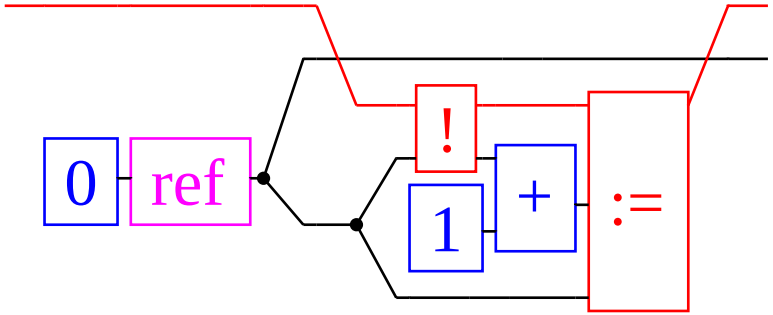
Premonoidal categories and a graphical view of programs



ALAN JEFFREY

UNIVERSITY OF
SUSSEX
AT BRIGHTON

Introduction

Two ways to give semantics to programs are:

- As data flow / control flow graphs, or
- Categorically.

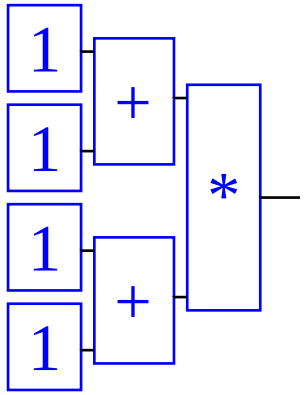
Flow graphs are usually semi-formal, common SE practice.

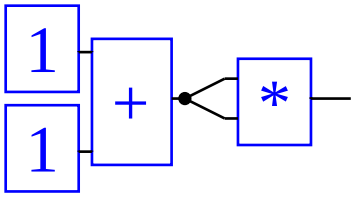
Categorical semantics is formal, specialist material.

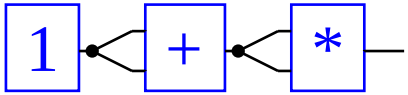
This talk describes a link between the two.

Single-coloured flow graphs

Some simple data flow graphs:





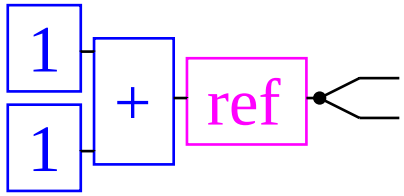


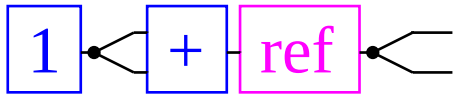
return ((1+1)*(1+1)); **let** *x*:int = 1+1; **let** *y*:int = 1;
return (*x***x*); **let** *x*:int = *y*+*y*;
return (*x***x*);

These graphs all have the same tree unfolding, so they should be considered equal.

Two-coloured flow graphs

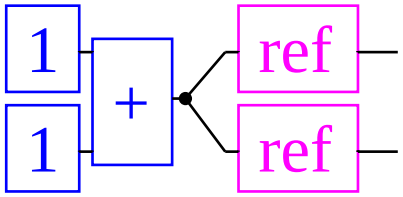
Not all graphs with the same unfolding should be considered equal:

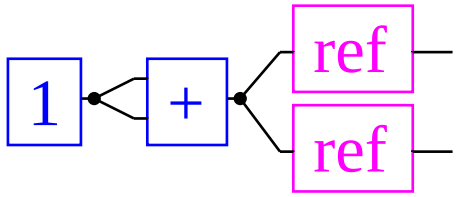




let *r*:ref = ref(1+1);
return (*r*,*r*);

let *y*:int = 1;
let *r*:ref = ref(*y*+*y*);
return (*r*,*r*);



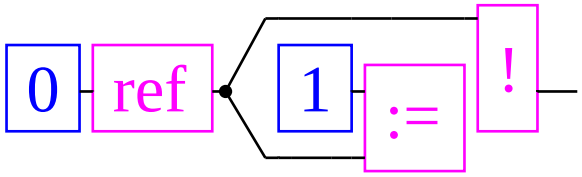


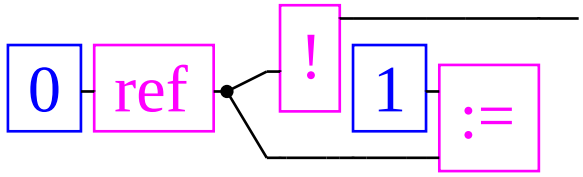
let *x*:int = 1+1;
return (ref(*x*),ref(*x*));

let *y*:int = 1;
let *x*:int = *y*+*y*;
return (ref(*x*),ref(*x*));

