A Functional Programming Language for Interaction Nets

14th International Workshop on Graph Computation Models, 2023

Marc Thatcher

m.thatcher@sussex.ac.uk

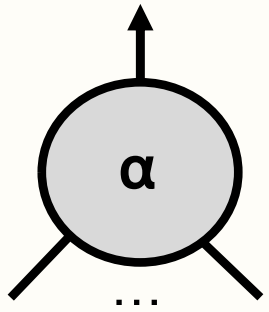
Department of Informatics

UNIVERSITY OF SUSSEX

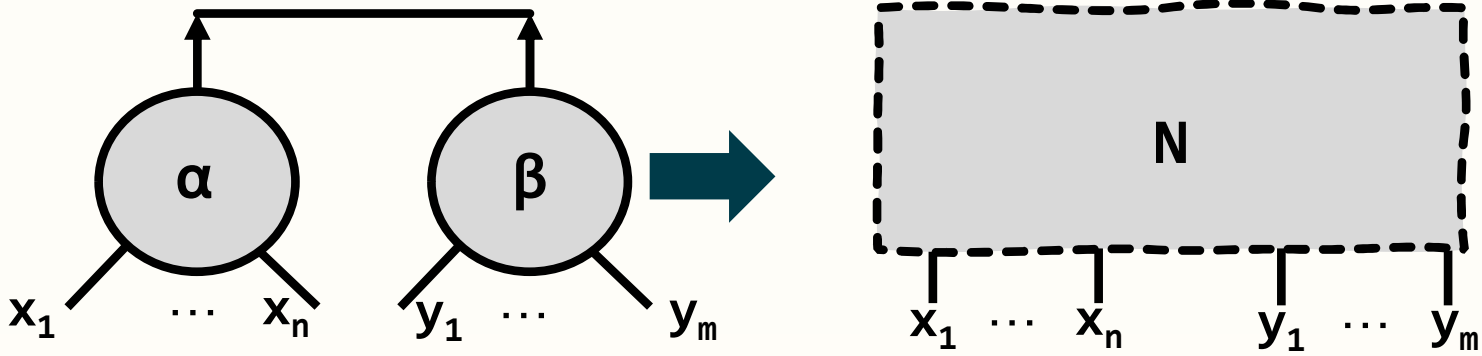
Interaction nets

Graph rewriting system (Lafont, 1990)

Finite set nodes (“agents”):



Finite set of rewrite rules:



At most one rule per agent pair.

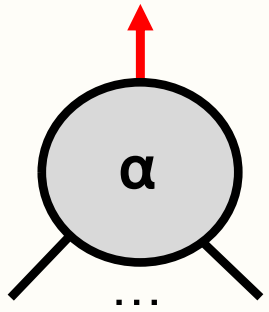
Interface preserved.

UNIVERSITY OF SUSSEX

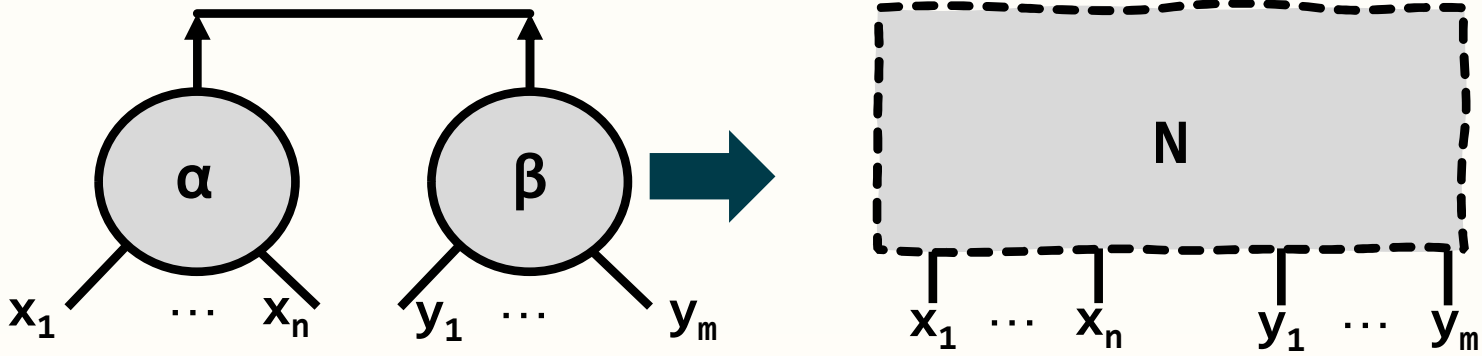
Interaction nets

Graph rewriting system (Lafont, 1990)

Finite set nodes (“agents”):



Finite set of rewrite rules:



At most one rule per agent pair.

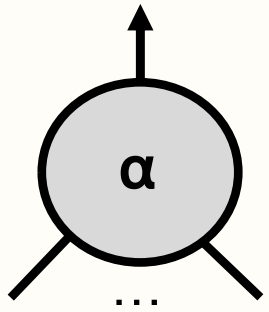
Interface preserved.

UNIVERSITY OF SUSSEX

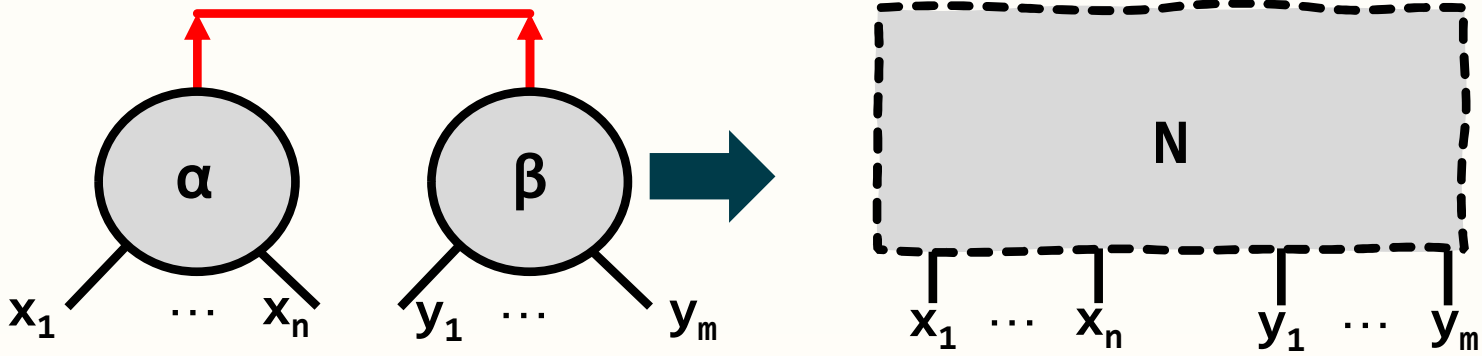
Interaction nets

Graph rewriting system (Lafont, 1990)

Finite set nodes (“agents”):



Finite set of rewrite rules:



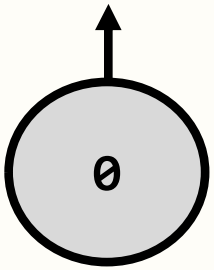
At most one rule per agent pair.

