Theories for Service Oriented Computing

Emilio Tuosto

Dipartimento di Informatica, Università di Pisa
etuosto@di.unipi.it − http://www.di.unipi.it/~etuosto

Napoli, 29 June 2005
Plan of the talk

What global computing and service oriented computing are?
- Abstractions for SOC
  - Constraint semirings
  - KoS
- APIs for SOC
- Hypergraphs Model
  - Programming model
  - Specification hypergraphs and related logic
- Automata Model

Semantic based verification
- Model checking
- Equivalence checking
Service Oriented Computing

- applications are made by gluing *services*
- “autonomous”
- independent (local choices, independently built)
- mobile/stationary
- “interconnected”

interactions governed by programmable coordination policies

services are searched and binded ... offline
Global Computing and Services

Service Oriented Computing

- applications are made by gluing services
  - “autonomous”
  - independent (local choices, independently built)
  - mobile/stationary
  - “interconnected”
- interactions governed by programmable coordination policies
- services are searched and binded ... offline

Can search/bind be dynamic and at run-time?
Service Oriented Computing

- applications are made by gluing services
  - “autonomous”
  - independent (local choices, independently built)
  - mobile/stationary
  - “interconnected”

- interactions governed by programmable coordination policies
- services are searched and binded ... offline

Can search/bind be dynamic and at run-time?

What should be considered relevant for searching?
SOC architectures are distributed, interconnected, and based on different communication infrastructures: IP, wireless, satellites... multi-layered: overlay networks (EC-GC2)

**Remark 1** Designers and end-users may ignore the stratification and complexity of the systems. They usually have a high-level view of the computations!

...hence, global computing is not just $\log(P)$, $\bar{s}(x)$ or $s(y)$ & search/bind cycle of SOAs must be defined abstractly. For instance,

- Application-level QoS
- Long Running Transactions
- ...
Abstractions for application-level QoS
Lifting QoS issues to application level...
...for programming global computers
with programmable application level QoS
QoS for designing and implementing SOC applications (SOA)
no longer considered only at the low-level layers

Search and bind wrt application level QoS
  application-related, e.g.
    price
    payment mode
  transactions
    data available in a given format

low-level related (e.g., throughput, response time) not directly referred but abstracted for expressing their “perception” at high-level.

Example 1  When downloading remote information QoS is specified as a constraint on the format of the file (e.g, DVI format instead of Postscript) instead of the underlying network speed (either for application-dependent constraints e.g., editing motivations or download time constraints).

First steps (extending Klaim) in [DFM+03] and recently in [DFM+05]
QoS as Constraint Semirings

C-Semirings \([\text{BMR95, BMR97]}\) for abstracting application level QoS \([\text{DFM}^+03]\)

\[
\langle S, +, \star, 0, 1 \rangle, \text{ where}
\]

- \(S\) is a set (containing 0 and 1),
- \(+, \star : S \times S \rightarrow S\)

<table>
<thead>
<tr>
<th>+</th>
<th>\star</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x + y = y + x)</td>
<td>(x \star y = y \star x)</td>
</tr>
<tr>
<td>((x + y) + z = x + (y + z))</td>
<td>((x \star y) \star z = x \star (y \star z))</td>
</tr>
<tr>
<td>(x + 0 = x)</td>
<td>(x \star 0 = 0)</td>
</tr>
<tr>
<td>(x + 1 = 1)</td>
<td>(x \star 1 = x)</td>
</tr>
<tr>
<td>(x + x = x)</td>
<td>((x + y) \star z = (x \star z) + (y \star z))</td>
</tr>
</tbody>
</table>
QoS as Constraint Semirings

C-Semirings [BMR95, BMR97] for abstracting application level QoS [DFM+03]

\( \langle S, +, \star, 0, 1 \rangle \), where

- \( S \) is a set (containing 0 and 1),
- \( +, \star : S \times S \rightarrow S \)

\[
\begin{array}{c|c}
+ & \star \\
\hline
x + y = y + x & x \star y = y \star x \\
(x + y) + z = x + (y + z) & (x \star y) \star z = x \star (y \star z) \\
x + 0 = x & x \star 0 = 0 \\
x + 1 = 1 & x \star 1 = x \\
x + x = x & (x + y) \star z = (x \star z) + (y \star z)
\end{array}
\]
QoS as Constraint Semirings

C-Semirings [BMR95, BMR97] for abstracting application level QoS [DFM+03]

\[ \langle S, +, \star, 0, 1 \rangle \], where

- \( S \) is a set (containing 0 and 1),
- \( +, \star : S \times S \rightarrow S \)

\[
\begin{align*}
    x + y &= y + x \\
    (x + y) + z &= x + (y + z) \\
    x + 0 &= x \\
    x + 1 &= 1 \\
    x + x &= x
\end{align*}
\]

\[
\begin{align*}
    x \star y &= y \star x \\
    (x \star y) \star z &= x \star (y \star z) \\
    x \star 0 &= 0 \\
    x \star 1 &= x \\
    (x + y) \star z &= (x \star z) + (y \star z)
\end{align*}
\]

