Models of concurrency, categories, and games

Pierre Clairambault and Glynn Winskel

A long-overdue marriage, Games and Concurrency

Where Games and Strategies belong.
Games and strategies, a theory of interaction which supports composition based on ideas of Conway, Joyal *et al*, one where
$Player =$ System/Program over which we have control;
$Opponent =$ Environment over which we have none.

Where Concurrency belongs.
After the pioneering work of the '60's (Petri) and late '70's (Hoare and Milner), concurrency became rather a separate study in need of a broader mathematical discipline.
Arguably, games and strategies, at the right level of generality, are as fundamental as relations and functions, so provide a broad foundation.
In fact, games and strategies lead to a review of approaches to concurrency, to composition, hiding and equivalences.

What model for concurrency?

Want mathematics not syntax!
Want a basis in a mathematical model not in a process algebra.

Want a model which captures the local nature of distributed computation as abstractly as possible but not too abstractly.

Want a model which is central in that it is related to many (perhaps all) other models. So that work in that one model can be generalised to others.

$\leadsto$
Event structures, the concurrent analogue of trees - a good place to start.

Applications of partial-order models

Security protocols, as strand spaces, event strs [Guttman et al, Basin, Constable];
Systems biology, analysis of chemical pathways [Danos-Feret-Fontana-Krivine];
Hardware, in the design of asynchronous circuits [Yakovlev];
Relaxed/weak memory, event structures [Jeffrey, Pichon, Castellan];
Types and proof, domain theory [Berry, Curien-Faggian, Girard];
Nondeterministic dataflow [Jonsson];
Network diagnostics [Benveniste et al];
Logic of programs, in concurrent separation logic;
Partial order model checking [McMillan];
Distributed computation, classically [Lamport] and recently in *e.g.* analysis of trust [Nielsen-Krukow-Sassone].

The three ingredients of this course

Models for distributed computation: $\textit{Event structures}$, central within models for concurrency, Petri nets, Mazurkiewicz trace languages, transition systems, ...

Category theory by example: Universal constructions such a product and pullback, functors and adjunctions, categories with structure.

Games: 2-party nondeterministic distributed/concurrent games between Player (team of players) and Opponent (team of opponents)

Motivation

Originally as **foundation for semantics of computation**. So as a successor to Domain Theory, the mathematical foundations of Denotational Semantics. *Distributed games and strategies [provide semantics for non-deterministic dataflow, probability with nondeterminism and higher types - all bugbears of traditional domain theory.*

A **structural game theory** in which one can program games and (optimal) strategies.

More distantly, there is a hope that the generality of distributed games can help bridge the **big divide** in CS between Algorithmics and Semantics. At the very least they go some way to providing a common vocabulary.

What is a computational process?

Pre 1930's: An algorithm *(informal)*

Post 1930's: An effective partial function $\mathbf{f} : \mathbb{N} \to \mathbb{N}$ *(mathematical)*

Mid 1960's : Christopher Strachey founded denotational semantics to understand *stored programs*, *loops*, *recursive programs* on *advanced datatypes*, often with *infinite objects* (at least conceptually): infinite lists, infinite sets, functions, functions on functions on functions, ...
A program denotes a term within the λ-calculus, a calculus of functions (but is it?): $\mathbf{t} ::= \mathbf{x} \mid \lambda\mathbf{x}.\mathbf{t} \mid (\mathbf{t}\,\mathbf{t}')$

Late 1960's: Dana Scott: Computable functions acting on infinite objects can only do so via approximations (topology!). **A computational process is an (effective) continuous function $\mathbf{f} : \mathbf{D} \to \mathbf{E}$ between special topological spaces, 'domains.'** Recursive definitions as least fixed points.

Basic domain theory

A *domain* is a complete partial order $(D, \sqsubseteq)$: any infinite chain

$$d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \cdots$$

has a least upper bound $\bigsqcup_{n\in\omega} d_n$.

A function $f : D \to E$ is *continuous* if f preserves $\sqsubseteq$ and for all chains $f(\bigsqcup_{n\in\omega} d_n) = \bigsqcup_{n\in\omega} f(d_n)$.

If D has a least element $\perp$ and $f : D \to D$ is continuous, then f has a least fixed point $\bigsqcup_{n\in\omega} f^n(\perp)$. *(Recursive definitions)*

Scott (1969): A nontrivial solution to $D \cong [D \to D]$ *(a recursively defined domain)*, so providing a model of the λ-calculus, and, by the same techniques, the semantics of recursive types.

But ... although denotational semantics and its mathematical foundation, domain theory, have had tremendous successes, amongst them functional programming, it suffers from certain anomalies:

- Nondeterministic dataflow;

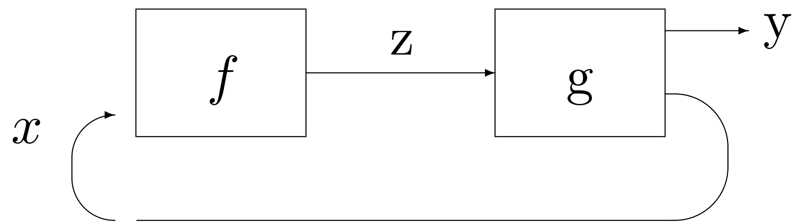
- Issues of full-abstraction;

- Concurrent/distributed computation is often captured too indirectly or too crudely;

- Although it can address probabilistic computation to some extent, it has difficulties with computation which combines probability with nondeterminism or higher types.

In summary, traditional domain theory has abstracted too early from operational concerns.

Deterministic dataflow—Kahn networks



A process built from basic processes connected by channels at which they input and output.

Simple semantics: Associate channels with streams x, y, z.

Provided f and g are continuous functions on streams there is a least fixed point

$$(x, y, z) = (g(z)_2, g(z)_1, f(x)) \ .$$

But, nondeterministic dataflow—the Brock-Ackerman anomaly!

Making domain theory more operational

In attacking the full-abstraction problem for PCF, there were several attempts to make domains more operational.

