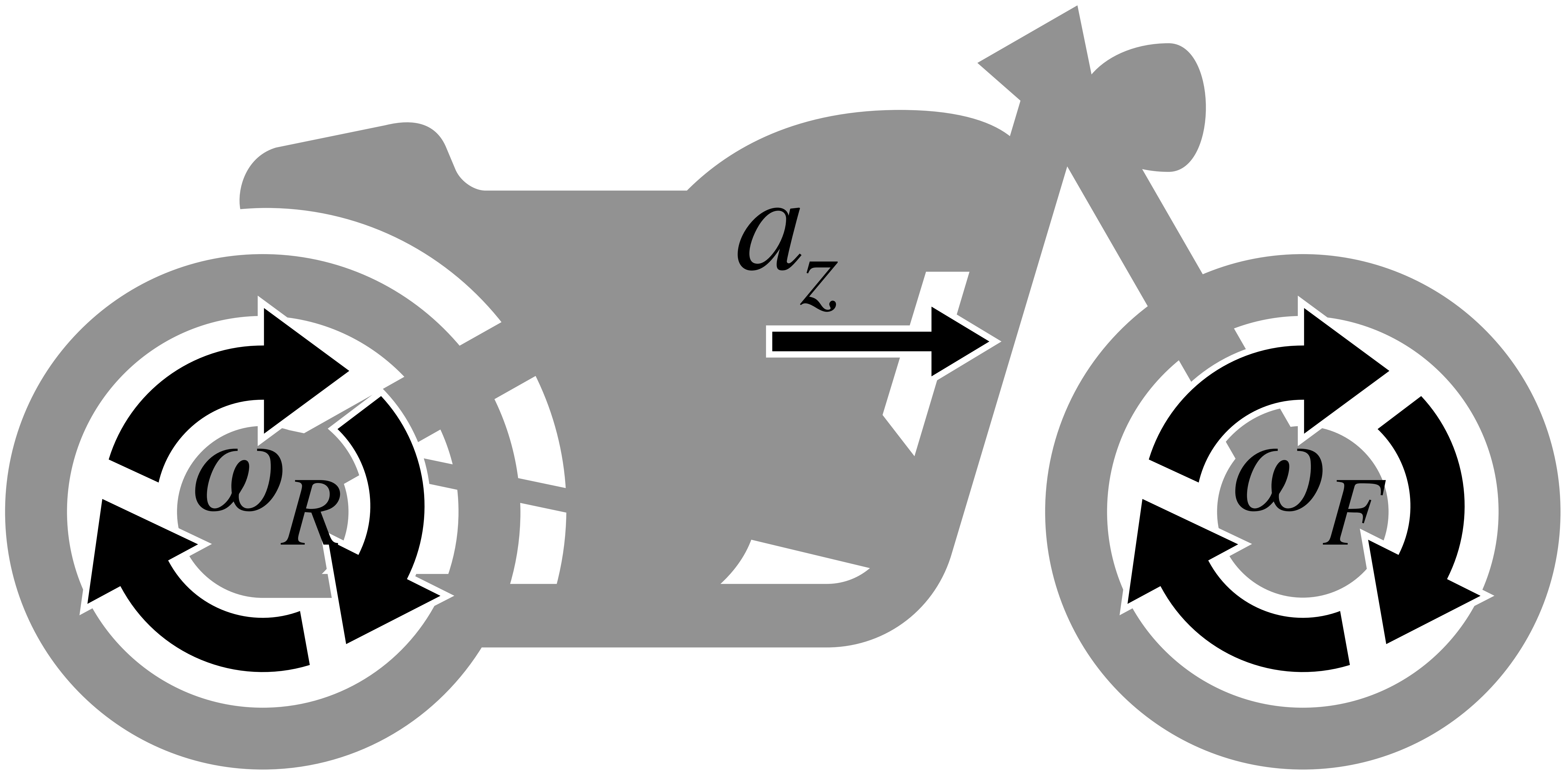