- Implicit partial order: \( a \leq b \iff a + b = b \) “\( b \) is better than \( a \)”
C-Semirings [BMR95, BMR97] for abstracting application level QoS [DFM+03]

\[ \langle S, +, \ast, 0, 1 \rangle, \text{ where} \]
\[ S \text{ is a set (containing 0 and 1),} \]
\[ +, \ast : S \times S \to S \]

\begin{align*}
  x + y &= y + x \\
  (x + y) + z &= x + (y + z) \\
  x + 0 &= x \\
  x + 1 &= 1 \\
  x + x &= x
\end{align*}

\begin{align*}
  x \ast y &= y \ast x \\
  (x \ast y) \ast z &= x \ast (y \ast z) \\
  x \ast 0 &= 0 \\
  x \ast 1 &= x \\
  (x + y) \ast z &= (x \ast z) + (y \ast z)
\end{align*}

Implicit partial order: \( a \leq b \iff a + b = b \) "b is better than a"

**Proposition 1**
Cartesian product, exponential and power constructions of c-semirings are c-semiring.
Examples of c-semirings

Example 2  The resource c-semiring \( R = \langle \omega_\infty, \max, \min, 0, \infty \rangle \) is defined on \( \omega_\infty \), the set of natural numbers with infinity.

C-semirings for specific synchronisation mechanisms:

Example 3  The Hoare synchronisation c-semiring is \( H = \langle H, +_H, \ast_H, 0_H, 1_H \rangle \) where

\[
H = \text{Act} \cup \{ 1_H, 0_H, \perp \}
\]

and

\[
\forall a \in \text{Act}. \ a \ast a = a \\
\forall a, b \in \text{Act} \cup \{ \perp \} : b \neq a \implies a \ast b = \perp
\]

plus commutative rules and the ones for 0 and 1

Operation \( +_H \) is obtained by extending the c-semiring axioms for the additive operation with

\[
a +_H a = a, \forall a \in H \\
a +_H b = \perp, \forall a, b \in \text{Act} \cup \{ \perp \}.b \neq a
\]

We can define a general synchronisation policy as a c-semiring that combines (using the cartesian product) a classical synchronisation algebra with the QoS requirements of interest

\[e.g., \ H \times R \text{ is the c-semiring of broadcast with priorities} \]
Another bunch of c-semiring examples

C-semirings structures can be defined for many frameworks:

- $\langle \{ \text{true}, \text{false} \}, \lor, \land, \text{false}, \text{true} \rangle$ (boolean): Availability
- $\langle \text{Real}^+, \text{min}, +, +\infty, 0 \rangle$ (optimization): Price, propagation delay
- $\langle \text{Real}^+, \text{max}, \text{min}, 0, +\infty \rangle$ (max/min): Bandwidth
- $\langle [0, 1], \text{max}, \cdot, 0, 1 \rangle$ (probabilistic): Performance and rates
- $\langle [0, 1], \text{max}, \text{min}, 0, 1 \rangle$ (fuzzy): Performance and rates
- $\langle 2^N, \cup, \cap, \emptyset, N \rangle$ (set-based, where $N$ is a set): Capabilities and access rights
## Process Algebraic Foundations of SOC

<table>
<thead>
<tr>
<th>π-calculus [MPW92]</th>
<th>Rich theory</th>
<th>basic wrt SOC (only link mobility)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Djoin [FG96, FGL+96]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dπ [HR98, HR00]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fusion [PV98]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ambient [CG00]</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Seal [VC98]</td>
<td></td>
<td>Hierarchical</td>
</tr>
<tr>
<td>Boxed [BCC01]</td>
<td></td>
<td>not very natural</td>
</tr>
<tr>
<td>Safe [LS00]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Klaim [BBD+03]</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchical [BLP02]</td>
<td></td>
<td>Very natural</td>
</tr>
<tr>
<td>OKlaim [BBV03]</td>
<td></td>
<td>Lack of observational semantics</td>
</tr>
<tr>
<td>MetaKlaim [FMPar]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| ...                       |             |                                   |
| ...                       |             |                                   |
| ...                       |             |                                   |
Twofold nature: calculus and language (logical vs. physical sites: but no “routing” features)
Twofold nature: calculus and language (logical vs. phisical sites: but no “routing” features)

Multiple tuple spaces
Twofold nature: calculus and language (logical vs. physical sites: but no “routing” features)

Multiple tuple spaces

Localities: first class citizens
Twofold nature: calculus and language (logical vs. physical sites: but no "routing" features)

