Conservation of Data

Ian Mackie

Isaac Newton Institute for Mathematical Sciences
January 2011

Conservation of ....

There are a number of fundamental laws of nature:

- energy can be neither created nor destroyed
- mass is neither created nor destroyed. For example: a chemical reaction:

$$2H_2 + O_2 \rightarrow 2H_2O$$

- momentum
- etc. etc. from physics, chemistry and biology.

What can be conserved in computation?

- We look at conserving data
- But what is data?

Or aim: define models and a syntax for computations that preserve data. Applications to memory management (efficient compilation, in-place algorithms, embedded systems, etc.)

But what is data?

Some suggestions:

- 0,1
- primitive data (numbers, Booleans, etc.)
- data structures
- memory locations
- programs (programs as data, first class functions)

Note:

- abstract away some details
- should syntax be allowed to artificially introduce "data" that should not be counted (cf. chemical reaction)?

What is computation

What kind of computational model shall we use?

- Term Rewriting Systems (constructor systems). Computation takes place when a function meets a constructor:

$$add(Z, y) \to y$$
$$add(S(x), y) \to add(x, S(y))$$

- Lambda calculus: application meets abstraction

$$(\lambda x.t)u \to t[u/x]$$

- Logic (natural deduction): introduction rule meets elimination rule (normalisation)
- Turing machine, recursive functions, chemical abstract machine, etc.

A starting point

R. Landauer: energy needed to erase data.

- This lead to much work on reversible computation (and quantum computation) to avoid erasing data

But there are other ways to avoid erasing data:

- Linearity (e.g. linear λ-calculus)
- Conservation of data

We very briefly say something about each of these.

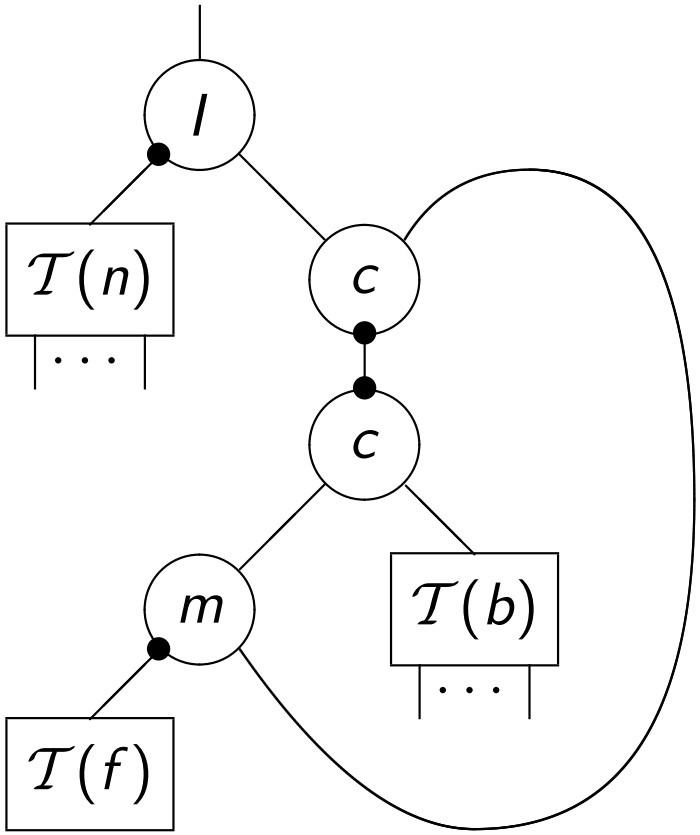
Reversible computation

- a bi-deterministic computation: the ability to move forwards or backwards in a deterministic way
- introduced in physics, relevant to quantum computation, and they also play a rule in computer science through programming language design and implementation

- it is easy to make computation reversible if we keep the history. Map $t \longrightarrow u$ into $t^{\sigma} \longrightarrow u^{t:\sigma}$
- but this is not interesting (it's cheating) as it has nothing to do with avoiding erasing bits (i.e. not thermodynamically reversible)
- other approaches though are possible, for example geometry of interaction (data flow model)