A problem with order of evaluation

Data flow graphs do not capture causal information:





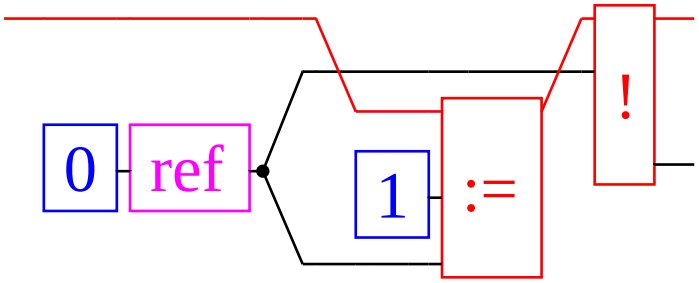
let *r*:ref = ref(0);
r := 1;
let *x*:int = !(*r*);
return *x*;

let *r*:ref = ref(0);
let *x*:int = !(*r*);
r := 1;
return *x*;

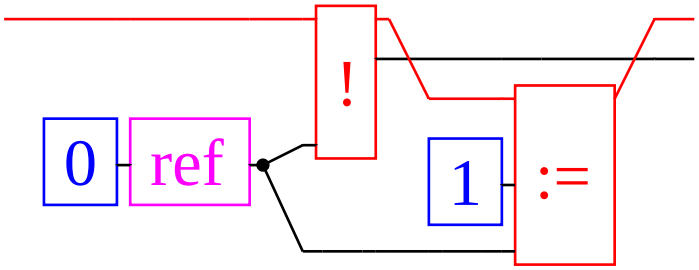
These graphs are isomorphic, but the programs have different semantics.

Three-coloured flow graphs

Add *control arcs* to nodes where order of evaluation is important:



let *r*:ref = ref(0);
r := 1;
let *x*:int = !(*r*);
return *x*;



let *r*:ref = ref(0);
let *x*:int = !(*r*);
r := 1;
return *x*;

Now the graphs are not isomorphic.

Summary of flow graphs

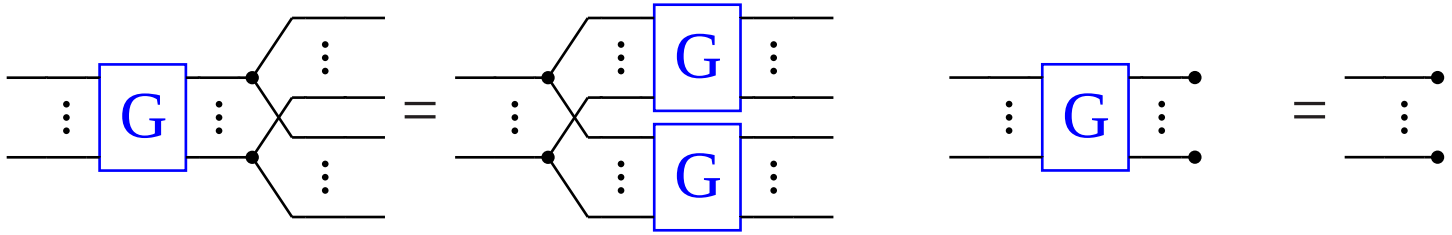
For languages with non-trivial computation, we can draw three-coloured flow graphs for programs.

- Value nodes can be copied, discarded, and evaluated in any order.
- Central nodes cannot be copied or discarded, but can be evaluated in any order.
- Process nodes cannot be copied or discarded, and order of evaluation is important.

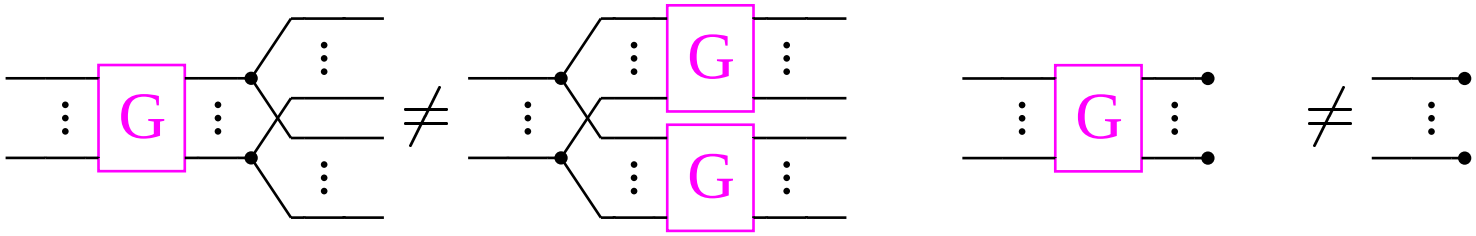
Edges are either data flow or control flow.