Interface preserved.

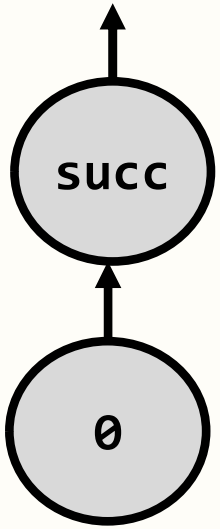
UNIVERSITY OF SUSSEX

Unary numbers

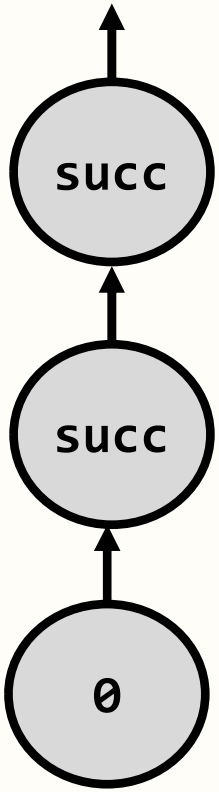
Zero



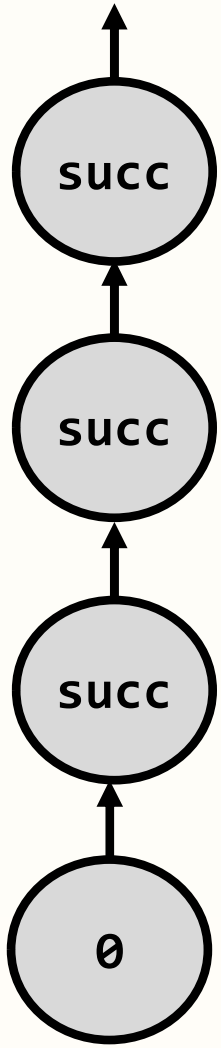
One



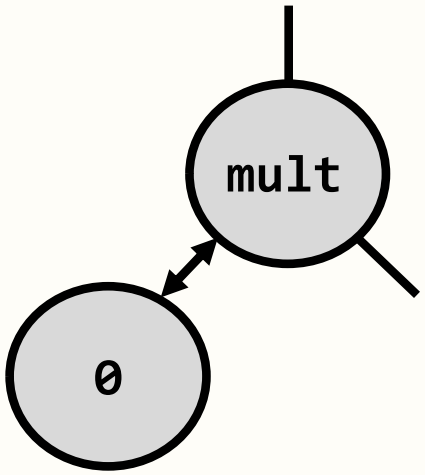
Two



