Towards a Hardware-Parallel Implementation of Interaction Nets

David Obwaller

Institute of Computer Science, University of Innsbruck

January 24, 2018

Overview

Motivation

Parallel programming is hard

- and load distribution and balancing
- data migration costs are often not taken into account, but are relevant on hardware or in a distributed setting
- desirable: automatic parallelization and load distribution

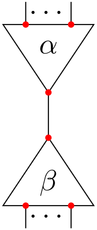
Interaction nets as candidate computation model

- parallel and asynchronous reduction
- interaction automata as refinement for hardware-like execution
- proof-of-concept implementation: ia2d

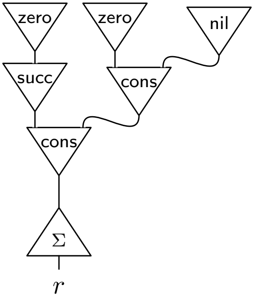
Overview

Interaction Nets

- computation model based on graph-rewriting
- an interaction net is a graph,
  vertices are called agents
- agents are labeled with symbols
- edges are called wires
- each node has a principal port as well as a number of auxiliary ports that is fixed for a given symbol
- when two agents are connected on their principal ports they are called an active pair
- unconnected ports are called free
- the set of free ports is called the interface



Example: Interaction Net



- list constructors: cons/2, nil/1
- nat constructors: succ/1, zero/0
- sum operation: sum/1

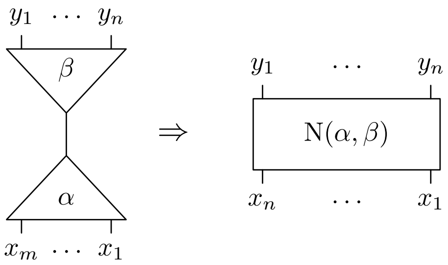
Interaction Net System

An interaction net system is a pair (S, R)

- S set of symbols (with fixed arity)
- R set of rewrite rules

Agents and rules are used to encode data and operations

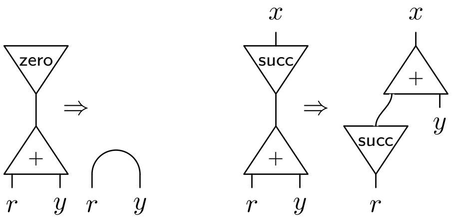
Rules



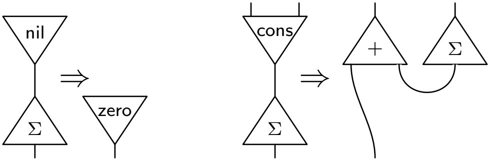
- left-hand side is an active pair of agents
- right-hand must preserve interface
- at most one rule per active pair
- if $\alpha = \beta$ the right-hand side must be top-down symmetric

Example: Rules

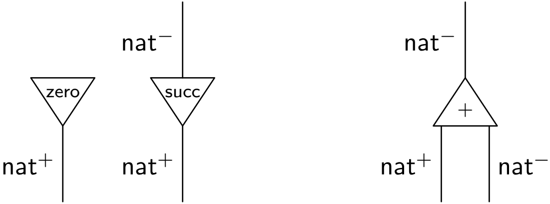
Rules for adding numbers in unary encoding:



Rules for summing up lists of numbers:

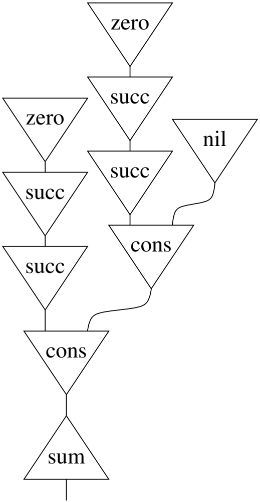


Typed Interaction Nets

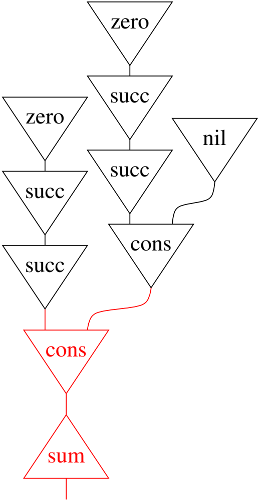


- typing ports avoids ill-formed nets
- clearly distinguishes constructors from destructors

