Talking to Frank

Craig McLaughlin

The University of Edinburgh

March 26, 2019

Joint work with Lukas Convent, Sam Lindley and Conor McBride
What’s Frank?

Frank:
- strict functional language
- effects as collections of *commands* (effect operations)

Novelties:
- effect type system for statically tracking effects
- effect handling arising from *generalising function application*

Implementation:

https://www.github.com/frank-lang/frank — try today!
map : {X -> Y} -> List X -> List Y

map f nil = nil
map f (x :: xs) = f x :: map f xs
map : \{X \to Y\} \to \text{List } X \to \text{List } Y

map f \text{ nil} = \text{ nil}

map f (x :: xs) = f x :: \text{ map } f \text{ xs}

map \{n \to n+1\} [1,2,3] \implies [2,3,4]
interface Abort = abort X : X

interface Write X = tell : X -> Unit

interface Read X = ask : X
Example: Writing a List

interface Write X = tell : X -> Unit

writeList : List X -> [Write X]Unit

writeList xs = map tell xs; unit
interface Write X = tell : X -> Unit

writeList : List X -> [Write X]Unit

writeList xs = map tell xs; unit

[Σ]

“Hi, I’m an ability.
The environment must be able to handle effects declared in Σ”
Example: Interpreting Read and Write

state : S \rightarrow <\text{Read } S, \text{Write } S> X \rightarrow X

state _ x = x

state s <\text{ask } \rightarrow k> = state s (k s)

state _ <\text{tell } s \rightarrow k> = state s (k \text{ unit})
Example: Interpreting Read and Write

\[
\text{state} : S \to \langle \text{Read } S, \text{Write } S \rangle X \to X
\]

\[
\text{state }_x x = x
\]

\[
\text{state } s \langle \text{ask } -> k \rangle = \text{state } s (k s)
\]

\[
\text{state }_x \langle \text{tell } s -> k \rangle = \text{state } s (k \text{ unit})
\]

\[
\langle \Delta \rangle
\]

“Hi, I’m an *adjustment*. The effects declared in $\Delta$ must be handled locally.”
Desugaring The Type of Map

map : \{X \to Y\} \to \text{List}\ X \to \text{List}\ Y

desugars to

\langle \iota \rangle \{ \langle \iota \rangle X \to [\varepsilon]Y \} \to \langle \iota \rangle \text{List}\ X \to [\varepsilon]\text{List}\ Y

Aside for Haskell programmers: We've got something that's equivalent to both map and mapM!
Desugaring The Type of Map

map : \{X -> Y\} -> List X -> List Y

desugars to

\langle \iota \rangle \{\langle \iota \rangle X -> [\varepsilon]Y\} -> \langle \iota \rangle \text{List} X -> [\varepsilon] \text{List} Y

Aside for Haskell programmers:

We’ve got something that’s equivalent to both \text{map} and \text{mapM}!
Demo
Conclusions:

- Application generalises to account for both functions & handlers
- Effect type system: effects tracked and pushed inwards
- Convenient syntactic sugars: rarely need specify effect variables
- Adaptors provide general rewiring of effects in the ambient ability
Conclusions:

- Application generalises to account for both functions & handlers
- Effect type system: effects tracked and pushed inwards
- Convenient syntactic sugars: rarely need specify effect variables
- Adaptors provide general rewiring of effects in the ambient ability

https://www.github.com/frank-lang/frank
catch : <Abort> X -> \{X\} -> X

catch x _ = x

catch <aborting -> _> h = h!
catch : <Abort> X -> {X} -> X

catch x _ = x

catch < aborting -> _ > h = h!

catchError :: -- Haskell
         MonadError () m => m a -> (() -> m a) -> m a

Imprecise typing (() -> m a) permits alternative to throw errors!