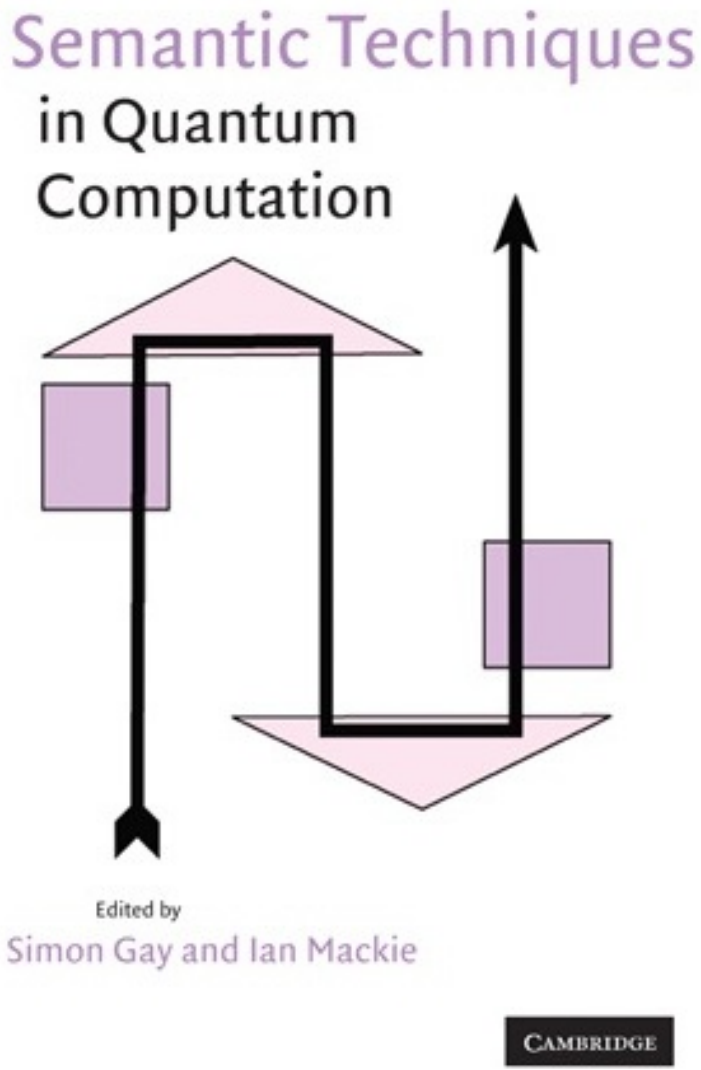
Data flow

- Geometry of interaction: data flow that keeps the "least information needed to go back"
- Works well as a model and as an implementation technique (operations on Hilbert space mapped to machine code)
- Example: $\mathit{Iter}(n, f, b) = f^n(b)$



Interlude

There is a very good book on the semantics of quantum computation.



Linearity

There are other ways to conserve data (not reversible).

- Linearity: model computation which does not create or destroy data
- A program is linear if it uses each input exactly once in producing its output. Can be enforced by dropping copying and erasing, so that computations just reorganise data
- Linear λ-calculus (symmetric monoidal closed category)

$$\begin{array}{ccccc} A \otimes (B \otimes C) & \xrightarrow{\alpha_{A,B,C}} & (A \otimes B) \otimes C & \xrightarrow{\sigma_{A \otimes B, C}} & C \otimes (A \otimes B) \\ \Big\downarrow{\scriptstyle \mathrm{id}_A \otimes \sigma_{B,C}} & & & & \Big\downarrow{\scriptstyle \alpha_{C,A,B}} \\ A \otimes (C \otimes B) & \xrightarrow[\alpha_{A,C,B}]{} & (A \otimes C) \otimes B & \xrightarrow[\sigma_{A,C} \otimes \mathrm{id}_B]{} & (C \otimes A) \otimes B \end{array}$$

Linear λ-calculus

- We can linearise any function by copying arguments. E.g. $\lambda x.xx$ becomes $\lambda xy.xy$, and we have to remember to give 2 copies of the argument (but this is cheating again).
- Can iterate linear functions to get power back
  - Gödel's System T
  - Primitive recursive functions can also be defined linearly, without projections.

