Synchronous Reactive Systems and the SR Domain

Stephen Edwards

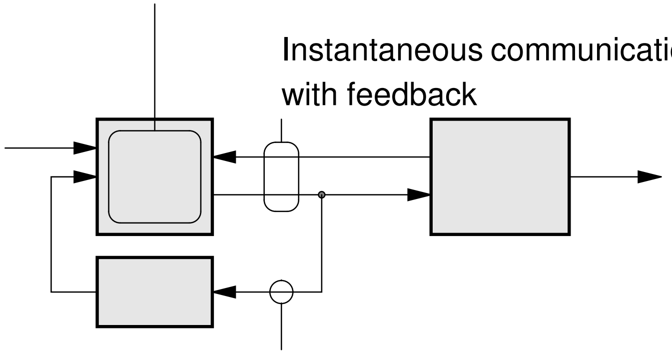
http://www.eecs.berkeley.edu/~sedwards/

University of California, Berkeley

SR Systems

Zero-delay blocks compute continuous functions

Instantaneous communication with feedback



Single driver, multiple receiver wires with values from flat CPOs

- Block functions may change between instants for time-varying behavior
- Block functions may be specified in any language

Reactive Embedded Systems

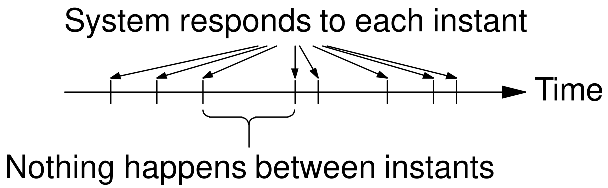
- Run at the speed of their environment
- *When* as important as *what*
- Concurrency for controlling the real world
- Determinism desired
- Limited resources (e.g., memory)
- Discrete-valued, time-varying
- Examples:
  - Systems with user interfaces
    * Digital Watches
    * CD Players
  - Real-time controllers
    * Anti-lock braking systems
    * Industrial process controllers

The SR Domain

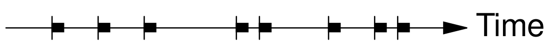
- A new model of computation in Ptolemy
  - Good for reactive systems
  - Good for describing control
  - Synchronous model of time
  - Supports heterogeneity: opaque blocks
  - Unbuffered multiple-receiver communication channels
- Deterministic
  - Guaranteed by fixed-point semantics
- Fast, predictable execution time
  - Chaotic iteration-based execution
  - Fully static scheduling

The Synchronous Model of Time

- Synchronous: time is an ordered sequence of instants
- Reactive: Instants initiated by environmental events

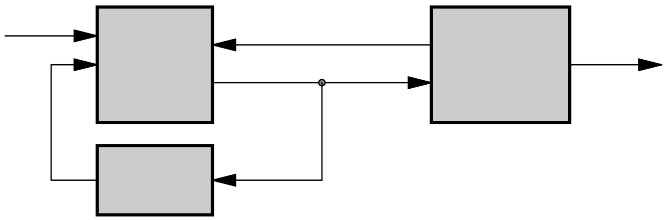


- A system only needs to be “fast enough” to simulate synchronous behavior



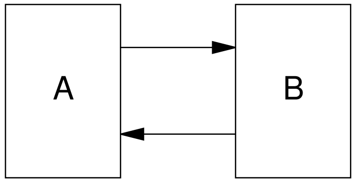
SR Systems

- Reactive systems need concurrency
- The synchronous model makes for deterministic concurrency
  - No “interleaving” semantics
  - Events are totally-ordered
  - “Before,” “after,” “at the same time” all well-defined and controllable
- Embedded systems need boundedness; dynamic process creation a problem
- SR system: fixed set of synchronized, communicating processes

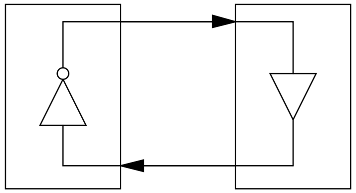


Zero Delay and Feedback

How to maintain determinism?



Which goes first?
Need an order-invariant semantics



Contradictory!
Need to attach meaning to such systems.

Fixed-point Semantics are Natural for Synchronous Specifications with Feedback

Why a fixed point?