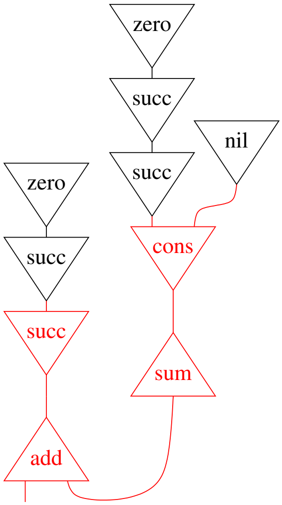
Example IN Computation



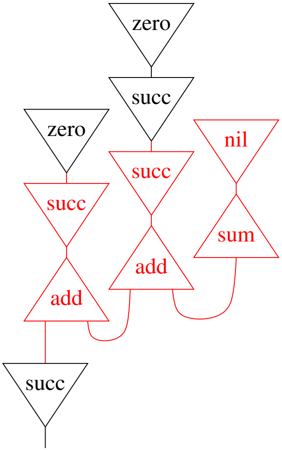
Example IN Computation



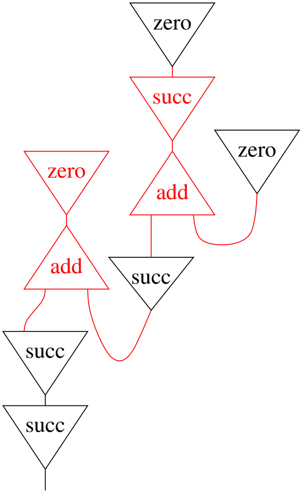
Example IN Computation



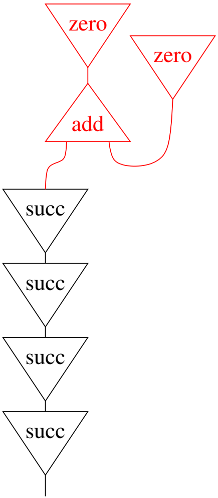
Example IN Computation



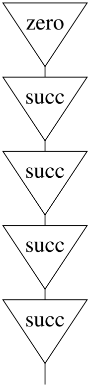
Example IN Computation



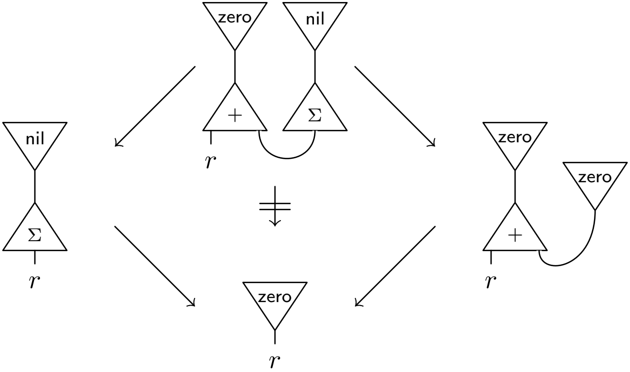
Example IN Computation



Example IN Computation



Properties



- linearity: preserves interface
- binary interaction
- no ambiguity
- → parallel/asynchronous reduction

Drawbacks

Some algorithms cannot be represented in interaction net systems

Example: parallel or

$$\begin{aligned} \text{por}(\text{True}, y) &\to \text{True} \\ \text{por}(x, \text{True}) &\to \text{True} \\ \text{por}(\text{False}, \text{False}) &\to \text{False} \end{aligned}$$

Solution:

- introduce special amb agent
- interaction nets with multiple principal ports
- $\to$ non-determinism

