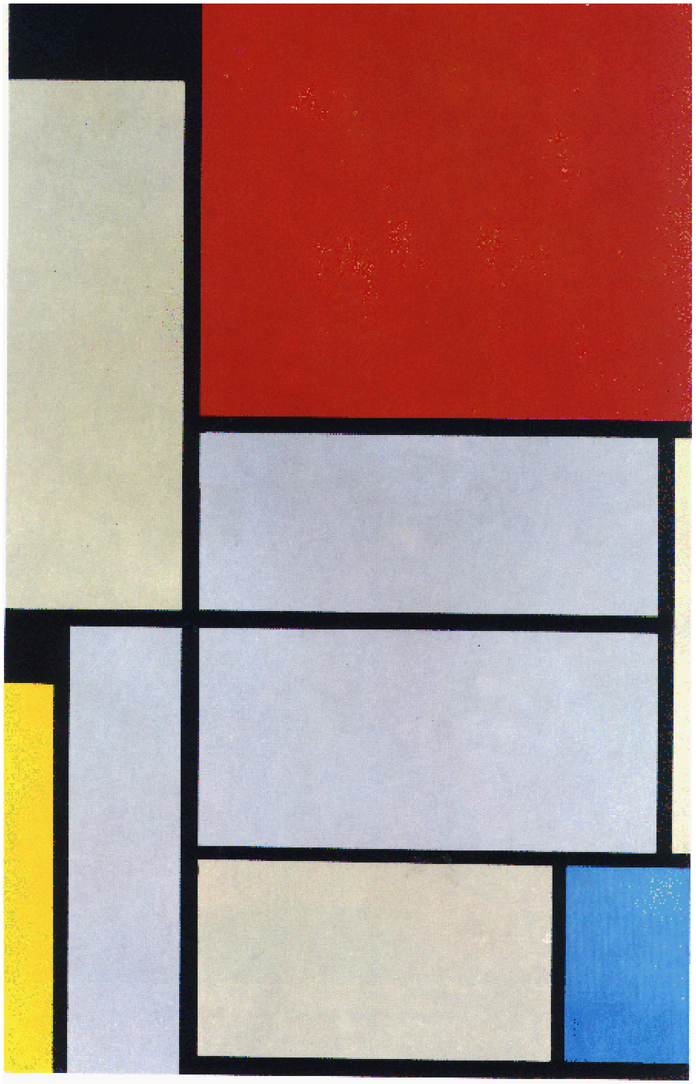
The LSV Tagged Signal Model

Edward A. Lee
Alberto Sangiovanni-Vincentelli

UC Berkeley
Dept. of EECS

Abstraction



Piet Mondrian, *Tableau I*, 1921

Abstraction in system-level design is the act of pulling away or withdrawing from the physical properties of the implementation...

... getting closer to the problem domain and, at design capture,

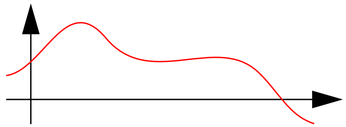
... avoiding overspecification.

Less Abstract, Closer to the Physical

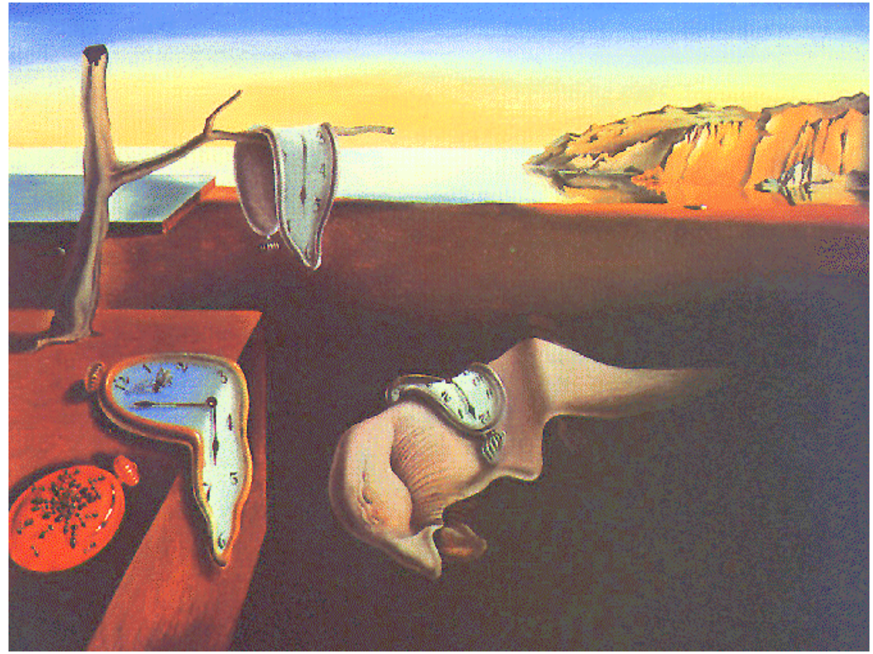
Piet Mondrian, ***The Grey Tree***, **1912**

Still Less Abstract

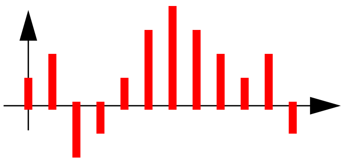
Piet Mondrian, ***The Red Tree***, **1908**



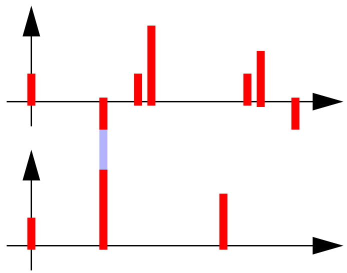
continuous time



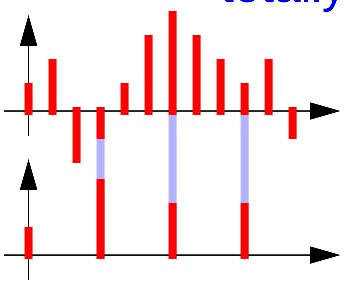
Salvador Dali, ***The Persistence of Memory***, 1931



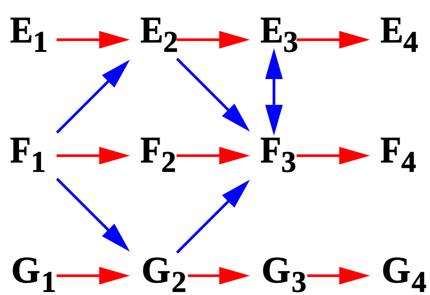
discrete time



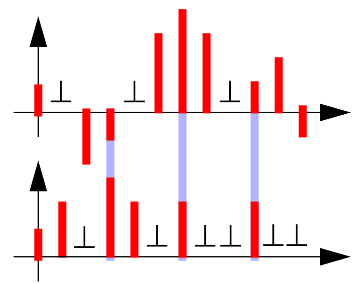
totally-ordered discrete events



multirate discrete time



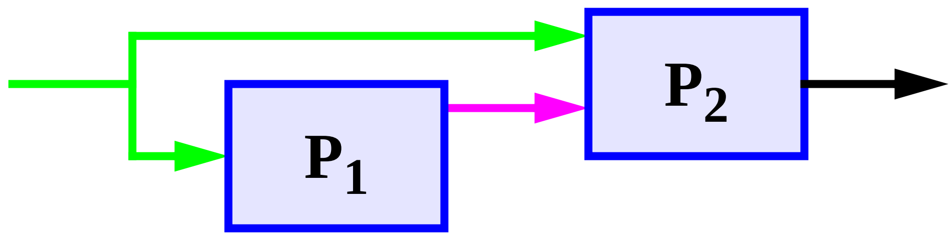
partially-ordered discrete events



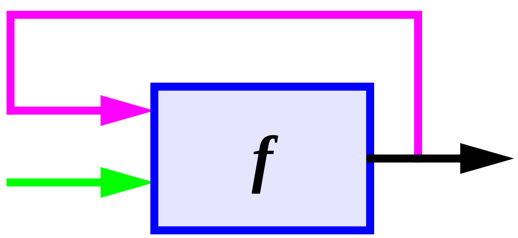
synchronous/reactive

Totally-Ordered Discrete-Event Models

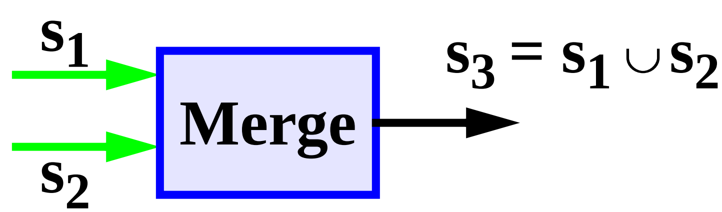
Examples of the sorts of problems that arise from a computerized model of physical time:



What if P_1 is causal but not strictly causal?