Self-reference:
The essence of a cycle

$$f(x_t) = x_t$$

System function (composition of block functions)

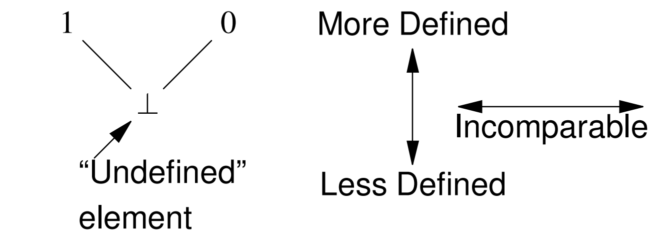
Vector of signals at time t (zero delay)

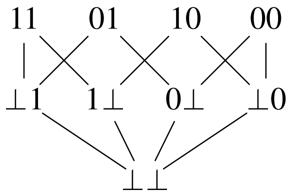
fixed point $\Longleftrightarrow$ stable state

determinism $\Longleftrightarrow$ unique solution

Vector of Signals is a CPO

Values along an upward path grow more defined.



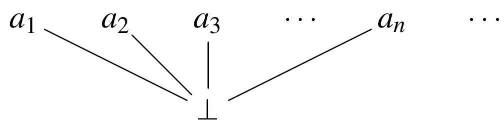


vector-valued extension

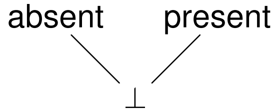
Formally, $x \sqsubseteq y$ if y is at least as defined as x.

Adding $\perp$ Is Enough

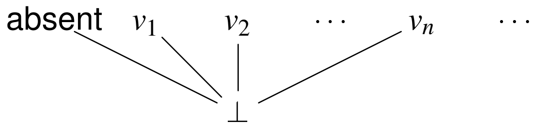
Any set $\{a_1, a_2, \ldots, a_n, \ldots\}$ can easily be "lifted" to give a flat partial order:



A CPO for signals with pure events:



A CPO for valued events:



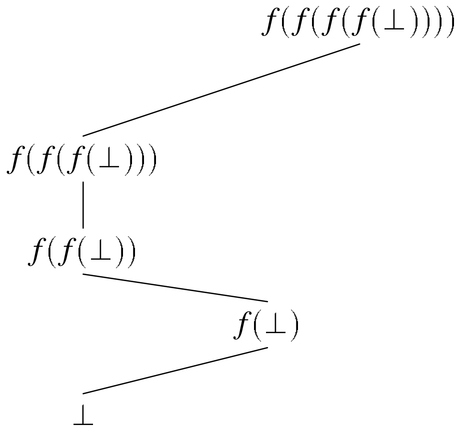
Why not absent $\sqsubseteq$ present?

```
present A then ... else ... end
```

Violates monotonicity

Monotonic Block Functions

Giving a more defined input to a monotonic function always gives a more defined output.



Formally, $x \sqsubseteq y$ implies $f(x) \sqsubseteq f(y)$.

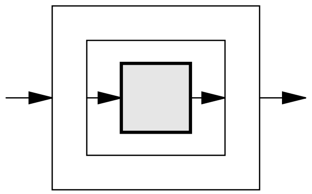
A monotonic function never recants ("changes its mind").

Many Languages Use Strict Functions, Which Are Monotonic

A strict function:

$$g(\underbrace{\ldots,\perp,\ldots}_{\text{inputs}}) = (\underbrace{\perp,\ldots,\perp}_{\text{outputs}})$$

Outside: A strict monotonic function



Inside: Simple "function call" semantics

Most common imperative languages only compute strict functions.

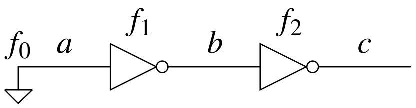
Danger: *Cycles of strict functions deadlock—fixed point is all $\perp$—need some non-strict functions.*

A Simple Way to Find the Least Fixed Point

$$\perp \sqsubseteq f(\perp) \sqsubseteq f(f(\perp)) \sqsubseteq \cdots \sqsubseteq \text{LFP} = \text{LFP} = \cdots$$

For each instant,

1. Start with all signals at $\perp$
2. Evaluate all blocks (in some order)
3. If any change their outputs, repeat Step 2



$$\begin{aligned}
(a,b,c) &= (\perp,\perp,\perp) \\
f_0(\perp,\perp,\perp) &= (0,\perp,\perp) \\
f_1(0,\perp,\perp) &= (0,1,\perp) \\
f_2(0,1,\perp) &= (0,1,0) \\
f_2(f_1(f_0(0,1,0))) &= (0,1,0)
\end{aligned}$$

Asymptotic Schedule Cost