Multiple tuple spaces

Localities: first class citizens

Process migration
Twofold nature: calculus and language (logical vs. physical sites: but no “routing” features)

- Multiple tuple spaces
- Localities: first class citizens
- Process migration
- Twofold nature: calculus and language (logical vs. physical sites: but no "routing" features)
- Multiple tuple spaces
- Localities: first class citizens
- Process migration
Twofold nature: calculus and language (logical vs. physical sites: but no “routing” features)

Multiple tuple spaces

Localities: first class citizens

Process migration

\[
P ::= \text{nil} \\
| \alpha.P \\
| P_1 | P_2 \\
\alpha ::= a@s \\
a ::= \ldots \ // \text{Klaim actions} \\
| \varepsilon(P)
\]
**KoS characteristics**

**KoS** aims at being a *minimal* calculus for SOC and

- builds on **Klaim** (e.g., processes are localised)...

  - only local communications (unlike **Klaim**)

- ...and $\pi$-calculus

  - link construction primitives

- naturally supports P2P model

  - only one remote action

- QoS as first class citizen

  - which relies on link topology

- QoS-driven semantics

  - semantic transitions report the “cost” of the execution
Remote actions in KoS

KoS ... graphically
Remote actions in KoS

KoS ... graphically
Remote actions in $\mathcal{K}oS$

$\mathcal{K}oS$ ... graphically
Remote actions in KoS

KoS ... graphically
Remote actions in KoS

KoS ... graphically
Remote actions in \( KoS \)

\( KoS \) ... graphically
Let $C$ be the c-semiring of QoS values (ranged over by $\kappa$).

\[
\begin{align*}
N, M &::= 0 \\
&| s :: P \\
&| (\nu s)N \\
&| N \parallel M \\
&| s\overset{\kappa}{\bowtie} t
\end{align*}
\]

\[
\begin{align*}
\gamma &::= (T) \\
&| \langle v_1, \ldots, v_n \rangle \\
&| \varepsilon_{\kappa}[P]@t \\
&| \text{con}_{\kappa}\langle t \rangle \\
&| \text{acc}_{\kappa}\langle t \rangle \\
&| \text{node}_{\kappa}\langle t \rangle
\end{align*}
\]

\[
\begin{align*}
P, Q &::= \text{nil} \\
&| \gamma.P \\
&| (\nu s)P \\
&| P \parallel Q \\
&| !P \\
&| \ldots
\end{align*}
\]

\[
\begin{align*}
T &::= \varepsilon \\
&| v \\
&| ?x \\
&| \neg v \\
&| T, T
\end{align*}
\]
The semantics of $KoS$ is defined by the relation

$$N \xrightarrow{\alpha, \kappa} M$$

which states that $N$ performs $\alpha$ with cost $\kappa$ and becomes $M$.

Local transitions (communications, node or link creations) have unitary QoS value, while the only non-trivial QoS values appear on the transitions that spawn processes or show the presence of links.
(PREF) \[ s :: \gamma.P \xrightarrow{\gamma @ s} s :: P, \quad \gamma \not\in \{ \text{node}_\kappa(t), \text{con}_\kappa(s), \text{acc}_\kappa(s) \} \]

\[ N \xrightarrow{s \text{ con}_\kappa(t)} N' \quad M \xrightarrow{t \text{ acc}_\kappa'(s)} M' \quad 0 < \kappa \leq \kappa' \]

(CON)

\[ N \parallel M \xrightarrow{\tau} N' \parallel M' \parallel s \xrightarrow{\kappa} t \]

(COMM)

\[ N \xrightarrow{s (T)} N' \quad M \xrightarrow{s t} M' \quad \bowtie (T, t) = \sigma \]

\[ N \parallel M \xrightarrow{\tau} N' \sigma \parallel M' \]
**KoS Operational Semantics**

\[
\text{(LINK)} \quad s \xrightarrow{\kappa} t \xrightarrow{s \text{ link } t} 0
\]

\[
\text{(NODE)} \quad s :: \text{node}_\kappa(t).P \xrightarrow{\text{node}(t)} s :: P \parallel s \xrightarrow{\kappa} t \parallel t :: 0, \quad s \neq t
\]

\[
\text{(PAR)} \quad \frac{N \xrightarrow{\alpha} N'}{N \parallel M \xrightarrow{\alpha} N' \parallel M} \quad \text{if} \quad \begin{cases} \text{bn}(\alpha) \cap \text{fn}(M) = \emptyset & \land \\ (\text{addr}(N') \setminus \text{addr}(N)) \cap \text{addr}(M) = \emptyset \end{cases}
\]