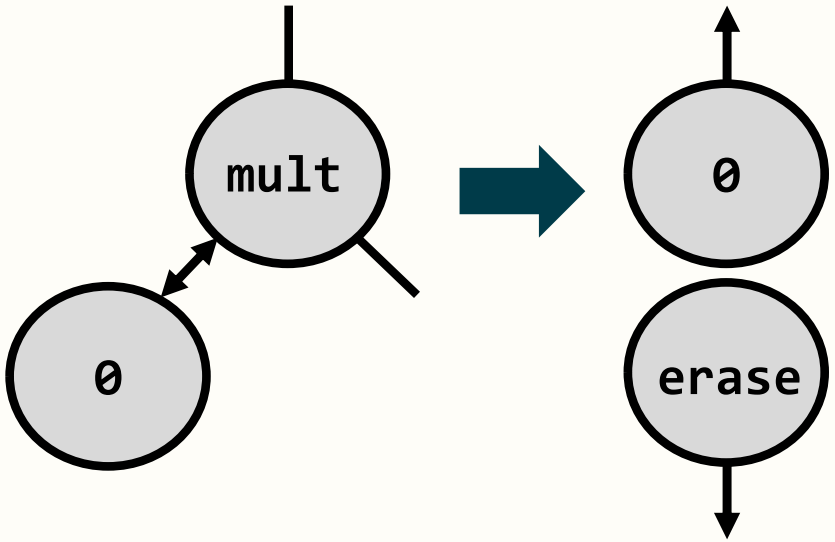
Three



Unary multiplication

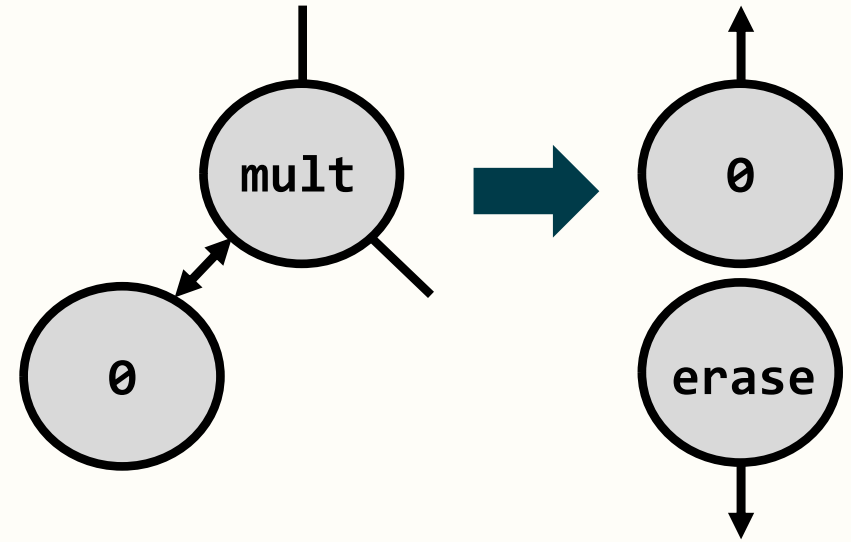


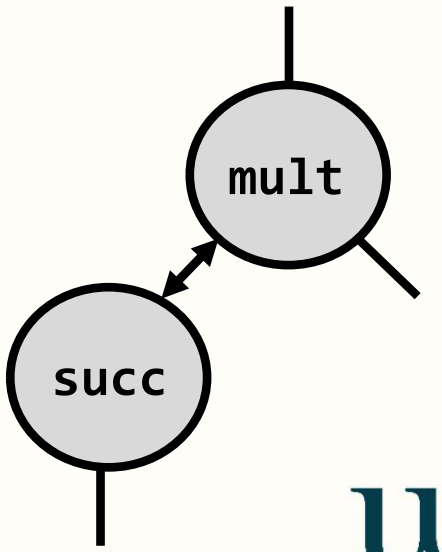
Unary multiplication



UNIVERSITY OF SUSSEX

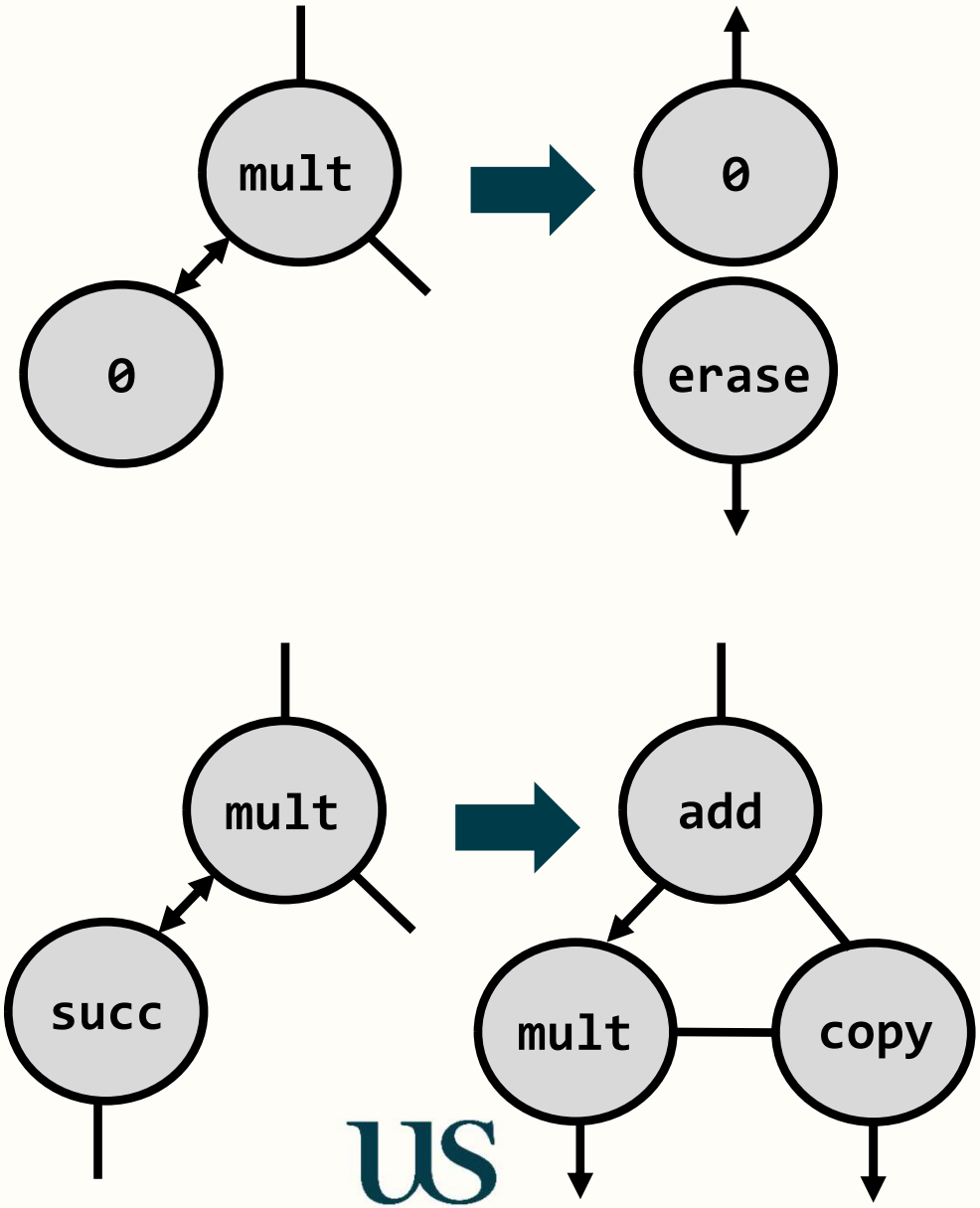
Unary multiplication