Equivalence between flow graphs

Equality between flow graphs is not just graph isomorphism, since graphs containing only value nodes can be copied and discarded:



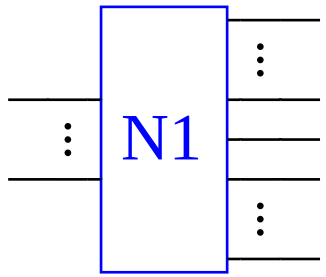
but the same is not true for graphs containing some central nodes:



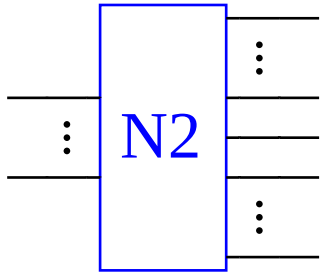
Is there an equivalence on graphs with these properties?

Equivalence between flow graphs

A relation *R* between single-coloured flow graphs *G1* and *G2* is a relation between nodes and a relation between edges, which respect labels, incoming edges and outgoing edges. An *almost simulation* is a relation *R* such that:



If *E1* in *G1* and *E1 R E2*

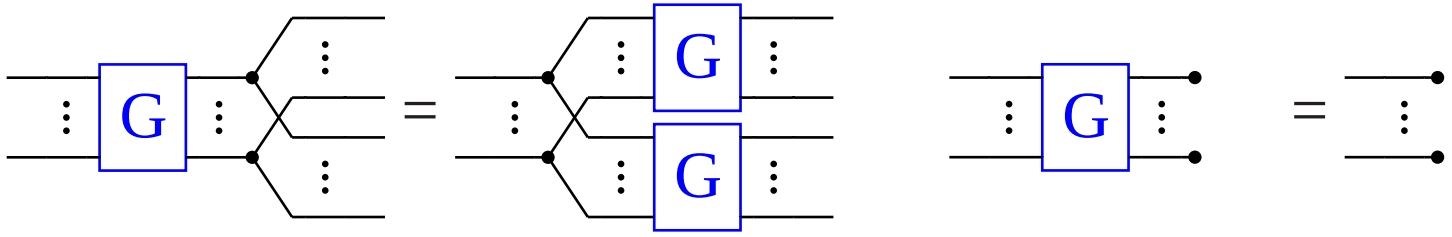


then *E2* in *G2* and *N1 R N2*.

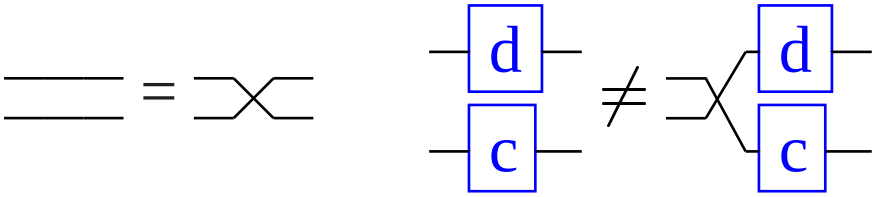
R is an *almost bisimulation* iff *R* and R^{-1} are almost simulations.

Equivalence between flow graphs

Almost bisimulation is almost the equivalence we would like between graphs. For example:



Unfortunately, it is too coarse, and fails to be a congruence:

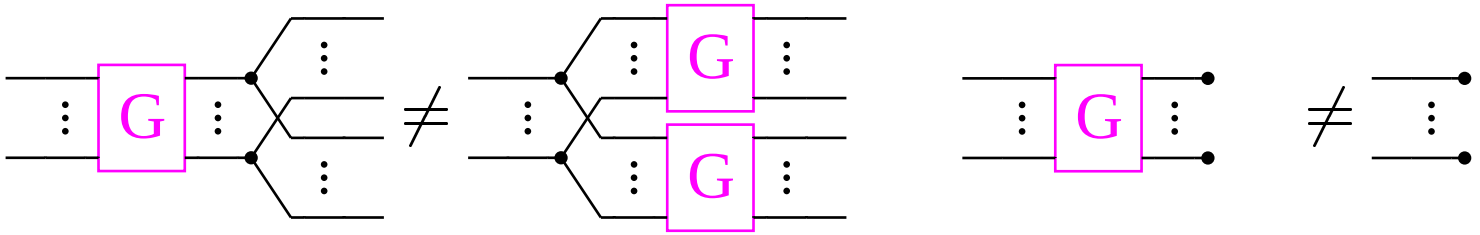


A *simulation* between *G1* and *G2* is an almost simulation which is a function on incoming edges of *G1*.