Overview

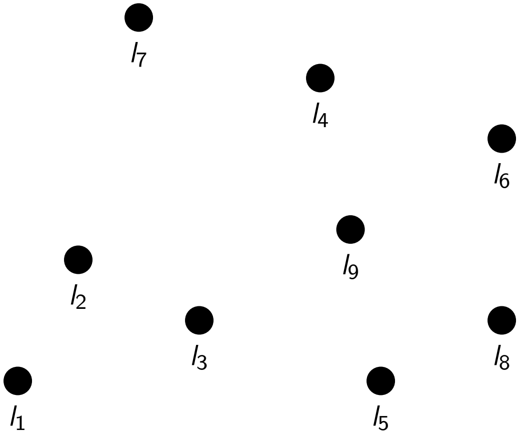
Interaction Automata

An interaction automaton $\mathcal{A}$ is a quadruple

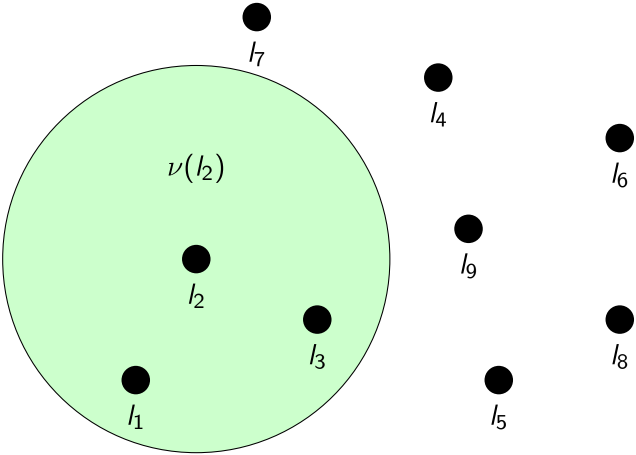
$$\mathcal{A} = (L, \nu, S, \rightarrow)$$

where

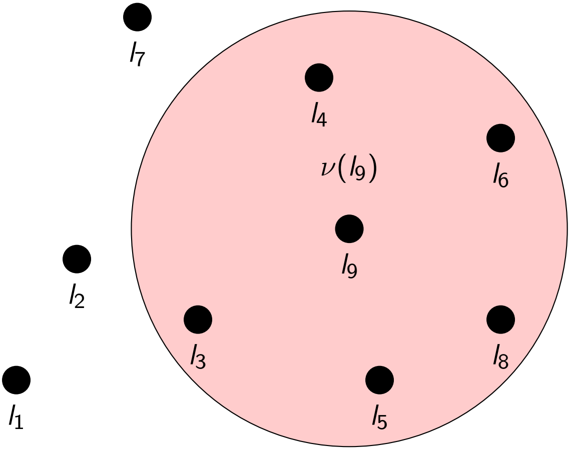
- L set of locations
- ν neighborhood
- S set of symbols
- $\rightarrow$ abstract transition



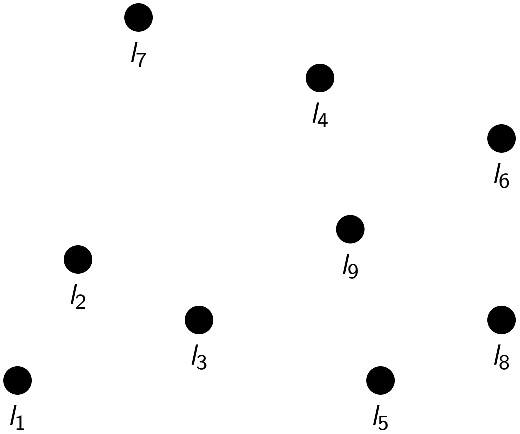
Set of locations L



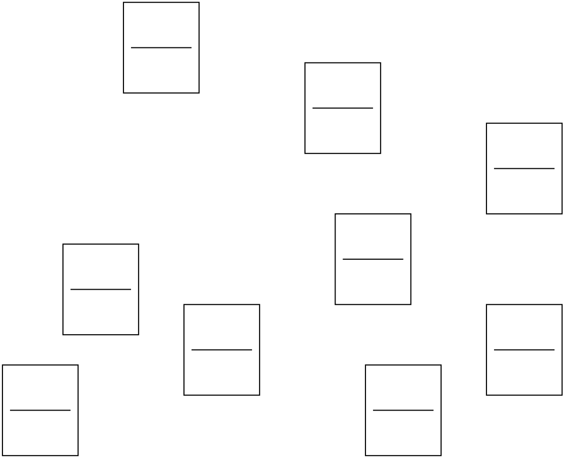
A neighborhood $\nu(l_i)$ is associated to each location