UNIVERSITY OF SUSSEX

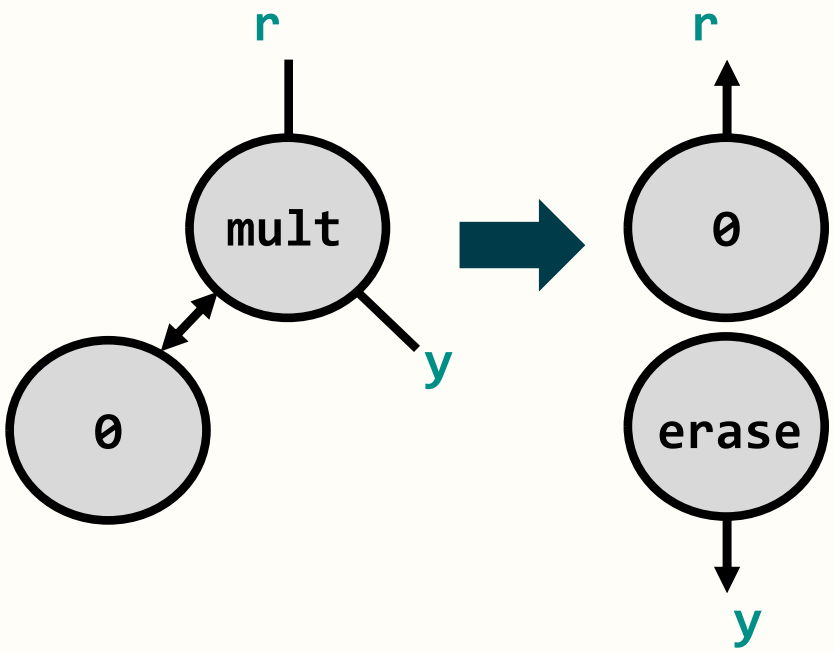
Unary multiplication



UNIVERSITY OF SUSSEX

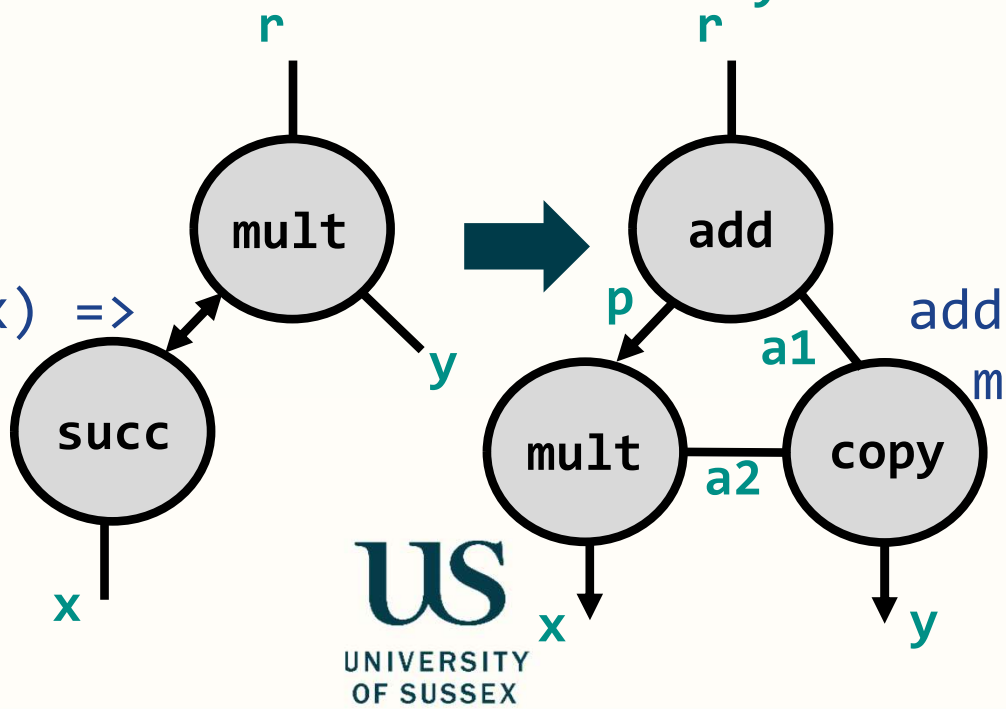
Explicit textual description

mult(r,y)><0 =>



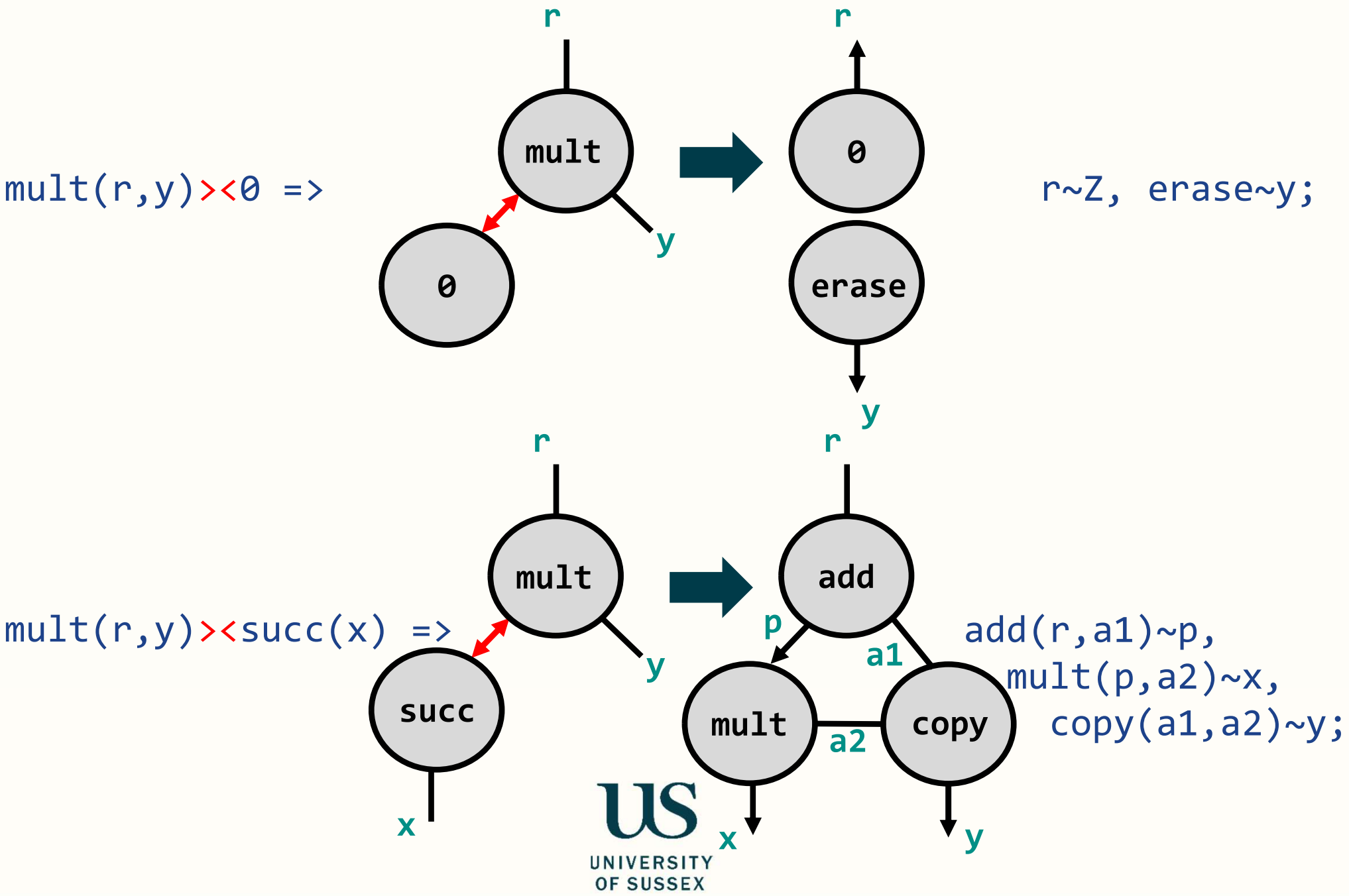
r~Z, erase~y;