Rule (NODE) allows a process allocated at \( s \) to use a name \( t \) as the address of a new node and to create a new link from \( s \) to \( t \) exposing the QoS value \( \kappa \). The side condition of (PAR) prevents new nodes (and links) to be created by using addresses of existing nodes.
### KoS Operational Semantics

**[LEVAL]**

\[
(LEVAL) \quad s :: \epsilon_\kappa [Q] \lhd s.P \xrightarrow{\tau} s :: P \parallel s :: Q
\]

**[ROUTE]**

\[
\begin{align*}
N \xrightarrow{r \in_s \langle P \rangle \lhd t} N' & \quad M \xrightarrow{r \text{ link } r'} M' & \quad \kappa' \ast \kappa'' \leq \kappa \\
\end{align*}
\]

**[LAND]**

\[
\begin{align*}
N \xrightarrow{r \in_s \langle P \rangle \lhd t} N' & \quad M \xrightarrow{\text{ link } t} M' & \quad \kappa' \ast \kappa'' \leq \kappa \\
\end{align*}
\]

Local spawning is always enabled while \( \epsilon_\kappa [P] \lhd t \) from \( s \) is not always possible: the net must contain a path of links from \( s \) to \( t \) suitable wrt \( \kappa \).

**[ROUTE]** states that \( P \) can traverse a link go an intermediate node \( r \) provided that costs are respected. **[LAND]** describes the last hop: in this case, \( P \) is spawned at \( t \), provided that the QoS value of the whole path that has been found is lower than \( \kappa \).
Links in $\mathcal{KoS}$ are public:

\[
N \triangleq s :: \varepsilon_3[P]@t \parallel s \xrightarrow{1} r \parallel r :: \text{con}_2\langle t \rangle.\varepsilon_2[Q]@t \parallel t :: \text{acc}_2\langle r \rangle,
\]

- $s$ and $r$ are trying to spawn a process on $t$ (but no path to $t$ exists).
- $r$ is aware that a link must be first created (and $t$ agrees on that).

Initially, only (CON) can be applied:

\[
N' \triangleq s :: \varepsilon_3[P]@t \parallel s \xrightarrow{1} r \parallel r :: \varepsilon_2[Q]@t \parallel r \xrightarrow{2} t \parallel t :: \text{nil}.
\]

$r \xrightarrow{2} t$ provides now a path (costing 3) from $s$ to $t$, hence using (PREF), (LINK), (ROUTE) and (LAND) we derive

\[
N' \xrightarrow{\tau} \frac{3}{3} s :: \text{nil} \parallel r :: \varepsilon_2[Q]@t \parallel t :: P.
\]

Noteworthy, the migration of $P$ prevents $Q$ to be spawned because the link created by $r$ has been used by $P$. 
**Private links**

Private links can be traversed only by those processes having the appropriate “rights”. Access rights are (particular) names.

\[
N \models s :: \varepsilon_{r,s} [P]@t \parallel s \overset{r}{\rightarrow} s' \quad M \models s :: \varepsilon_{r,s} [P]@t \parallel s \overset{r,u}{\rightarrow} s'
\]

\(P\) can traverse the link in \(N\) but not the one in \(M\).

Access rights c-semiring:

\[
\mathcal{R} = \langle \emptyset_{\text{fin}}(S) \cup \{S\}, \text{glb}, \cup, S, \emptyset \rangle
\]

\[X \leq Y \iff Y \subseteq X\]

A private link between the nodes \(s\) and \(t\) can be specified as

\[
(\nu p)(s :: P \parallel s \overset{p}{\rightarrow} t \parallel t :: Q)
\]
Permanent and stable links

\( \text{KoS} \) links are vanishing but permanent links can be easily encoded:

\[
s :: !\text{con}_\kappa \langle t \rangle \parallel t :: !\text{acc}_\kappa \langle s \rangle
\]

A slight variation are stable links, which are links existing until a given condition is satisfied.

\[
\text{Stable}_s \ G t \triangleq !\text{con}_\kappa \langle t \rangle \mid \varepsilon[\text{While } G \text{ do } \text{acc}_\kappa \langle s \rangle \text{ od } \text{nil}@t]
\]
SOC & Transactions

- Long-Running Transactions (LRTs) equip SOAs with the possibility of atomic execution.
- LRTs use **compensations** for undoing the effects of partial executions when the overall orchestration cannot be completed.
- Primitives for LRTs have been informally specified in most of the languages for orchestrating WS (WSCL [WSC], BPML [BPM], WSFL [Ley01], XLANG [XLA], BPEL4WS ["BP]).

We propose **Java Transactional Web Services (JTWS)**, a set of JAVA APIs supporting coordination of SOA and their transactional composition.

