

Synchronous Reactive Systems

Stephen Edwards

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

University of California, Berkeley

Outline

- Synchronous Reactive Systems
- Heterogeneity and Ptolemy
- Semantics of the SR Domain
- Scheduling the SR Domain

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 Digital Approach

Why do we build digital systems?

- Voltage noise is unavoidable
- Discretization plus non-linearity can filter out low-level noise completely
- Complex systems becomes predictable and controllable
- Incredibly successful engineering practice

The Synchronous Approach

Idea: Use the same trick to filter out “time noise.”

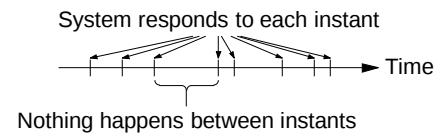
- Noise: Uncontrollable and unpredictable delays
- Discretization $\Leftrightarrow$ global synchronization
- The synchrony hypothesis:

Things compute instantaneously

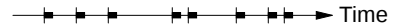
- Already widespread:
 - Synchronous digital systems
 - Finite-state machines

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



Outline

- Synchronous Reactive Systems
- Heterogeneity and Ptolemy
- Semantics of the SR Domain
- Scheduling the SR Domain

Heterogeneity

Why are there so many system description languages?

- Want a succinct description for *my* system.
- “Let the language fit the problem”

Bigger systems have more diverse problems; use a fitting language for each subproblem.

Want a heterogeneous coordination scheme that allows many languages to communicate.

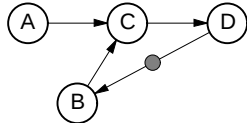
Heterogeneity in Ptolemy

Ptolemy: A system for rapid prototyping of heterogeneous systems

A Ptolemy *domain* (model of computation):

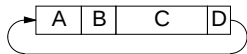
- Set of blocks:

Atomic pieces of computation that can be “fired” (evaluated).



- Scheduler:

Determines block firing order before or during system execution.



Schedulers Support Heterogeneity

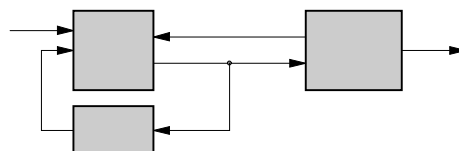
- Scheduler does not know block contents, only how to fire
- Block contents may be anything
- “Wormhole”: A block in one domain that behaves as a system in another
- Hierarchical heterogeneity: Any system may contain subsystems described in different domains

Outline

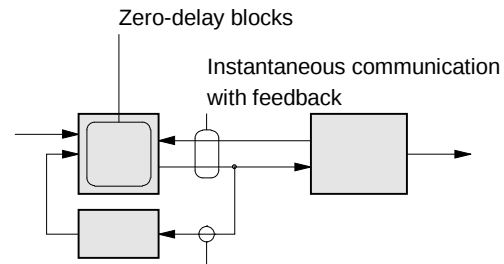
- Synchronous Reactive Systems
- Heterogeneity and Ptolemy
- **Semantics of the SR Domain**
- Scheduling the SR Domain

The SR Domain

- 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



The SR Domain (2)

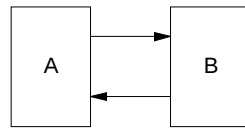


Single driver, multiple receiver channels

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

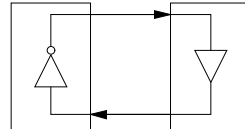
Zero Delay and Feedback

How to maintain determinism?



Which goes first?

Need an order-invariant semantics



Contradictory!

Need to attach meaning to such systems.

Dealing with Feedback

Why bother at all?

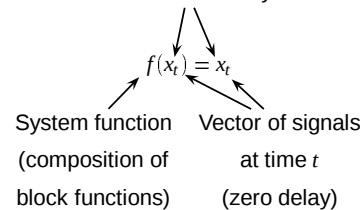
Answer: *Heterogeneity*

- Cycles are usually broken by delay elements *at the lowest level*
- Some schemes insist on this
- False feedback often appears at higher levels
- Data dependent cycles can appear when sharing resources
- *Virtually all cycles are "false," yet must be dealt with.*

Fixed-point Semantics are Natural for Synchronous Specifications with Feedback

Why a fixed point?

