Parallel Functional Programming with Interaction Nets

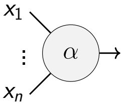
Marc Thatcher
m.thatcher@sussex.ac.uk
Department of Informatics, University of Sussex

Fun in the REPL, Bristol
1st November, 2023

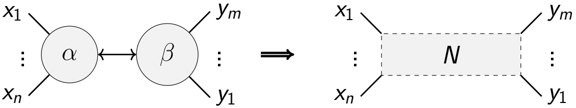
What are interaction nets?

Graph rewriting system (Lafont 1990).
"A new kind of programming language"

Finite set of *user-defined* agents:



Finite set of *user-defined* rewrite rules:



Maximum one rule per agent pair.

Set of auxiliary ports preserved.

Properties as programming language

- Turing complete
- Pattern matching
- Constant time rewrite operations
- Visual debugging

- Local reductions
- Diamond property

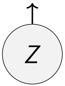
$$\begin{tikzcd} & N \arrow[dl] \arrow[dr] & \\ N1 \arrow[dr] & & N2 \arrow[dl] \\ & N' & \end{tikzcd}$$

- Explicit mandatory memory management

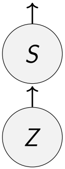
$\longrightarrow$ **Natural parallel execution**

Example constructor - Unary numbers

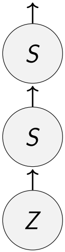
Zero



One



Two

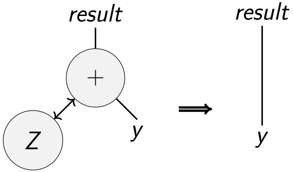


etc . . .