**JTWS** exploit a signal passing mechanism and consists of:
- **JSCL**: provides the signal handling primitives.
- **JTL**: provides primitives for transactional flows.
The **JSCL** layer allows to specify generic components that
- creates a signal link between two components
- sends a signal to the components
- handles signals received by other components

**JTL** is defined on top of **JSCL** and allows to specify transactional flows by composing according
- transactional sequences
- transactional parallel

We have fruitfully exploited **JTWS** for implementing the Sagas calculus [BMM05].
Modelling SOC with Synchronized Hyperedge Replacement
Hypergraphs Programming model

- Edge replacement for graph rewritings \[ \text{[Fed71, Pav72]} \]
- Graphs for distributed systems \[ \text{[CM83, DM87]} \]
- Edge replacement/distributed constraint solving problem \[ \text{[MR96]} \]
- Graphs grammars for software architecture styles \[ \text{[HIM00]} \]
- Synchronised Hyperedge Replacement (SHR) with mobility for name passing calculi \[ \text{[HM01]} \]
- Extension to node fusions \[ \text{[FMT01]} \]
- ...
Using SHR, we aim at

- defining a uniform framework
- tackling new *non-functional* computational phenomena of SOC

The metaphor is

- “SOC systems *as* Hypergraphs”
- “SOC computations *as* SHR”

In other words:

- Components are represented by hyperedges
- Systems are *bunches* of (connected) hyperedges
- Computing means to synchronously rewrite hyperedges...
- ...according to a synchronisation policy
Replacement of Hyperedges

\[ L \rightarrow G \]
Replacement of Hyperedges

$L \rightarrow G$

![Diagram of Replacement of Hyperedges](image)
Replacement of Hyperedges

$L \rightarrow G$
Replacement of Hyperedges

$L \rightarrow G$

- Edge replacement: local
- Synchronisation as distributed constraint solving
- Multiple synchronisation
- New node creation
- Node fusion: model of mobility and communication
**Replacement of Hyperedges**

\[ L \rightarrow G \]

- Edge replacement: local
- Synchronisation as distributed constraint solving
- Multiple synchronisation
- New node creation
- Node fusion: model of mobility and communication

**Benefits:**
- Uniform framework for $\pi$, $\pi$-I, fusion
- LTS for Ambient ...
- ... for Klaim ...
Replacement of Hyperedges

\[ L \rightarrow G \]

- Edge replacement: local
- Synchronisation as distributed constraint solving
- **Multiple synchronisation**
- New node creation
- **Node fusion:** model of mobility and communication

**Benefits:**
- Uniform framework for \( \pi, \pi\text{-}I, \) fusion
- LTS for Ambient ...
- ... for Klaim ...
- ... and *path reservation* for KAOS
- Expressive for distributed coordination
- Wireless networks
Replacement of Hyperedges

$L \rightarrow G$

- Edge replacement: local
- Synchronisation as distributed constraint solving
- Multiple synchronisation
- New node creation
- Node fusion: model of mobility and communication

Benefits:
- Uniform framework for $\pi$, $\pi$-I, fusion
- LTS for Ambient ...
- ... for Klaim ...
- ... and path reservation for KAOS
- Expressive for distributed coordination
- Wireless networks

SHR can combine QoS & sophisticated synchronisations [HT05, LT05, HLT]
A hyperedge connects more than two nodes (generalisation of edge)

\[ L : 3, \ L(y, z, x), \]

\[ x \rightarrow_L 3 \rightarrow_L y \rightarrow_L 1 \rightarrow_L z \]
Hyperedges and Hypergraphs Syntax

A hyperedge connects more than two nodes (generalisation of edge)

\[ L : 3,\ L(y, z, x), \]

\[ G ::= \text{nil} \mid L(\bar{x}) \mid G \mid \nu\ y.G \]

\[ x \rightarrowL^3 \bullet \quad y \quad \bullet \rightarrowL^1 \]

\[ x \rightarrowL^3 \bullet \rightarrowL^2 \bullet z \]
Hyperedges and Hypergraphs Syntax

A hyperedge connects more than two nodes (generalisation of edge)

\[ L : 3, \; L(y, z, x), \]

\[ \begin{array}{c}
\bullet \\
\downarrow_1 \\
\bullet
\end{array} \]

\[ G ::= \text{nil} \mid L(x) \mid G\mid G \mid \nu y.G \]

Syntactic Judgement

\[ x_1 : s_1, \ldots, x_n : s_n \vdash G, \quad fn(G) \subseteq \{x_1, \ldots, x_n\} \]
A hyperedge connects more than two nodes (generalisation of edge)

\[ L : 3, \quad L(y, z, x), \]

\[
\begin{array}{c}
\text{Syntactic Judgement} \\
\vdots \\
G ::= \text{nil} \mid L(x) \mid G \mid v \ y. G
\end{array}
\]