What does this mean?



What if s_1 and s_2 have synchronous events?

The Tagged Signal Model

A mathematical framework within which the essential properties of models of computation can be understood and compared.

A denotational framework.

Events and Signals

Abstractions of *time* give us tools to deal with these questions.

- **set of *values*** V
- **set of *tags*** T
- **an event** $e \in T \times V$
- **a *signal* is a set of events**
- **a *functional signal* is a (partial) function** $s: T \to V$
- **the set of all signals** $S = 2^{(T \times V)}$ **(the powerset)**
- N**-tuples of signals** $\mathbf{s} \in S^N$

Possible Interpretations of Tags

- **Universal time ($T = \Re$)**
- **Discrete time (T is a *totally ordered* discrete set)**
- **Precedences (T is a *partially ordered* discrete set)**

Why not always use the "most physical" model:

universal time?

- **In specifying systems, avoid over-specifying.**
- **In modeling systems, recognize the inherent difficulty of maintaining a globally consistent notion of time.**

Empty Signals and Events

- **The empty signal:** λ
- **The tuple of empty signals:** Λ
- **Note:** $\lambda \in S$ **and** $\Lambda \in S^N$**.**
- **For any signal** s**,** $s \cup \lambda = s$**.**
- **For any tuple** $\mathbf{s}$**,** $\mathbf{s} \cup \Lambda = \mathbf{s}$ **(pointwise union).**

In some models of computation, the set V **of values includes a special value** $\perp$ **(pronounced "bottom"), which indicates the absence of a value.**

Processes and Connections

Processes

- **a** ***process*** $P \subseteq S^N$ **for some** N
- **a** ***behavior*** **s** $\in P$ **(s** ***satisfies*** **the process)**
- **a** ***process*** **is a set of possible** ***behaviors***

Connections (a type of process)

- **a** ***connection*** $C \subset S^N$**: s** $= (s_1, \ldots, s_N) \in C \Leftrightarrow s_i = s_j$

Systems

Given a tuple P of processes, a *system* is another process:

$$Q = \left(\bigcap_{P_i \in \mathbf{P}} P_i \right)$$

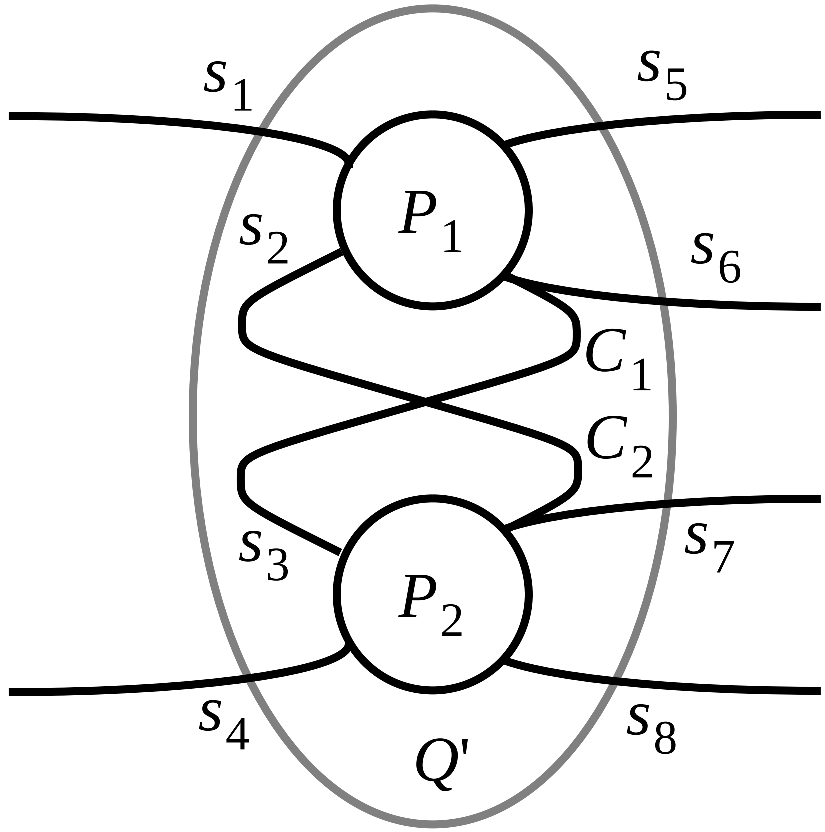
Given a process $P \subseteq S^N$, the *projection* $\Pi_j(P) \subseteq S^{N-1}$ is defined by

$$(s_1, \ldots, s_{j-1}, s_{j+1}, \ldots, s_N) \in \Pi_j(P)$$

if there exists $s_j \in S$

such that $(s_1, \ldots, s_{j-1}, s_j, s_{j+1}, \ldots, s_N) \in P$

An Example



$$P_1, P_2, C_1, C_2 \subseteq S^8, Q = P_1 \cap P_2 \cap C_1 \cap C_2$$

$$Q' = \Pi_2(\Pi_3(Q)) \subseteq S^6$$

Determinacy

- **An *input* to a process is an externally imposed constraint $I \subseteq S^N$ such that $I \cap P$ is the total set of acceptable behaviors.**

- **The *set of all possible inputs* $B \subseteq 2^{S^N}$ is a further characterization of a process.**

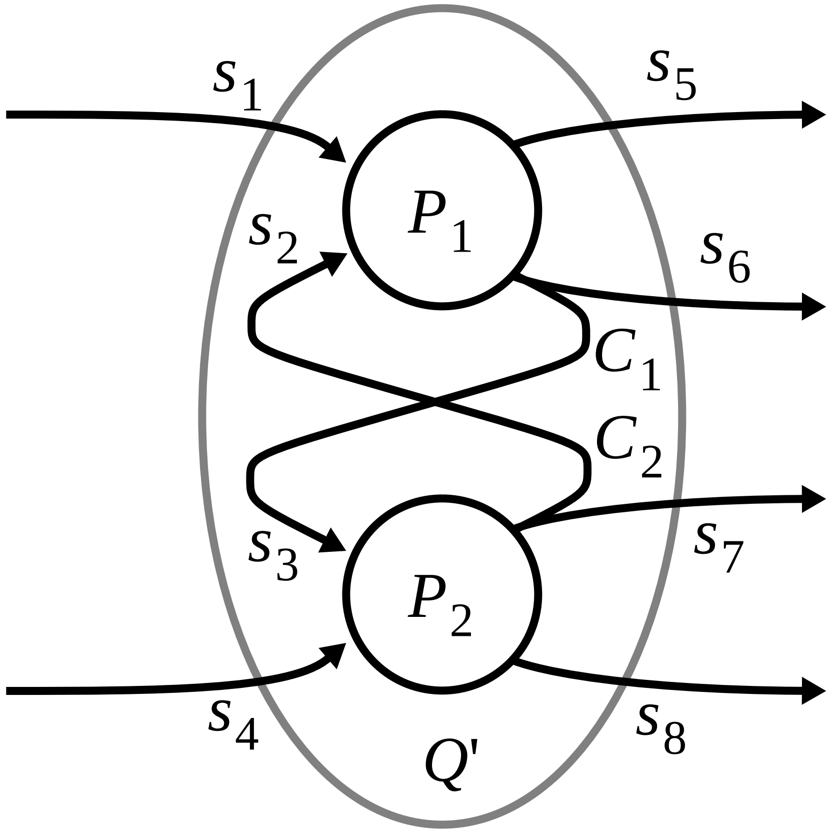
- **For example, $B = \{I ; I \subseteq S^N, \pi_1(I) = s, s \in S\}$ means that the first signal is specified externally and can take any value in the set of signals.**

- **A process is *determinate* if for all inputs $I \in B$, $|I \cap P| = 1$ or $|I \cap P| = 0$.**

Input/Output Partitions

- (S^m, S^n) **is a *partition* of** S^N **if** $N = m + n$**.**
- **A *process*** P **with** m ***inputs*, **n ***outputs*** **is a subset of** $S^m \times S^n$
- **A process with inputs and output is a *relation* between them**
- **A *functional process* is a single-valued mapping (a possibly partial function)** $P: S^m \rightarrow S^n$**.**
- **A process that is functional with respect to some partition is *determinate* for** $B = \{\{(\boldsymbol{p}, \boldsymbol{q}); \boldsymbol{q} \in S^n\}; \boldsymbol{p} \in S^m\}$**.**

Example



Suppose:

P_1 **is functional with respect to the partition**

$((s_1, s_2, s_3, s_4, s_7, s_8), (s_5, s_6))$

P_2 **is functional w.r.t.**

$((s_1, s_2, s_3, s_4, s_5, s_6), (s_7, s_8))$

Key question: is Q' **functional w.r.t.** $((s_1, s_4), (s_5, s_6, s_7, s_8))$**?**

Answer: It depends.