Kahn and Plotkin: *Concrete data-structures* and *sequential functions*;

Berry (and later Girard): *stable domain theory* - in which the order of information is a temporal order;

Berry and Curien: *sequential algorithms* - in which functions are replaced by special algorithms;

Abramsky-Jagadeesan-Malacaria and Hyland-Ong: *game semantics* - in which types denote games and programs strategies.

A common feature: in all cases domains are (or can be) described in terms of explicit dependencies between events.

Game semantics—a simple example

Type with a single value, the game:

$$\begin{tikzcd} \oplus \\ \ominus \arrow[u] \end{tikzcd}$$

Type with a pair of values, the game:

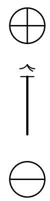
$$\begin{tikzcd} \oplus & \oplus \\ \ominus \arrow[u] & \ominus \arrow[u] \end{tikzcd}$$

Type of ‘algorithms’ from pairs to value, the game:

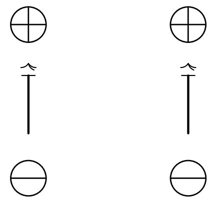
$$\begin{tikzcd} \ominus & \ominus & \oplus \\ \oplus \arrow[u] & \oplus \arrow[u] & \ominus \arrow[u] \end{tikzcd}$$

Game semantics—a simple example

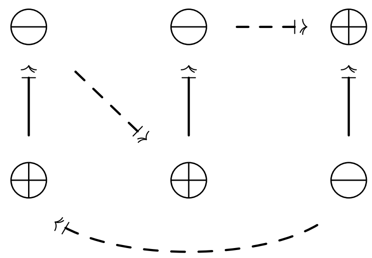
Type with a single value, the game:



Type with a pair of values, the game:



Type of ‘algorithms’ from pairs to value, the game:

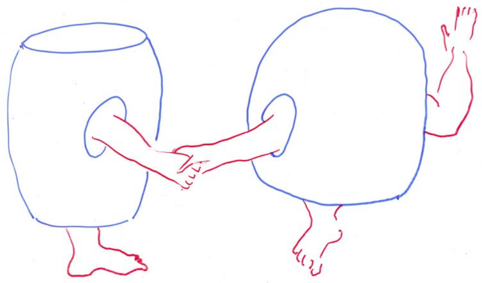


E.g. “after left then right input yield output”

Making concurrency a separate study

Difficulties with domain theory led Robin Milner (after LCF, ML) to forsake denotational semantics in favour of operational semantics; there he followed Plotkin's lead in "structural operational semantics" (SOS).

His idea: to create a fundamental basic *Calculus of Communicating Systems* into which other concurrent languages could be interpreted and reasoned about. He took as the basic primitive of communication, synchronised communication, *"synchronised handshake"* (Tony Hoare had similar ideas though based on domains of failure sets.)



(Pure) CCS

Actions: a, b, c, . . .
Complementary actions: $\overline{a}$, $\overline{b}$, $\overline{c}$, . . .
Internal action: τ
Notational convention: $\overline{\overline{a}} = a$

Processes:

p	$::=$	$\lambda.p$	prefix	λ ranges over τ, a, $\overline{a}$ for any action a
		$\sum_{i \in I} p_i$	sum	I is an indexing set
		$p_0 \parallel p_1$	parallel	
		$p \backslash L$	restriction	L a set of actions
		$p[f]$	relabelling	f a function on actions
		P		process identifier, accompanied by

Process definition: $P \stackrel{\text{def}}{=} p$,

Transition rules for CCS

nil has no rules.

Guarded processes:

$$\lambda.p \xrightarrow{\lambda} p$$

Sums:

$$\frac{p_j \xrightarrow{\lambda} q}{\sum_{i\in I} p_i \xrightarrow{\lambda} q}\ j \in I$$

Composition:

$$\frac{p_0 \xrightarrow{\lambda} p_0'}{p_0 \parallel p_1 \xrightarrow{\lambda} p_0' \parallel p_1} \qquad \frac{p_1 \xrightarrow{\lambda} p_1'}{p_0 \parallel p_1 \xrightarrow{\lambda} p_0 \parallel p_1'}$$

$$\frac{p_0 \xrightarrow{l} p_0' \quad p_1 \xrightarrow{\bar{l}} p_1'}{p_0 \parallel p_1 \xrightarrow{\tau} p_0' \parallel p_1'}$$

Restriction:

$$\frac{p \xrightarrow{\lambda} q}{p \backslash L \xrightarrow{\lambda} q \backslash L} \ \lambda \notin L \cup \overline{L}$$

Relabelling:

$$\frac{p \xrightarrow{\lambda} q}{p[f] \xrightarrow{f(\lambda)} q[f]}$$

Identifiers:

$$\frac{p \xrightarrow{\lambda} q}{P \xrightarrow{\lambda} q} \text{ where } P \stackrel{\text{def}}{=} p$$

As operations on transition systems

A CCS process p represents a transition system with states

$$\{p' \ \mid \ p \rightarrow^* p'\}\,,$$

where $p \rightarrow p'$ means $p \xrightarrow{\lambda} p'$ for some λ.

Operations of guarding, sum, parallel composition, restriction, relabelling as operations on transition systems:

The fuller story

Milner showed how to translate a variety of languages and language constructions into CCS. In particular, it is easy to interpret (early) synchronised value-passing within CCS.

CCS supports equational reasoning via equivalences such as bisimulation and weak bisimulation - the primary methods advocated by Milner.

CCS also supports the compositional proof of logical assertions, *e.g.* within the modal mu-calculus.

