Asynchronous Functional Reactive Processes

Daniel Winograd-Cort
Paul Hudak

Department of Computer Science
Yale University

IBM PL Day
Yorktown Heights, NY
Tuesday, November 18, 2014

The Context:
Functional Reactive Programming

- Programming with *continuous values* and *streams of events*.
- Like drawing *signal processing diagrams*:

signal processing diagram

≡

equivalent arrow syntax in Haskell

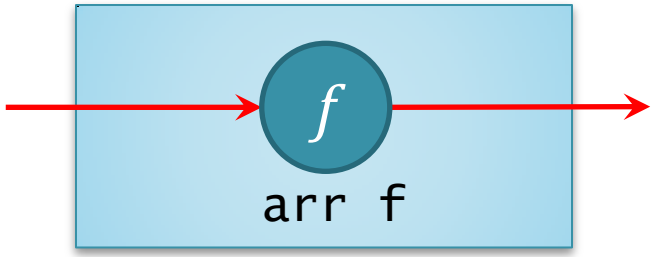
$$y \leftarrow sigfun \prec x$$

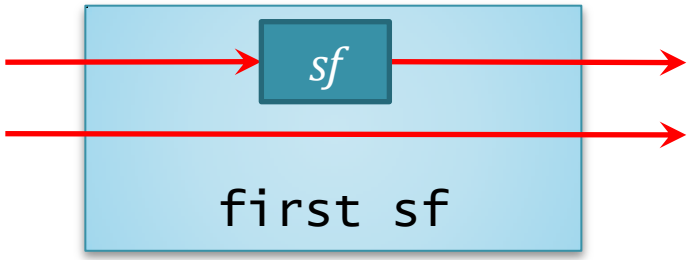
- Previously used in:
  - Yampa: robotics, vision, animation
  - Nettle: networking
  - Euterpea: sound synthesis and audio processing

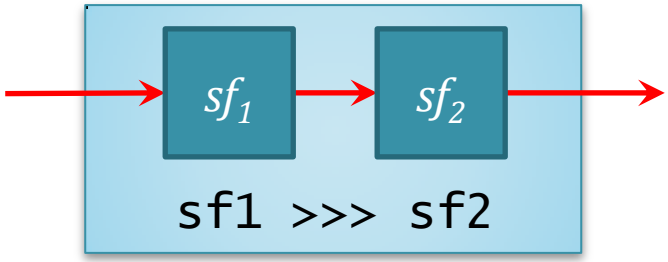
Fundamental Abstraction

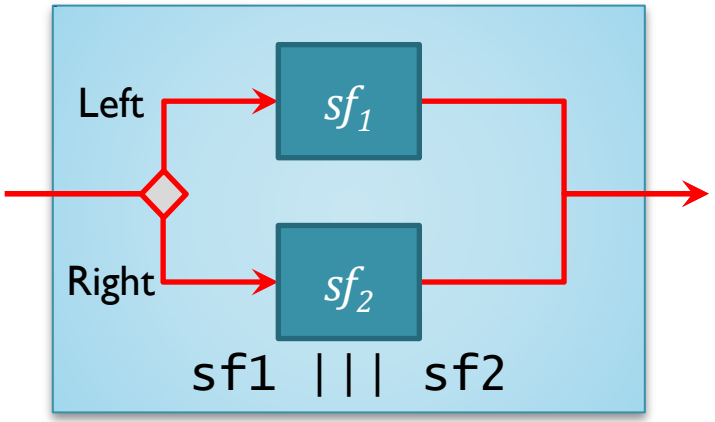
- Signal functions process infinitely fast, infinitely often
  - Within the signal function, there is no notion of time.
  - The data itself governs the passage of time.
- Clear, commutative design
- Synchronization as a given

Standard Arrow Operators









Adding Effects

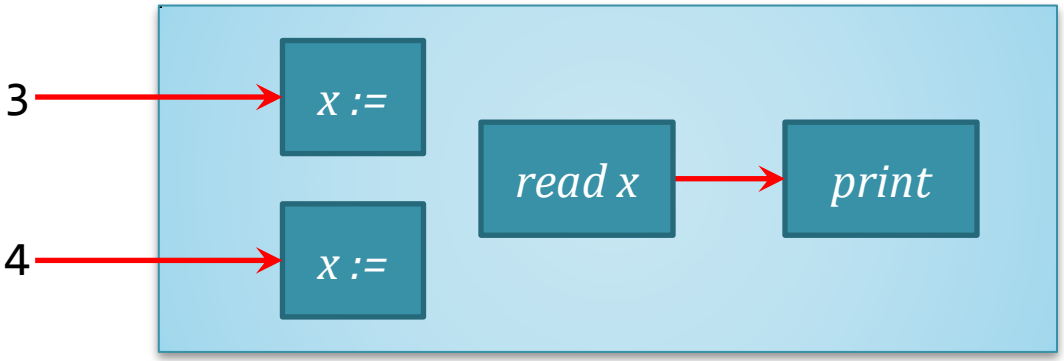
- Typically, effects are sequenced by the structure of the program
  - Consider the following program:

```
x := 3;
print x;
x := 4;
```

When the program completes, x will be 4 and we will have printed 3.