Partial Ordering of Tags and Events

- ***Partially ordered***: there exists an reflexive, antisymmetric, transitive relation "≤" between tags.
- **Version of this relation: "<".**
- **Ordering of the tags ⇒ ordering of events. Given two events** $e_1 = (t_1, v_1)$ **and** $e_2 = (t_2, v_2)$, $e_1 < e_2 \Leftrightarrow t_1 < t_2$.

Timed Systems

- ***Timed system***: T **is totally ordered.**
- ***Metric time***: T **is a metric space.**

Continuous Time

Let $T(s) \subseteq T$ denote the set of tags in a signal s in a timed system.

- **A *continuous-time system* is a metric timed system where T is a continuum (a *closed connected set*) and $T(s) = T$ for each signal s in any tuple $\mathbf{s}$ that satisfies the system.**

A connected set is one where there do not exist two disjoint open sets O_1 and O_2 such that $O_1 \cup O_2$ is the entire set.

Discrete Event Systems

Given a system Q, and a tuple of signals $\mathbf{s} \in Q$ that satisfies the system, let $T(\mathbf{s})$ denote the set of tags (time stamps) appearing in any signal in the tuple $\mathbf{s}$.

- **A *two-sided discrete-event system* Q is a timed system where for each $\mathbf{s} \in Q$, there exists an order-preserving bijection from some subset of the integers to $T(\mathbf{s})$.**

Intuitively

Any pair of events in a signal have a finite number of intervening events.

One-Sided Discrete-Event Systems

- **A *one-sided discrete-event system* Q is a timed system where for each $s \in Q$, there exists an order-preserving bijection from some subset of the natural numbers to $T(s)$.**

Intuitively

Every signal has a first event.

Synchrony

- **Two events are *synchronous* if they have the same tag.**
- **Two *signals* are synchronous if all events in one signal are synchronous with an event in the other signal and vice versa.**
- **A *system* is synchronous if every signal in the system is synchronous with every other signal in the system.**
- **A *discrete-time system* is a synchronous discrete-event system.**

By this definition, synchronous dataflow (SDF) is not synchronous. The “synchronous languages” (Argos, Lustre, Esterel) are synchronous only if $\perp \in V$.

Causality in DE Systems (Intuitively)

- **A *causal* process has a non-negative (but possibly zero) time delay from inputs to outputs.**
- **A *strictly causal* process has a positive time delay from inputs to outputs.**
- **A *delta causal* process has a time delay from inputs to outputs of at least Δ for some constant $\Delta > 0$.**

A Metric Space for DE Signals

In a one-sided DE system, where WOLG $T \subseteq [0, \infty)$, define the *Cantor metric* to be

$$d(\mathbf{s}_1, \mathbf{s}_2) = \frac{1}{2^t}$$

where t is the smallest time where the two signals differ, or if $\mathbf{s}_1 = \mathbf{s}_2$, then $d(\mathbf{s}_1, \mathbf{s}_2) = 0$.

With this metric, behaviors of a discrete-event system become points in a metric space!

Causality in the Cantor Metric Space

Causality: $d(F(\mathbf{s}), F(\mathbf{s}')) \le d(\mathbf{s}, \mathbf{s}')$.

Strict causality: $d(F(\mathbf{s}), F(\mathbf{s}')) < d(\mathbf{s}, \mathbf{s}')$.

Delta causality**: there exists a $k < 1$ such that**

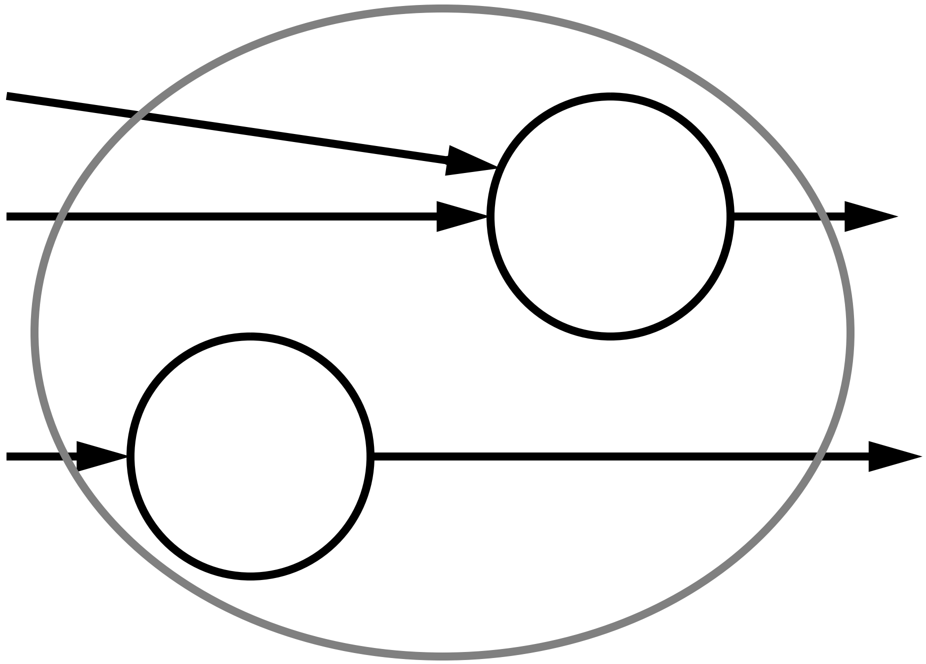
$$d(F(\mathbf{s}), F(\mathbf{s}')) \le k d(\mathbf{s}, \mathbf{s}')$$

F **is a *contraction mapping*.**

Note: $k = \frac{1}{2^{\Delta}}$.

Composing Functional Processes (1)

Parallel composition:

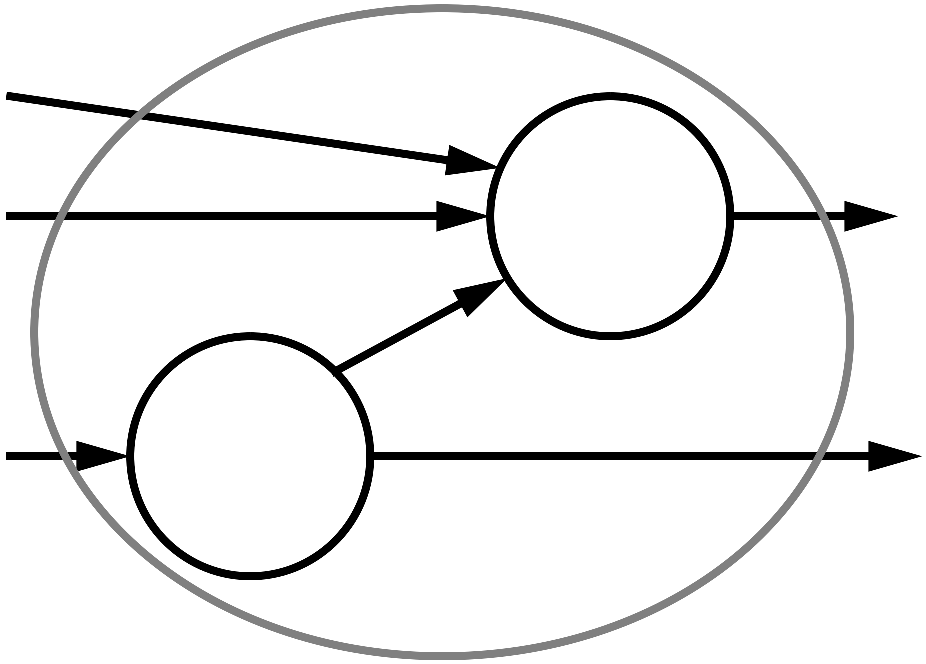


The composition is functional and (strictly, delta) causal if the components are functional and (strictly, delta) causal.

Determinacy is preserved.