Example function - Unary number addition

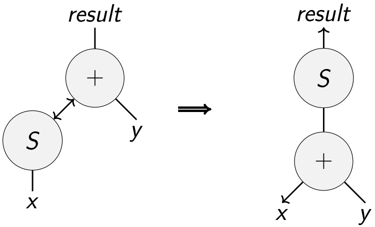
```
Z + y = y
```

```
add Z y = y
```



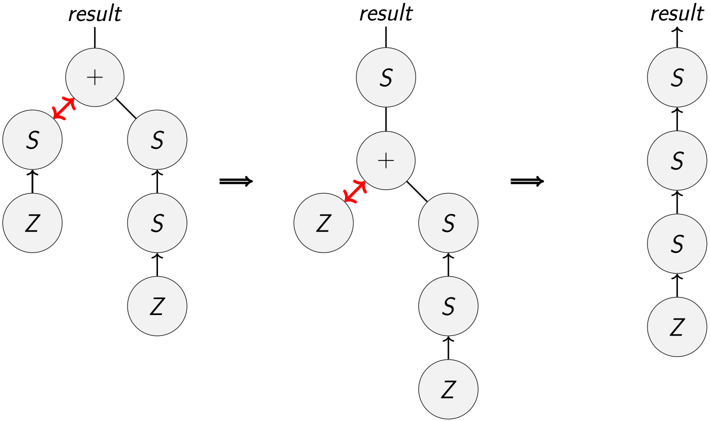
```
(S x) + y = S (x + y)
```

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

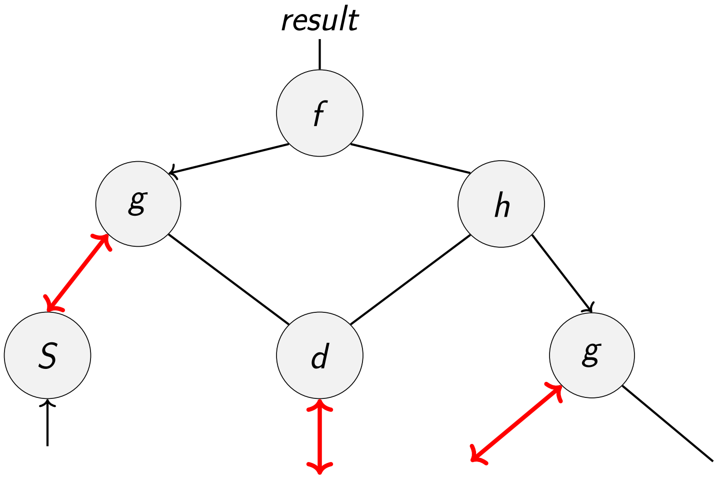


Example function - Unary number addition

$1 + 2 = 3$

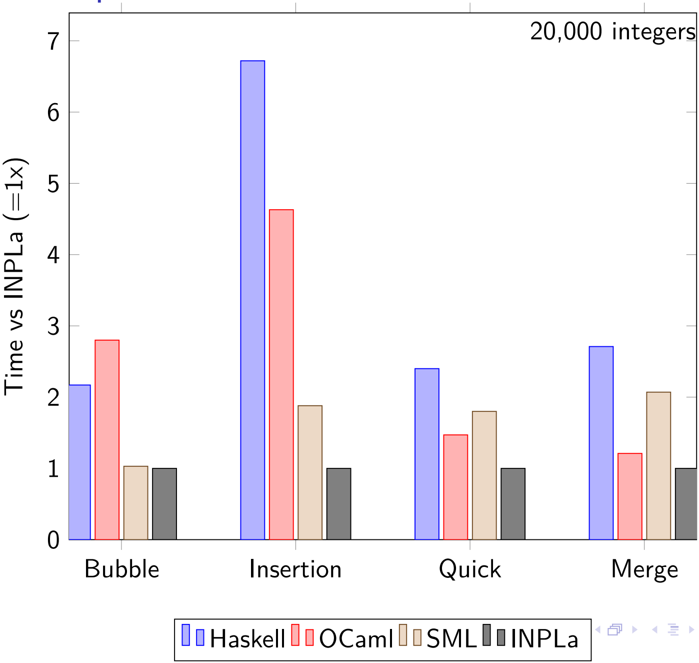


Parallel evaluation

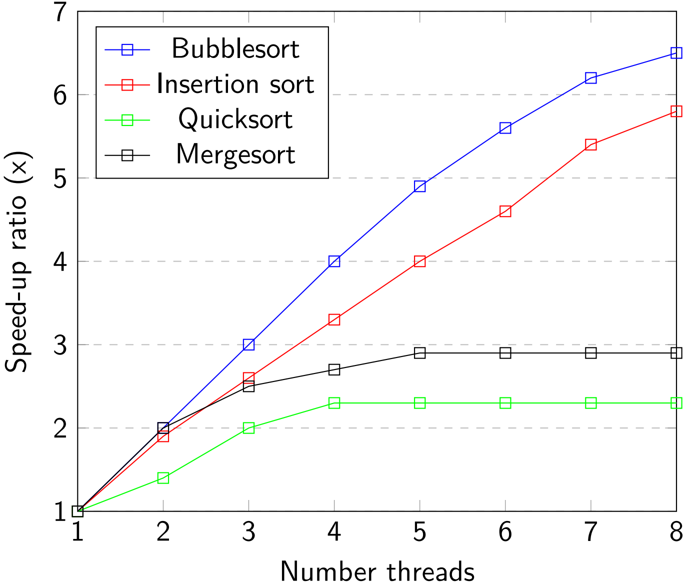


Sequential algorithm ≡ parallel algorithm.

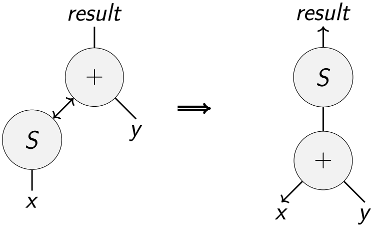
Impact of parallelism - benchmark results



Impact of parallelism - benchmark results



Towards a programming language

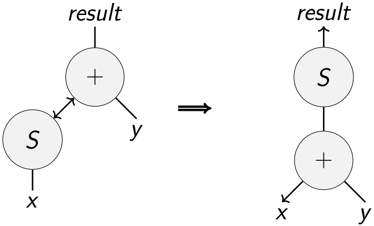


Flatten net[1]:

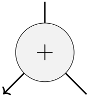
```
add(result,y)><S(x) => result~S(aux), add(aux,y)~x
```

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

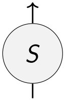
Towards a programming language



Agents whose principal port acts as input → *functions*.

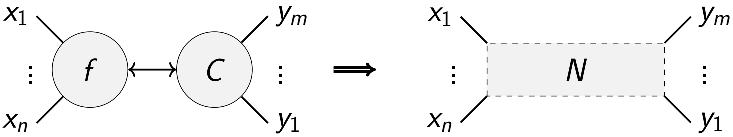


Agents whose principal port acts as output → *constructors*.



FLIN - a Functional Language for Interaction Nets

If f is a function and C is a constructor:



then:

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

where:

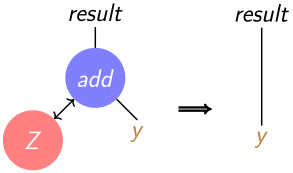
$\texttt{N = f ...| C ...| } \vec{y} \texttt{ | ...}$

and

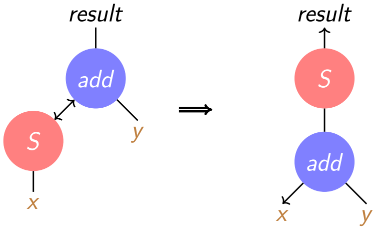
$\vec{x}' = \vec{x}$ adjusted for output.

