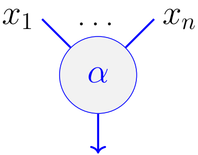
Parallel Functional Programming with Interaction Nets

Marc Thatcher
m.thatcher@sussex.ac.uk
Department of Informatics, University of Sussex

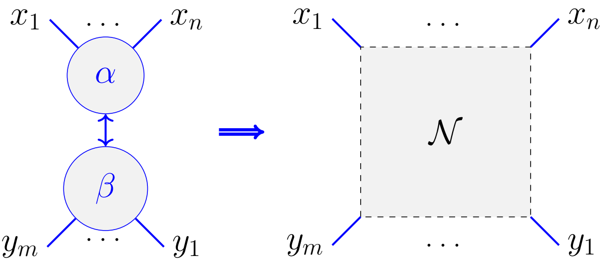
40th British Colloquium for Theoretical Computer Science
University of Bath, April 4th–5th 2024

Interaction nets (Lafont,1990)

Finite set of *user-defined* agents:

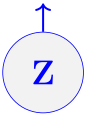


Finite set of *user-defined* rewrite rules:

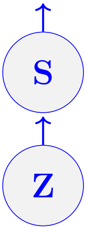


Example - Unary numbers

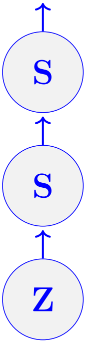
Zero



One

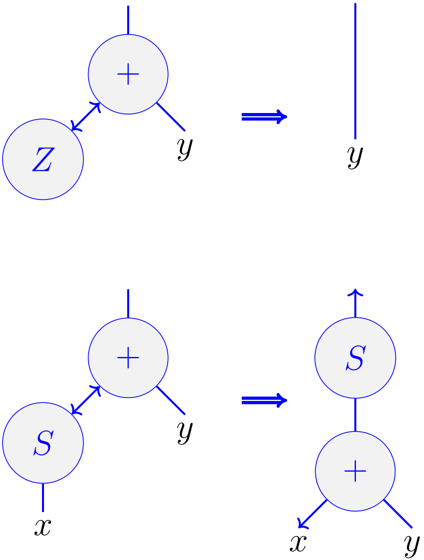


Two

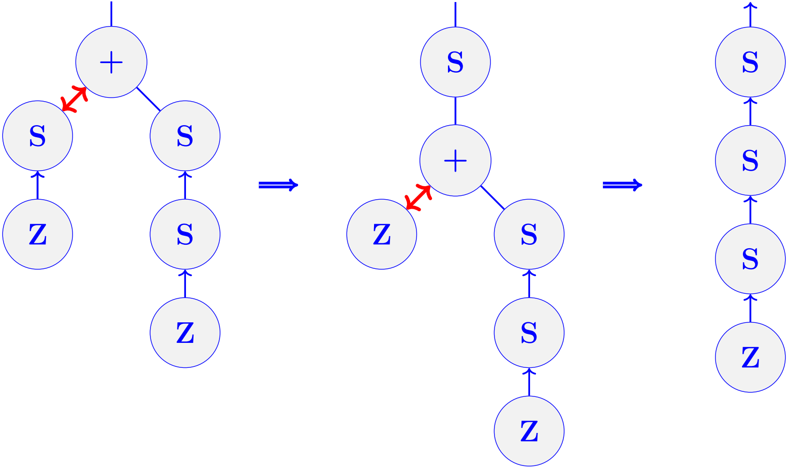


etc. . . .