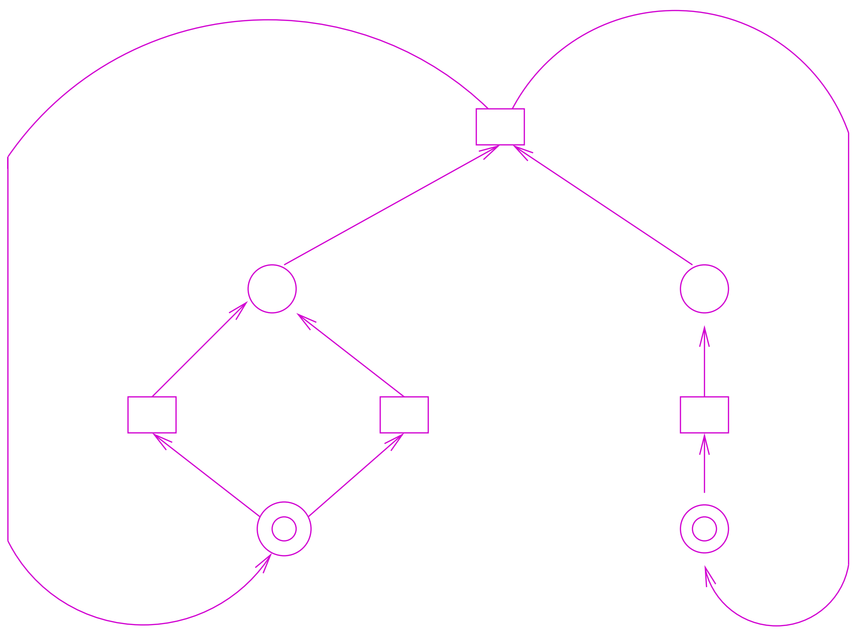
Note CCS reduces parallelism/concurrency to nondetermism: a parallel composition is represented by the nondeterministic interleaving (a shuffle) of the actions of its components.

Taking locality seriously ...

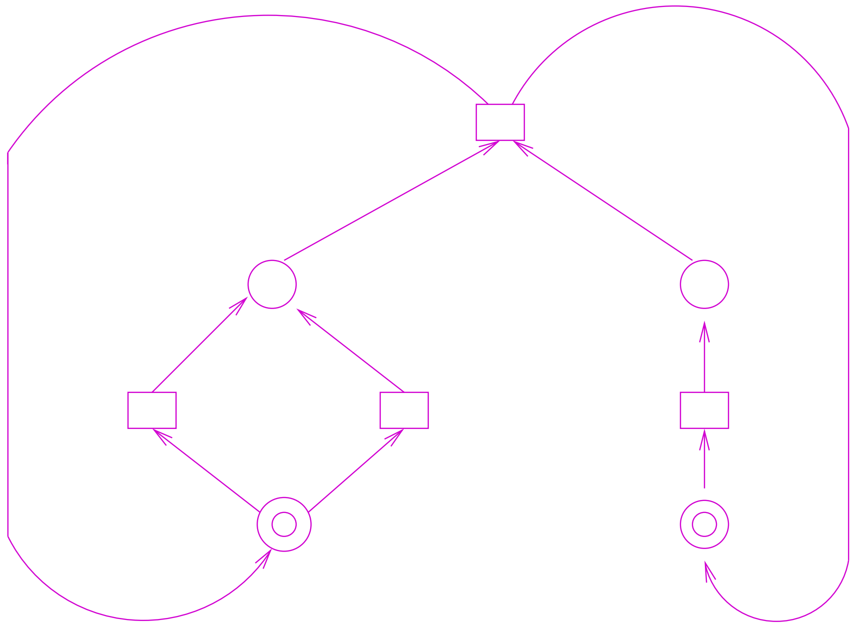
Carl Adam Petri introduced *Petri nets* in 1962.

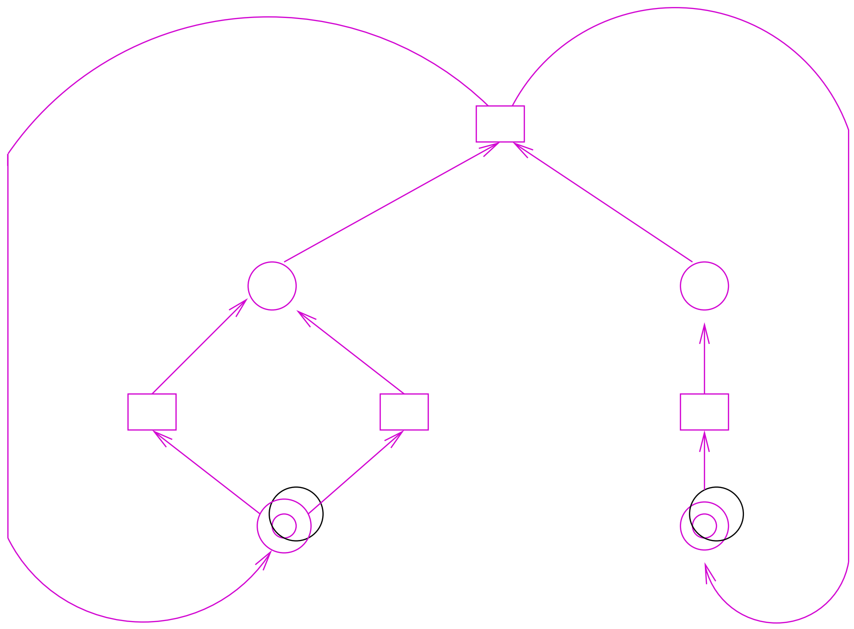
Petri nets are closely related to *partial-order models*, such as Petri's *causal nets*, *Mazurkiewicz traces* and *event structures*, in which a history of a process determines a partial order of events.