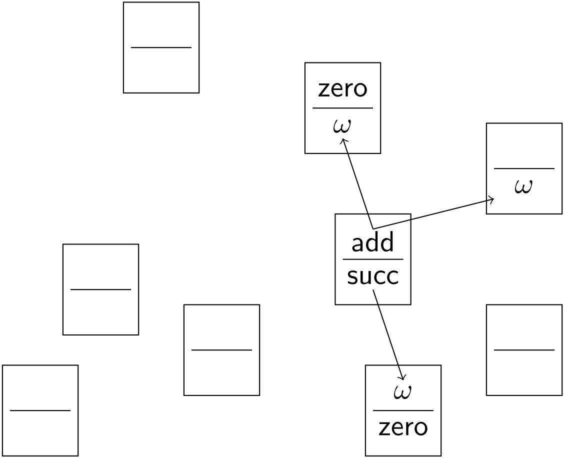
A neighborhood $\nu(l_i)$ is associated to each location



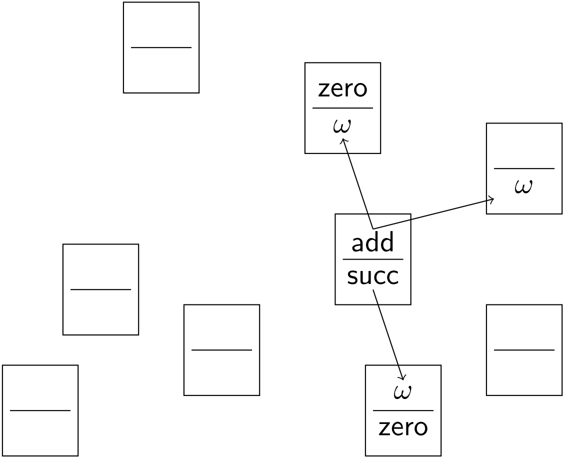
Each location may contain 0, 1 or 2 nodes



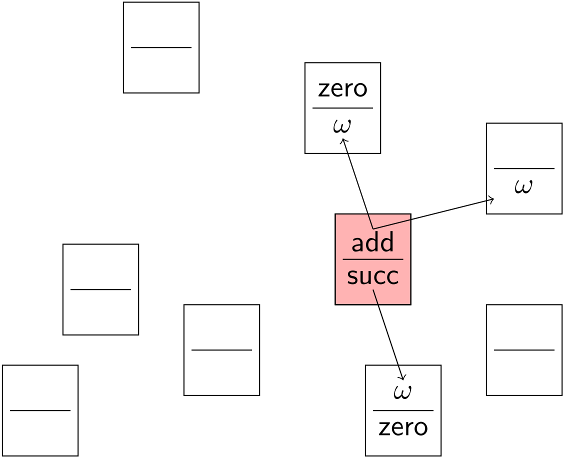
Each location may contain 0, 1 or 2 nodes



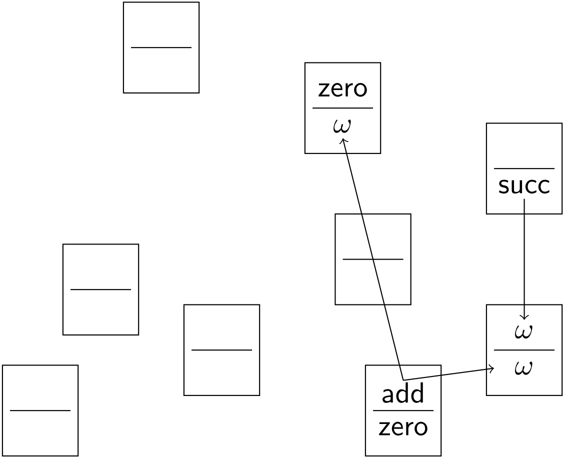
A node is either $s(l_1, \ldots, l_k)$, $s \in S$ where k is the arity of s, or ω