Composing Functional Processes (2)

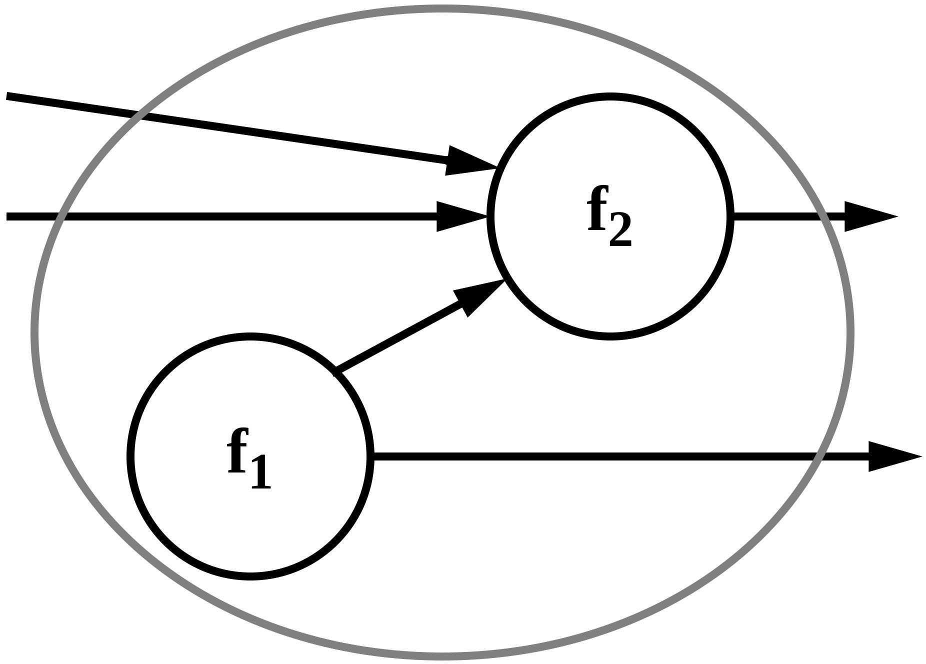
Cascade composition:



The composition is functional and (strictly, delta) causal if the components are functional and (strictly, delta) causal.

Determinacy is preserved.

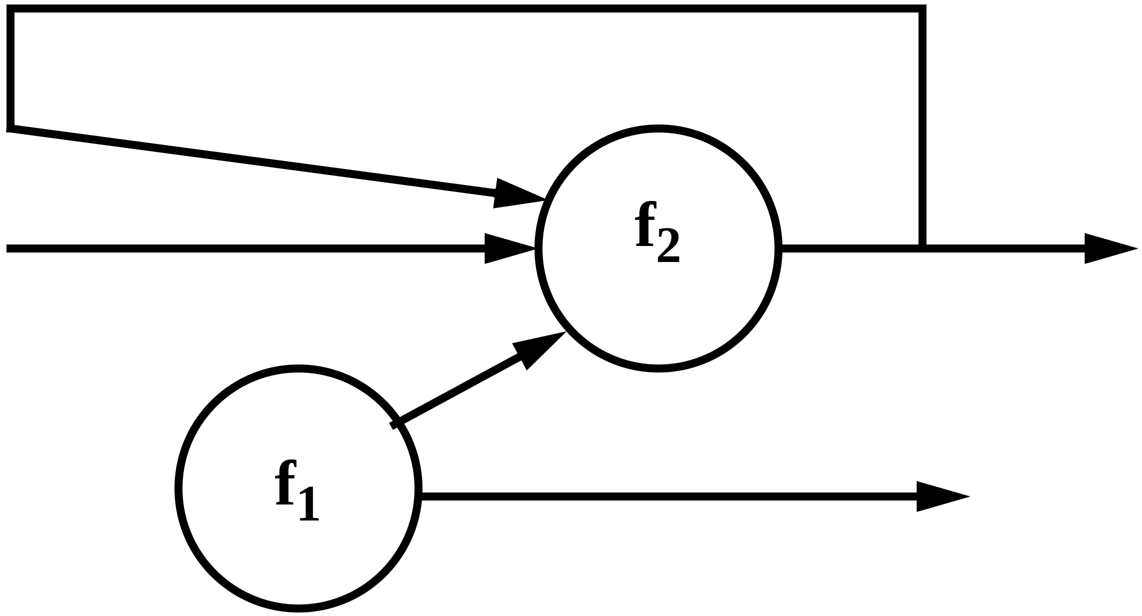
Source composition:



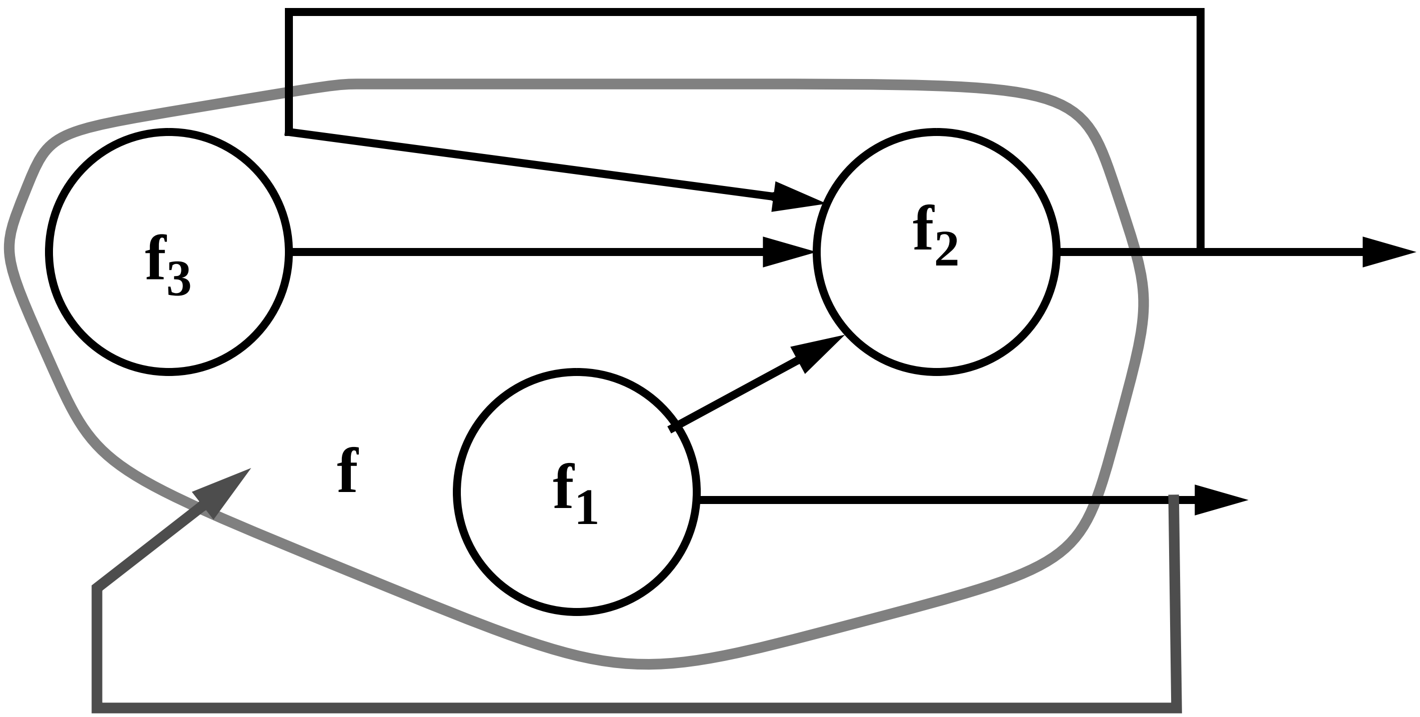
The composition is functional and (strictly, delta) causal if f_2 is functional and (strictly, delta) causal and $f_1 \subset S^2$ is determinate.

Composing Functional Processes (3)

Feedback composition:

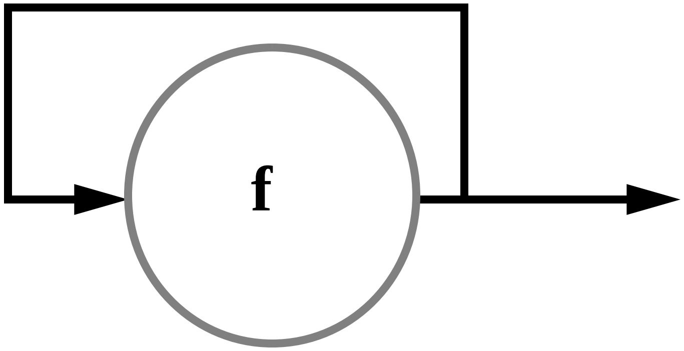


Device: Capture Inputs with Source Processes



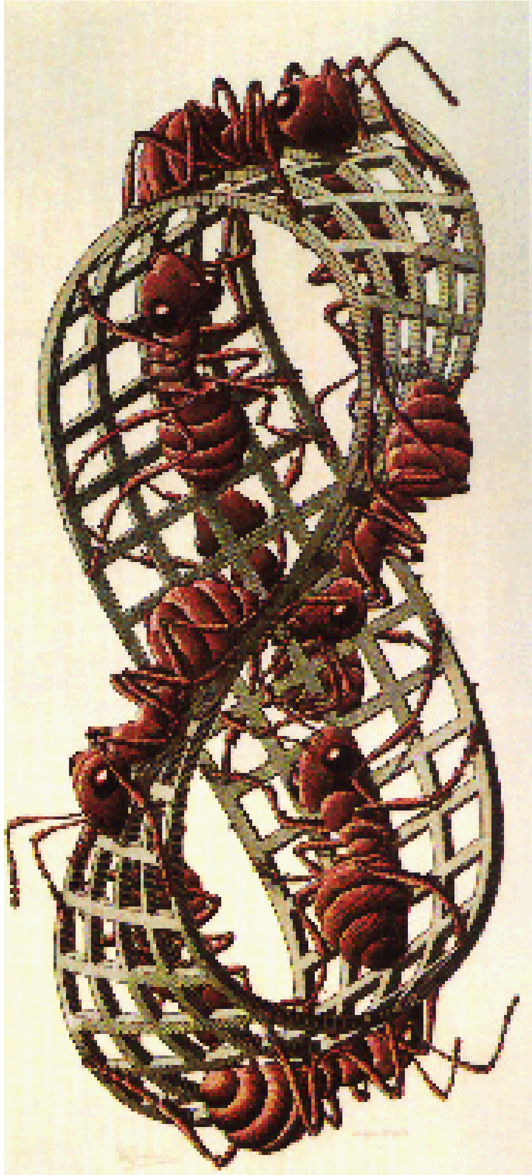
If f_1 and f_3 are determinate, and f_2 is causal, strictly causal, or delta causal, then $f : S^2 \to S^2$ will be causal, strictly causal, or delta causal, respectively.

The Semantics of Feedback



For $f : S \to S$**, define the** ***seman-tics*** **to be a** ***fixed point*** **of** f

i.e. s **such that** $f(s) = s$**.**



M. C. Escher, *Moebius Strip* II, 1963

Fixed Point Theorems Applied to Discrete-Event Systems

- **If f is strictly causal, then it has at most one fixed point. Hence the feedback composition is determinate.**
- **(*Banach fixed point theorem*) If the metric space is complete (it is) and f is delta causal, then it has exactly one fixed point, and that fixed point can be found by starting with any signal tuple s_0 and finding the limit of:**

$$s_1 = f(s_0), s_2 = f(s_1), s_1 = f(s_0), \ldots$$

- **If the metric space is compact (it is if V is a finite set and all signals are discrete-event), then f only needs to be strictly causal to apply the Banach fixed point theorem.**

Lessons

- **If subsystems are delta causal, then the Banach fixed point theorem gives us a *constructive* way to find their *one unique* behavior.**

- **Specification languages often only insist on *strict causality* (VHDL, for example, has a so-called "delta time" model that, despite the similar name, only ensures strict causality).**

- **The set of VHDL signals is not compact.**

- **The lack of a constructive solution manifests itself in practice (VHDL simulators, for example, can get stuck, where time fails to advance).**

Ordered Signal Process Networks

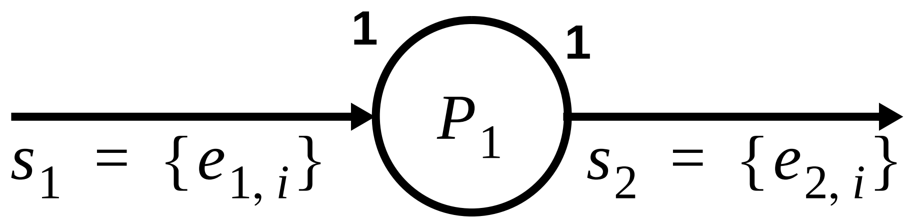
Let $T(s)$ denote the tags in the signal s and $T(\mathbf{s})$ the union of the tags in the signals in the tuple s.

- **In a *two-sided ordered signal process network*, $T(s)$ is order isomorphic with Z, the integers, for each signal s, but the set of all tags $T(\mathbf{s})$ is typically partially ordered.**
- **In a *one-sided ordered signal process network*, $T(s)$ is order isomorphic with N, the natural numbers, for each signal s.**

For any two distinct signals s_1 and s_2, it could be that $T(s_1) \cap T(s_2) = \varnothing$.

- **Events are often called *tokens*, and a signal is a sequence of tokens with no notion of time.**

Example of an OSP



- **Because tokens in each signal are ordered,**

$$i < j \Rightarrow e_{k,i} < e_{k,j} \textbf{ for } k = 1, 2.$$

- **Suppose each token consumed on the input results in exactly one token produced on the output. Then**

$$e_{1,i} < e_{2,i} \textbf{ for all } i.$$

Dataflow

- **A *dataflow process* or *actor* is an OSP with a firing signal.**
- **A *firing signal* is both an input and an output and contains only events that are comparable with all events in other input and output signals.**
- **A *firing* is an event in the firing signal.**

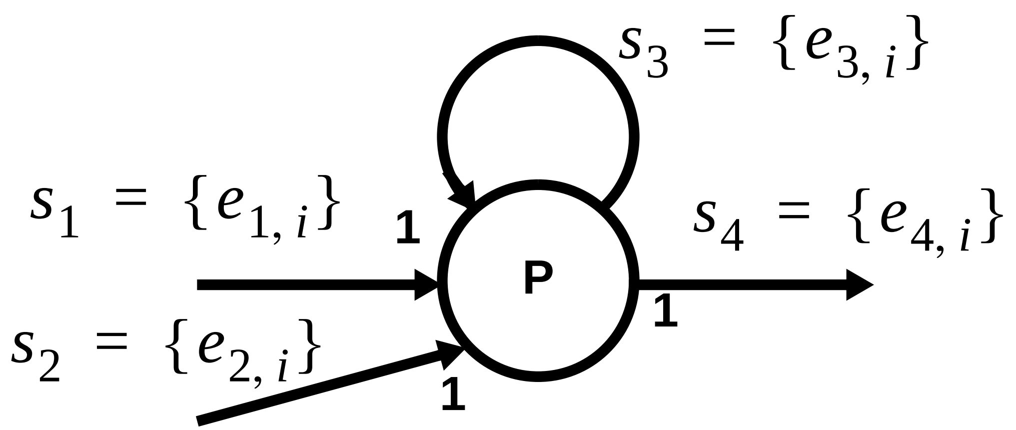
i.e., if e is a firing, and e' is an event in any other input or output signal of the process, then either $e < e'$ or $e' < e$.

Consumption and Production of Tokens

Given two successive firings $e_1 < e_2$:

- **An output event e' where $e_1 < e' < e_2$ is said to be *produced* by firing e_1.**
- **An input event e' where $e_1 < e' < e_2$ is said to be *consumed* by firing e_2.**

Example of a Dataflow Process



- s_3 **is the firing signal.**
- **The process consumes one token from each of** s_1 **and** s_2 **on each firing, and produces one token on** s_4**.**
- **For each** i**,** $e_{1,i} < e_{3,i}$**,** $e_{2,i} < e_{3,i}$**, and** $e_{3,i} < e_{4,i}$**.**

Partially Ordered Signals

- **By *set inclusion*:** $s_1 \subseteq s_2$ **if every event in** s_1 **is also in** s_2**.**

- **By *prefix ordering*:** $s_1 \sqsubseteq s_2 \Leftrightarrow s_1 \subseteq s_2$ **and for all** $e_1 \in s_1$ **and** $e_2 \in s_2 - s_1$, $e_2 > e_1$.

Complete Partial Orders

The set of tuples of ordered signals S^N is a ***complete partial order*** (**cpo**) under the prefix order. I.e.,

- **A *chain* in S^N is a (possibly infinite) sequence** $C = \{\mathbf{s}_1, \mathbf{s}_2, \ldots\}$ **where** $\mathbf{s}_i \sqsubseteq \mathbf{s}_j \Leftrightarrow i \leq j$.
- **Every chain in a cpo has a *least upper bound*, written** $\sqcup C$.

Notation:

- **Given a set C of N-tuples of signals and a function** $F: S^N \to S^M$, $F(C)$ **denotes the set of M-tuples of signals resulting from applying the function to each member of C.**

Kahn Process Networks

- **A *Kahn process* is an OSP that is also a continuous function.**
- **A function is *continuous* if for every chain C**

$$F(\sqcup C) = \sqcup F(C)$$

This is exactly topological continuity in the *Scott topology*.

Another fixed point theorem (based on the *Knaster-Tarski* fixed-point theorem) shows that networks of continuous processes in a cpo have a unique *least-fixed-point*, which we take to be the semantics of feedback loops (we will develop this idea).

