Three philosophers consider the nature of existence:

To be is to do

— Socrates

To do is to be

— Sartre

Do be do be do

— Sinatra

Introducing...

# FRANK

a new (eventually)
programming language
from

Alta Systems Northern Ireland

# Doing and Being:
## an unfunny pun

In Haskell, `[Int]` is both

- a type of pure lists
- a type of nondeterministic numbers

```
[1,2] ⊕ [3,4] = [1,2,3,4]

⁅(⊕) [1,2] [3,4]⁆
= do x ← [1,2]
     y ← [3,4]
     return (x ⊕ y)
= [4,5,5,6]
```

# Types for Doing and Being

(thanks Paul)

$V ::= D\ V^*$ datatype

$\mid \{C\}$ suspended computation

$\mid \Sigma\langle\Sigma^*\rangle V$ request

$C ::= V \to C$ function

$\mid [\Sigma^*]V$ value, given capabilities

# Frank – syntax

$$
\begin{aligned}
E ::= {} & X \\
& \mid \mathsf{Con}(E, \ldots, E) \\
& \mid \{(P, \ldots, P) \mapsto E \\
& \quad \mid \ldots\} \\
& \mid E(E, \ldots, E) \\
& \mid E \,?\, E
\end{aligned}
$$

$$
\begin{aligned}
P ::= {} & X \\
& \mid \mathsf{Con}(P, \ldots, P) \\
& \mid \mathsf{Msg}(P, \ldots, P \mid X)
\end{aligned}
$$

# Frank – typing (I)

- $\Gamma ::= (X : V)^*$ variables have values
- judgment carries context and capability

$$\boxed{\Gamma\,[\Sigma^*] \vdash E : V}$$

$$\frac{}{\Gamma\,[\Sigma_s] \vdash x : V}\; x : V \in \Gamma$$

$$\frac{\Gamma\,[\Sigma_s] \vdash e_i : V_i[\vec{\alpha} ::= \vec{T}]}{\Gamma\,[\Sigma_s] \vdash Con(e_1, \ldots, e_n) : D\,\vec{T}}\; Con(V_1, \ldots, V_n) : D\vec{\alpha}$$

$$\frac{\Gamma\,[\Sigma_s] \vdash e : \{V_1 \to \cdots \to V_n \to [\Sigma_s']V\} \quad \Gamma\,[\Sigma_s] \vdash e_i : V_i}{\Gamma\,[\Sigma_s] \vdash e(e_1, \ldots, e_n) : V}\; \Sigma_s' \subseteq \Sigma_s$$

# Frank – typing (II)

- pattern typing is the obvious linear thing

$$\boxed{\Gamma \vdash P : V}$$

$$\frac{}{x : V \vdash x : V} \qquad \frac{\Gamma_i \vdash p_i : V_i[\vec{\alpha} := \vec{T}] \quad Con(V_1, \ldots, V_n) : D\,\vec{\alpha}}{\Gamma_1, \ldots, \Gamma_n \vdash Con(p_1, \ldots, p_n) : D\,\vec{T}}$$

- functions delay effects

$$\frac{\Gamma_{ij} \vdash p_{ij} : V_j \qquad \Gamma_i, \Gamma_{i1}, \ldots, \Gamma_{in} \; [\Sigma s'] \vdash e_i : V}{\Gamma \; [\Sigma s] \vdash \{ (\vec{p_i}) \mapsto e_i \} : \{ V_1 \to \ldots \to [\Sigma s'] V \}}$$

no connection

# Frank - signatures (interfaces?)

$$\mathsf{sig}\ S\ \vec{\alpha} = \{\ M(V, \ldots, V) : V \mid \ldots\ \}$$

results in (for all $\vec{\alpha}$)

$$M : \{V_1 \to \ldots \to V_n \to [S\ \vec{\alpha}]V\}$$

Examples, please!

$$\mathsf{sig}\ \mathsf{State}\ \sigma = \{\ \mathsf{get}\ () : \sigma \mid \mathsf{put}\ (\sigma) : 1\ \}$$

$$\mathsf{sig}\ \mathsf{Fail} = \{\ \mathsf{fail}\ () : 0\ \}$$

# Frank – semantics

- a signature determines a functor (indeed, a container)

$$\hat{\Sigma}\, X = \sum_{(V_1,\ldots,V_n):V \in \Sigma} V_1 \times \ldots \times V_n \times (V \to X)$$

  so does a bunch of signatures, additively

- we can (sweeping much under carpet) interpret doings in $[\Sigma_S]V$ as elements of $\hat{\Sigma}_S^* V$, where