An example:

\[ L : 3, \quad M : 2 \]

\[ x : 1, \quad y : 0 \vdash v \ z. (L(y, z, x) \mid M(y, z)) \]
Hyperedges and Hypergraphs Syntax

A hyperedge connects more than two nodes (generalisation of edge)

\[ L : 3, \quad L(y, z, x), \]

\[ y \]

\[ x \]

\[ M : 2 \]

\[ G ::= \text{nil} \mid L(x) \mid G|G \mid \nu y.G \]

Syntactic Judgement

\[ x_1 : s_1, \ldots, x_n : s_n \vdash G, \quad \text{fn}(G) \subseteq \{x_1, \ldots, x_n\} \]

An example:

\[ L : 3, \quad M : 2 \]

\[ x : 1, \quad y : 0 \vdash \nu z. (L(y, z, x)|M(y, z)) \]
Productions of **SHReQ** are based on requirements: $\mathcal{R} = S \times \mathcal{N}^*$ (where $\langle S, +, *, 0, 1 \rangle$ is a fixed c-semiring)

**production**

$$\chi \triangleright L(\tilde{x}) \xrightarrow{\Lambda} G$$
Productions

Productions of SHReQ are based on requirements: $\mathcal{R} = S \times \mathcal{N}^*$ (where $\langle S, +, \times, 0, 1 \rangle$ is a fixed c-semiring)

- production $\chi \triangleright L(\tilde{x}) \xrightarrow{\Lambda} G$

- $\tilde{x}$ is a tuple of pairwise distinguished nodes and $L : |\tilde{x}|$
Productions of SHReQ are based on requirements: $\mathcal{R} = S \times \mathcal{N}^*$ (where $\langle S, +, \times, 0, 1 \rangle$ is a fixed c-semiring)

- $\tilde{x}$ is a tuple of pairwise distinguished nodes and $L : |\tilde{x}|$
- $\chi : \{|\tilde{x}|\} \rightarrow S$ is the applicability function
Productions of SHReQ are based on requirements: $\mathcal{R} = S \times \mathcal{N}^*$ (where $\langle S, +, \ast, 0, 1 \rangle$ is a fixed c-semiring)

- $\hat{x}$ is a tuple of pairwise distinguished nodes and $L : |\hat{x}|$
- $\chi : \{\hat{x}\} \rightarrow S$ is the applicability function
- $\Lambda : \{\hat{x}\} \rightarrow \mathcal{R}$ is the communication function.

$n(\Lambda)$ communicated nodes of $\Lambda$: those nodes appearing in a requirement in the range of $\Lambda$.

The set of new nodes of $\Lambda$ is $\text{new}(\Lambda) = n(\Lambda) \setminus \text{dom}(\Lambda)$
Productions of SHReQ are based on requirements: $\mathcal{R} = S \times \mathcal{N}^*$
(where $\langle S, +, \cdot, 0, 1 \rangle$ is a fixed c-semiring)

- production $\chi \triangleright L(\tilde{x}) \xrightarrow{\Lambda} G$

- $\tilde{x}$ is a tuple of pairwise distinguished nodes and $L : |\tilde{x}|$
- $\chi : \{|\tilde{x}|\} \rightarrow S$ is the applicability function
- $\Lambda : \{|\tilde{x}|\} \rightarrow \mathcal{R}$ is the communication function.
  $n(\Lambda)$ communicated nodes of $\Lambda$: those nodes appearing in a requirement in the range of $\Lambda$
  The set of new nodes of $\Lambda$ is $\text{new}(\Lambda) = n(\Lambda) \setminus \text{dom}(\Lambda)$
- $G$ is a graph s.t. $\text{fn}(G) \subseteq \{|\tilde{x}|\} \cup n(\Lambda)$
Interpreting SHReQ productions

Consider

$$\pi : \chi \triangleright L(\tilde{x}) \xrightarrow{\Lambda} G$$

and a graph $H$ having an arc labelled by $L$, e.g.:
Consider

\[ \pi : \chi \triangleright L(\tilde{x}) \xrightarrow{\Lambda} G \]

and a graph \( H \) having an arc labelled by \( L \), e.g.:

Replacing \( L \) with \( G \) in \( H \) according to \( \pi \) requires that \( H \) satisfies the conditions expressed by \( \chi \) on the attachment nodes of \( L \).
Interpreting **SHReQ** productions

Consider

\[ \pi : \chi \triangleright L(\tilde{x}) \xrightarrow{\Lambda} G \]

and a graph \( H \) having an arc labelled by \( L \), e.g.:

Replacing \( L \) with \( G \) in \( H \) according to \( \pi \) requires that \( H \) satisfies the conditions expressed by \( \chi \) on the attachment nodes of \( L \).
Consider

$$\pi : \chi \triangleright L(\tilde{x}) \xrightarrow{\Lambda} G$$

and a graph $H$ having an arc labelled by $L$, e.g.:

Replacing $L$ with $G$ in $H$ according to $\pi$ requires that $H$ satisfies the conditions expressed by $\chi$ on the attachment nodes of $L$.