Example - Unary number addition



Example function - Unary number addition



Properties as a programming language

- Turing complete
- Pattern matching
- Constant time rewrites
- Visual debugging

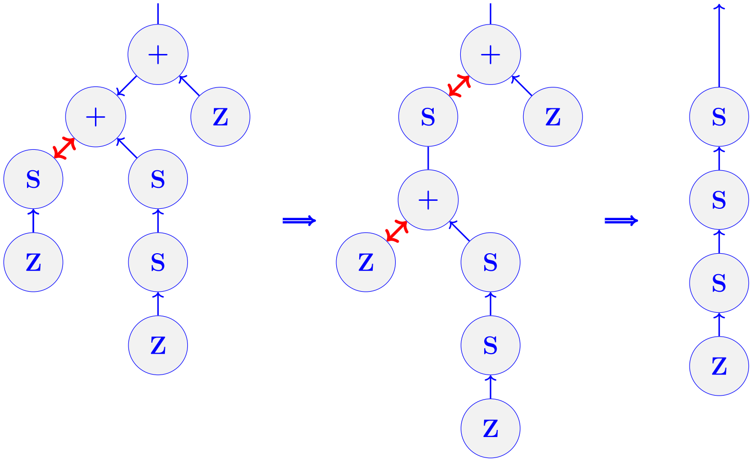
- Local reductions ; Shared computations
- One-step confluence

$$\begin{array}{ccccc} & & N & & \\ & \swarrow & & \searrow & \\ N_1 & & & & N_2 \\ & \searrow & & \swarrow & \\ & & N' & & \end{array}$$

- Explicit mandatory memory management – **no GC!**

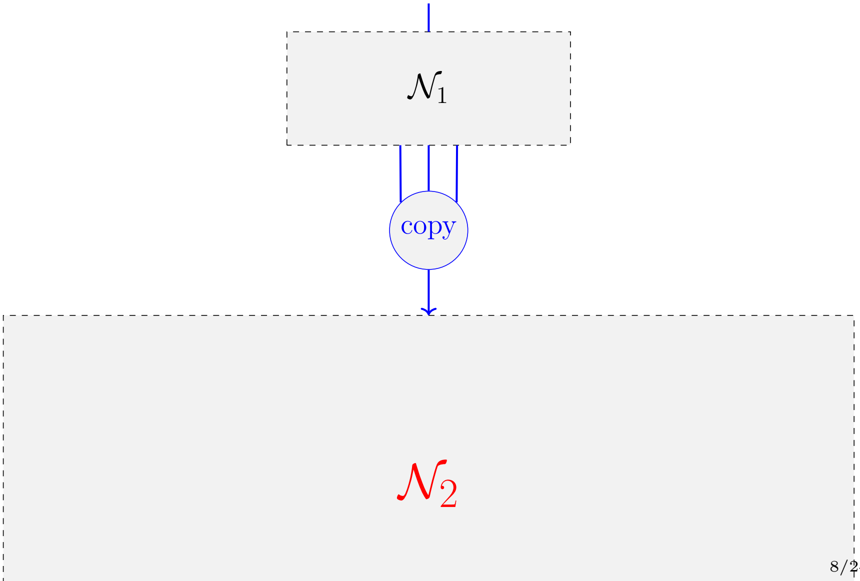
$\longrightarrow$ **Natural parallel execution**

Parallel evaluation

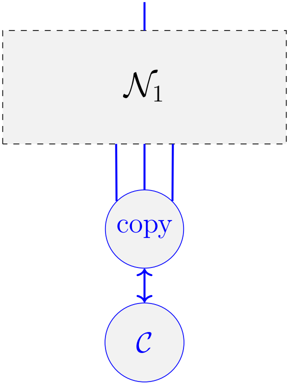


Sharing

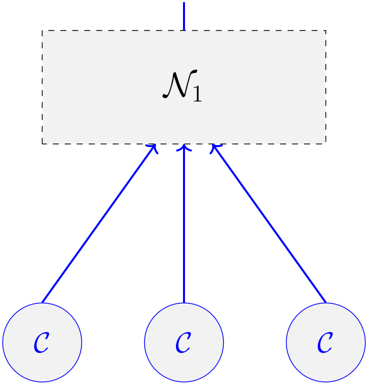
Assume $\mathcal{N}_2 \to^* \mathcal{C}$



