

Compiling Interaction Nets

Abubakar.Hassan at kcl.ac.uk

Department of Computer Science
Kings College London

April 4, 2006

Outline

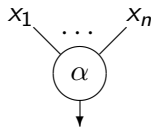
- 1 Interaction Nets Programming Language
 - Background
 - Language Syntax
 - Foreign Language Interface
 - Module System
- 2 Compilation of Interaction nets
 - Abstract Machine
 - Compilation schemes
- 3 Conclusions and Future work

Part I

Interaction Nets Programming Language

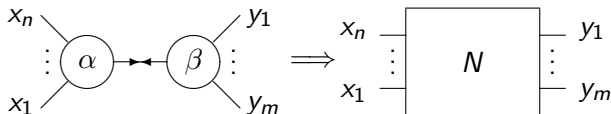
Interaction nets system

Define a set Σ of Agents



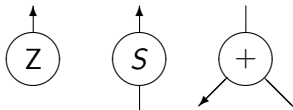
$$y = \alpha(x_1, \dots, x_n)$$

and a set \mathcal{R} of Rules

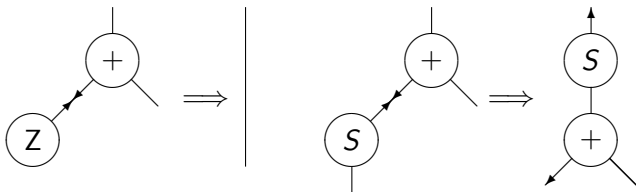


Example

$$\Sigma = \{Z, S, +\}:$$



$$x = Z() \quad y = S(a) \quad w = +(d, e)$$

 $\mathcal{R} =$


$$\begin{aligned}
 a = \text{Add}(x, y), a = Z &\implies x = y \\
 a = \text{Add}(x, y), a = S(z) &\implies x = S(b), z = \text{Add}(b, y)
 \end{aligned}$$

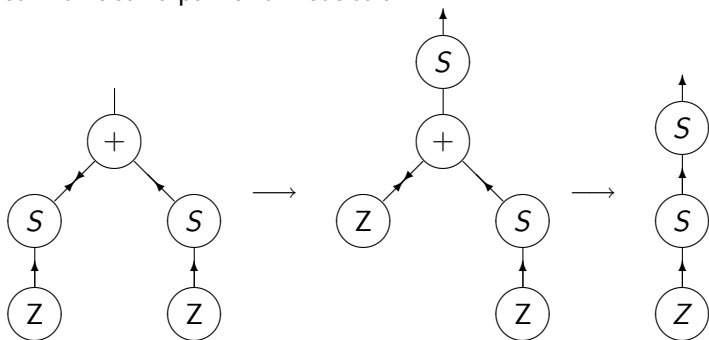
Replace equals for equals:

$$\begin{aligned}
 Z = \text{Add}(x, x) &\implies \\
 S(\text{Add}(b, y)) = \text{Add}(S(b), y) &\implies
 \end{aligned}$$

we replace the '=' by '><' and remove \implies if empty.

$$\begin{aligned}
 Z >< \text{Add}(x, x) \\
 S(\text{Add}(b, y)) >< \text{Add}(S(b), y)
 \end{aligned}$$

Net with active pair and Reduction



After replacing equals for equals:

$$\begin{aligned}
 S(Z) = \text{Add}(z, S(Z)) &\longrightarrow Z = \text{Add}(S, S(Z)) \\
 &\longrightarrow S(S(Z))
 \end{aligned}$$

Language Syntax

We can see interaction nets as a configuration

$$c = (\Sigma, \langle \vec{t} \mid u_1 = v_1, \dots, u_n = v_n \rangle, \mathcal{R})$$

P ::= $(agent \mid rule)^* net$
 $agent$::= $agentName : arity$
 $rule$::= $[ruleName:] term \triangleright \triangleleft term (\Rightarrow net \mid)$
 net ::= $[netName:] equation (, equation)^* \mid$
 $equation$::= $term = term$
 $term$::= $agentName(term, \dots, term) \mid var$

Example

Z:0 S:1

Add:2

AddZ: $Z \triangleright \triangleleft \text{Add}(x,x)$

AddS: $S(\text{Add}(b,y)) \triangleright \triangleleft \text{Add}(S(b),y)$

$S(Z) = \text{Add}(b,S(Z))$

Foreign Language Interface

P ::= $(agent \mid rule)^* net$
 $agent$::= $agentName : arity$
 $rule$::= $[ruleName:] term \triangleright \triangleleft term (\Rightarrow net \mid)$
 net ::= $[netName:] equation (, equation)^* \mid$
 $equation$::= $term = term$
 $term$::= $agentName\{\ell\}(term, \dots, term)$

Example

```
fact:1 num:0 ans:0
```

```
num{int n} >< fact(ans{Factorial.fact(n)})
```

```
fact(x) = num{8}
```

Module System

```

P ::= module { [import modName[. compName]]
           (agent | rule)* net }
agent ::= [*]agentName : arity
rule ::= [[*]ruleName: ] term >< term ( => net | )
net ::= [[*]netName: ] equation (, equation)* |
equation ::= term = term
term ::= [*]agentName{ℓ}(term, ..., term)
  
```

Example

```
module add {  
import arith.Z  
import arith.S  
*Add:2  
*AddZ: Z >< Add(x,x)  
*AddS: S(Add(b,y)) >< Add(S(b),y)  
S(Z) = Add(b,S(Z)) }
```

Part II

Compilation of Interaction nets

Compilation

Compiling a program $p.net$ gives a *linkable* $p.l$ and a $p.sym$. We link $p.l$ to get a *loadable* $p.o$ which is ready for execution by the abstract machine.