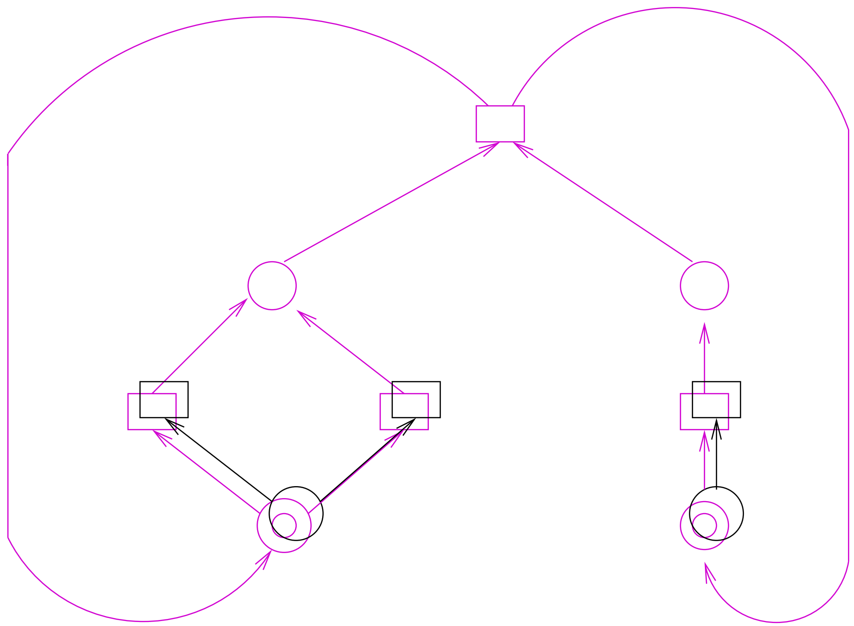
A (safe) Petri net

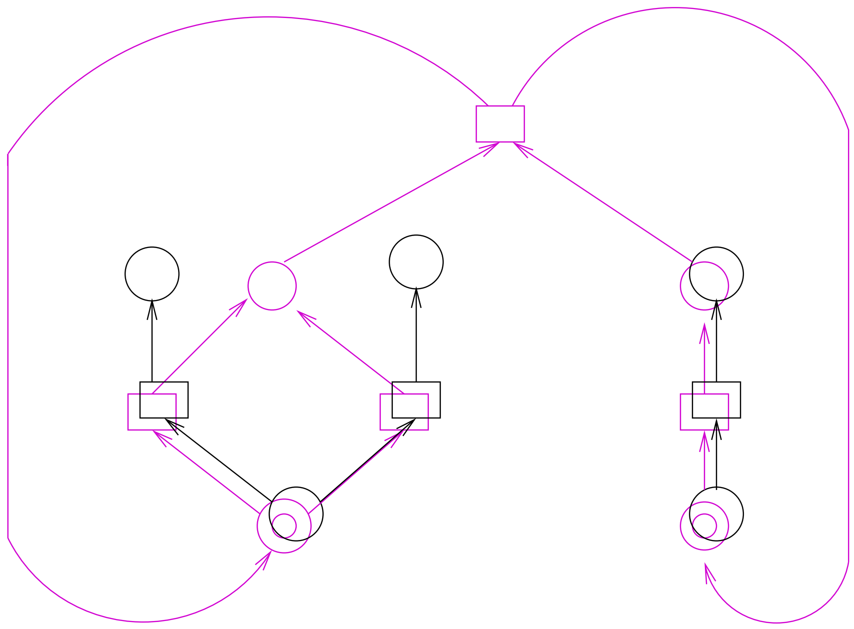


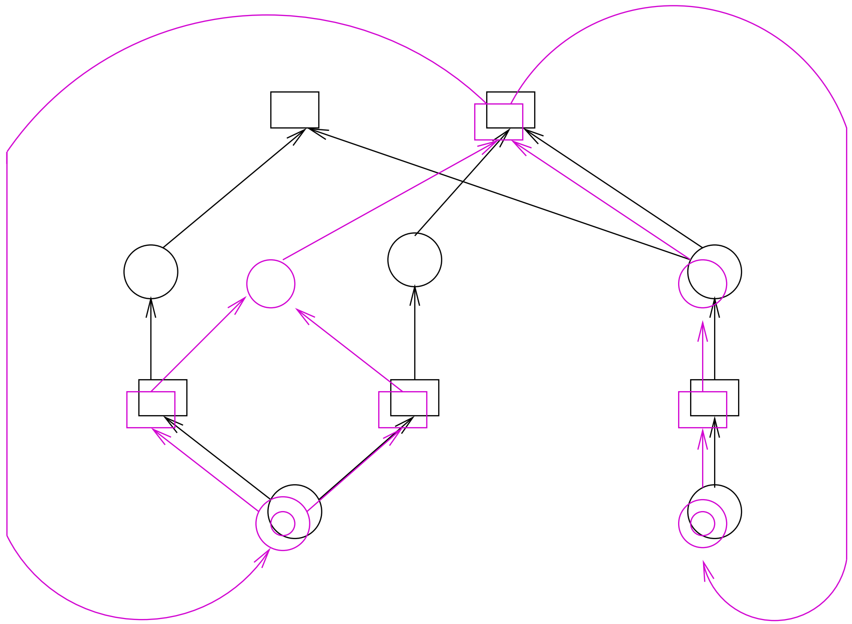
Unfolding a (safe) Petri net:

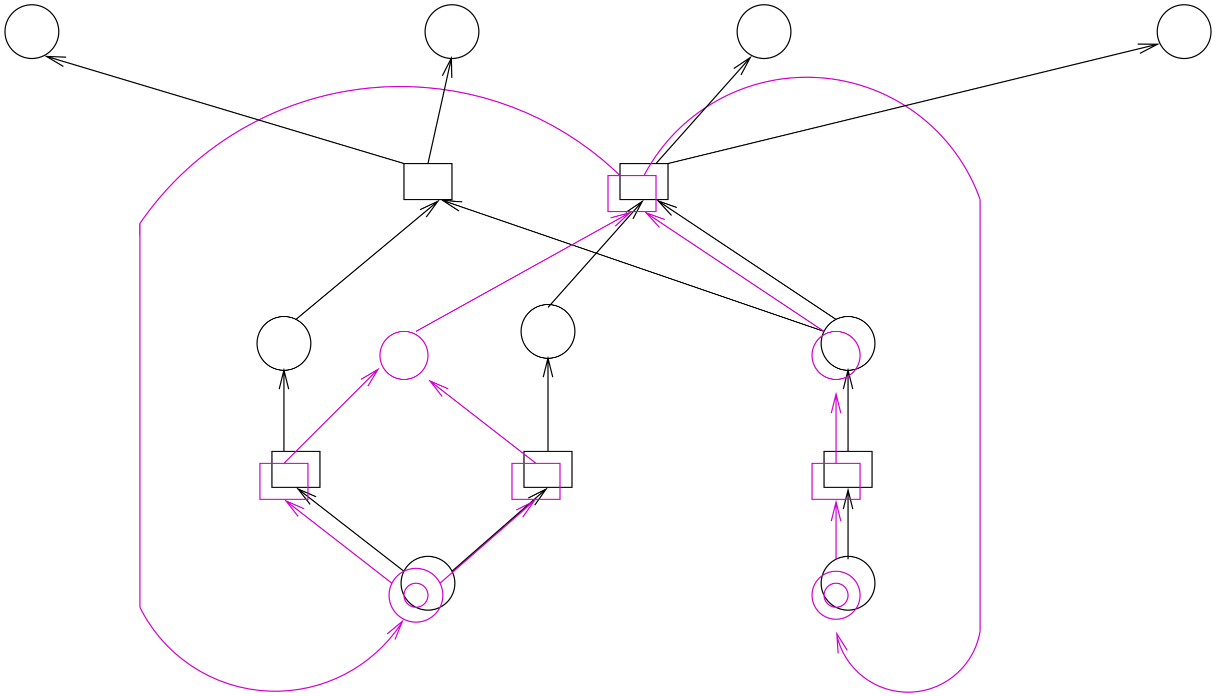


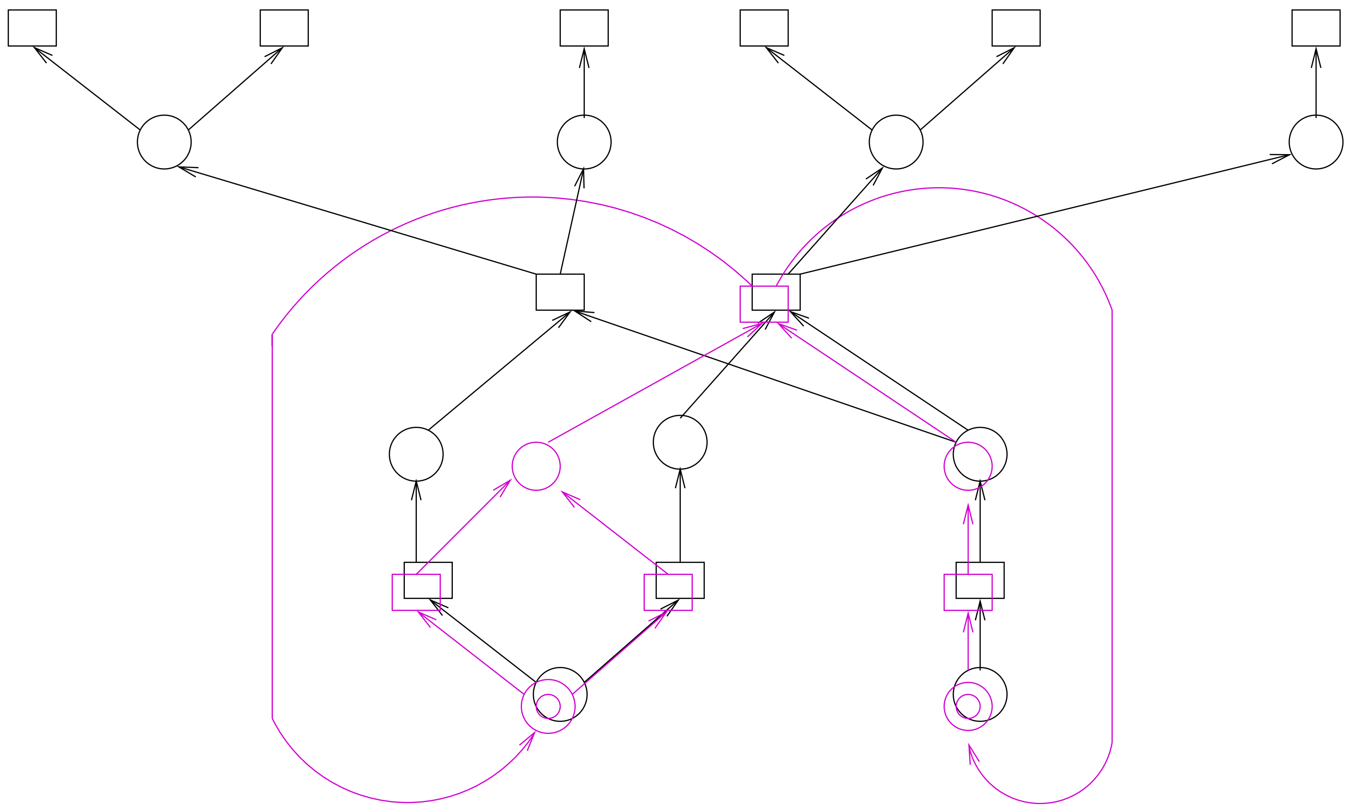


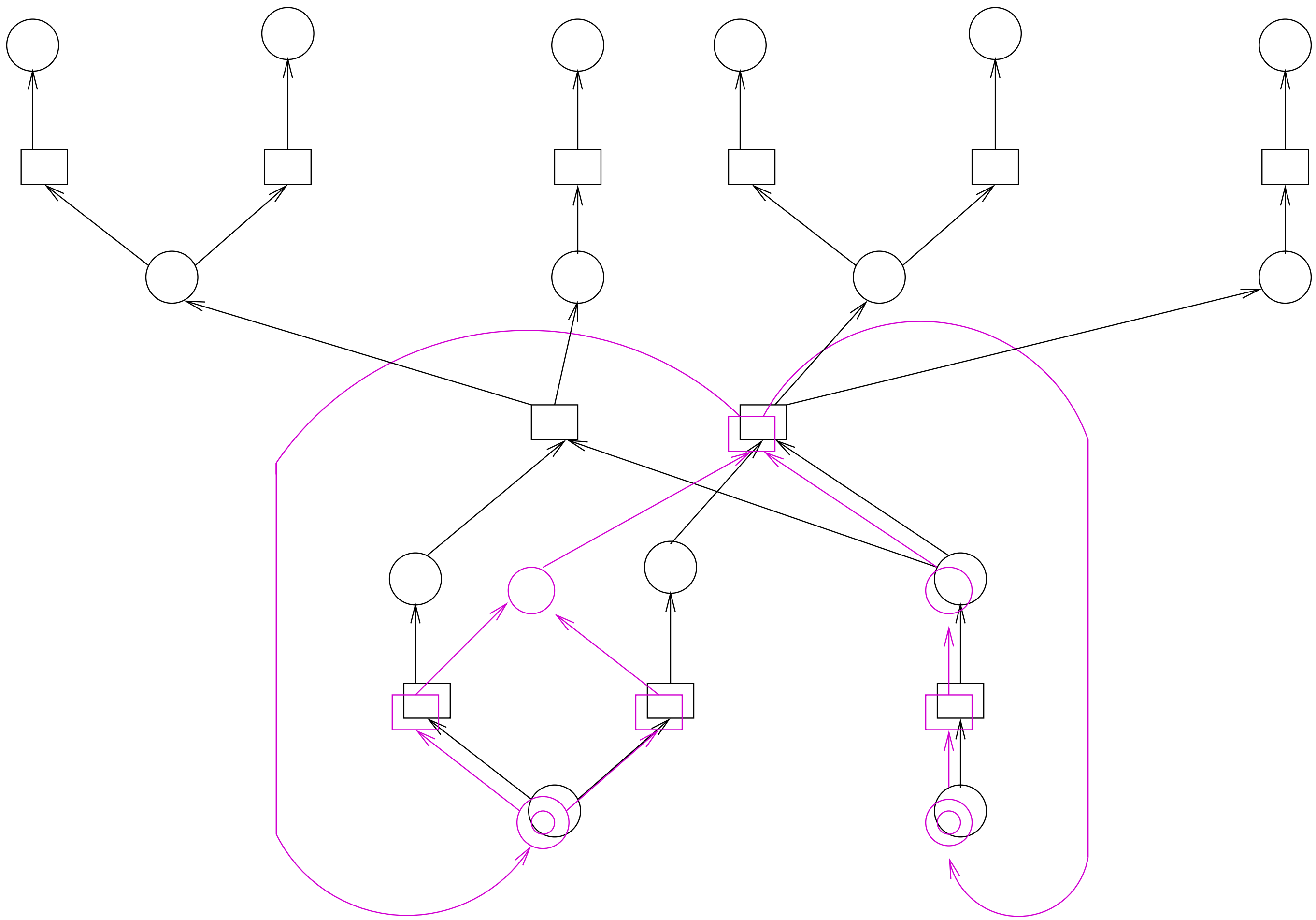


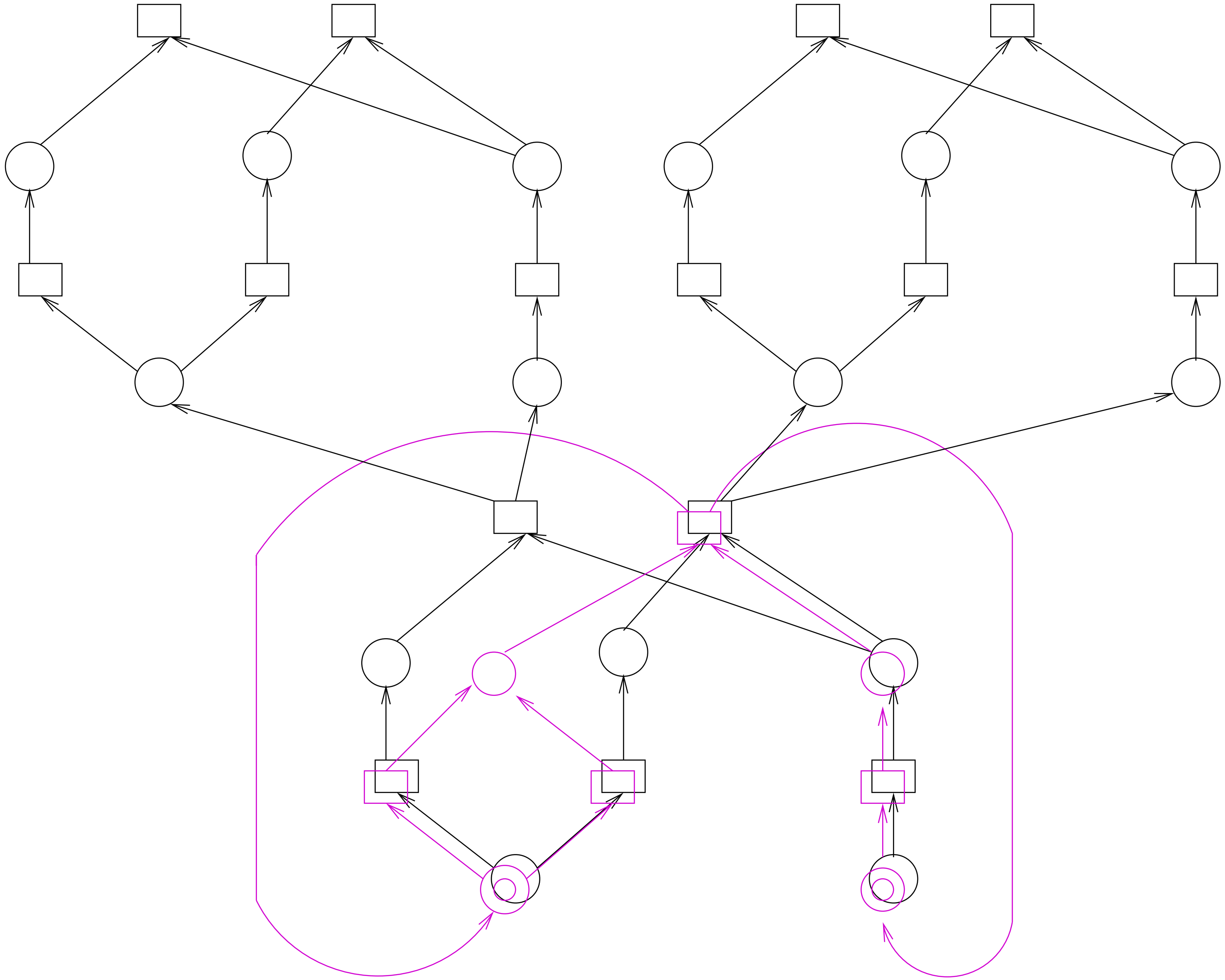