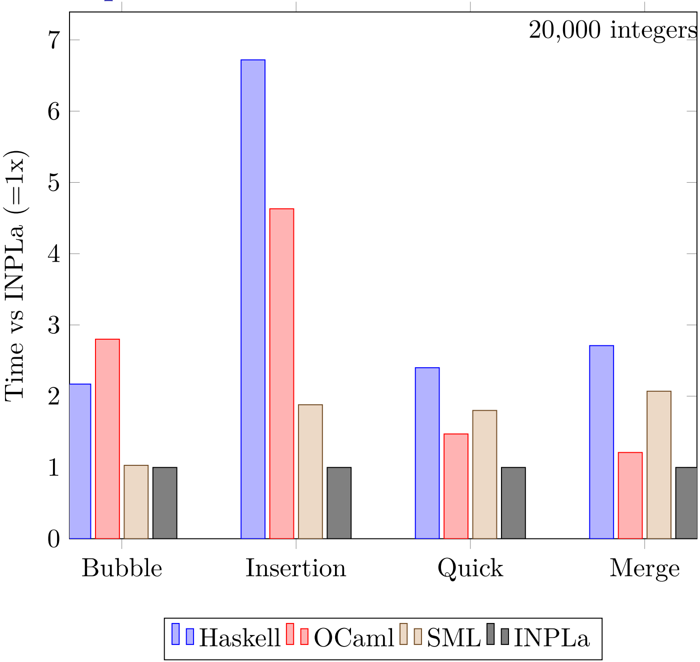
Sharing



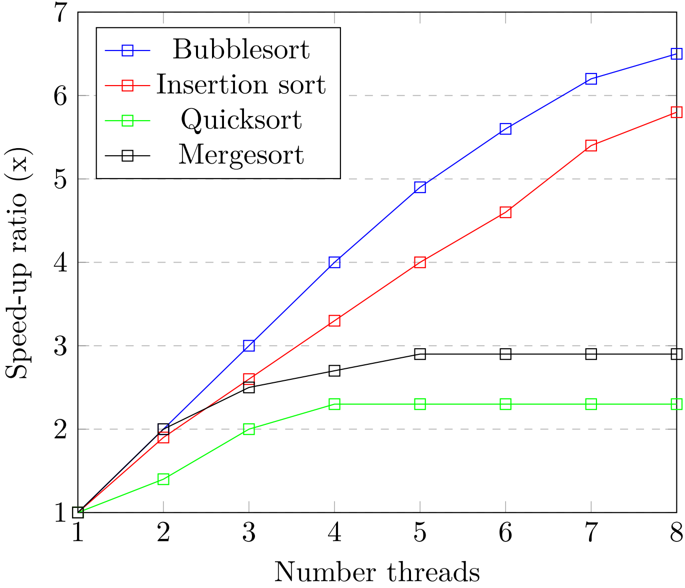
Sharing



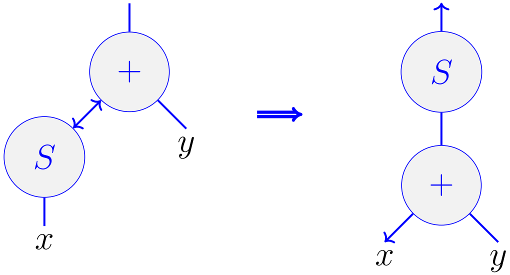
Impact of parallelism - benchmark results



Impact of parallelism - benchmark results



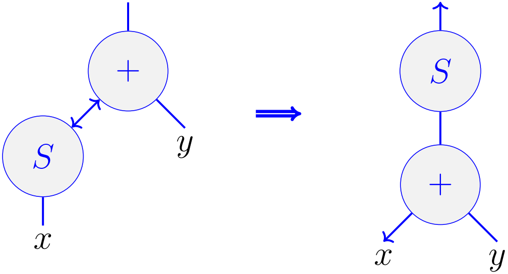
Towards a programming language v.1[1]



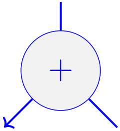

```
add(result,y)><S(x) => result~S(aux), add(aux,y)~x
```

[1] Sato, 2014 ; https://github.com/inpla/inpla

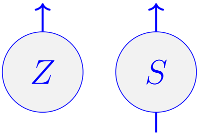
Towards a programming language v.2



Functions:

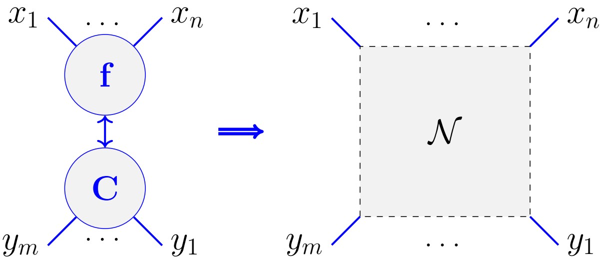


Constructors:



FLIN - a Functional Language for Interaction Nets[2]

If f is a function and C is a constructor:



then:

`f(C(`$\vec{y}$`),`$\vec{x}'$`) = `$N(\vec{x}',\vec{y})$

where:

`N = f ...| C ...|` $\vec{y}$ `| ...`

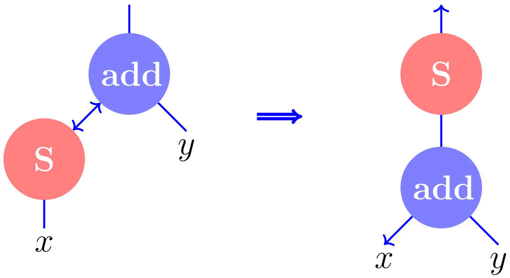
and

$\vec{x}' = \vec{x}$ adjusted for output.

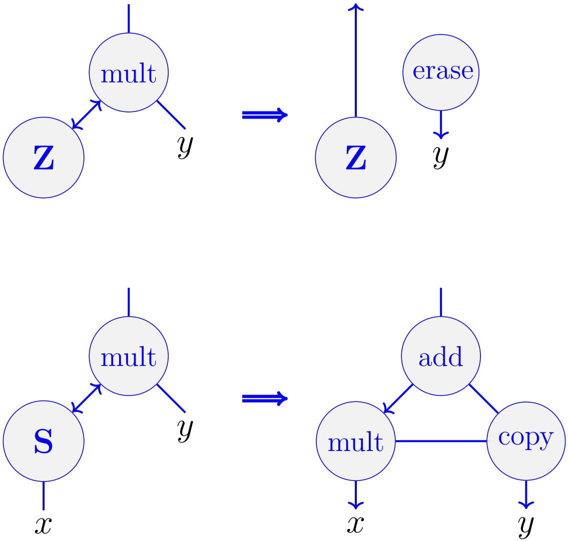
[2] https://github.com/inpla/train

FLIN ≅ Interaction Nets

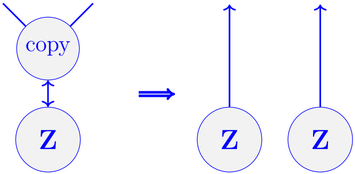
```
add S (x, y) = S (add (x, y))
```



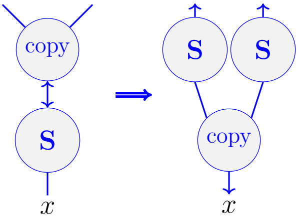
Non-functional interaction net rules



FLIN syntax for non-functions

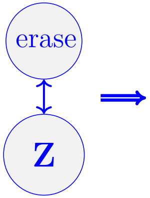



```
copy Z = (Z,Z)
```

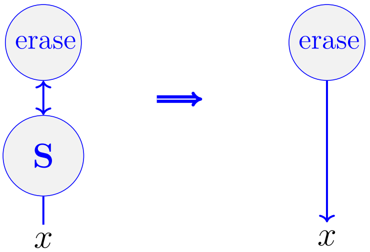



```
copy S(x) = let (x1,x2) = copy x in (S(x1),S(x2))
```