Self-reference:
The essence of a cycle



fixed point $\iff$ stable state

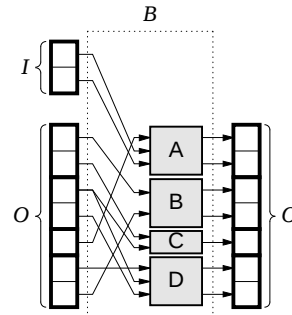
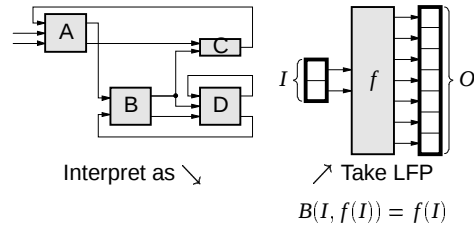
determinism $\iff$ unique solution

Unique Least Fixed Point Theorem

A monotonic function on a complete partial order (with $\perp$) has a unique least fixed point.

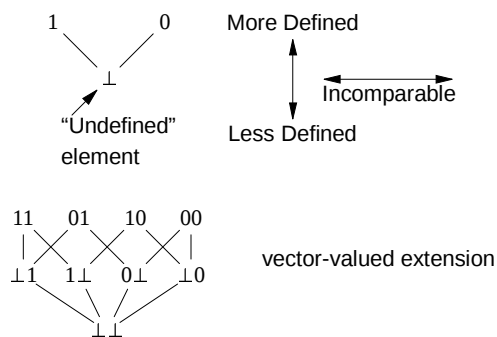
What does it mean to make the system function f monotonic and the signal values a CPO?

The Least Fixed Point of What?



Vector of Signals is a CPO

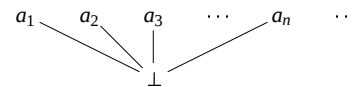
Values along an upward path grow more defined.



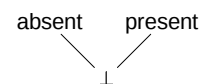
Formally, $x \sqsubseteq y$ if y is at least as defined as x .

Adding $\perp$ Is Enough

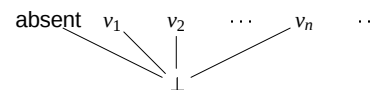
Any set $\{a_1, a_2, \dots, a_n, \dots\}$ can easily be "lifted" to give a flat partial order:



A CPO for signals with pure events:



A CPO for valued events:



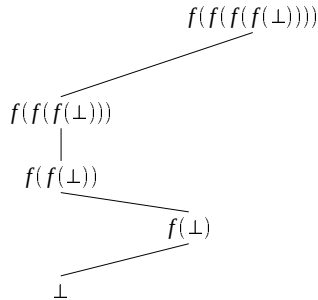
Why not $\text{absent} \sqsubseteq \text{present}$?

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").

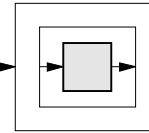
Many Languages Use Strict Functions, Which Are Monotonic

A strict function:

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

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.*

Outline

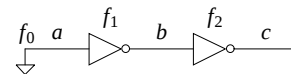
- Synchronous Reactive Systems
- Heterogeneity and Ptolemy
- Semantics of the SR Domain
- Scheduling the SR Domain

A Simple Way to Find the Least Fixed Point

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

For each instant,

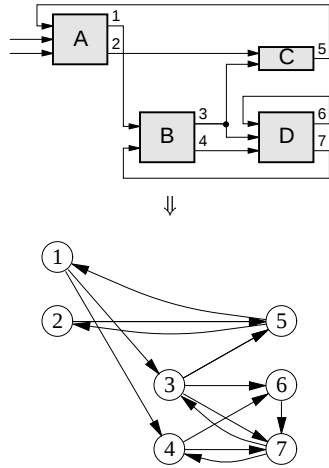
1. Start with all signals at $\perp$
2. Evaluate all blocks (in some order)
3. If any change their outputs, repeat Step 2