But these approaches cannot be seen to conserve data. An alternative is to focus on the data and try to conserve it directly.

Conservation of data

A direct approach to not erasing (or copying) data:

- Easy to define if we cheat again: start any computation with some spare data to be used when needed, and accumulate garbage (unused data)
- So computation can be given as:

$$(P, D) \to (R, G)$$

  we just need to make sure D is big enough....
- Can we do any better?

Using existing models?

Can we use constraints on existing models of computation?

- Linear λ-calculus. Syntax does not fit well:
  $(\lambda x.x)(\lambda x.x) \rightarrow \lambda x.x$
  as data (the program) is consumed (separation theorem)
- Term Rewriting Systems
- Interaction nets: some interesting developments
- Data flow models, grouping dual operations

Need to decide/agree on a reasonable abstraction.
We look at some examples where some progress has been made

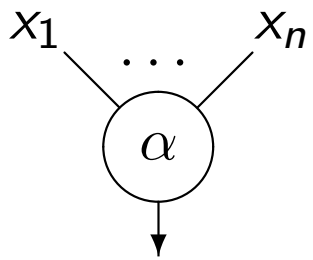
Interaction Nets (Lafont - 1990)

- A diagrammatic model of computation where both programs and data are given the same status
- Computation is rule based, and *all* the computation is expressed by the rules
  - no garbage collection
  - no copying machinery
- The rewrite rules transform the diagrams: algorithm animation

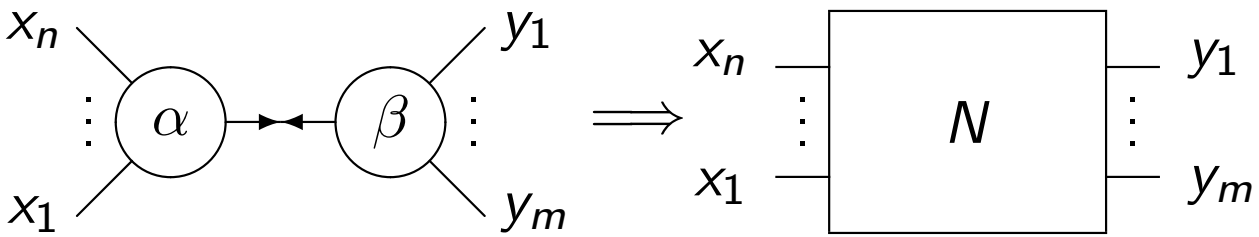
Can interaction nets be used as a model for conservation of data?

Definition

Analogous to term rewriting systems:
A set of agents:

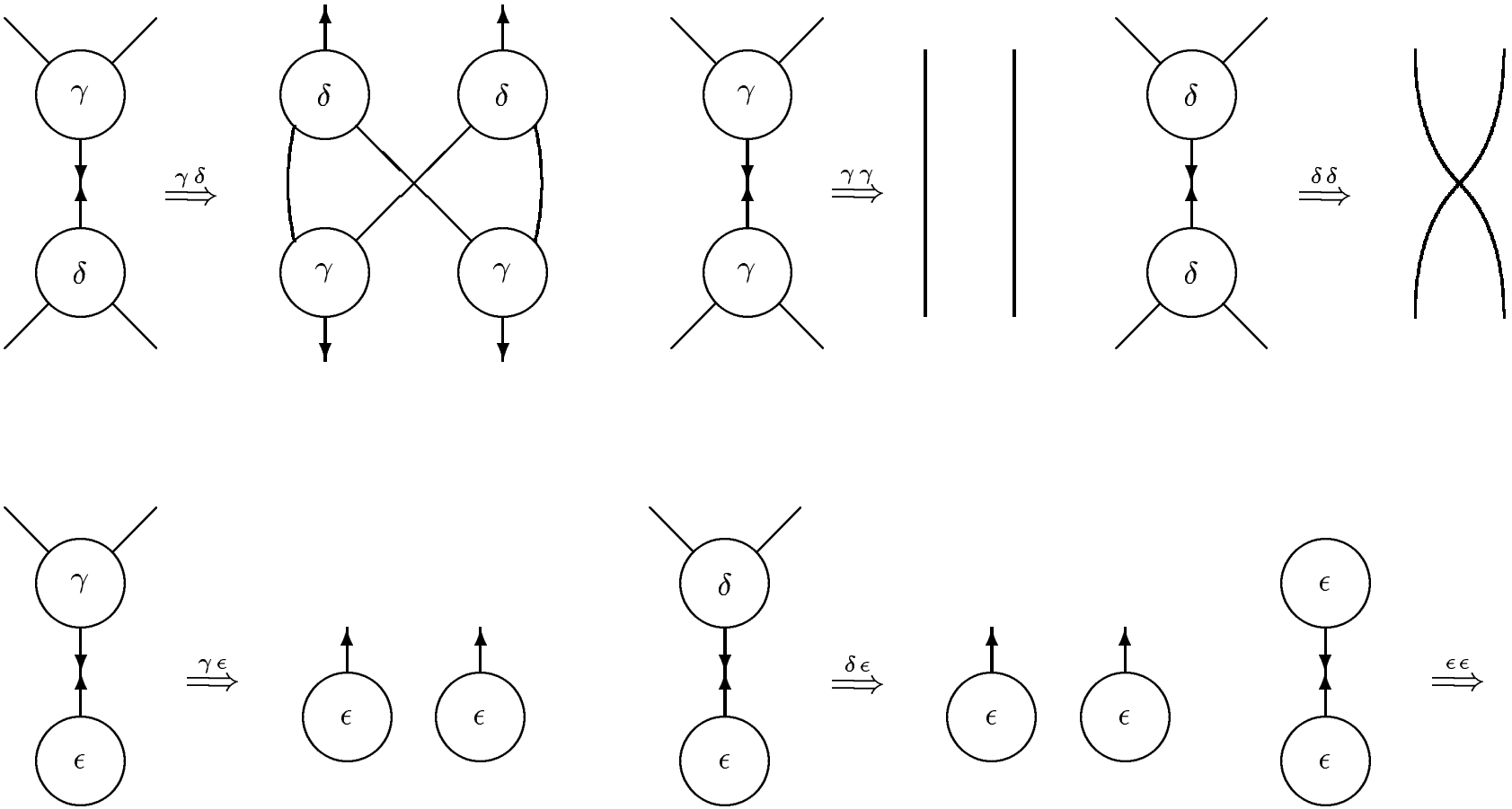


A set of rewrite rules:



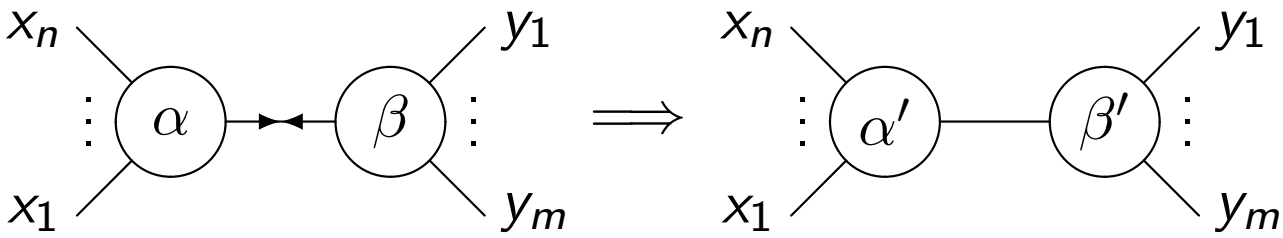
At most one rule for each pair of agents; interface is preserved

