

Reversible Session-Based Concurrency

An implementation in Haskell

Folkert de Vries

University of Groningen, The Netherlands,

Supervisors:
Jorge Pérez
Gerard Renardel

Bachelor Symposium 2018
June 26, 2018

TFP Conference



Concurrency is hard

Bridging the gap between theory and practice

Goal

A functional implementation of the model in the paper
“Causally Consistent Reversible Choreographies” (PPDP 2017)
by Mezzina and Pérez

Our contributions:

- ▶ practical validation of theoretical ideas
- ▶ a natural connection between reversibility and immutable data structures in pure functional languages
- ▶ a step toward reversible concurrent debuggers and failure guiding evaluation

Core Concepts

- ▶ **Reversible** moving backwards through a program
- ▶ **Session-Based** sessions and their types
- ▶ **Concurrency** calculi for concurrent computation

Defining Concurrent Computation

The lambda-calculus

Our favorite model for sequential computation: The lambda calculus

$M, N ::= x$	Variable
$(\lambda x.M)$	Function definition
$(M N)$	Applying a function to an argument

β -reduction:

$$(\lambda x.M) E \rightarrow (M[x/E])$$

The pi-calculus

In contrast, the pi-calculus defines

$P, Q, R ::= \bar{x}\langle y \rangle.P$	Send the value y over channel x , then run P
$x(y).P$	Receive on channel x , bind the result to y , then run P
$P Q$	Run P and Q simultaneously
$(\nu x)P$	Create a new channel x and run P
$!P$	Repeatedly spawn copies of P
0	Terminate the process
$P + Q$	(Optionally) Nondeterministic choice

reduction:

$$\bar{x}\langle z \rangle.P | x(y).Q \rightarrow P|Q[z/y]$$

Sessions & Session Types

Session Types

Data types prevent us from making silly mistakes with **data**.

Session Types

Session types prevent common mistakes in **communication**.

- ▶ Send without a receive & receive without a send
- ▶ type mismatch between sent and expected value
- ▶ Sending/Receiving before you're supposed to
- ▶ undesired infinite recursion

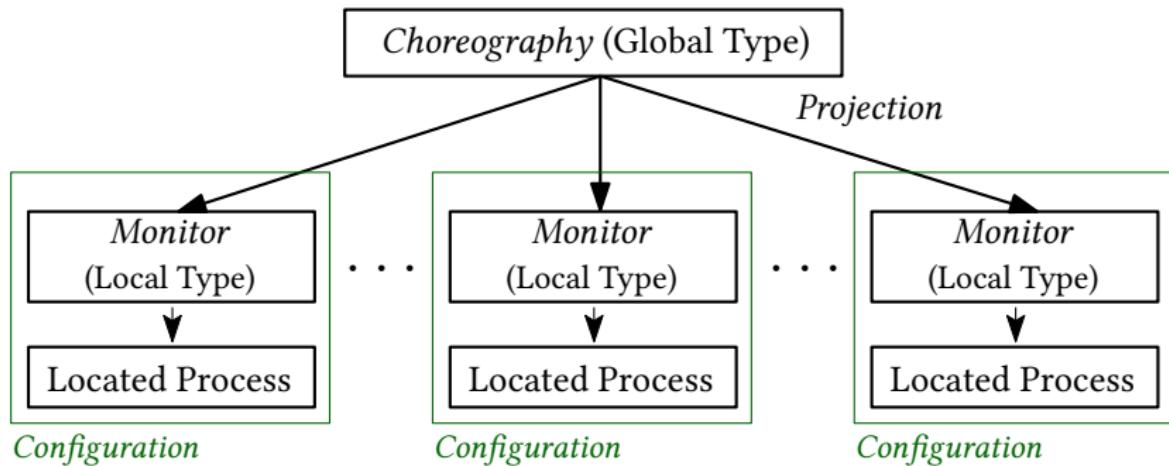
An idea introduced in the late 90s by Honda, Kubo, and Vasconcelos (ESOP 1998), and widely studied ever since.

Session Types

Global Type

- ▶ There is a Transaction between Carol and Bob, a Bool is sent.
- ▶ There is a Transaction between Alice and Carol, an Int is sent.

Projection



Session Types

Global Type

- ▶ There is a Transaction between Carol and Bob, a Bool is sent.
- ▶ There is a Transaction between Alice and Carol, an Int is sent.

Local Type (for Carol)

- ▶ I will send Bob a Bool
- ▶ I expect an Int from Alice

Session Types

Global Type

- ▶ There is a Transaction between Carol and Bob, a Bool is sent.
- ▶ There is a Transaction between Alice and Carol, an Int is sent.