Equivalence between flow graphs

Bisimulation on single-coloured flow graphs is the same as tree unfolding.

For two-coloured flow graphs we require that value nodes can be copied and discarded, but that central nodes cannot:



We require a simulation between two-coloured flow graphs to be a function on central nodes.

Similarly, we require a simulation between three-coloured flow graphs to be a function on process nodes.

Semantics

Consider a typed programming language where we divide the constructors into three colours:

- Value constructors, which can be copied or discarded, such as 0, succ, true, cons.
- Central constructors, which cannot be copied or discarded, such as ref or rand.
- Process constructors, where order of evaluation is important, such as := or print.

We have seen example flow graphs for such a language, but what structures are necessary to give it a semantics?

Semantics

Programs can be intuitionistically typed:

- $\Gamma \vdash M : T$ in val (if M only uses value constructors).
- $\Gamma \vdash M : T$ in central (if M only uses value and central constructors).
- $\Gamma \vdash M : T$ in proc (M can use any constructor).

We can look at each of these cases in turn...

Semantics

To give a semantics for value terms over a signature Σ, we need a category with chosen finite products, and an interpretation for Σ. For example:

- Set (sets and functions).
- Cpo (complete partial orders, not necessarily with bottom).
- Term(Σ) (the appropriate term algebra).
- Graph(Σ) (single-coloured flow graphs up to bisimulation).

Call such a category a *cartesian category over* Σ.

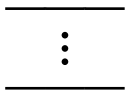
Proposition *Graph(Σ) is the initial strict cartesian category over* Σ.

As a corollary, giving a flow graph semantics for a language *uniquely* defines a semantics in any other strict cartesian category over Σ.

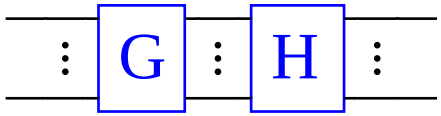
Semantics

The category Graph(Σ) has vectors of sorts as objects, and single-coloured flow graphs as morphisms, considered up to bisimulation.

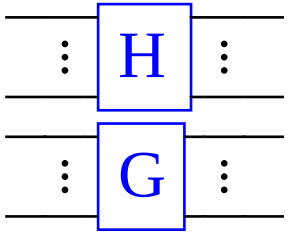
Identity:



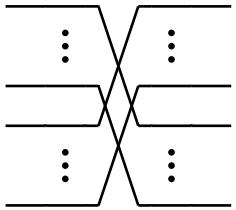
Composition:



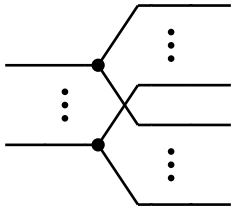
Tensor:



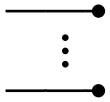
Symmetry:



Diagonal:



Terminal:



Semantics

To give a semantics for central terms over signatures Σ and Σ, we need a symmetric monoidal category $\mathbb{C}$ with an interpretation for Σ, and $\mathbb{V}$ a subcategory with the same objects. For example:

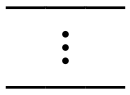
- $\mathbf{Set} \hookrightarrow \mathbf{Rel}$.
- $\mathbf{Cpo} \hookrightarrow \mathbf{Cpo}_{\vee}$.
- $\mathbb{V} \hookrightarrow \mathbb{V}_T$, where T is any commutative monad satisfying the monic condition, and $\mathbb{V}_T$ is the Kliesli category.
- $\mathbf{Term}(\Sigma) \hookrightarrow \mathbf{Term}(\Sigma,\Sigma)$.
- $\mathbf{Graph}(\Sigma) \hookrightarrow \mathbf{Graph}(\Sigma,\Sigma)$.

Proposition *$\mathbf{Graph}(\Sigma,\Sigma)$ is the initial strict symmetric monoidal category over over Σ with $\mathbf{Graph}(\Sigma)$ as a sub-smc.*

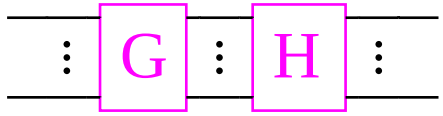
Semantics

The category Graph(Σ,Σ) has vectors of sorts as objects, and two-coloured flow graphs as morphisms, considered up to bisimulation.