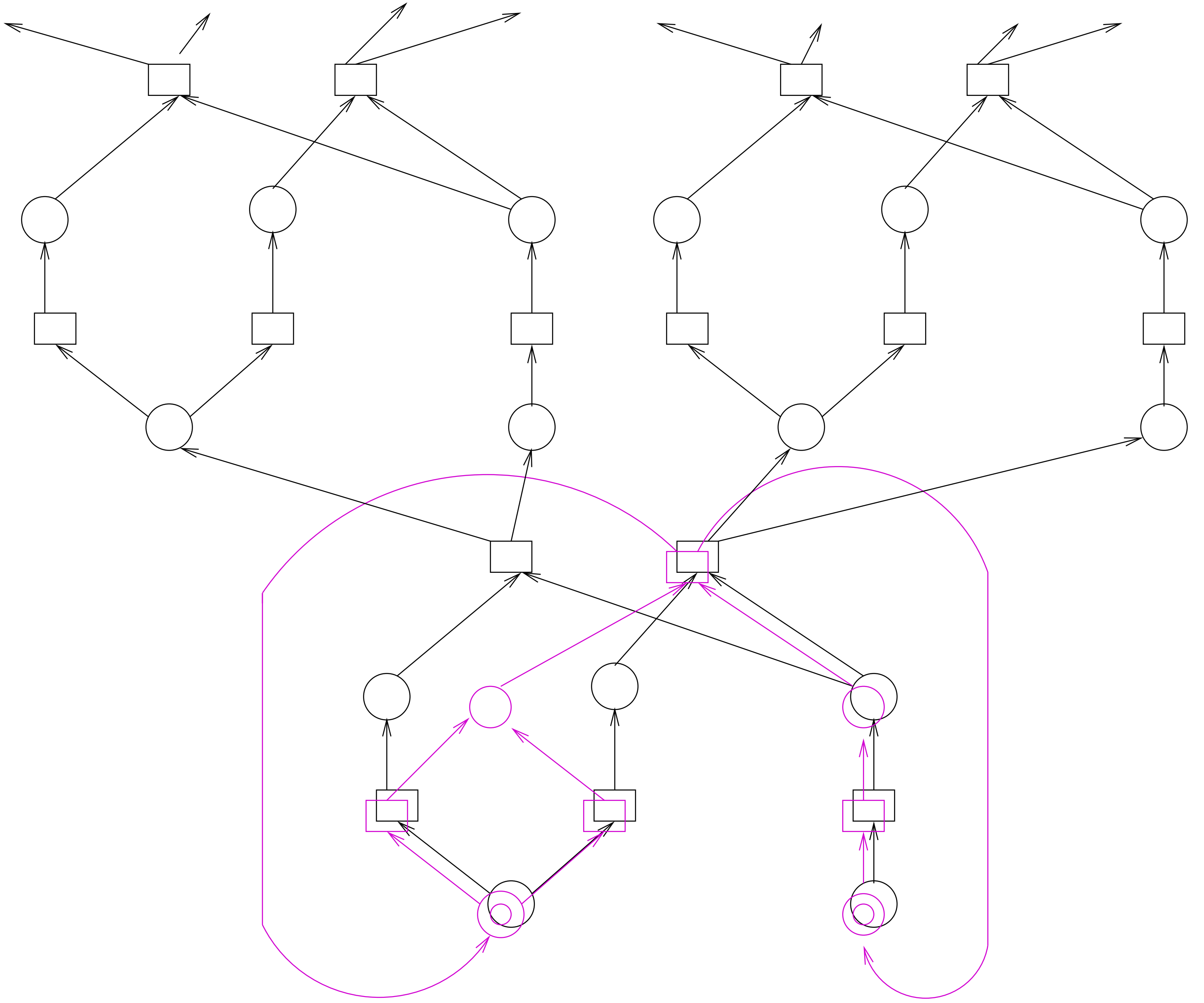


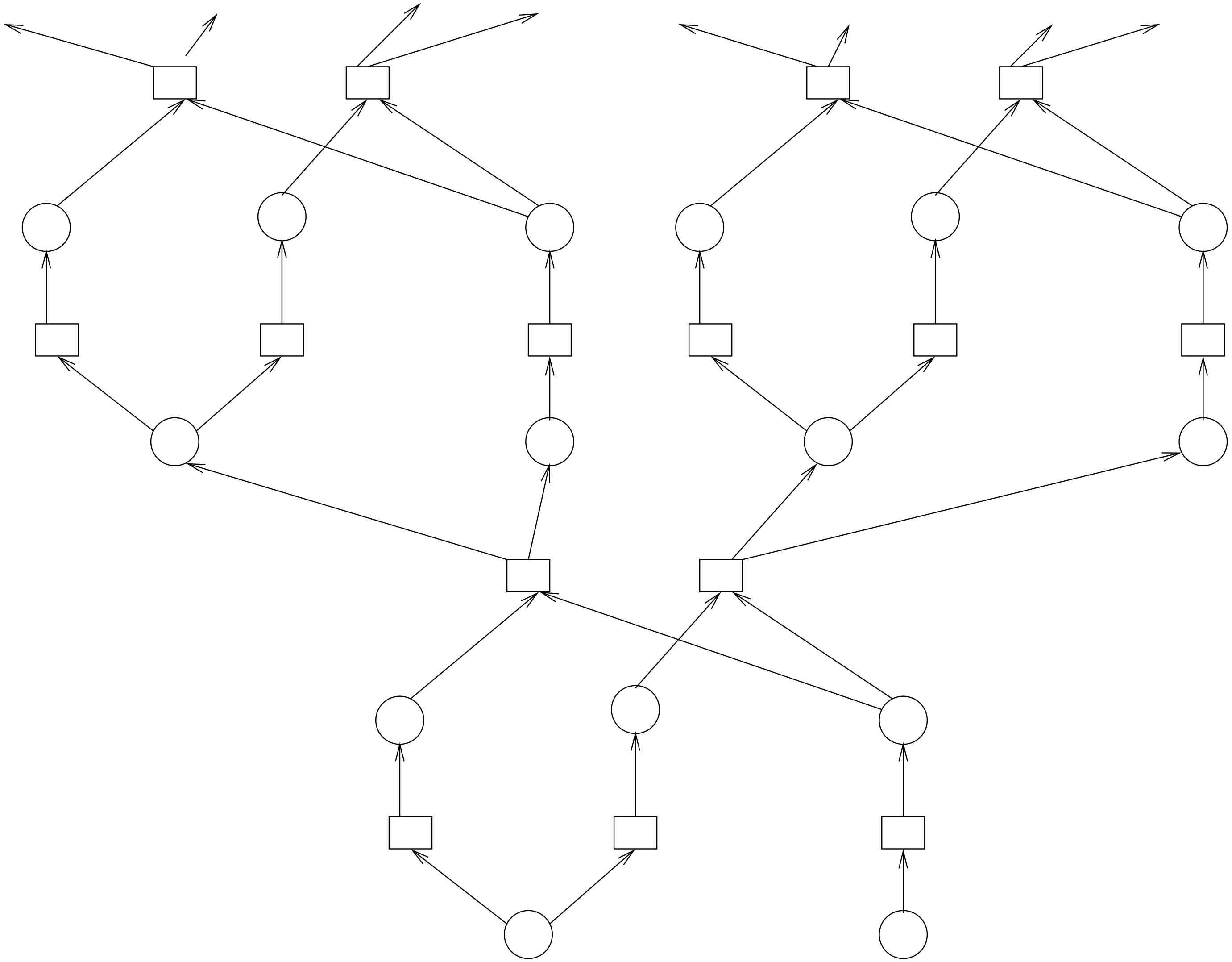


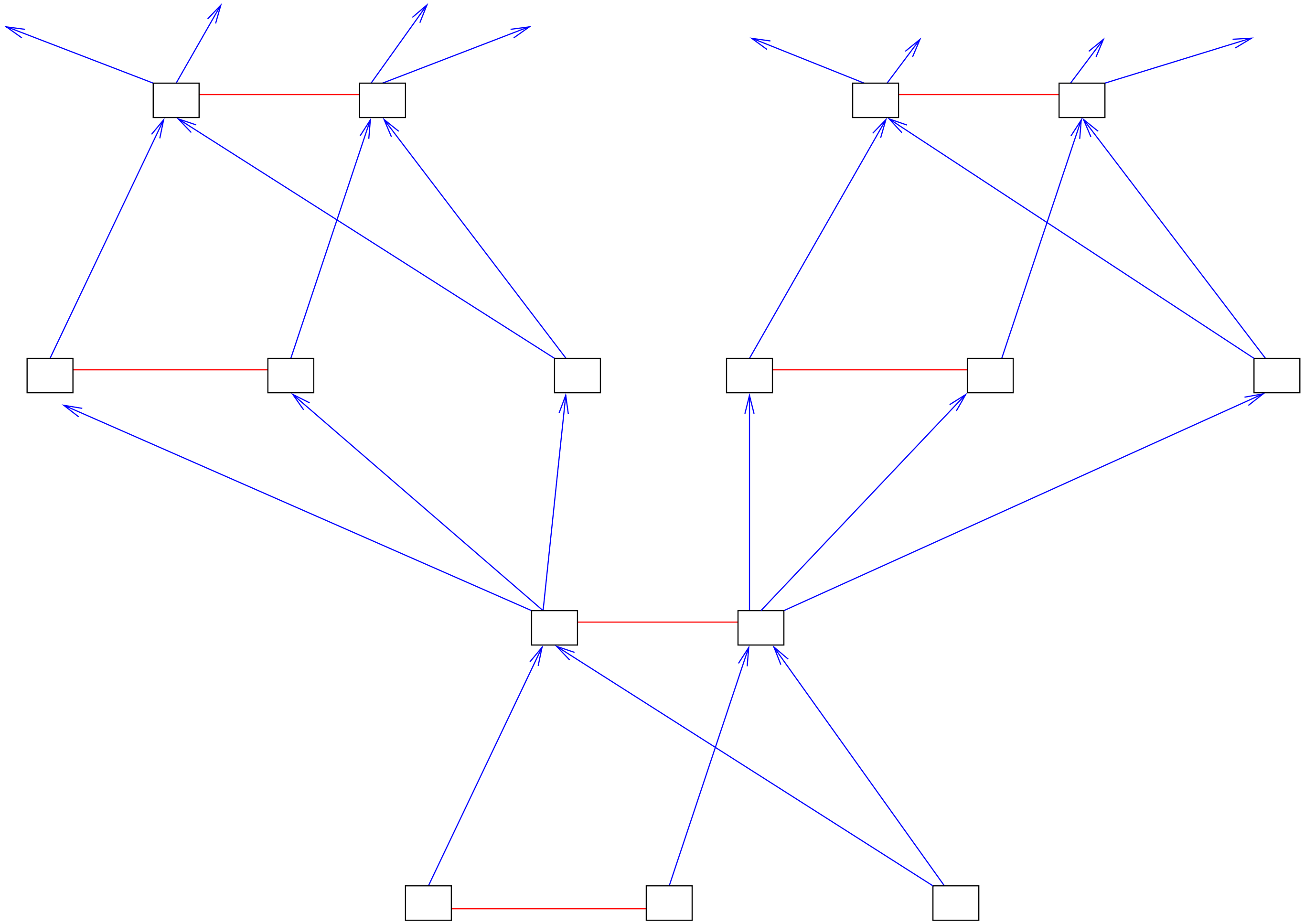












An event structure

The operations of CCS on Petri nets and event structures?

Issues:
Operations only defined up to isomorphism!
The constructions of CCS suggest ideas of map on transition systems, on nets, on event structures. (These structures aren't just graphs.)
Universal characterisations of the operations?
Relations between models;
preservation of operations in moving between models.
We need category theory!