Adding Effects

- In FRP, the data controls the flow of time rather than the program structure.
  - It does not make sense to assign a variable in more than one place.

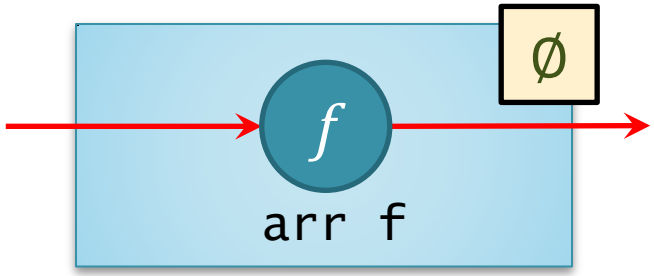


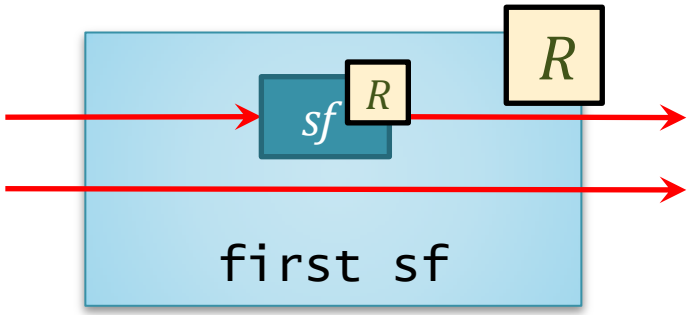
What should the value of x be?
What value should be printed?

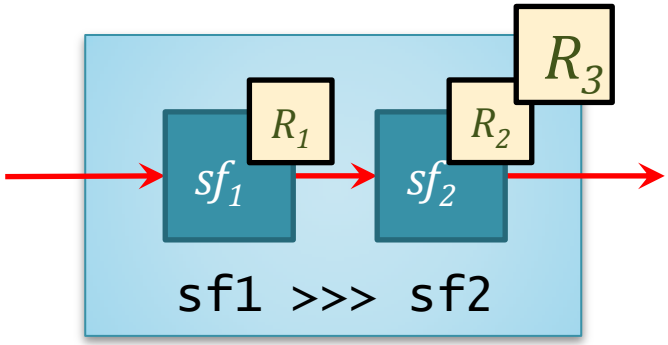
Adding Effects

- To make effects safe, we must limit how we use effectful signal functions.
  - If an effect is used, it can only be used in one place.
- We achieve this by tagging signal functions at the type level with *resource types* and restricting their composition.