Abstract transition (→) between configurations



Abstract transition ($\rightarrow$) between configurations



Abstract transition ($\rightarrow$) between configurations

Interaction Automata

Properties

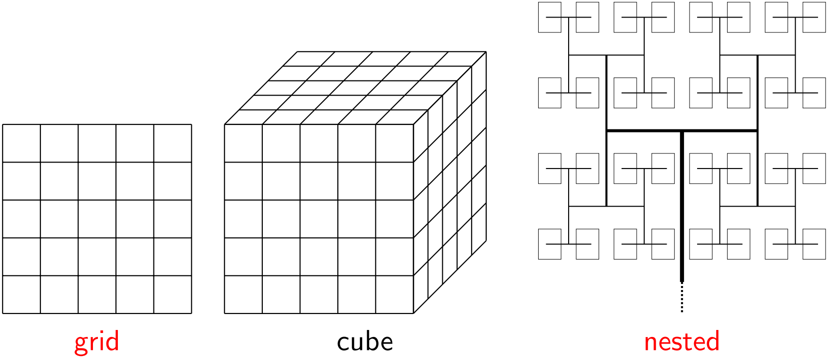
- reduction is performed locally
- allocated cells can be placed freely within the neighborhood
- the allocation strategy is unspecified in the model

ia2d interpreter

- proof-of-concept implementation with simple local allocation strategy
- the locations are layed out in a 2-dimensional grid
  $\rightarrow$ in some sense close to hardware

Demo

Memory Topologies



Simple Input Language

- takes care of non-linear functions, i.e. erasure and duplication
- supports anonymous and higher-order functions
- implemented as source-to-source compiler

Example

Haskell

```
foldr f z Nil          = z
foldr f z (Cons x xs) = f x (foldr f z xs)
```

Simple input language

```
foldr(Nil(),      f,z) = z;
foldr(Cons(x,xs),f,z) = f(x,foldr(xs,f,z));
```

Practical Considerations

- limit number of rules
  $\rightarrow$ interaction combinators: 3 symbols and 6 rules suffice to encode arbitrary interaction net systems
- standard arithmetic
  $\rightarrow$ special agents that reference value or computation
- input and output
  $\rightarrow$ singleton IO agent and corresponding operations

Overview

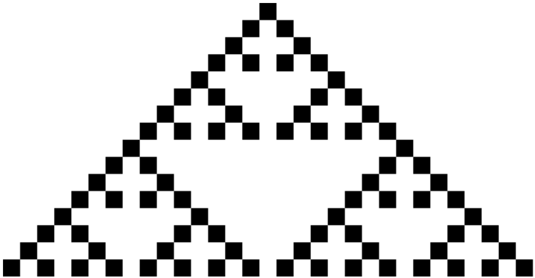
Related Systems: Cellular Automata

- interaction automata share some similarities with cellular automata (CA)
- $\sigma_i(t)$ state of cell at position i at time t
- transition function Φ

  $$\sigma_i(t+1) = \Phi(\sigma_{i-r}(t), \sigma_{i-r+1}(t), \ldots, \sigma_{i+r-1}(t), \sigma_{i+r}(t))$$

  depends on state of neighbors
- CA are synchronous parallel computational model
- Turing-complete: Rule 110

Example: Rule 90



___	__#	_#_	_##	#__	#_#	##_	###
_	#	_	#	#	_	#	_
0	1	2	3	4	5	6	7

Related Systems

Open Multi-Processing (OpenMP)

- shared memory
- parallelization by adding annotations (pragmas)

Message Passing Interface (MPI)

- distributed memory
- complete restructuring of program

Open Compute Language (OpenCL)

- run small kernels on many CPUs (GPUs)
- program-in-program

Related Systems: Data-Flow Graph

Fresh Breeze

- data-flow processor
- graph of tasks

Parallel Haskell

- Par monad

Overview

Conclusion

Proof-of-concept implementation

- parallel computation model with localized reduction on top of memory scheme with limited connectivity and locally bounded storage
- functional programs can be run on this model
- the implementation shows that reasonable memory management strategies can be implemented locally

Limits

- grid layout is limiting
- nested layout scheme seems more promising