Stack based machine that executes the bcodes.

$$\begin{aligned} bcodes & ::= S \mid A \mid M \\ S & ::= loadc \mid store \mid load \mid pop \\ A & ::= mkAgent \mid mkVar \mid mkNet \mid mkRule \\ M & ::= getVar \mid connectPorts \mid eval \end{aligned}$$

A state of the machine is a configuration:

$$\langle PC, \rho, S, A, R \rangle$$

Compilation schemes

A program may consist of a list of modules M_1, \dots, M_n . The compilation scheme \mathcal{C}_p compiles each module using the scheme \mathcal{C}_m . This in turn compiles each of the component

$$c = (\Sigma, \langle \vec{t} \mid u_1 = v_1, \dots, u_n = v_n \rangle, \mathcal{R})$$

in the module by using the appropriate scheme:

$$\mathcal{C}_p = \mathcal{C}_m \llbracket M_1 \rrbracket, \dots, \mathcal{C}_m \llbracket M_n \rrbracket$$

The compilation of a module generates the following code:

$$\mathcal{C}_m[(\Sigma, \langle \vec{t} \mid u_1 = v_1, \dots, u_n = v_n \rangle, \mathcal{R})] = \left\{ \begin{array}{l} \textit{ruleName} : \\ \mathcal{C}_R[r_1] \\ \vdots \\ \textit{ruleName} : \\ \mathcal{C}_R[r_n] \\ \textit{netName} : \\ \mathcal{C}_N[u_1 = v_1] \\ \vdots \\ \textit{netName} : \\ \mathcal{C}_N[u_n = v_n] \\ \textit{interface} : \\ \mathcal{C}_T[t] \end{array} \right.$$

Example

The code generated by compiling:

```
module Erase{
Eps:0
EpsEpsRule:  Eps >< Eps
EpsNet:  Eps = Eps
}
```

is:

EpsEpsRule:	loadc 2	loadc 1
loadc eps	load 0	mkAgent
loadc 1	load 1	store 0
mkAgent	mkRule	load 0
store 0	pop	store 0
load 0	EpsNet:	load 0
store 1	loadc eps	load 1
loadc eps	loadc 1	mkNet

Execution starts from the initial configuration:

$$P \vdash \langle 0, [], [], [], [] \rangle$$

The machine starts executing the instructions under the label `EpsEpsRule`. After interpreting all the code in this label, we obtain the configuration:

$$p \vdash \langle 16, [], [], [], [\text{Eps}, \text{Eps}] \rangle$$

The configuration obtained after executing the code in the label `EpsNet`, before the instruction `eval`:

$$p \vdash \langle 32, [], [], [\text{Eps}, \text{Eps}], [\text{Eps}, \text{Eps}] \rangle$$

The final configuration obtained after the `eval` instruction:

$$p \vdash \langle 33, [], [0 \mapsto \epsilon], [], [\text{Eps}, \text{Eps}] \rangle$$

Future work

We have seen for the first time a compilation scheme for Interaction nets.

- extend the core language
- study alternative compilation schemes
- compilation for parallel hardware
- optimisations
- type system

Compilation Schemes

$$\mathcal{C}_v \llbracket v \rrbracket = \begin{cases} \text{loadc } v \\ \text{mkVar} \\ \text{store}_i \end{cases}$$

$$\mathcal{C}_t \llbracket \alpha(t_1 \cdots t_n) \rrbracket \begin{cases} \mathcal{C}_a \llbracket \alpha \rrbracket \\ \text{store}_i \\ \mathcal{C}_a \llbracket t_1 \rrbracket \\ \text{store}_{i+1} \\ \text{load } i \\ \text{loadc } 1 \\ \text{load } i + 1 \end{cases}$$

$$\mathcal{C}_n[\alpha(u_1, \dots, u_n) = \beta(v_1, \dots, v_n)] = \begin{cases} \mathcal{C}_t[\alpha(u_1 \cdots u_n)] \\ \mathcal{C}_t[\beta(v_1 \cdots v_n)] \\ \text{mkNet} \\ \text{eval} \\ \text{pop} \end{cases}$$

$$\mathcal{C}_r[\alpha(t_1, \dots, t_n) \gg \beta(u_1, \dots, u_n)] = \begin{cases} \mathcal{C}_t[\alpha(t_1 \cdots t_n)] \\ \mathcal{C}_t[\beta(u_1 \cdots u_n)] \\ \text{mkRule} \\ \text{pop} \end{cases}$$

$$C_a[\alpha] = \left\{ \begin{array}{l} \text{loadc } \alpha \\ \text{loadc } ar \\ \text{mkAgent} \\ \text{store}_i \\ \text{loadc } 0 \\ \text{connectPorts} \\ \text{store}_i \\ \vdots \\ C_a[t_n] \end{array} \right. \begin{array}{l} \text{store}_i + 1 \\ \text{load } i \\ \text{loadc } n \\ \text{load } i + 1 \\ \text{loadc } 0 \\ \text{connectPorts} \\ \text{store}_i \end{array}$$