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

The Dependency Graph

Transform into single-output functions:

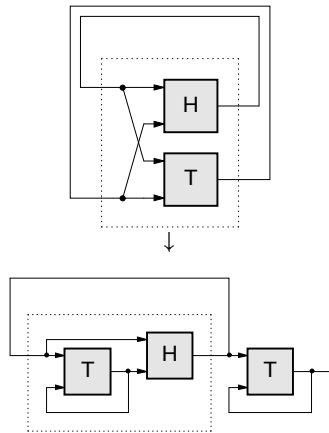


The Scheduling Algorithm

1. Decompose into strongly-connected components
2. Remove a head (set of vertices) from each SCC, leaving a tail
3. Recurse on each tail

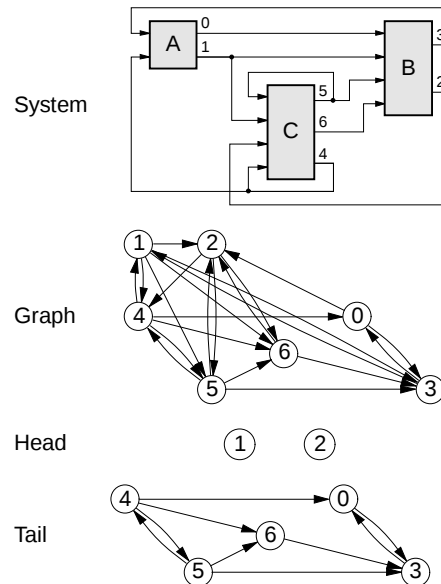
Evaluating SCCs

Split a strongly-connected graph into a head and tail:

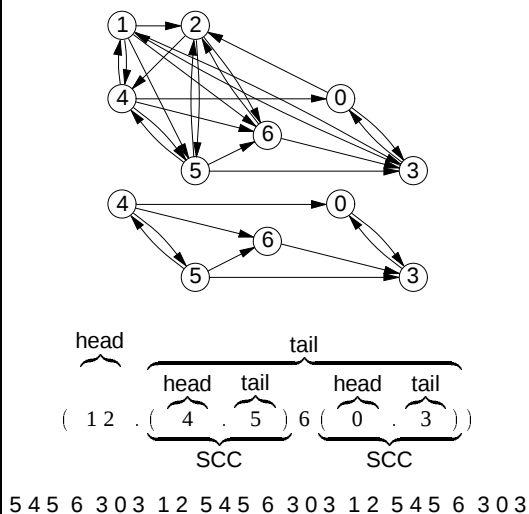


Good heads break T's strong connectivity.

Example

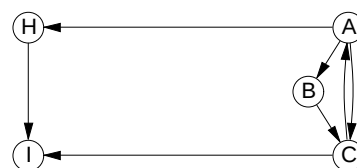
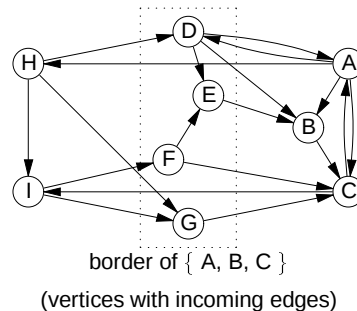


Schedules



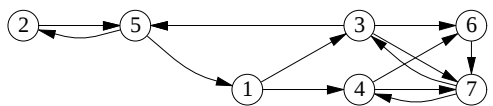
Finding Good Heads

Must break strong connectivity—remove a border of a set of vertices:



Choosing Good Border Sets

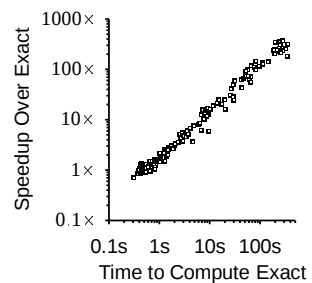
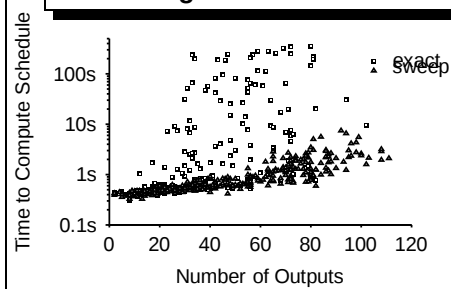
Heuristic: "Grow" a set starting from a vertex and greedily include the best border vertex:

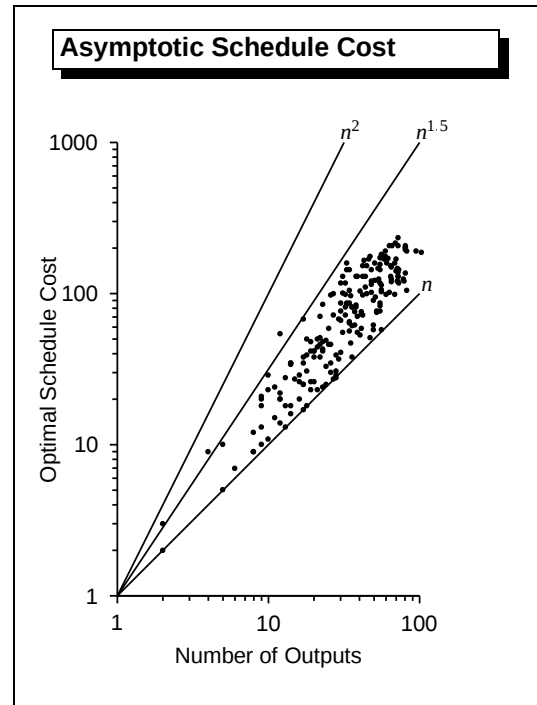
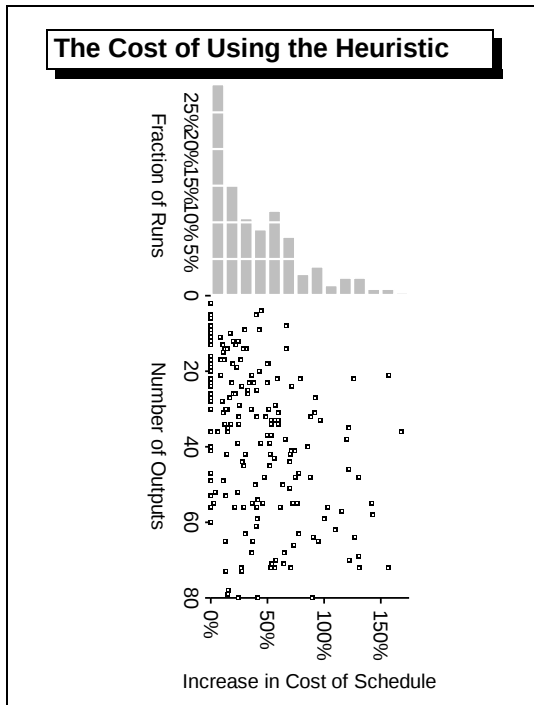


Set	Border
1	5
1 5	2 3
1 5 2	3
1 5 2 3	7
1 5 2 3 7	4 6
1 5 2 3 7 4	6

2 is better (3 would increase border)

Scheduling Results





Conclusions

- Reactive embedded systems
 - Run at the speed of their environment
 - *When* as important as *what*
 - Concurrent, deterministic, bounded, discrete-valued
- The synchronous approach
 - Discrete instants, globally synchronized
 - Assumes instantaneous computation
- Heterogeneity in Ptolemy
 - Domain: Blocks and Scheduler
 - Hierarchical heterogeneity through domain embedding

Conclusions (2)

- The SR domain
 - Concurrent zero-delay blocks
 - Semantics: the least fixed point of a monotonic function on a CPO
 - Values include “undefined” ($\perp$)
- Scheduling the SR Domain
 - Use single-output dependency graph
 - Decompose into SCCs; remove a head from each; recurse
 - Head is the border of the tail
 - Choose a head by greedily growing a set of vertices
 - Fast, efficient. $O(n^{1.25})$ execution