Once $\chi$ is satisfied in $H$, $L(\tilde{x})$ contributes to the rewriting by offering $\Lambda$ in the synchronisation with all the edges connected to nodes in $\tilde{x}$. 
Interpreting SHReQ productions

Consider

\[ \pi : \chi \triangleright L(\tilde{x}) \xrightarrow{\Lambda} G \]

and a graph \( H \) having an arc labelled by \( L \), e.g.:

Replacing \( L \) with \( G \) in \( H \) according to \( \pi \) requires that \( H \) satisfies the conditions expressed by \( \chi \) on the attachment nodes of \( L \).

Once \( \chi \) is satisfied in \( H \), \( L(\tilde{x}) \) contributes to the rewriting by offering \( \Lambda \) in the synchronisation with all the edges connected to nodes in \( \tilde{x} \).
Synchronised Rewriting for \textbf{SHReQ}

Events for 

\textbf{Synchronisation} $\text{Sync}$ and $\text{Fin}$ s.t.

1. $\text{Sync} \subseteq \text{Fin} \subseteq S$
2. $1 \in \text{Sync}$

\textbf{No synchronisation} $\text{NoSync} \subseteq S \setminus \text{Fin}$ s.t.

1. $S \times \text{NoSync} \subseteq \text{NoSync}$
2. $0 \in \text{NoSync}$

\textbf{SHReQ} semantics exploits a mgu accounting for node fusions.

Let $\Omega$ be a finite multiset over $\mathcal{N} \times \mathcal{R}$: $\text{mgu} (\Omega)$ for denoting an idempotent substitution is defined iff

\[(x, s, \tilde{u}), (x, s', \tilde{v}) \in \Omega \oplus x \]

implies

\[|\tilde{u}| = |\tilde{v}|\]

\[\forall i \in \{1, \ldots, |\tilde{u}|\} : \tilde{u}_i \in \text{new} (\Omega) \lor \tilde{v}_i \in \text{new} (\Omega)\]

\[|\Omega \oplus x| > 1 \implies \prod_{(x, s, \tilde{y}) \in \Omega \oplus x} s \notin \text{NoSync}\]
Synchronised Rewriting for SHReQ

Events for

**Synchronisation** $Sync$ and $Fin$ s.t.
- $Sync \subseteq Fin \subseteq S$
- $1 \in Sync$

**No synchronisation** $NoSync \subseteq S \setminus Fin$ s.t.
- $S \ast NoSync \subseteq NoSync$
- $0 \in NoSync$

SHReQ semantics exploits a mgu accounting for node fusions.

Let $\Omega$ be a finite multiset over $\mathcal{N} \times \mathcal{R}$: $\text{mgu}(\Omega)$ for denoting an idempotent substitution is defined iff

\[
(x, s, \tilde{u}), (x, s', \tilde{v}) \in \Omega \land x
\]

implies

\[
|\tilde{u}| = |\tilde{v}|
\]

\[
\forall i \in \{1, \ldots, |\tilde{u}|\} : \tilde{u}_i \in \text{new}(\Omega) \lor \tilde{v}_i \in \text{new}(\Omega)
\]

\[
|\Omega \land x| > 1 \implies \prod_{(x, s, \tilde{y}) \in \Omega \land x} s \notin NoSync
\]
Synchronised Rewriting for **SHReQ**

Events for **Synchronisation** $\text{Sync}$ and $\text{Fin}$ s.t.
- $\text{Sync} \subseteq \text{Fin} \subseteq S$
- $1 \in \text{Sync}$

**No synchronisation** $\text{NoSync} \subseteq S \setminus \text{Fin}$ s.t.
- $S \star \text{NoSync} \subseteq \text{NoSync}$
- $0 \in \text{NoSync}$

**SHReQ** semantics exploits a mgu accounting for node fusions.

Let $\Omega$ be a finite multiset over $\mathcal{N} \times \mathcal{R}$: $\text{mgu}(\Omega)$ for denoting an idempotent substitution is defined iff

\[
(x, s, \tilde{u}), (x, s', \tilde{v}) \in \Omega@x \\
\text{implies} \\
|\tilde{u}| = |\tilde{v}| \\
\forall i \in \{1, \ldots, |\tilde{u}|\} : \tilde{u}_i \in \text{new}(\Omega) \lor \tilde{v}_i \in \text{new}(\Omega) \\
|\Omega@x| > 1 \implies \prod_{(x, s, \tilde{y}) \in \Omega@x} s \not\in \text{NoSync}
\]