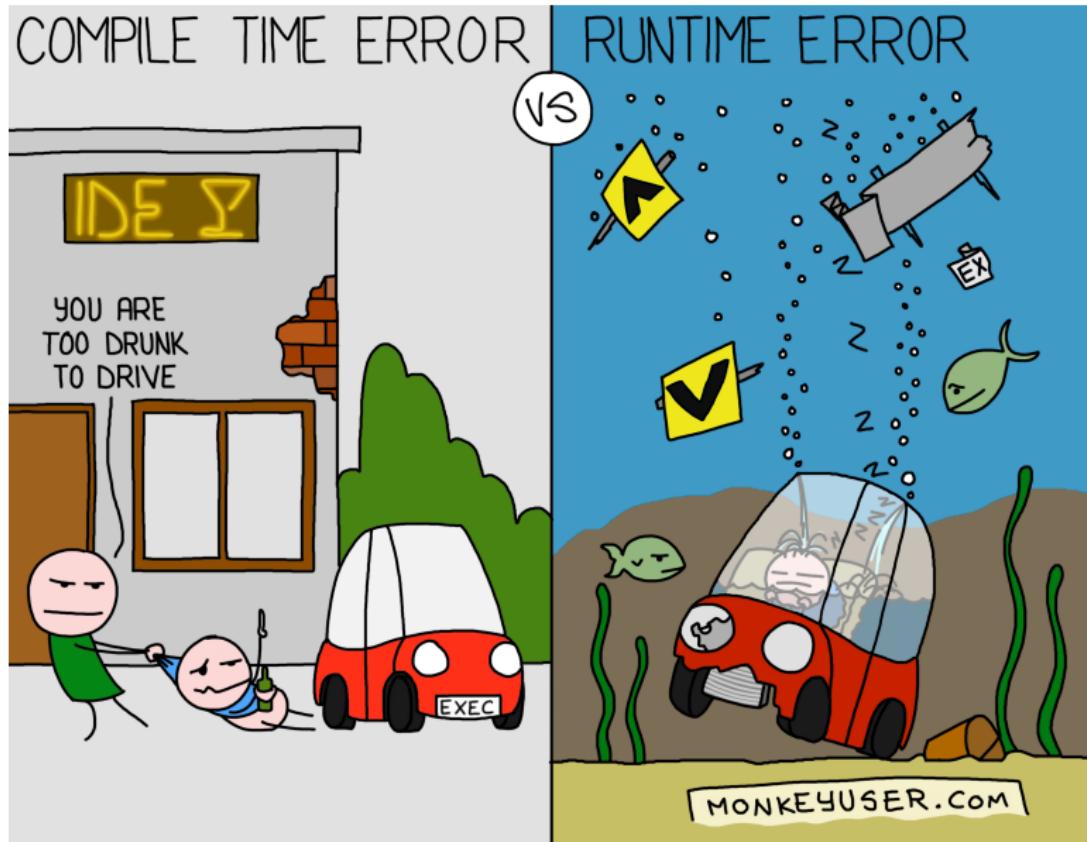
Local Type (for Carol)

- ▶ I will send Bob a Bool
- ▶ I expect an Int from Alice

Properties:

- ▶ Order: enforced ordering of communication over a channel
- ▶ Progress: every sent message is eventually received
- ▶ Safety: sent values are typed, there can be no mismatch

Session Types: Static vs. Dynamic



Session Types: Static vs. Dynamic

We will always need dynamic verification of our session types

Because in practice systems are:

- ▶ opaque
- ▶ written in multiple languages

Reversibility

Reversibility

Leave behind a trail of breadcrumbs



So we can always find our way back

Reversibility

Causal Consistency: Reversible steps lead to states that can be reached with forward steps only. No extra new states are introduced.

$$\text{backward} \circ \text{forward} \approx \text{identity}$$

Implementation

Implementation

We can define 3 data types

```
-- Our Process Calculus
data Program = ...
-- Our Global Session Type
data GlobalType = ...
-- Our "bread crumbs"
data TypeContext = ...
```

And then implement:

```
type Session a = StateT ExecutionState (Except Error) a
forward :: Location -> Session ()
backward :: Location -> Session ()
```

Advantages of a pure functional language

- ▶ Algebraic Data Types (Union Types)
 - ▶ We can stay close to the theory
 - ▶ invalid states are impossible to represent

$$G, G' ::= p \rightarrow q : \langle U \rangle . G \mid p \rightarrow q : \{I_i : G_i\}_{i \in I} \mid \mu X . G \mid X \mid \text{end}$$

```
data GlobalTypeF u next =
  Transaction {...} | Choice {...} | R next | V | End | Wk next
```

- ▶ Immutability & checked effects
 - ▶ Reversibility is easier and testable

Conclusion

An implementation of reversible, session-based concurrency in Haskell

- ▶ Embedding a sophisticated operational semantics in Haskell
- ▶ The first functional implementation of a reversible debugger
- ▶ <https://github.com/folkertdev/reversible-debugger/>

Current and future work:

- ▶ Graphical interfaces
- ▶ Controlled reversibility (checkpoints)

Questions