mult(r,y)><succ(x) =>



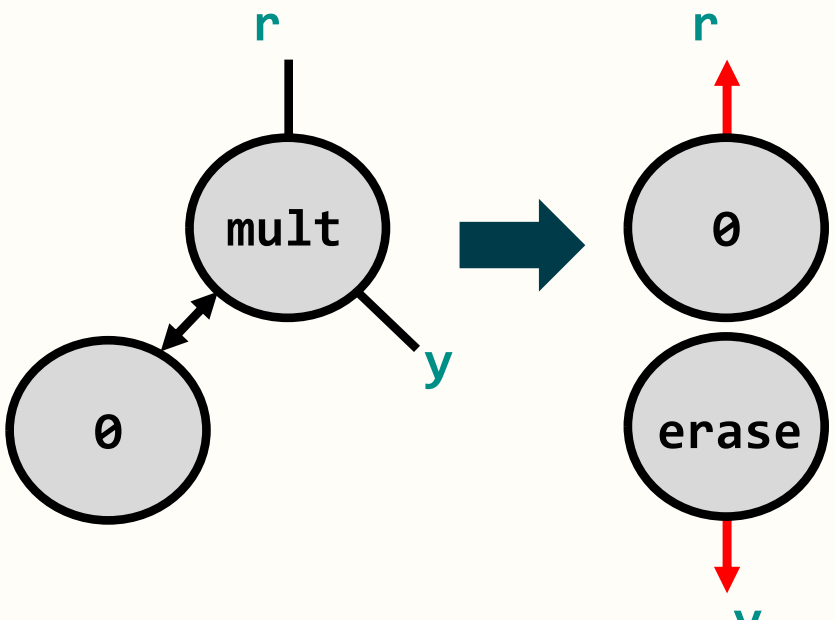
add(r,a1)~p,
mult(p,a2)~x,
copy(a1,a2)~y;

Explicit textual description



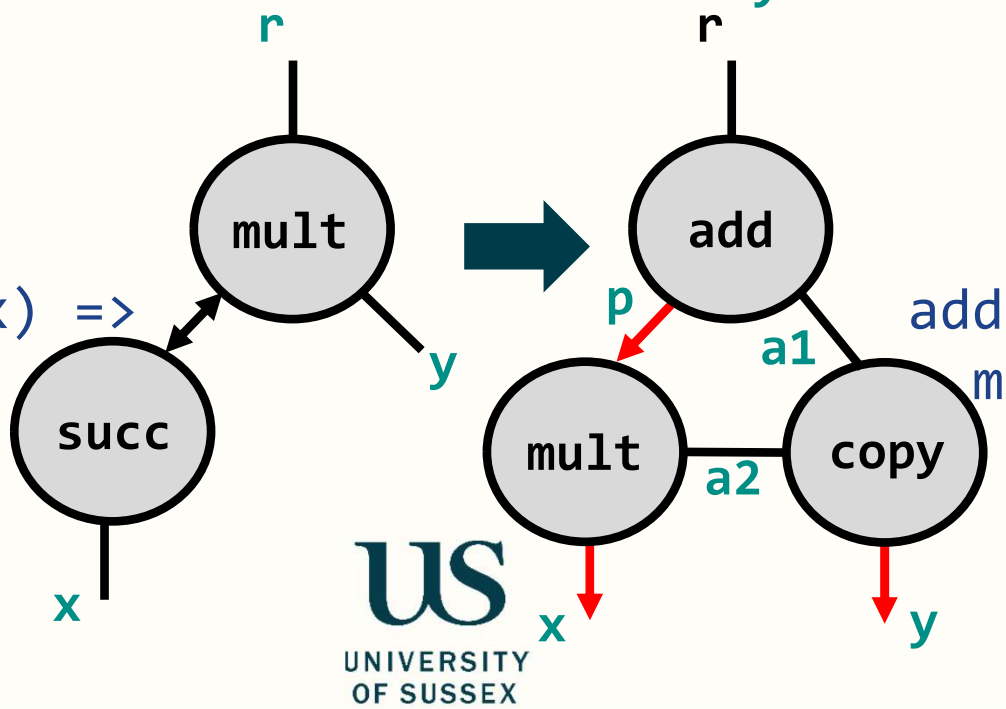
Explicit textual description

mult(r,y)><0 =>



r~Z, erase~y;