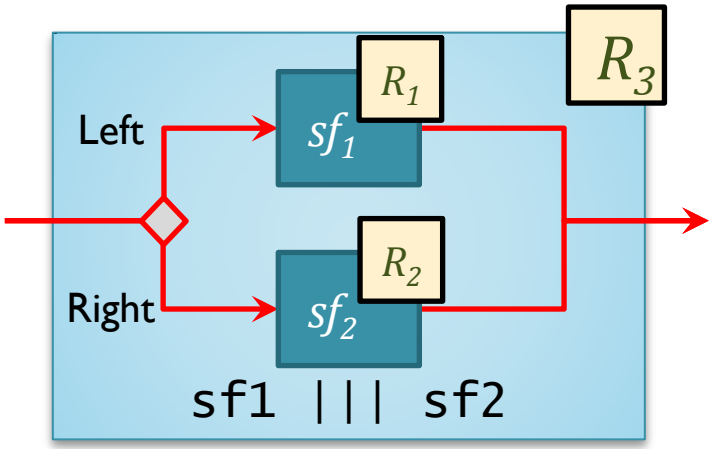
Resource Typed Arrow Operators







$R_1 \cup R_2 = R_3$
$R_1 \cap R_2 = \emptyset$



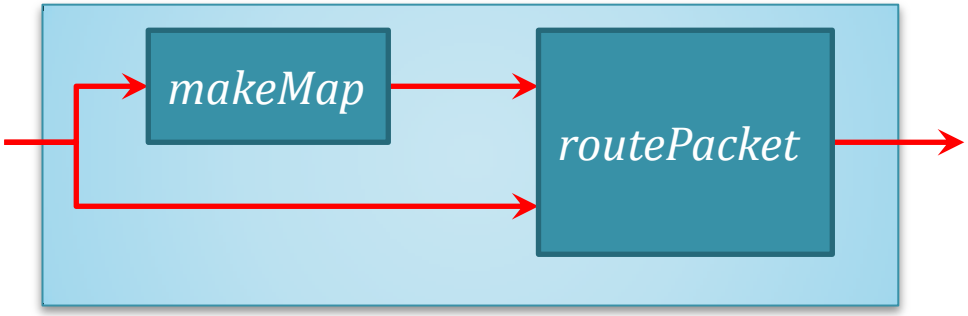
$R_1 \cup R_2 = R_3$

Asynchrony

- In some cases, our synchronous assumption is too strong.
- Perhaps the processing rates of two functions would be better off different.
  - Memory reads running synchronously with hard drive seeks
  - A GUI that should be run at ~60 FPS along with sound generation at 44.1 KHz
  - Packet routing together with network map updating

Asynchrony

- Packets are used to make new routing maps, which are then used to route the packets
- Making maps is slow, but routing must be fast



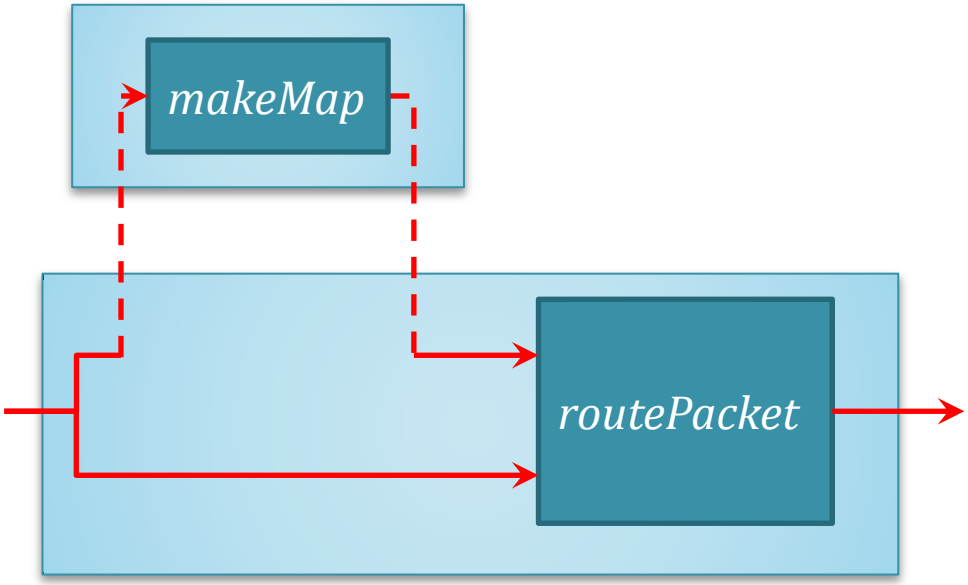
- What if we allow the relaxation that we do not always need the newest map?

Asynchrony

- Let us allow multiple processes, each with its own notion of time.
  - Each will individually retain the fundamental abstraction ("infinitely fast, infinitely often").
  - Each will still respect the others' resources.
  - However, they will no longer synchronize.

Asynchrony

- Now we can make maps asynchronously.