Example: Interaction Combinators



Hard interaction nets

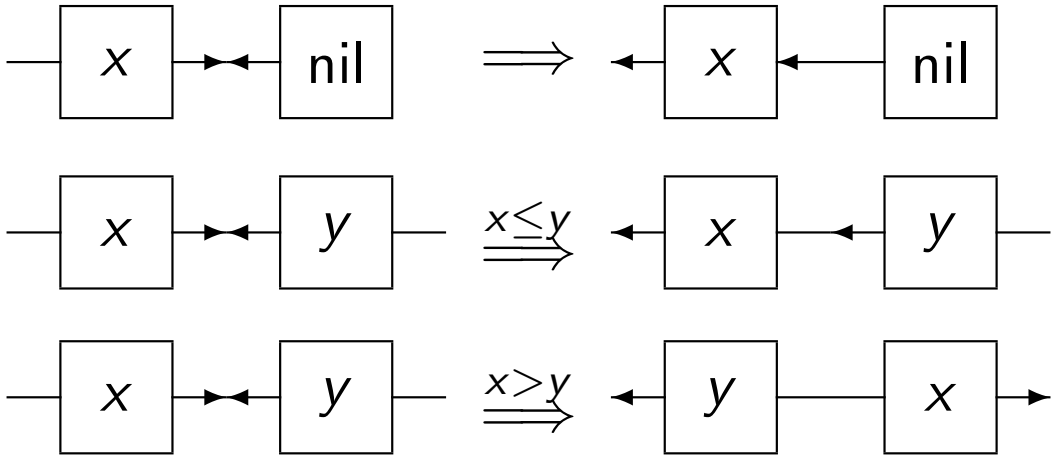
Restrict interaction to the form where the topology of the network is fixed:



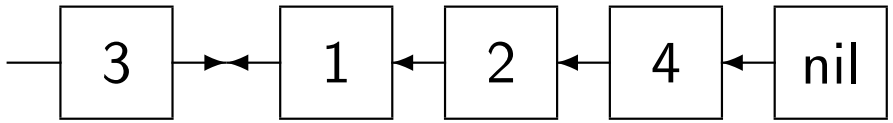
with appropriate principal ports in RHS.

Example: Insertion

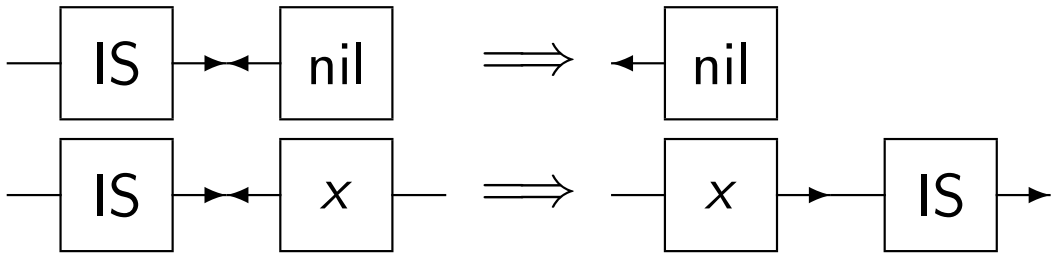
Here are the rules to insert an element into a sorted list:



- Try this example:



- It is then easy to define insertion sort:



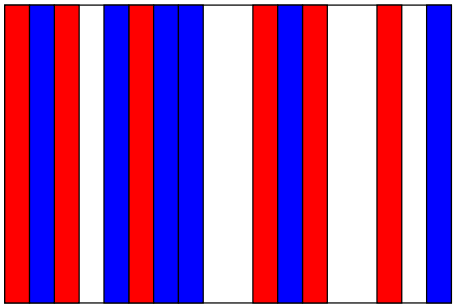
- Try bubble sort, and compare with insertion sort