Monotonic Functions

Theorem: **A continuous process** F **is** ***monotonic*****, meaning that**

$$\mathbf{s}_3 \sqsubseteq \mathbf{s}_1 \Rightarrow F(\mathbf{s}_3) \sqsubseteq F(\mathbf{s}_1).$$

Proof: **Suppose** $F: S^N \to S^M$ **is continuous and consider two signals** $\mathbf{s}_1$ **and** $\mathbf{s}_2$ **in** S^N **where** $\mathbf{s}_1 \sqsubseteq \mathbf{s}_2$**. Define the increasing chain** $C = \{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_2, \mathbf{s}_2, \ldots\}$**. Then** $\sqcup C = \mathbf{s}_2$**, so from continuity,**

$$F(\mathbf{s}_2) = F(\sqcup C) = \sqcup F(C) = \sqcup \{F(\mathbf{s}_1), F(\mathbf{s}_2)\}.$$

Therefore $F(\mathbf{s}_1) \sqsubseteq F(\mathbf{s}_2)$**, so the process is monotonic.**

Monotonicity does not imply Continuity

Example:

$$F(s) = \begin{cases} [0]; & \text{if } s \text{ is finite} \\ [0, 1]; & \text{otherwise} \end{cases} .$$

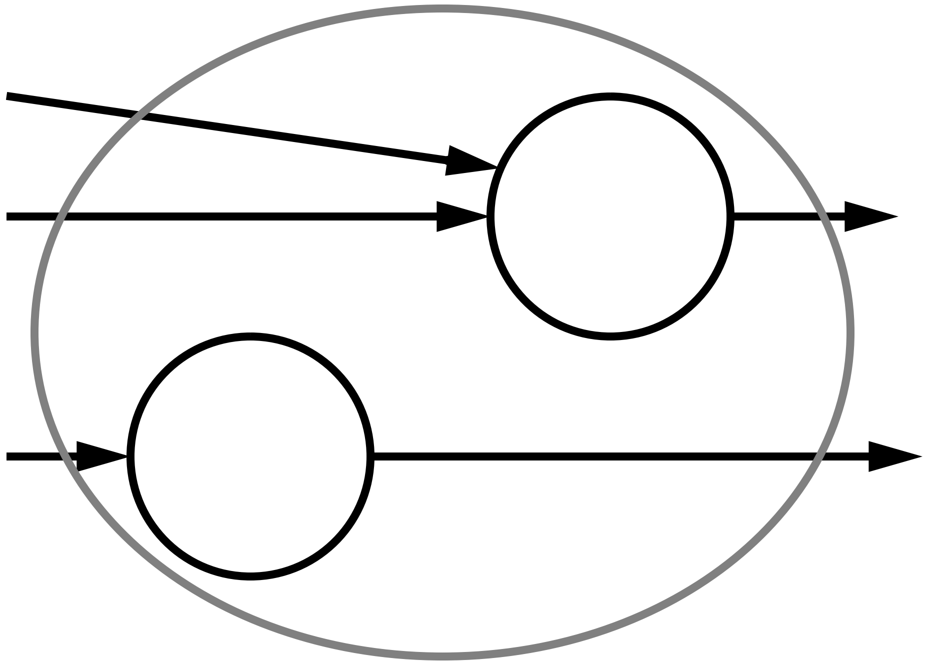
To show that this *is monotone*, note that if s is infinite and $s \sqsubseteq s'$, then $s = s'$, so $F(s) \sqsubseteq F(s')$. If s is finite, then $F(s) =$ $[0]$, which is a prefix of all possible outputs.

To show that it is *not continuous*, consider any chain

$$C = \{ s_0 \sqsubseteq s_1 \sqsubseteq \ldots \},$$

where each s_i has exactly i events in it. Then $\sqcup C$ is infinite, so $F(\sqcup C) = [0, 1] \neq \sqcup F(C) = [0]$.

Parallel composition:

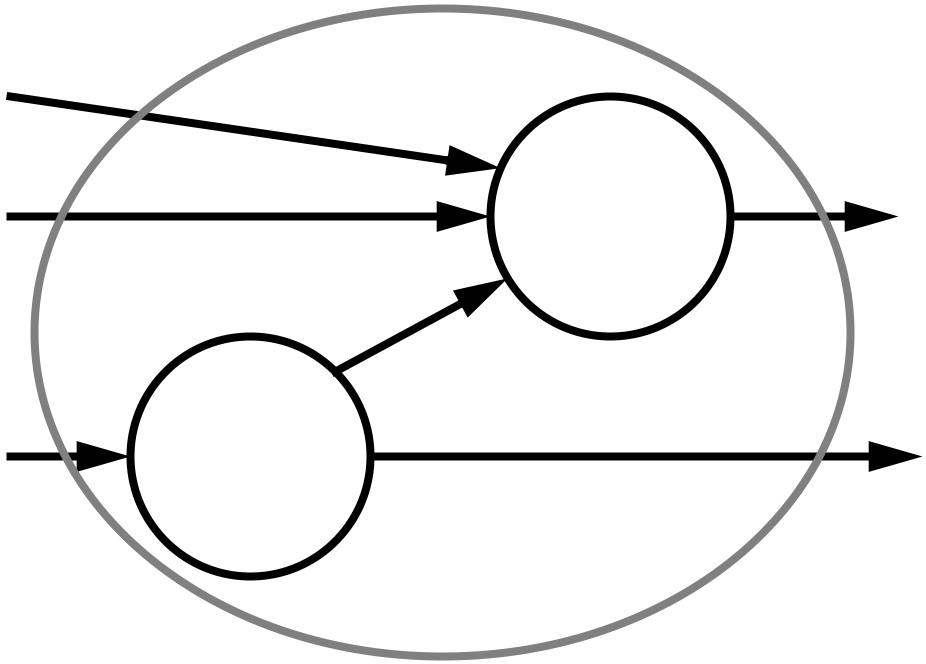


The composition is functional and continuous or monotonic if the components are functional and continuous or monotonic.

Determinacy is preserved.

Composing Continuous Processes (2)

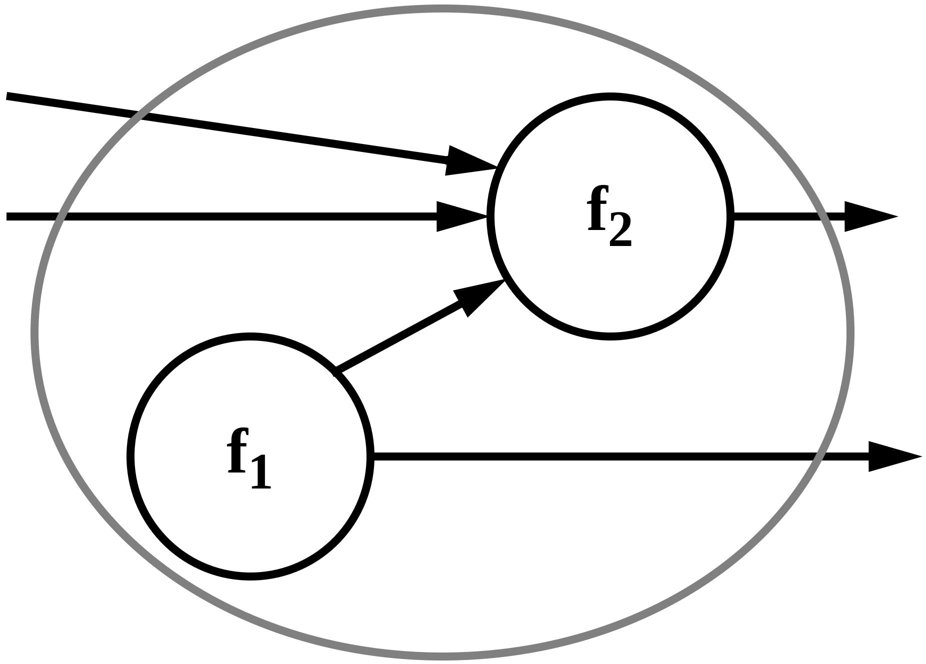
Cascade composition:



The composition is functional and continuous or monotonic if the components are functional and continuous or monotonic.

Determinacy is preserved.