FLIN ≅ Interaction Nets

```
add Z y = y
```

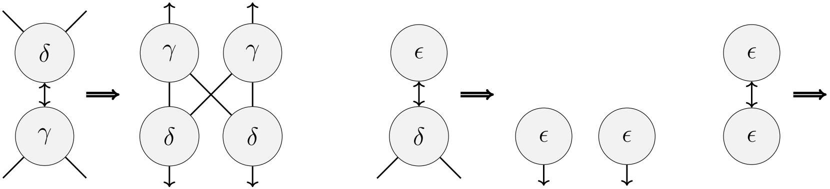


```
add (S x) y = S (add x y)
```



But not all interaction rules are functions!

Interaction combinators (Lafont 1997)



δ has two outputs.

ϵ consumes its input!

None are function-constructor systems!!

FLIN syntax for non-functions

Multiple outputs:

```
delta Z = Z,Z
delta (S x) = let x1,x2 = delta x in (S x1),(S x2)
```

No output or not function-constructor:

```
{eps><delta(a1,a2) => eps~a1, eps~a2}
{eps><eps => }
```

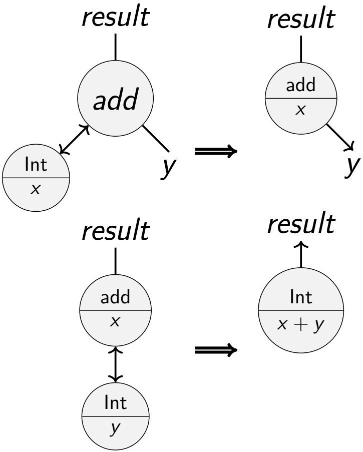
Or rewrite the algorithm!

FLIN examples

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

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

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)
```

Extension - Attributes

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



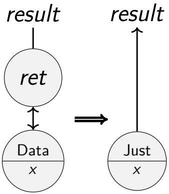
cf. λ-calculus → PCF.

Monads

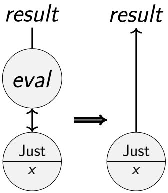
Translate: `bind m f` → `f (eval m)`

e.g. Maybe monad (following Jiresch 2010)

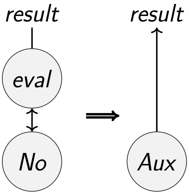
```
return Data.x = Just.x
```

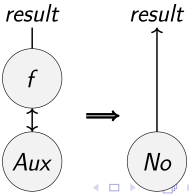


```
eval (Just x) = Just x
```



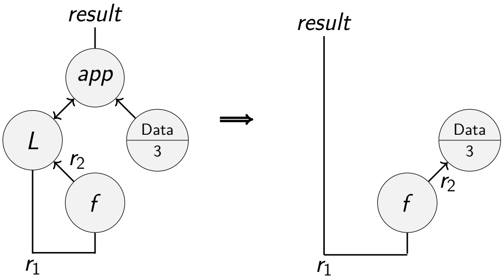
```
eval Nothing = Aux ; f Aux = Nothing
```





Higher order functions

"Package up" function in a constructor (λ-abstraction):

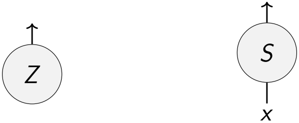


Rule: `app (L r2 r1) a = let r2=a in r1`

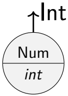
Application: `{app(result,a)><L(r2,r1), f(r1)~r2, a~Data(3)}`

Type system

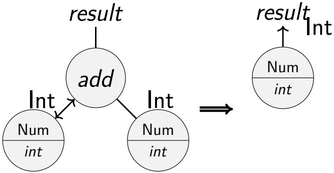
```
data Nat = Z | S Nat
```



```
type i::int => Num.i:Int
```



```
add ::  Int -> Int -> Int
```



Conclusions

- Interaction nets provide inherently parallel evaluation.
- INPLa implementation has encouraging benchmarks.
- FLIN - a simple function-constructor language maps 1:1 to interaction nets.
- FLIN can encode standard functional programming structures.
- FLIN programs run sequentially or in parallel based on resources.
- We have implemented a FLIN → INPLa transpiler.
- Language can be used directly for programming or as an intermediate language for a more complete language.