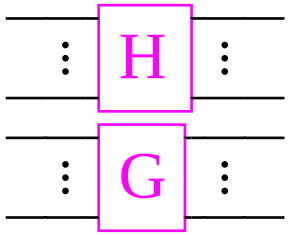
Identity:



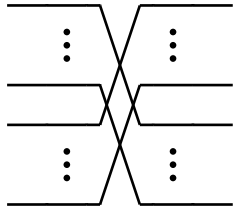
Composition:



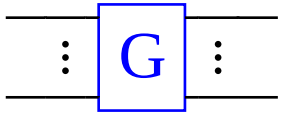
Tensor:



Symmetry:



Subcategory:



Semantics

A *strict symmetric premonoidal category* (proposed by Power and Robinson) is a category P together with:

- A strict symmetric monoidal category C with the same objects as P (called the *centre* of P).
- An identity on objects inclusion $\mathrm{C} \hookrightarrow \mathrm{P}$.
- Two functors:

$$\oslash : \mathrm{C} \times \mathrm{P} \to \mathrm{P}$$
$$\obar : \mathrm{P} \times \mathrm{C} \to \mathrm{P}$$

  such that:
  - the three functors $\otimes$, $\oslash$ and $\obar$ coincide on objects,
  - the three 'obvious' functors from $\mathrm{C} \times \mathrm{C}$ to P coincide, and
  - the symmetry in C is a natural isomorphism $X \oslash Y \simeq Y \obar X$ in P.

Semantics

Premonoidal structure (together with an interpretation of base types and constructors) is exactly what is required to give a semantics of process terms. For example:

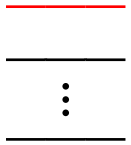
- $\mathbf{V} \hookrightarrow \mathbf{C} \hookrightarrow \mathrm{State}_Z(\mathbf{C})$, where $\mathrm{State}_Z(\mathbf{C})[X,Y]$ is $\mathbf{C}[X \otimes Z, Y \otimes Z]$.
- $\mathbf{V} \hookrightarrow \mathbf{V}_T \hookrightarrow \mathbf{V}_U$, where T is any commutative monad satisfying the monic condition, U is any monad satisfying the monic condition, and there is an appropriate natural transform from T to U.
- $\mathrm{Term}(\Sigma) \hookrightarrow \mathrm{Term}(\Sigma,\Sigma) \hookrightarrow \mathrm{Term}(\Sigma,\Sigma,\Sigma)$.
- $\mathrm{Graph}(\Sigma) \hookrightarrow \mathrm{Graph}(\Sigma,\Sigma) \hookrightarrow \mathrm{Graph}(\Sigma,\Sigma,\Sigma)$.

Proposition *$\mathrm{Graph}(\Sigma,\Sigma,\Sigma)$ is the initial strict symmetric premonoidal category over over Σ with $\mathrm{Graph}(\Sigma,\Sigma)$ as its centre.*

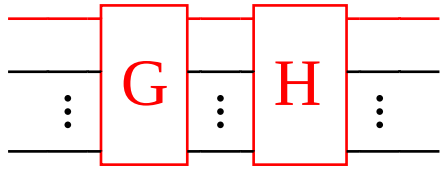
Semantics

The category Graph(Σ,Σ,Σ) has vectors of sorts as objects, and three-coloured flow graphs as morphisms, considered up to bisimulation.

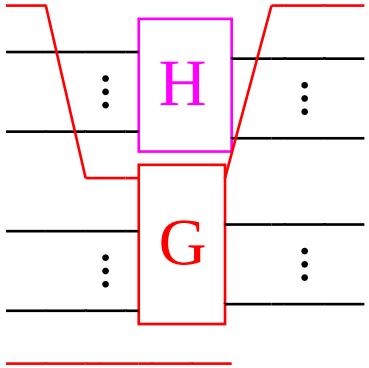
Identity:



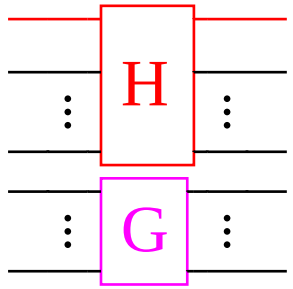
Composition:



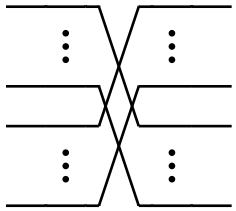
Tensor left:



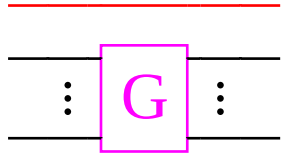
Tensor right:



Symmetry:



Centre:



Semantics

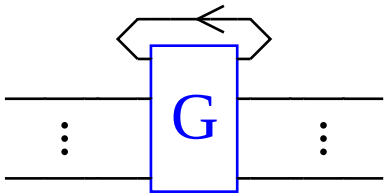
Another presentation of this result about flow graphs is to say that for any strict symmetric premonoidal category $C \hookrightarrow P$ we can find a symmetric monoidal category D such that:

$$\begin{array}{ccc} C & \hookrightarrow & P \\ \downarrow\wr & & \downarrow\wr \\ D & \hookrightarrow & \mathrm{State}(D) \end{array}$$

So the state transformer premonoidal category is in this sense canonical.

Cyclic graphs

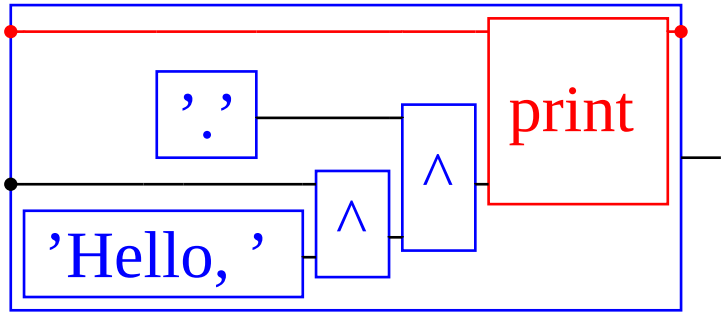
So far, all graphs have been acyclic, but we can generalize these results to show that the appropriate categories of cyclic graphs give the intial category with a *partial trace*, that is a category with a feedback operator:



The notion of partial trace is a generalization of Joyal, Street and Verity's traced monoidal categories, and adapts Hasegawa's semantics for call-by-name languages to call-by-value.

Higher-order graphs

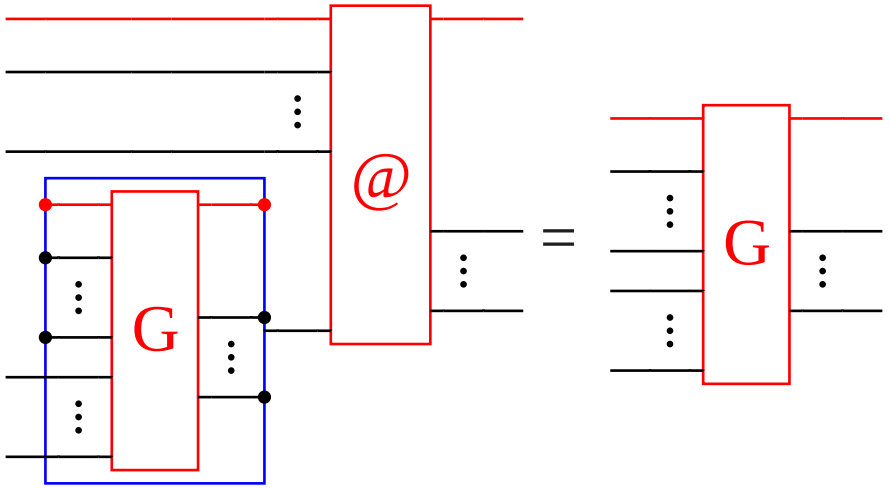
We can extend flow graphs with functions, for example:




```
proc hello (name:string) {
  print (’Hello, ’ ^ name ^ ’.’);
}
```

and add application nodes. We factor the graphs by equations such as:

Beta:

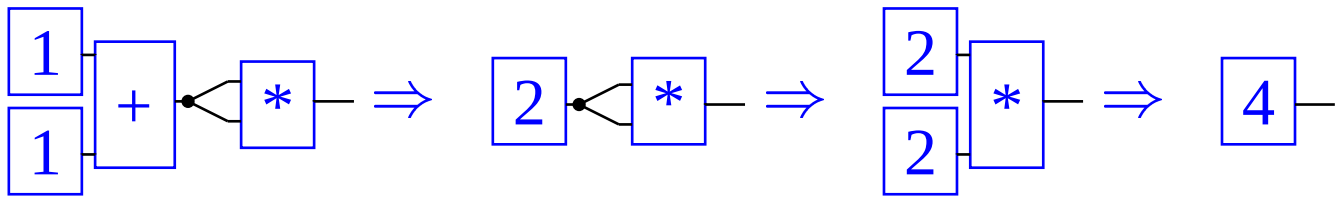


Operational semantics

Following Gardner's name-free presentation of Milner's action calculi, we would expect to be able to give a graphical operational semantics to a language, where:

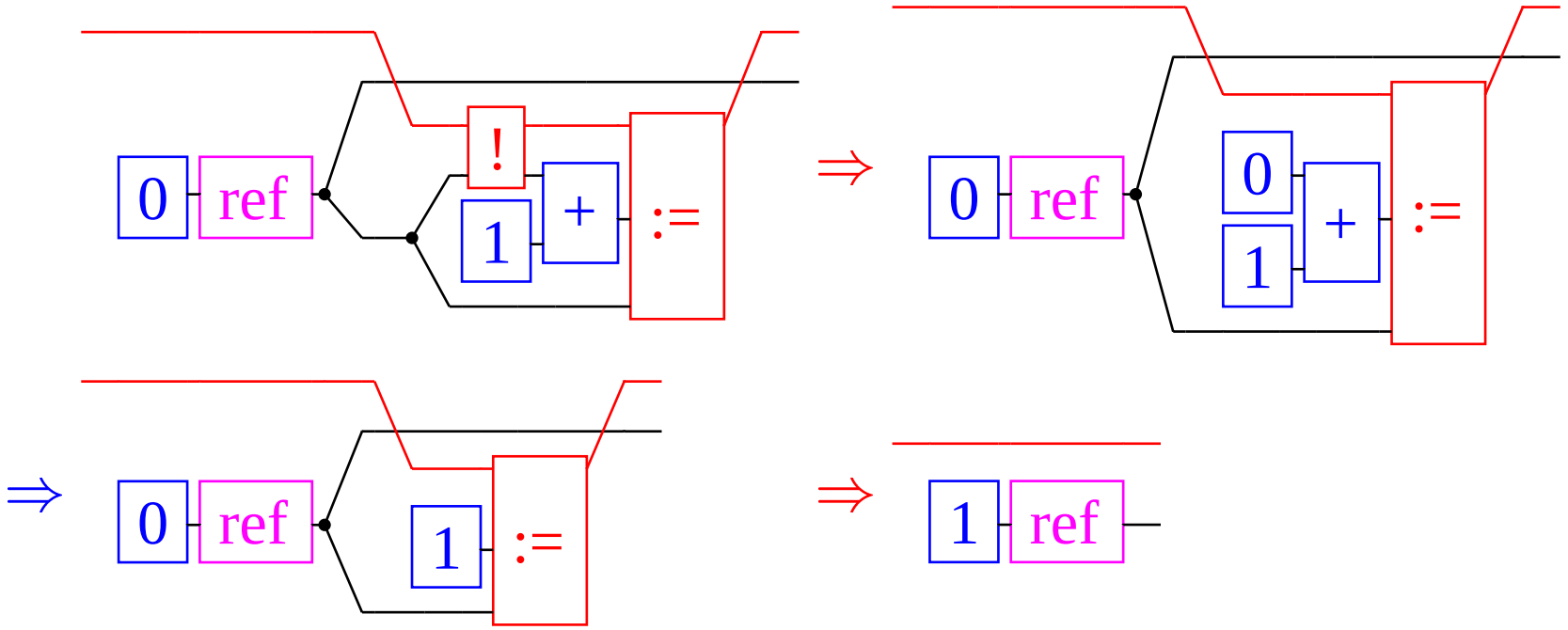
- Objects are vectors of types.
- Morphisms are graphs.
- Two-cells are reductions between graphs.

For example:



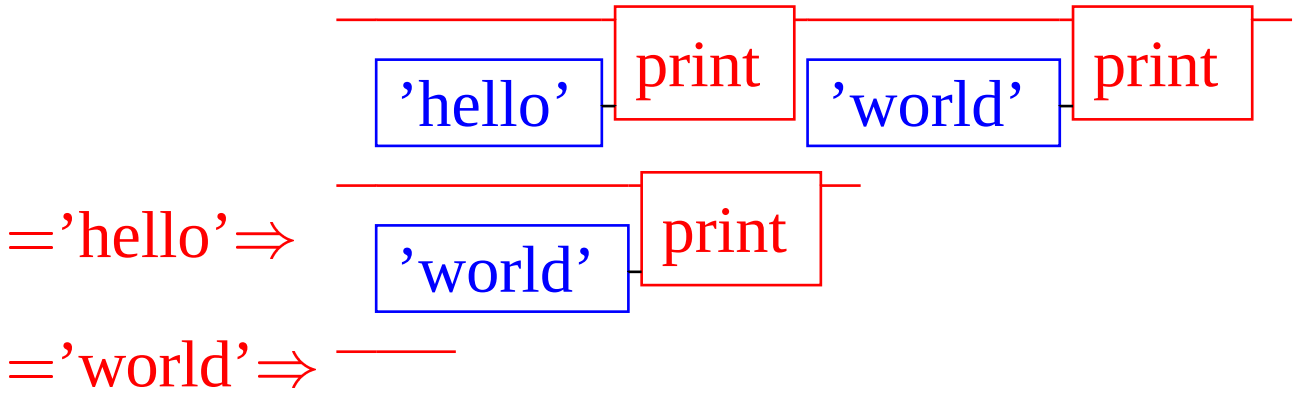
Operational semantics

For example:



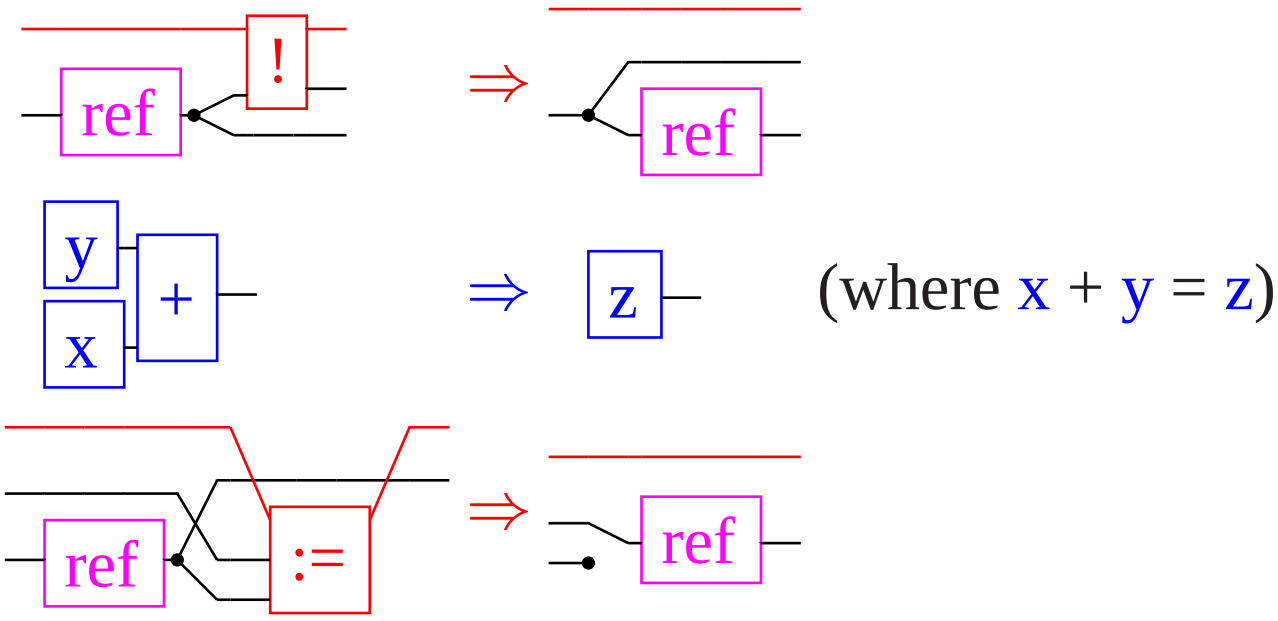
Operational semantics

As another example, we can allow 2-cells labelled with strings, so we can model a language with print statements, for example:



Operational semantics

In each case, the operational semantics can be given by taking the free premonoidal pre-2-category given by some generators together with some primitive reductions, for example:



Conclusions and future work

We have shown that the appropriate categories of flow graphs up to bisimulation are the intial categories with certain structure.

As a corollary, providing a data flow / control flow semantics for a language is exactly the same as providing a semantics in this categorical framework.

We have presented some tentative notions of higher-order graph and graphical operational semantics, similar to Milner's. These need future work.

This semantics has been implemented as a Java applet (which drew all of the graphs in this talk).

http://www.cogs.susx.ac.uk/users/alanje/premon/