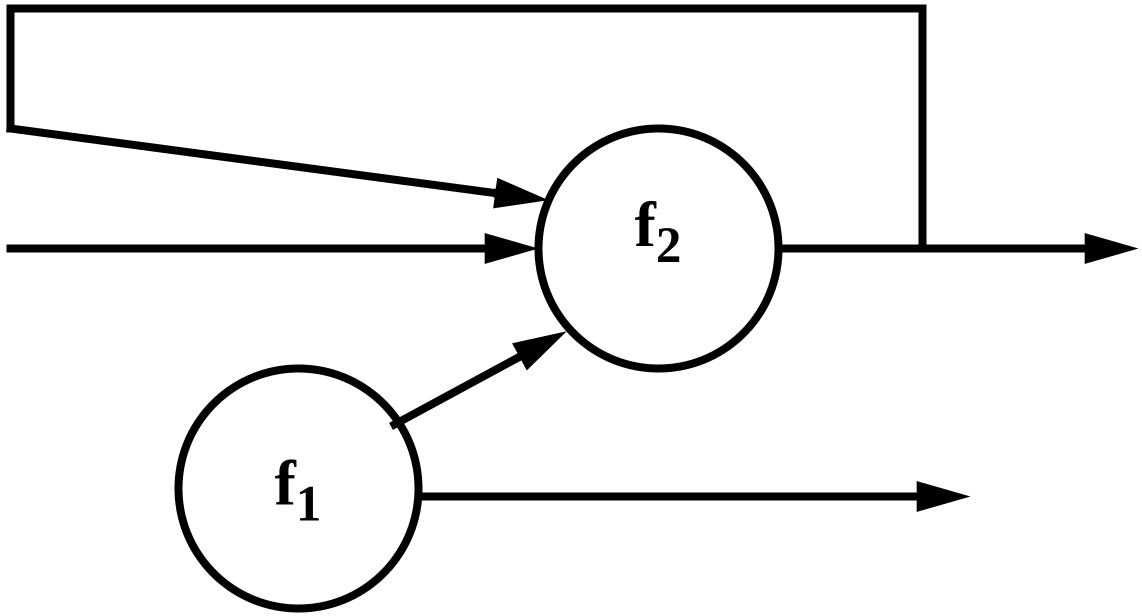
Source composition:



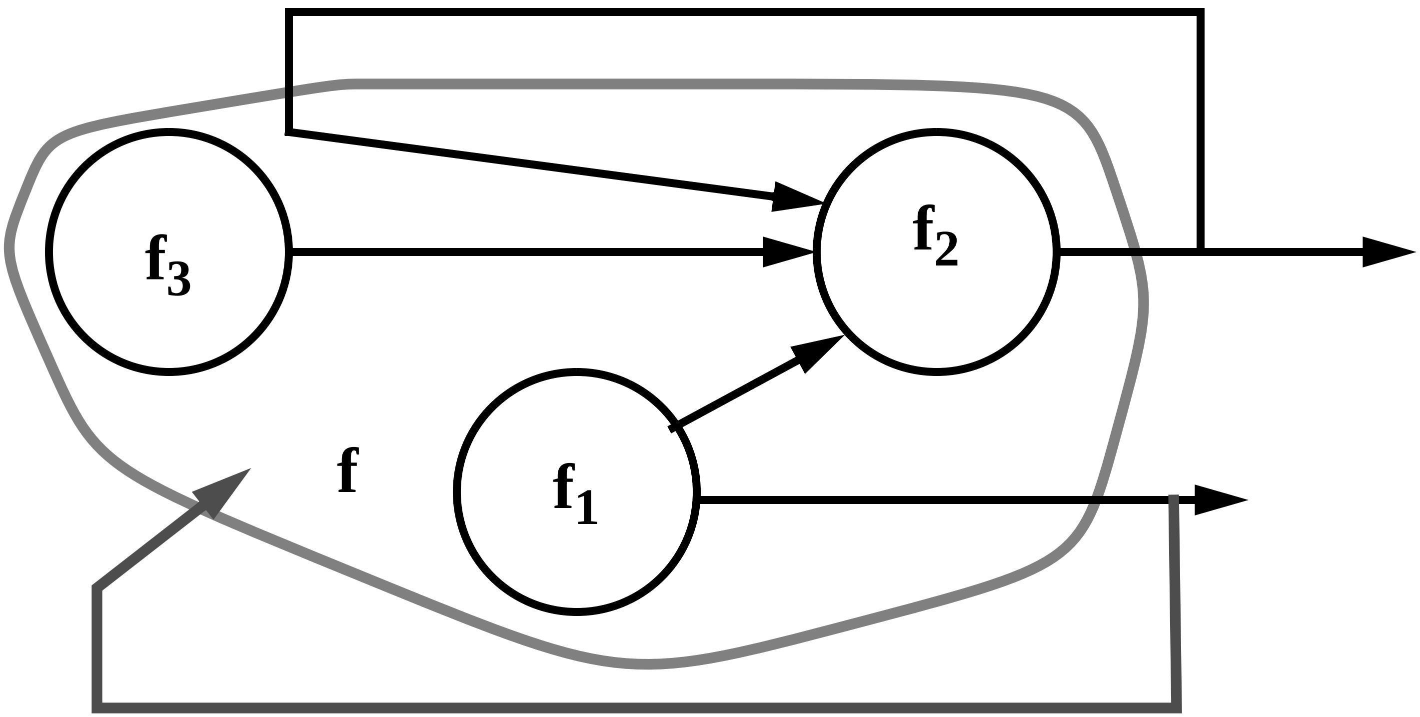
The composition is functional and continuous or monotonic if f_2 is functional and continuous or monotonic and $f_1 \subset S^2$ is determinate.

Composing Continuous Processes (3)

Feedback composition:

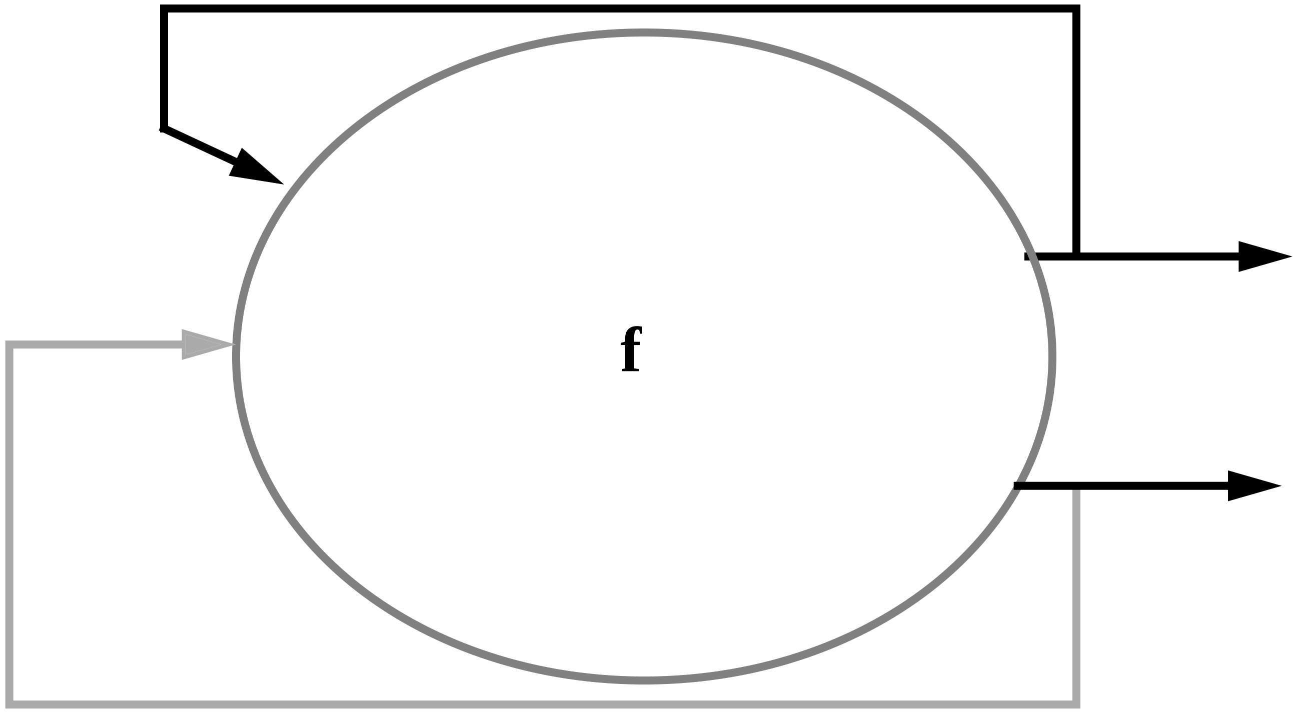


Device: Capture Inputs with Source Processes



If f_1 and f_3 are determinate, and f_2 is continuous or monotonic, then $f : S^2 \to S^2$ will be continuous or monotonic, respectively.