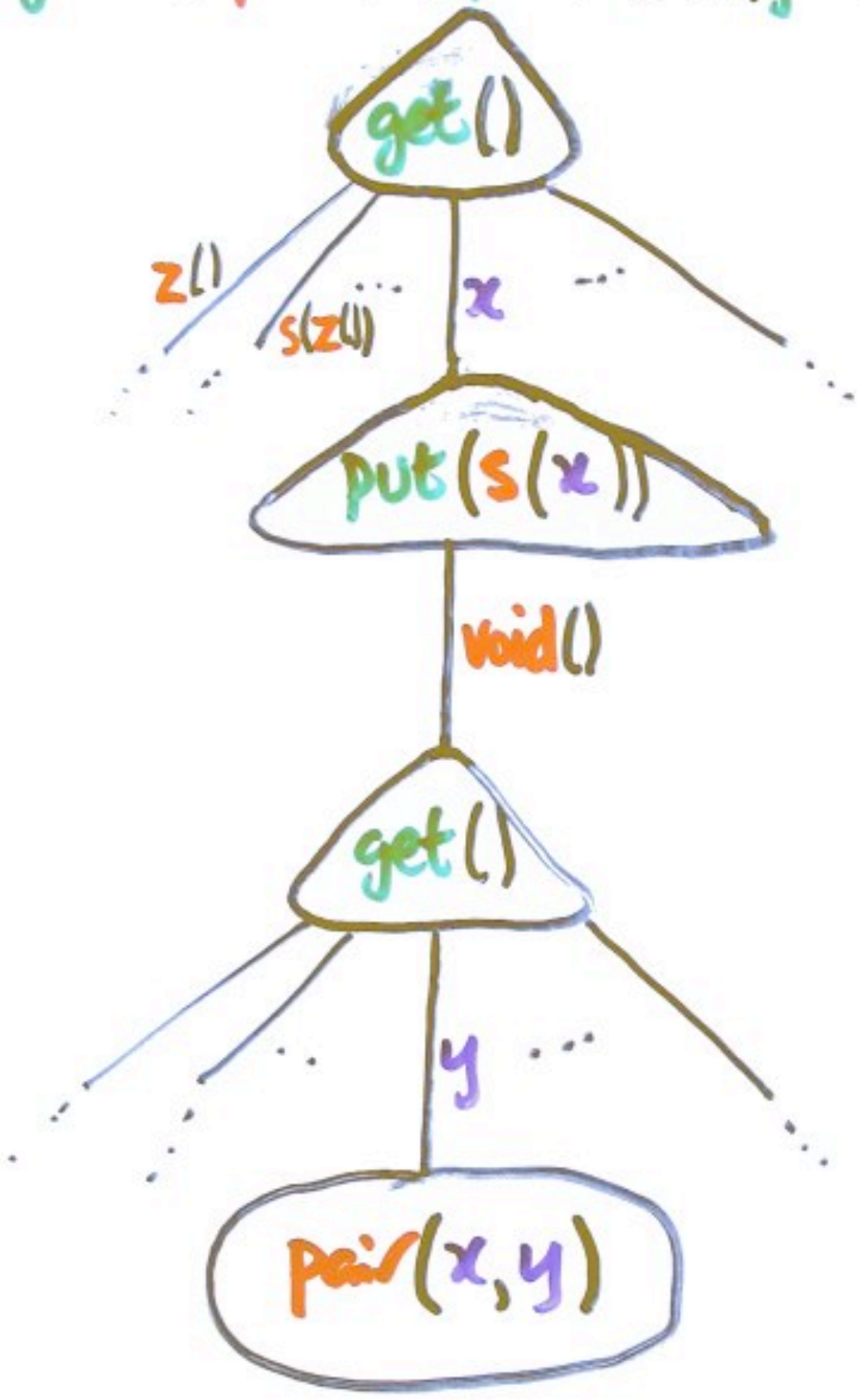
$$F^* V = V + F(F^* V)$$

  is the usual free monad construction

- the remaining types are interpreted directly

# Frank – semantics in a picture

`x ← get() ; pair(x, put(s(x)); get())`



# Is anybody listening?

- types of messages
- message patterns

$$\frac{m\,(V_1,\ldots,V_n) : V \in \Sigma \qquad \Gamma_i \vdash p_i : V_i}{\Gamma_1;\ldots;\Gamma_n;r : V \to [\Sigma;\Sigma_s]U \vdash m(p_1,\ldots,p_n|r) : \Sigma\langle\Sigma_s\rangle U}$$

$$\frac{\Gamma \vdash p : U}{\Gamma \vdash \mathbf{ret}(p) : \Sigma\langle\Sigma_s\rangle U}$$

(so $\Sigma\langle\Sigma_s\rangle U = U + \hat{\Sigma}([\Sigma;\Sigma_s]U)$ )

- so is anybody listening?

I'm listening

$$\frac{\Gamma \vdash l : \{\, \Sigma\langle \Sigma_s \rangle U \to [\Sigma_s] V \}
\qquad \Gamma \vdash e : [\Sigma; \Sigma_s] U}{\Gamma \vdash l\,?\,e : [\Sigma_s] V}$$

- you what?

```
state : σ → {State σ ⟨Σs⟩ u → [Σs]u}
state (s) = {
  ret(u) ↦ u
| get (|r) ↦ state(s) ? r(s)
| put (s'|r) ↦ state(s') ? r (void())}

try : {Fail⟨Σs⟩u → [Σs]Maybe u}
try = {
  ret(u) ↦ just(u)
| fail (|r) ↦ nothing }
```

# So, what should effectful programs look like?

- I'm sure I have too much punctuation.
- Is everything the right colour?

- Type system book-keeps values separately from messages.
- Polymorphism? Type inference?
- main as a listener for events, sending to display, file system, etc
- $\mu$ versus $\nu$ ?
- static state in signatures?
- high apple pie in the sky hopes?