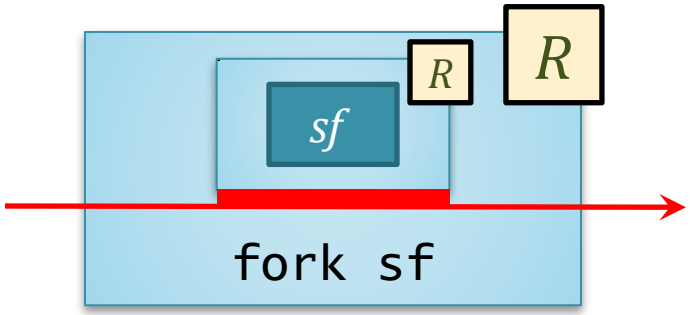


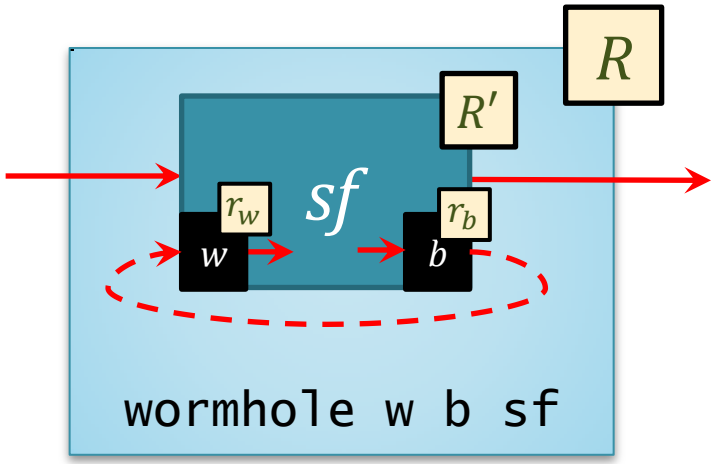
- But what are those dashed lines?

Inter-process Communication

- We need a way to communicate data from one time stream to another.
- Data needs to get time dilated – either stretched or compressed
- Wormholes!
  - Wormholes have a blackhole for writing to and a whitehole for reading from.
  - Wormhole access is made safe with resource types.
  - Wormholes automatically dilate their data.

New Operators

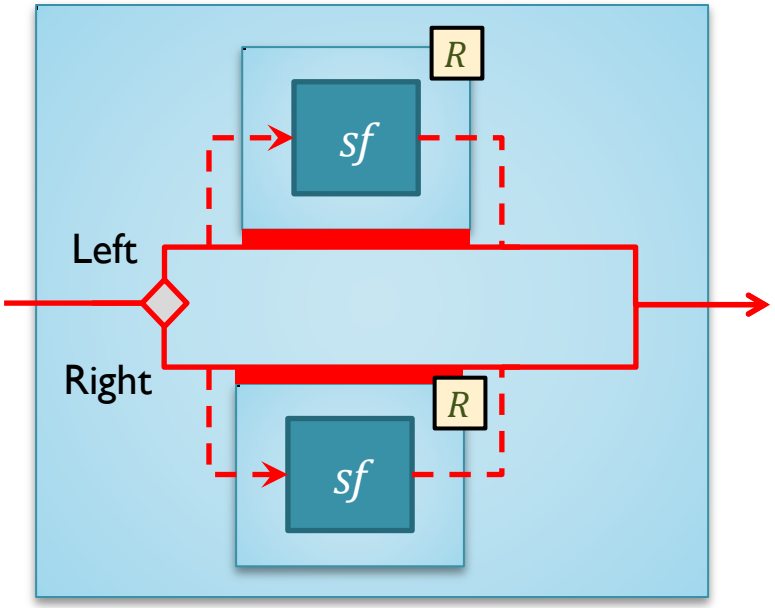




$$R' = R \cup \{r_b, r_w\}$$

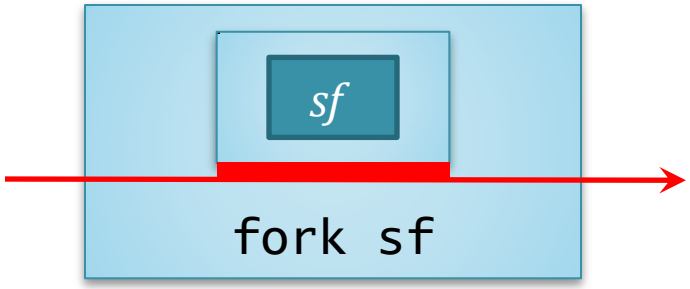
Maintaining Effect Safety

- Are effects still safe in the presence of asynchrony?



Asynchronous Choice

- Remember that the data controls time.
  - When a signal function has no incoming data, it must freeze.
  - Likewise, if a fork has no incoming data, it freezes its forked process.



Asynchronous Choice

- Remember that the data controls time.
  - When a signal function has no incoming data, it must freeze.
  - Likewise, if a fork has no incoming data, it freezes its forked process.
- We achieve this while guaranteeing safety with our fundamental abstraction of FRP
  - Treat every moment in time as a transaction.
  - Freezing only occurs between transactions.

Parallelizing Signal Functions

- Forking and wormholes allow us to create asynchronous, concurrent behavior, but what about parallel behavior?
  - For instance, we may fork multiple processes but then want to wait for their results before continuing.
  - "Waiting" is nonsensical in FRP
- We can achieve the same idea with event streams.

Thank you!

- There is a prototype of this work available at: github.com/dwincort/CFRP

- I would be happy to take questions