Feedback composition:



For $f : S^N \to S^N$, define the ***semantics*** **to be the** ***least fixed point*** **of f, i.e. the smallest s (in a prefix order sense) s.t. $f(s) = s$.**

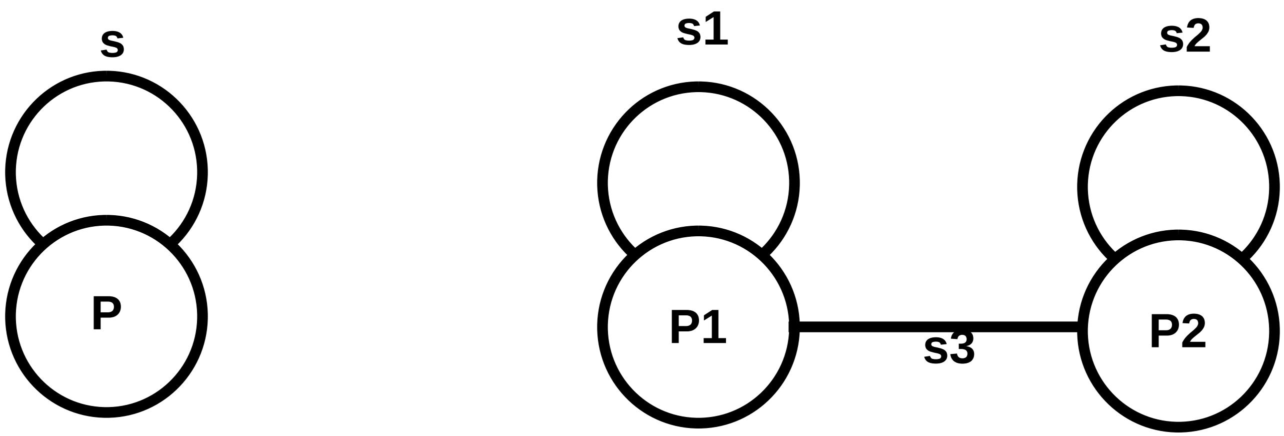
Least Fixed Point Theorems

- ***Principle***: The desired behavior is the "smallest" one consistent with the specification.
- ***Fixed-point theorem I***: A continuous function F has a least fixed point, the least upper bound of the sequence

$$
\begin{aligned}
\mathbf{s}_1 &= F(\Lambda) \\
\mathbf{s}_2 &= F(\mathbf{s}_1) \\
\mathbf{s}_3 &= F(\mathbf{s}_2) \\
&\ldots
\end{aligned}
$$

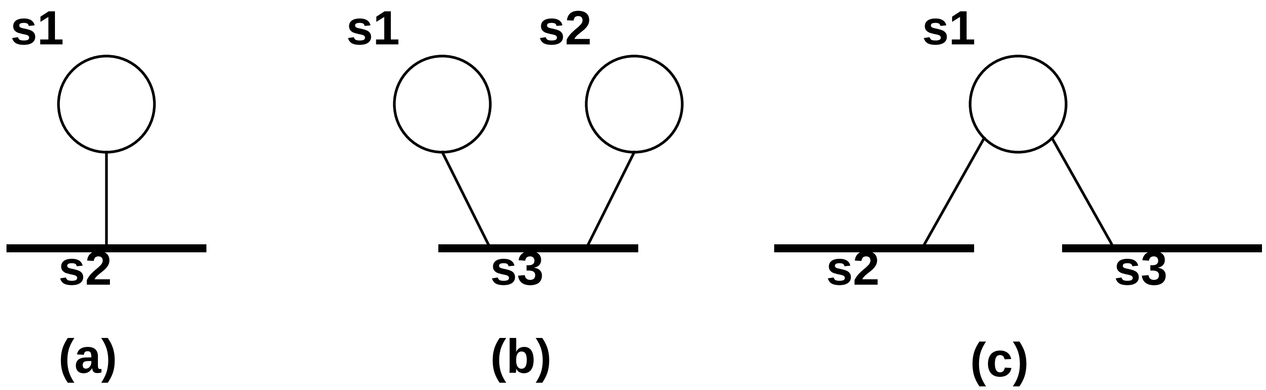
- ***Fixed-point theorem II***: A monotonic function F has a least fixed point.
- Given an input constraint $I \subseteq S^N$, the least fixed point is the unique minimum value $\min(Q \cap I)$ under the prefix order.

Sequential Processes and Rendezvous



- The events on the self-loops are totally ordered.
- There exist events in s_3 with the same tag as events in s_1 and s_2 (rendezvous).

Petri Nets (part 1)

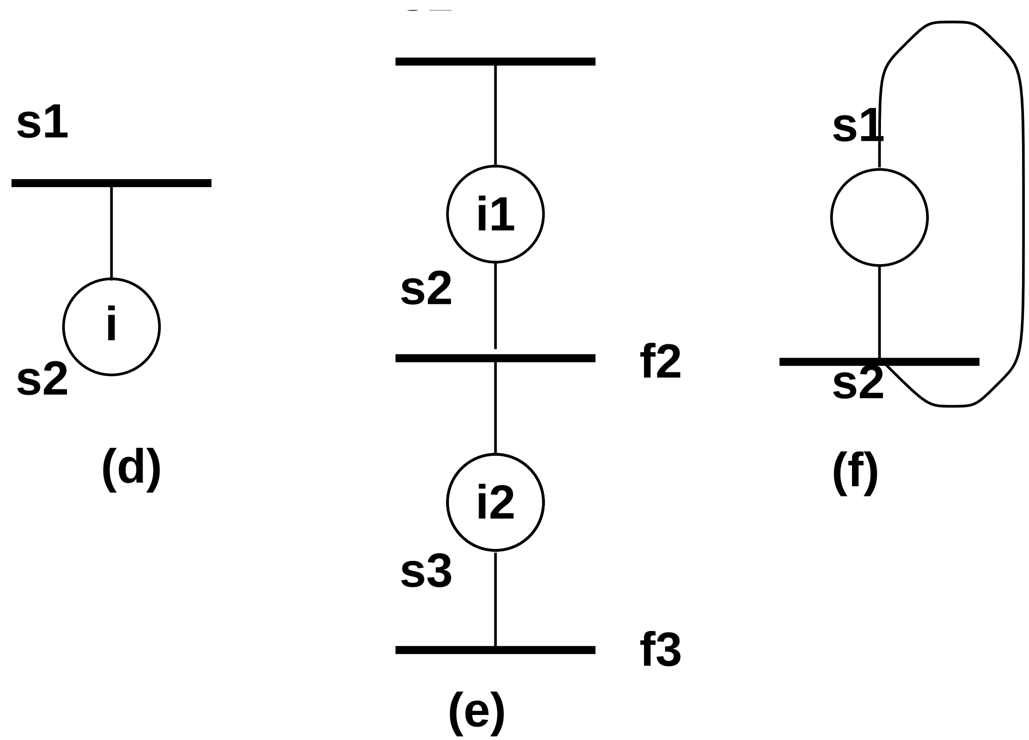


(a) There exists a one-to-one function $f : s_2 \rightarrow s_1$ **such that** $f(e) < e$ **for all** $e \in s_2$**.**

(b) There exist two one-to-one functions $f_1 : s_3 \rightarrow s_1$ **and** $f_2 : s_3 \rightarrow s_2$ **such that** $f_1(e) < e$ **and** $f_2(e) < e$ **for all** $e \in s_3$

(c) There exist two one-to-one functions $f_1 : s_2 \rightarrow s_1$ **and** $f_2 : s_3 \rightarrow s_1$ **such that** $f_1(e) < e$ **for all** $e \in s_2$ **and** $f_2(e) < e$ **for all** $e \in s_3$ **with** $f_1(s_2) \cap f_2(s_3) = \emptyset$

Petri Nets (part 2)



(d) $s_2 = s_1 \cup i$.

(e) $f_2{:}s_2 \to s_1 \cup i_1$, $f_3{:}s_3 \to s_2 \cup i_2$, $f_2(e) < e$ **and** $f_3(e) < e$, **so** $f_3(f_2(e)) < e$.

(f) $s_2 = \varnothing$.

Related Models

- Fidge, 1991 (processes that can fork and join increment a counter on each event)
- Lamport, 1978 (gives a mechanism in which messages in an asynchronous system carry time stamps and processes manipulate these time stamps)
- Mattern, 1989 (vector time)
- Mazurkiewicz, 1984 (uses partial orders in developing an algebra of concurrent “objects” associated with “events”)
- Pratt, 1986 (generalizes the notion of formal string languages to allow partial ordering).
- Winskel 1993 (describes “event structures,” a closely related framework for concurrent systems).
- Yates, 1993 (works with Δ-causal functional processes in a timed model with metric time).

Conclusions

Presented:

- **The beginnings of a framework for understanding and comparing models of computation.**
- **A suite of mathematical techniques for analyzing intrinsic properties of these models of computation.**

In the future:

- **Use this framework to understand heterogeneous mixtures of models of computation.**