FLIN syntax for non-functions




```
{erase><Z => }
```



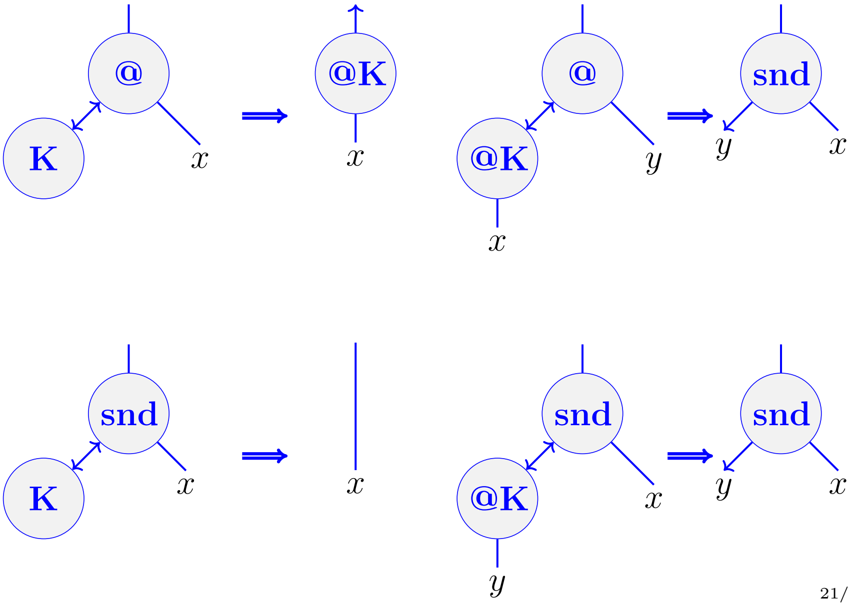

```
{erase><S(x) => erase~x}
```

FLIN examples

```
mult (Z,y)    = (Z,{erase~y})
mult (S(x),y) = let (y1,y2)=dup y in
                    add (y1,(mult (x,y2)))


---
mult' (Z,y)    = snd(y,Z)
snd   (Z,x)    = x
snd   (S(y),x) = snd(y,x)
---


fib Z     = Z
fib S(x)  = fibS x
fibS Z    = S(Z)
fibS S(x) = let (x1,x2)=dup x in
                add ((fibS x1),(fib x2))

append ([],ys)     = ys
append ((x:xs),ys) = x:(append (xs,ys))
```

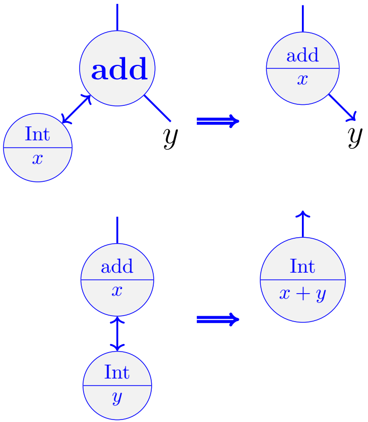
Computational power of Functional Interaction Nets

$\mathbf{K}xy = x$



Extension - Attributes

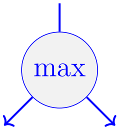
Hold values within agents - ints, bools, strings etc. & tuples of. (Fernández, Mackie, Pinto 2001)



cf. λ-calculus $\rightarrow$ PCF.

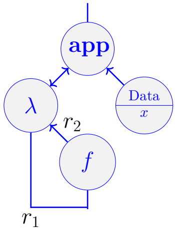
Extensions – further work

Multiple principal ports

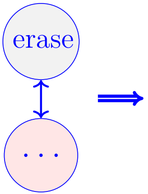


INMPP (Alexiev, 1999);
Macros (Sinot, Mackie 2005)

Higher order functions

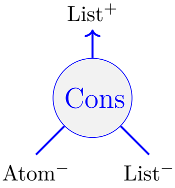


Generic rules



(Jiresch, 2012)

Type system



(Lafont, 1990); (Fernández, 1998)

Conclusions

- Interaction nets: asynchronous parallel computation.
- INPLa implementation has encouraging benchmarks.
  github.com/inpla/inpla
- FLIN - function-constructor language maps 1:1 to interaction nets.
- FLIN $\rightarrow$ INPLa transpiler.
  github.com/inpla/train
- FLIN – programming or intermediate language for more complete language.

Lots more to do!