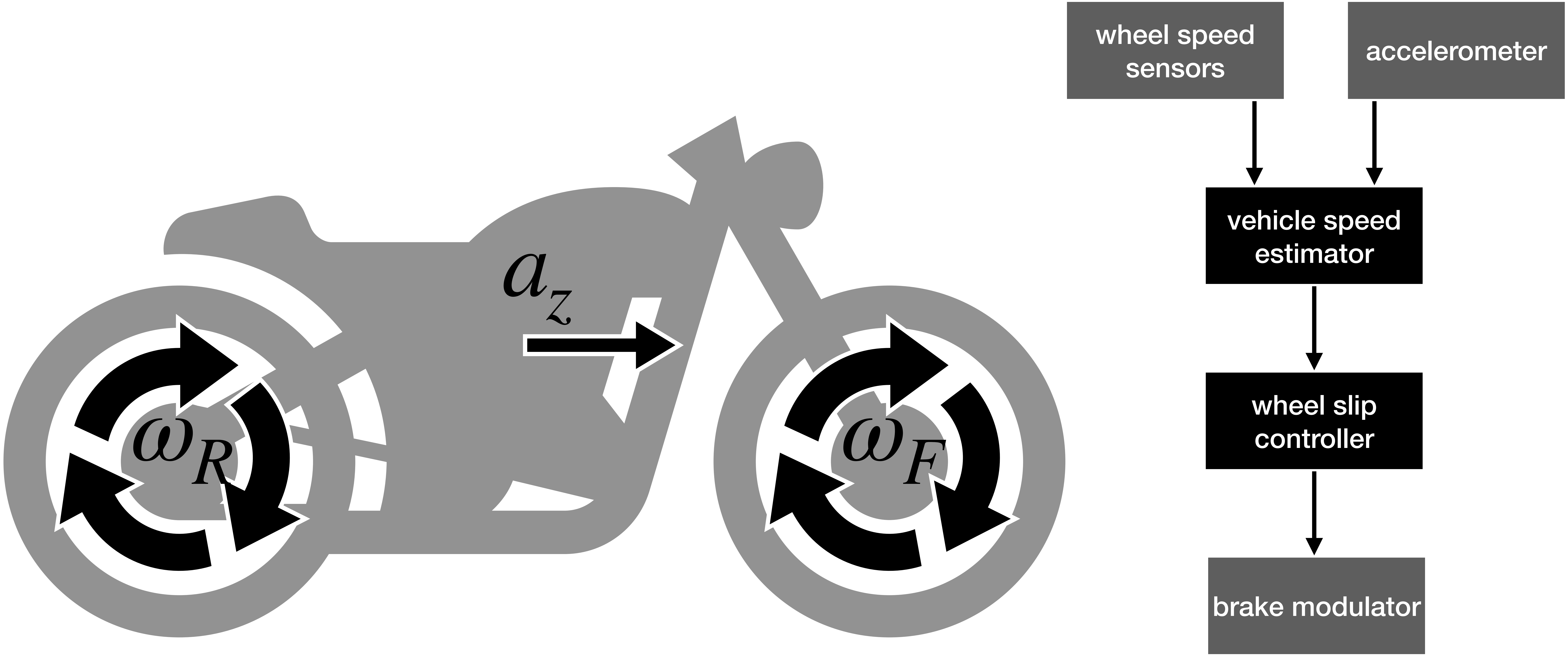
Pipit: reactive systems in F*