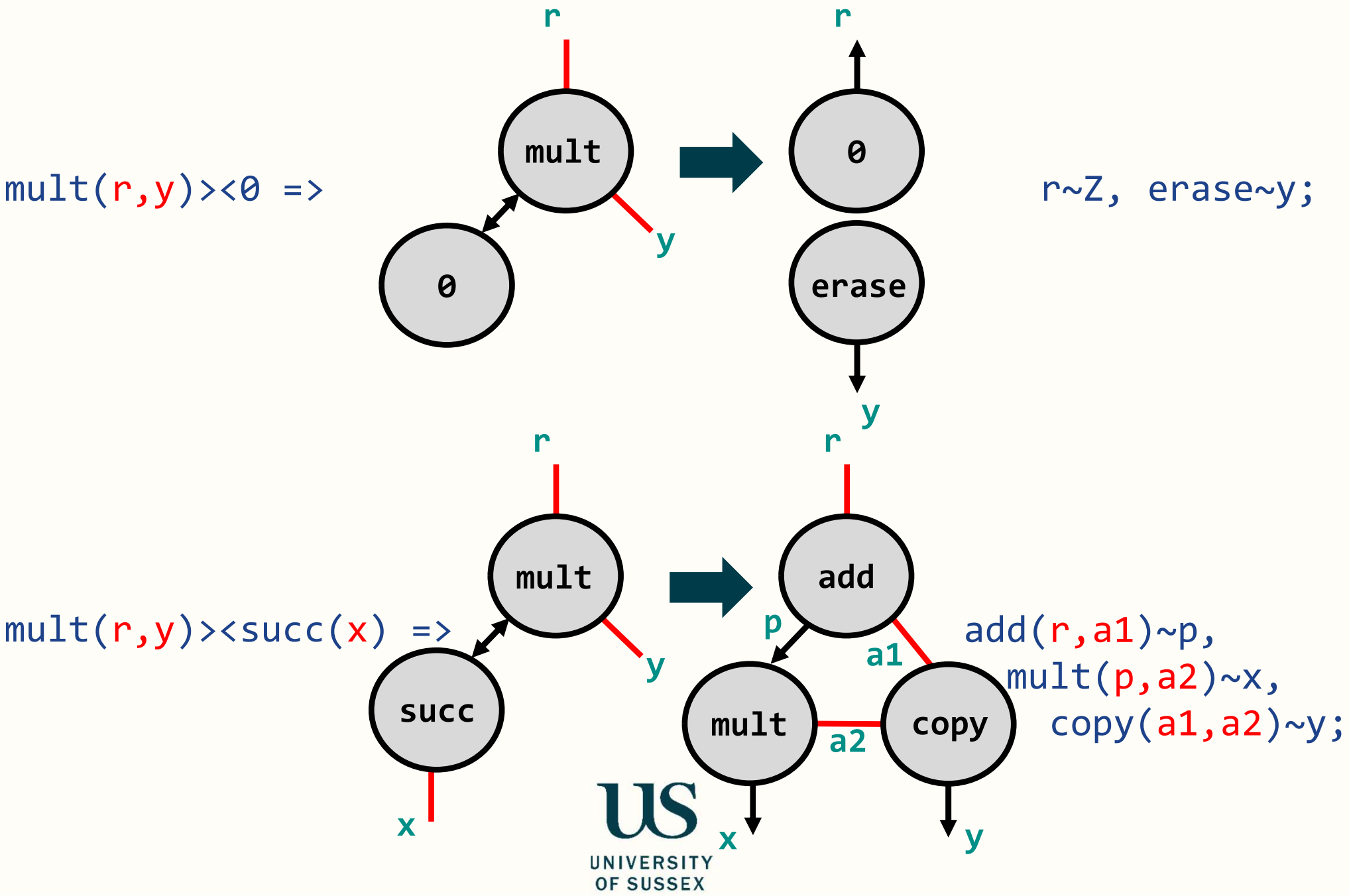
mult(r,y)><succ(x) =>



add(r,a1)~p,
mult(p,a2)~x,
copy(a1,a2)~y;