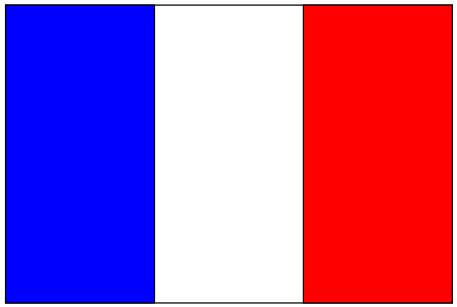
Notes

- Insertion sort can be implemented in-place. It is not clear (visually) if any other programming language has this property.
- There is a choice of application of the rules in the example given.
  But the order of application does not matter, as this system of rewriting is deterministic (all reductions lead to the same answer).
  So the algorithm is in-place no matter what order we take, which is very strong property.

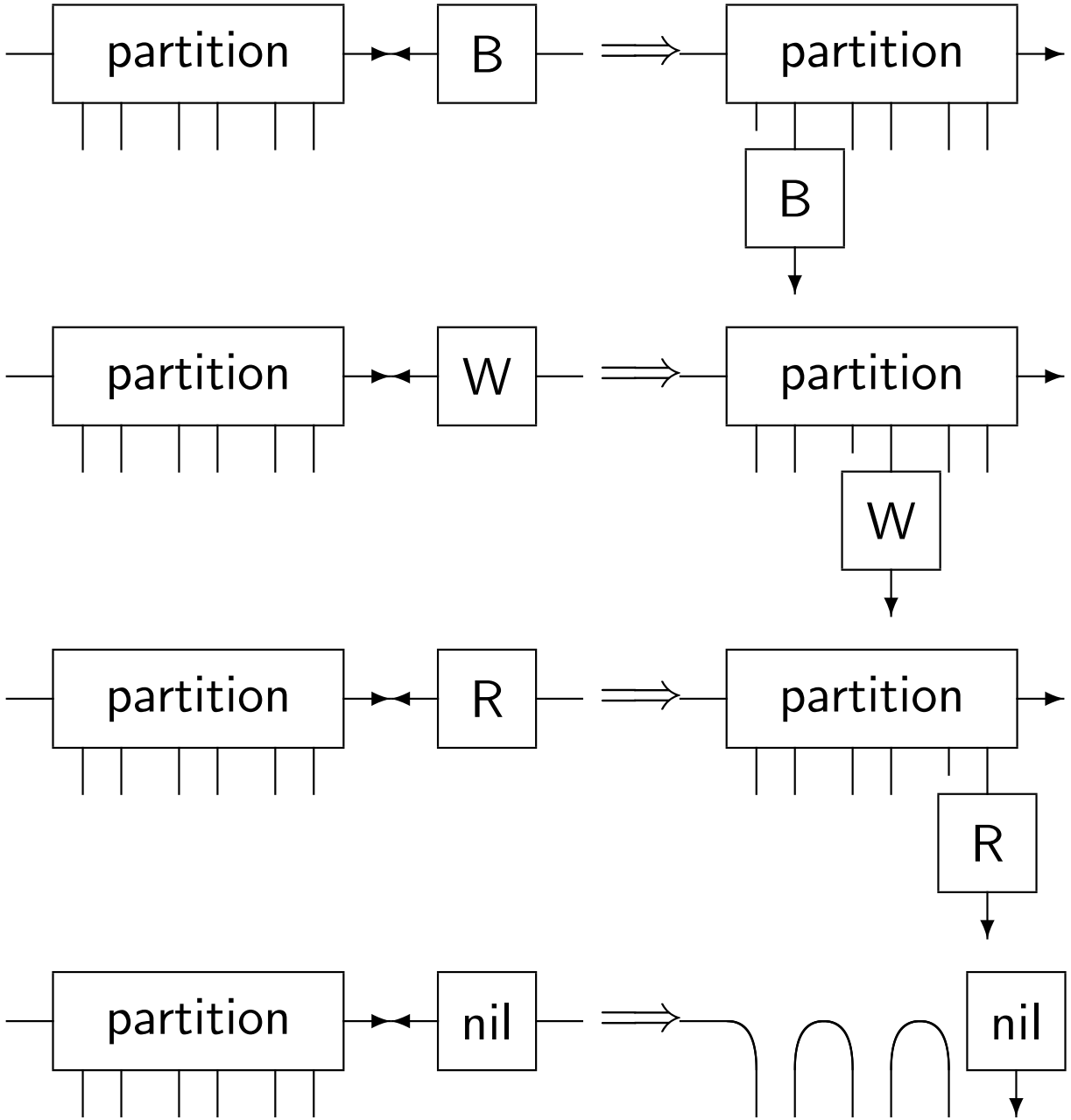
Partition and sorting

Dijkstra's partition algorithm, known as the Dutch national flag algorithm, is a 3-way partition algorithm of complexity $O(n)$. Here we recast the algorithm using the French national flag. The idea is to sort the flag from strips, as shown on the left below to give the flag on the right:



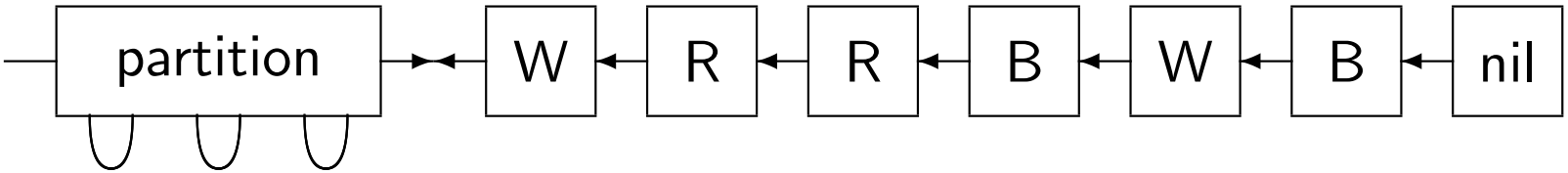


Partition

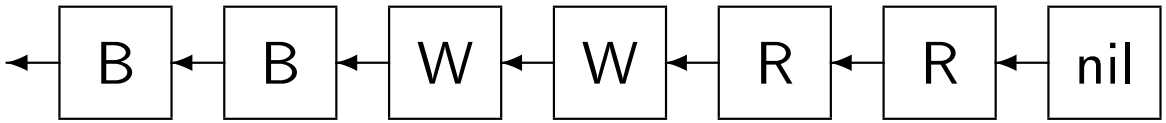

partition
B
partition
B
partition
W
partition
W
partition
R
partition
R
partition
nil
nil

Putting it all together

This algorithm is clearly linear—each element is examined once. It is also interesting to note that we can extend this algorithm to any number of partitions and still maintain a linear algorithm (which is not the case with arrays). It is also in-place by observing the rules.



After seven rewrite steps (one for each element of the list, plus one for nil), we get the following net in normal form, which is the correctly partitioned list:



Main points of these examples

1. The graphical representation of the problem is directly cast into the graphical language. The algorithm given can be understood as programming directly with the internal data structures, rather than some syntax describing it.
2. All rewrite steps correspond to steps in the computation: there is no need to introduce additional data structures and operations that are not part of the problem.
3. Because of point 1 above, if we single-step the computation we get an animation of the algorithm directly from the rewriting system.

Brief look at another approach: pairing up

Here are two functions that do not conserve data:

- succ: $n \mapsto S(n)$
- pred: $S(m) \mapsto m$

But if we do them together:

$$(n, S(m)) \mapsto (S(n), m)$$

then we restore the property.

- Synchronous or asynchronous
- Idea can be generalised with the use of buffers

Compile a program by grouping dual operations.

Conclusion

- Only given a snapshot of work in this area - many other works
- Other models: chemical abstract machine, gamma model of programming...
- Developing programming language (syntax) at the same time as understanding the problem semantically seems fruitful
- Programming directly with the model of computation
- Main message: there are many more approaches possible to study conservation of data