Amos Robinson, Australian National University

Anti-lock brakes for a motorcycle



Anti-lock brakes for a motorcycle



Vehicle speed estimator

let veh_speed_estimator $\omega_F\ \omega_R\ a_z\ \lfloor\hat{v}\rfloor\ \lceil\hat{v}\rceil$ =

…called every 10ms…

let $\lfloor\hat{v}'\rfloor$ = …updated lower bound… **in**

let $\lceil\hat{v}'\rceil$ = …updated upper bound… **in**

$(\lfloor\hat{v}'\rfloor, \lceil\hat{v}'\rceil)$

Vehicle speed estimator

let veh_speed_estimator $\omega_F\ \omega_R\ a_z\ \lfloor \hat{v} \rfloor\ \lceil \hat{v} \rceil$ =

let $v_F = \omega_F \cdot$ circumference **in**

let $v_R = \omega_R \cdot$ circumference **in**

let $\lfloor \hat{v}' \rfloor$ = **if** $v_F \approx_\epsilon v_R$ **then** $\min v_F\ v_R$ **else** $\lfloor \hat{v} \rfloor + a_z - \epsilon$ **in**

let $\lceil \hat{v}' \rceil$ = **if** $v_F \approx_\epsilon v_R$ **then** $\max v_F\ v_R$ **else** $\lceil \hat{v} \rceil + a_z + \epsilon$ **in**

$(\lfloor \hat{v}' \rfloor, \lceil \hat{v}' \rceil)$

What properties do we want our estimator to have?

- if the wheels agree, the estimate is pretty good

$$v_F \approx_\epsilon v_R \implies \lfloor \hat{v}' \rfloor \approx_\epsilon \lceil \hat{v}' \rceil$$

What properties do we want our estimator to have?

- if the wheels agree, the estimate is pretty good

$$v_F \approx_\epsilon v_R \implies \lfloor \hat{v}' \rfloor \approx_\epsilon \lceil \hat{v}' \rceil$$

easy proof:

$\lfloor \hat{v}' \rfloor$ = **if** $v_F \approx_\epsilon v_R$ **then** $\min v_F\ v_R$ **else** …

$\lceil \hat{v}' \rceil$ = **if** $v_F \approx_\epsilon v_R$ **then** $\max v_F\ v_R$ **else** …

What properties do we want our estimator to have?

- if the wheels agreed within time t, the estimate is not *too* bad

$$\blacklozenge_t (v_F \approx_\epsilon v_R) \implies \lfloor \hat{v}' \rfloor \approx_{t\epsilon} \lceil \hat{v}' \rceil$$

What properties do we want our estimator to have?

- if the wheels agreed within time t, the estimate is not *too* bad

$$\blacklozenge_t (v_F \approx_\epsilon v_R) \implies \lfloor \hat{v}' \rfloor \approx_{t\epsilon} \lceil \hat{v}' \rceil$$

how do we even state this? not trivial!

val veh_speed_estimator (ω_F ω_R: wheel) (a_z: accel) ($\lfloor \hat{v} \rfloor$ $\lceil \hat{v} \rceil$: vel)

: (vel & vel)

As a reactive system

let node veh_speed_estimator $\omega_F\ \omega_R\ a_z$=

let $v_F = \omega_F \cdot$ circumference **in**

let $v_R = \omega_R \cdot$ circumference **in**

let rec $\lfloor \hat{v} \rfloor$ = **if** $v_F \approx_\epsilon v_R$

then $\min v_F\ v_R$

else $(\min v_F\ v_R \rightarrow$ **pre** $\lfloor \hat{v} \rfloor) + a_z - \epsilon$ **in**

let rec $\lceil \hat{v} \rceil$ = **if** $v_F \approx_\epsilon v_R$

then $\max v_F\ v_R$

else $(\max v_F\ v_R \rightarrow$ **pre** $\lceil \hat{v} \rceil) + a_z + \epsilon$ **in**

$(\lfloor \hat{v} \rfloor, \lceil \hat{v} \rceil)$

As a reactive system

let node veh_speed_estimator $\omega_F\ \omega_R\ a_z$=

let $v_F = \omega_F \cdot$ circumference **in**

let $v_R = \omega_R \cdot$ circumference **in**

let rec $\lfloor \hat{v} \rfloor$ = **if** $v_F \approx_\epsilon v_R$

then $\min v_F\ v_R$

else $(\min v_F\ v_R \rightarrow$ **pre** $\lfloor \hat{v} \rfloor) + a_z - \epsilon$ **in**