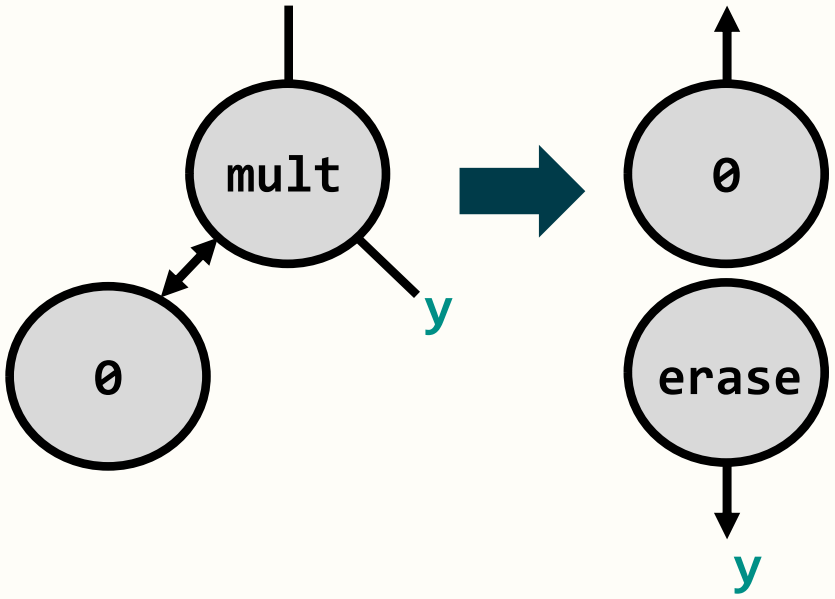
UNIVERSITY OF SUSSEX

Explicit textual description



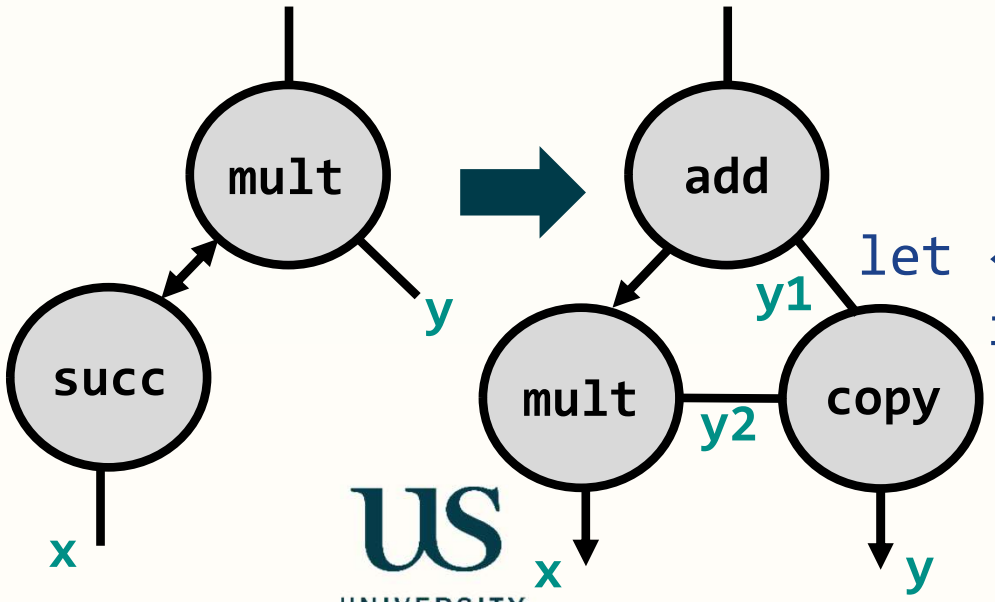
Functional style description

mult(0,y)=



0 {erase~y}

mult(Succ(x),y)=

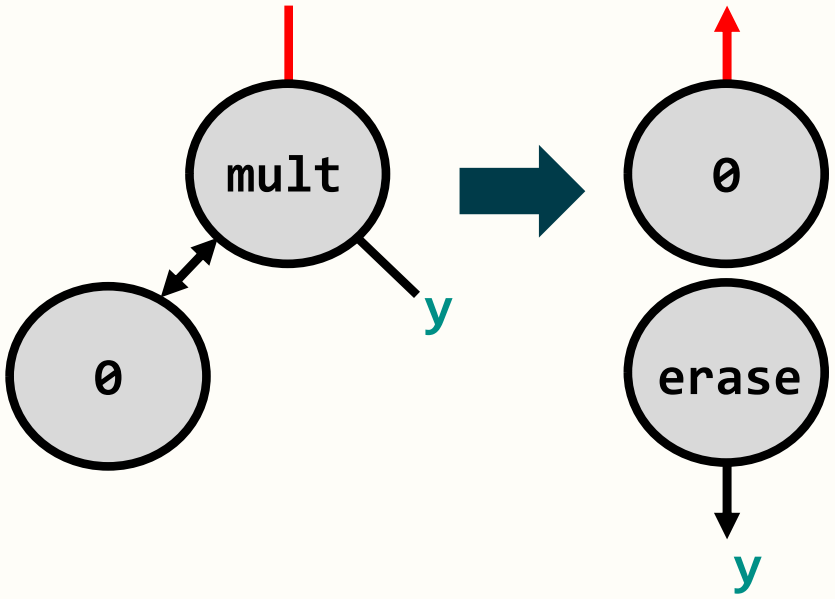



```
let <y1,y2>=copy(y)
in add(
     mult(x,y2),
          y1))
```

UNIVERSITY OF SUSSEX

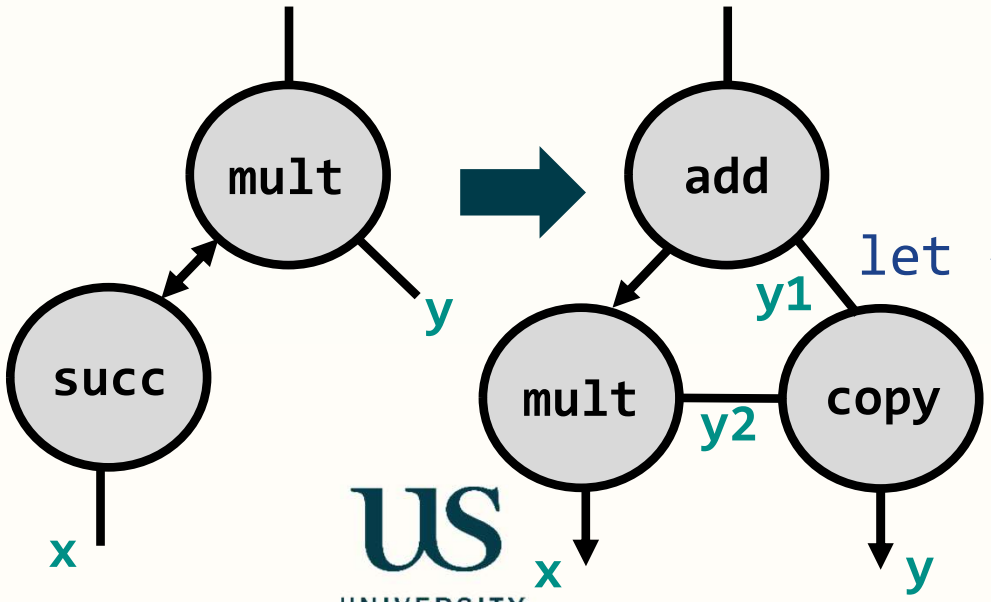
Functional style description

mult(0,y)=



0 {erase~y}

mult(Succ(x),y)=

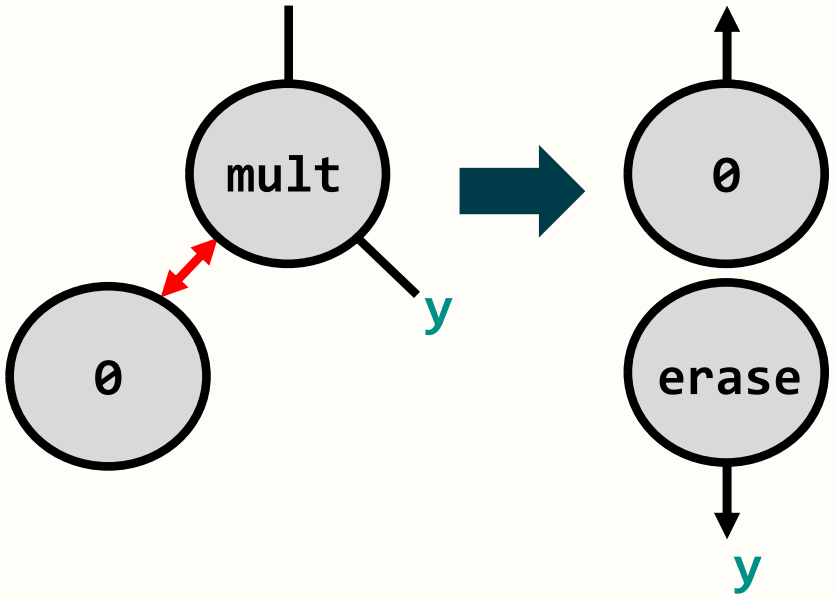



```
let <y1,y2>=copy(y)
in add(
     mult(x,y2),
          y1))
```

UNIVERSITY OF SUSSEX

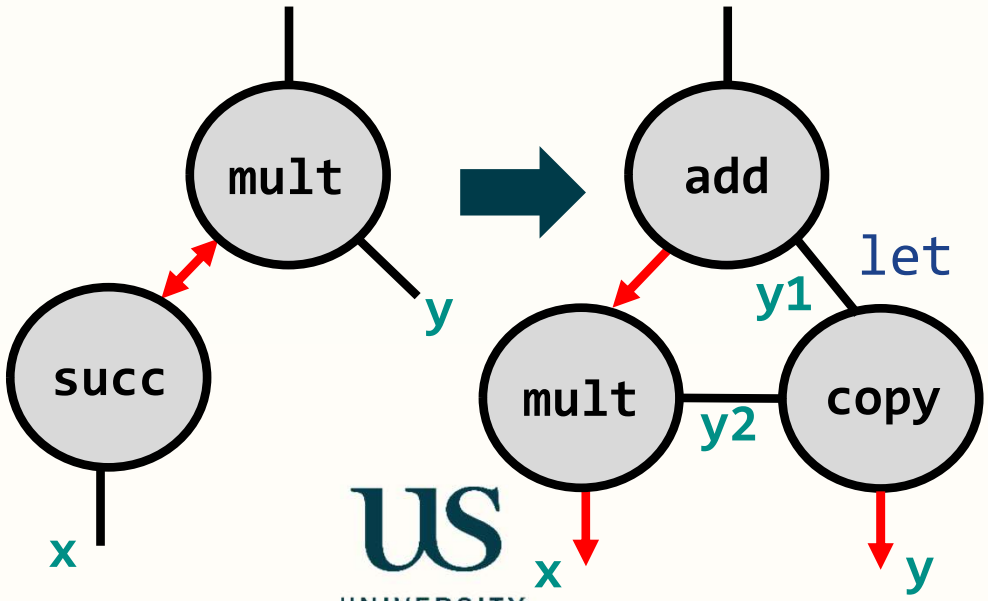
Functional style description

```
mult(0,y)=
```




```
0 {erase~y}
```