let rec $\lceil \hat{v} \rceil$ = **if** $v_F \approx_\epsilon v_R$

then $\max v_F\ v_R$

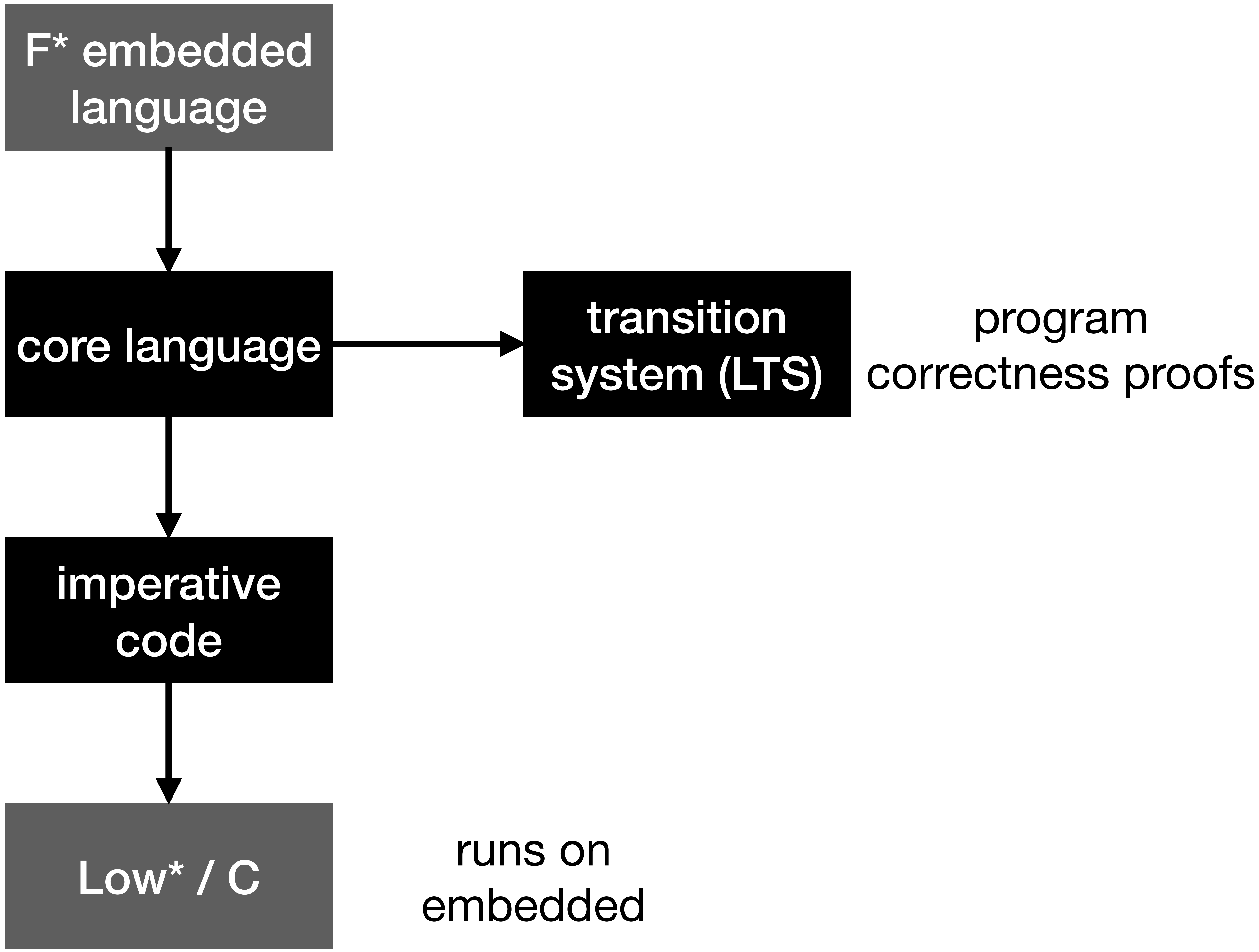
else $(\max v_F\ v_R \rightarrow$ **pre** $\lceil \hat{v} \rceil) + a_z + \epsilon$ **in**

check $(\blacklozenge_t (v_F \approx_\epsilon v_R) \implies \lfloor \hat{v} \rfloor \approx_{t\epsilon} \lceil \hat{v} \rceil)$;

$(\lfloor \hat{v} \rfloor, \lceil \hat{v} \rceil)$

Another pipit

Pipit structure



Core language

$$
\begin{aligned}
e :=\ & \\
& \mid v \mid x \mid e\ e' \\
& \mid \text{pre } e \\
& \mid e \to e' \\
& \mid \mu x.\ e[x] \\
& \mid \text{let } x = e \text{ in } e'[x] \\
& \mid \text{check } p\ e \text{ in } e'
\end{aligned}
$$

Core language

$e :=$

$| \, v \, | \, x \, | \, e \; e'$

$| \, \text{pre} \; e$

$| \, e \to e'$

$| \, \mu x. \; e[x]$

$| \, \text{let } x = e \text{ in } e'[x]$

$| \, \text{check } p \; e \text{ in } e'$

$$\frac{}{\Sigma \vdash v \Downarrow v} \text{(Value)}$$

$$\frac{}{\Sigma; \sigma \vdash x \Downarrow \sigma(x)} \text{(Var)}$$

Core language

$$e := $$
$$| v \mid x \mid e\ e'$$
$$| \text{pre } e$$
$$| e \to e'$$
$$| \mu x.\ e[x]$$
$$| \text{let } x = e \text{ in } e'[x]$$
$$| \text{check } p\ e \text{ in } e'$$

$$\frac{\Sigma \vdash e \Downarrow v}{\Sigma; \sigma \vdash \text{pre } e \Downarrow v}\ (\text{Pre})$$

Core language

e :=

$| \; v \; | \; x \; | \; e \; e'$

| pre e

$| \; e \rightarrow e'$

$| \; \mu x. \; e[x]$

| let $x = e$ in $e'[x]$

| check p e in e'

$$\frac{\sigma \vdash e \Downarrow v}{\sigma \vdash e \rightarrow e' \Downarrow v} \; (\rightarrow_1)$$

$$\frac{\Sigma; \sigma \vdash e' \Downarrow v'}{\Sigma; \sigma \vdash e \rightarrow e' \Downarrow v'} \; (\rightarrow_S)$$

Core language

$e :=$

$\mid v \mid x \mid e\ e'$

$\mid \text{pre}\ e$

$\mid e \to e'$

$\mid \mu x.\ e[x]$

$\mid \text{let}\ x = e\ \text{in}\ e'[x]$

$\mid \text{check}\ p\ e\ \text{in}\ e'$

$$\frac{\Sigma \vdash e[x := \mu x. e] \Downarrow v}{\Sigma \vdash \mu x. e \Downarrow v}\ (\mu)$$

Core language

e :=

$\mid v \mid x \mid e\ e'$

$\mid$ pre e

$\mid e \rightarrow e'$

$\mid \mu x.\ e[x]$

$\mid$ let $x = e$ in $e'[x]$

$\mid$ check $p\ e$ in e'

$$\frac{\Sigma \vdash e'[x := e] \Downarrow v}{\Sigma \vdash \text{let } x = e \text{ in } e' \Downarrow v}\ (\text{Let})$$

Core language

$e :=$

$\mid v \mid x \mid e\ e'$

$\mid \text{pre}\ e$

$\mid e \to e'$

$\mid \mu x.\ e[x]$

$\mid \text{let}\ x = e\ \text{in}\ e'[x]$

$\mid \text{check}\ p\ e\ \text{in}\ e'$

$$\frac{\Sigma \vdash e \Downarrow \top \qquad \Sigma \vdash e' \Downarrow v'}{\Sigma \vdash \text{check}\ p\ e\ \text{in}\ e' \Downarrow v'}\ \text{(Check)}$$