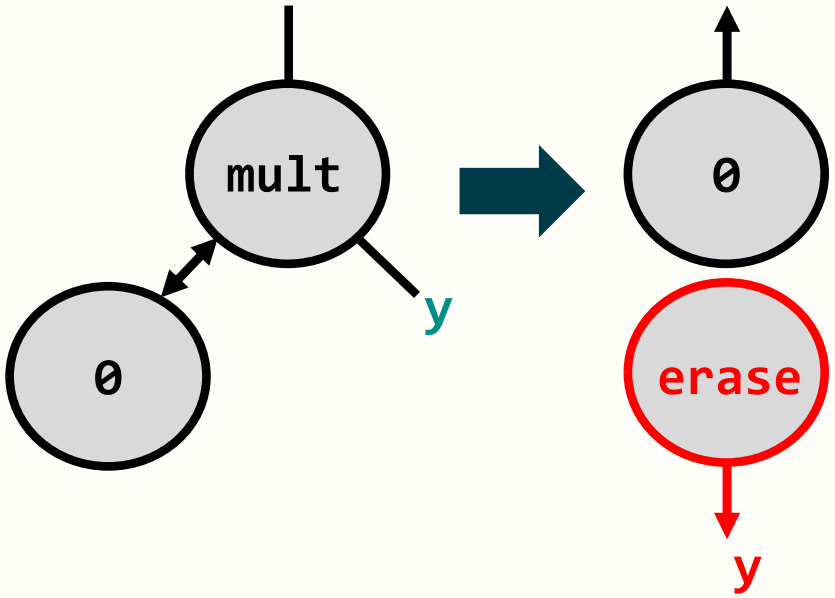
```
mult(Succ(x),y)=
```




```
let <y1,y2>=copy(y)
in add(
    mult(x,y2),
          y1))
```

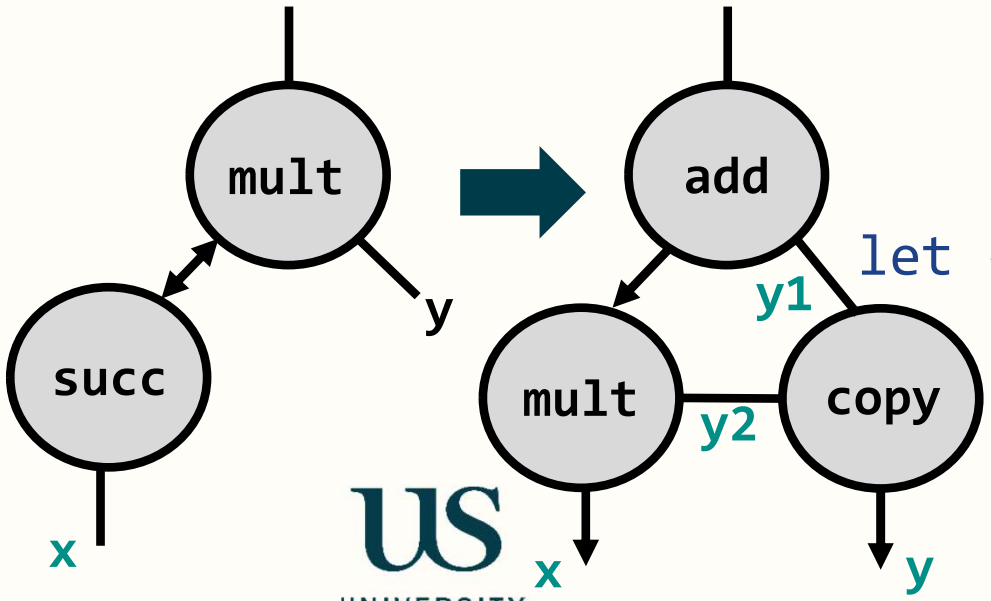
Functional style description

mult(0,y)=



0 {erase~y}

mult(Succ(x),y)=




```
let <y1,y2>=copy(y)
in add(
     mult(x,y2),
          y1))
```

UNIVERSITY OF SUSSEX

Further examples

```
add(0,y)          = y
add(Succ(x),y)    = add(x,add(x,y))

len(Nil)          = 0
len(Cons(x,xs))= Succ(len(xs)) {erase~x}

bubble(Nil)               = Nil
bubble(Elem|x|,list)      = bx|x|(list)
bx|x|(Nil)                = Elem|x|(Nil)
bx|x|(list)               = bxy|x,y|(list)
bxy|x,y|(Nil)
     | x<= y              = Elem|x|(Elem|y|(Elem(Nil))
     | _                  = Elem|y|(Elem|x|(Elem(Nil))
bxy|x,y|(list)
     | x<= y              = Elem|x|(bx|y|(list))
     | _                  = Elem|y|(bx|x|(list))
```

On-going work

- Lists — *[ ] [Nil, Z]*
- Integer arithmetic — *incr(Num|x|) = Num|x+1|*
- Conditional expressions — *bubble(x:xs) | Num(n) = …*
- Templates / polymorphism — *copy(_y) = <_y, _y>*
- Modules — *import sort as …*
- Types — *mult :: nat x nat → nat*

UNIVERSITY OF SUSSEX

A Functional Programming Language for Interaction Nets

14th International Workshop on Graph Computation Models, 2023

Marc Thatcher

m.thatcher@sussex.ac.uk

Department of Informatics

UNIVERSITY OF SUSSEX