Flying pipit

Deep embedding: applicative functor

type stream α = name_supply $\to$ (exp α & name_supply)

val pure : $\alpha \to$ stream α

val (<$>) : $(\alpha \to \beta) \to$ stream $\alpha \to$ stream β

val (<*>) : stream $(\alpha \to \beta) \to$ stream $\alpha \to$ stream β

Deep embedding: applicative functor

```
type stream α = name_supply → (exp α & name_supply)

val pure  :                α                      → stream α
val (<$>) :           (α → β) → stream α → stream β
val (<*>) : stream (α → β) → stream α → stream β

let if_then_else (p: stream 𝔹) (s1 s2: stream α): stream α =
  (λ p' s1' s2'. if p then s1' else s2')
    <$> p <*> s1 <*> s2
```

Deep embedding: streaming

type stream α = name_supply → (exp α & name_supply)

— *delay*

val pre: stream α → stream α

— "*then*"

val (→): stream α → stream α → stream α

Deep embedding: bindings

type stream α = name_supply → (exp α & name_supply)

—*let bindings*

val let': stream α → (stream α → stream β) → stream β

— *recursive stream (μ)*

val rec': (stream α → stream α) → stream α

Idealised program

let node veh_speed_estimator $\omega_F\ \omega_R\ a_z$=

let $v_F = \omega_F \cdot$ circumference **in**

let $v_R = \omega_R \cdot$ circumference **in**

let rec $\lfloor \hat{v} \rfloor$ = **if** $v_F \approx_\epsilon v_R$

then $\min v_F\ v_R$

else $(\min v_F\ v_R \rightarrow$ **pre** $\lfloor \hat{v} \rfloor) + a_z - \epsilon$ **in**

let rec $\lceil \hat{v} \rceil$ = **if** $v_F \approx_\epsilon v_R$

then $\max v_F\ v_R$

else $(\max v_F\ v_R \rightarrow$ **pre** $\lceil \hat{v} \rceil) + a_z + \epsilon$ **in**

check ($\blacklozenge_t (v_F \approx_\epsilon v_R) \implies \lfloor \hat{v} \rfloor \approx_{t\epsilon} \lceil \hat{v} \rceil$);

$(\lfloor \hat{v} \rfloor, \lceil \hat{v} \rceil)$

Actual program

let veh_speed_estimator (ω_F ω_R: stream wheel) (a_z: stream accel) =

let' ($\omega_F \cdot$ circumference) (λv_F .

let' ($\omega_R \cdot$ circumference) (λv_R .

letrec' ($\lambda \lfloor \hat{v} \rfloor$. **if_then_else** ($v_F \approx_\epsilon v_R$)

$(\min\ v_F\ v_R)$

$((\min\ v_F\ v_R \rightarrow$ **pre** $\lfloor \hat{v} \rfloor) + a_z - \epsilon))$ ($\lambda \lfloor \hat{v} \rfloor$.

letrec' ($\lambda \lceil \hat{v} \rceil$. **if_then_else** ($v_F \approx_\epsilon v_R$)

$(\max\ v_F\ v_R)$

$((\max\ v_F\ v_R \rightarrow$ **pre** $\lceil \hat{v} \rceil) + a_z + \epsilon))$ ($\lambda \lceil \hat{v} \rceil$.

check ($\blacklozenge_t (v_F \approx_\epsilon v_R) \implies \lfloor \hat{v} \rfloor \approx_{t\epsilon} \lceil \hat{v} \rceil$)

$(\lambda\ a\ b. (a, b))$ <$> $\lfloor \hat{v} \rfloor$ <*> $\lceil \hat{v} \rceil$

Problems with the embedding

- "stream" isn't a monad (no bind)
  - meta let-bindings duplicate expressions
  - no if-then-else syntax
- constants must be wrapped (pure 100)

Problems with the embedding

- "stream" isn't a monad (no bind)
  - meta let-bindings duplicate expressions
    $\Longrightarrow$ do sharing recovery / CSE on core
  - no if-then-else syntax
    $\Longrightarrow$ arrows in F*?
- constants must be wrapped (pure 100)
  $\Longrightarrow$ implicit coercions?

Future work

- verification:
  - finish proof of transition system
  - start proof of imperative codegen
- case studies:
  - anti-lock braking?
- improvements:
  - common subexpression elimination
- language features:
  - clocks, letrecs and contracts


Last pipit