and obtained by computing the mgu of the equations

\[
\{\tilde{u}_i = \tilde{v}_i \mid \exists s, t \in S : (x, s, \tilde{u}), (x, t, \tilde{v}) \in \Omega \land 1 \leq i \leq |\tilde{u}|\}
\]
The set $\mathcal{QP}$ of quasi-productions on $\mathcal{P}$ is the smallest set s.t. $\mathcal{P} \subseteq \mathcal{QP}$ and

$$\chi \triangleright L(\tilde{x}) \xrightarrow{\Omega} G \in \mathcal{QP} \quad \land \quad y \in \mathcal{N} \setminus \text{new}(\Omega)$$

$$\downarrow$$

$$\chi' \triangleright L(\tilde{x}\{y/x\}) \xrightarrow{\Omega\{y/x\}} G\{y/x\} \in \mathcal{QP}$$

where

$$\chi' : \{\tilde{x}\} \setminus \{x\} \cup \{y\} \rightarrow S \quad \chi'(z) = \begin{cases} 
\chi(z), & z \in \{\tilde{x}\} \setminus \{x, y\} \\
\chi(x) + \chi(y), & z = y \land y \in \tilde{x} \\
\chi(x), & z = y \land y \not\in \{\tilde{x}\}
\end{cases}$$

**SHReQ rewriting system:** $(\mathcal{QP}, \Gamma \vdash G)$
Graph transitions

(REN)
\[ \chi \triangleright L(\tilde{x}) \xrightarrow{\Omega} G \in Q\mathcal{P} \quad \rho = \text{mgu}(\Omega) \quad \bigwedge_{x \in \text{dom}(\chi)} \chi(x) \leq \Gamma(x) \]

\[ \Gamma \vdash L(\tilde{x}) \xrightarrow{\Omega} \Gamma_{\Omega} \vdash (\nu Z)(G\rho) \]

(COM)
\[ \Gamma_1 \vdash G_1 \xrightarrow{\Lambda_1} \Gamma_1' \vdash G_1' \quad \Gamma_2 \vdash G_2 \xrightarrow{\Lambda_2} \Gamma_2' \vdash G_2' \quad \rho = \text{mgu}(\Lambda_1 \cup \Lambda_2) \quad \bigwedge_{x \in \text{dom}(\Gamma_1) \cap \text{dom}(\Gamma_2)} \Gamma_1(x) = \Gamma_2(x) \]

\[ \Gamma_1 \cup \Gamma_2 \vdash G_1 \mid G_2 \xrightarrow{\Lambda_1 \cup \Lambda_2} (\Gamma_1 \cup \Gamma_2)_{(\Lambda_1 \cup \Lambda_2)} \vdash (\nu Z)(G_1' \mid G_2')\rho \]

where \( Z = \text{new}(\Omega) \setminus \text{new}(\Omega) \)
Graph transitions

\[(\text{REN})\]
\[\chi \triangleright L(\tilde{x}) \xrightarrow{\Omega} G \in \mathcal{QP} \quad \rho = \text{mgu}(\Omega) \quad \bigwedge_{x \in \text{dom}(\chi)} \chi(x) \leq \Gamma(x)\]

\[\Gamma \vdash L(\tilde{x}) \xrightarrow{\Omega} \Gamma \Omega \vdash (\nu Z)(G\rho)\]

\[(\text{COM})\]
\[\Gamma_1 \vdash G_1 \xrightarrow{\Lambda_1} \Gamma'_1 \vdash G'_1 \quad \Gamma_2 \vdash G_2 \xrightarrow{\Lambda_2} \Gamma'_2 \vdash G'_2 \quad \rho = \text{mgu}(\Lambda_1 \cup \Lambda_2) \quad \bigwedge_{x \in \text{dom}(\Gamma_1) \cap \text{dom}(\Gamma_2)} \Gamma_1(x) = \Gamma_2(x)\]

\[\Gamma_1 \cup \Gamma_2 \vdash G_1 \mid G_2 \xrightarrow{\Lambda_1 \cup \Lambda_2} (\Gamma_1 \cup \Gamma_2 )_{(\Lambda_1 \cup \Lambda_2)} \vdash (\nu Z)(G'_1 \mid G'_2)\rho\]

where \(Z = \text{new}(\Omega) \setminus \text{new}(\Omega)\)
Induced communication functions

Let $\rho = \text{mgu}(\Omega)$. The communication function induced by $\Omega$ is the function $\Omega : \text{dom}(\Omega) \rightarrow \mathcal{R}$ defined as

$$\Omega(x) = \begin{cases} (t, \tilde{y}\rho), & t = \prod_{(x,s,\tilde{y}) \in \Omega \otimes x} s \notin \text{Sync} \\ (t, \langle \rangle), & t = \prod_{(x,s,\tilde{y}) \in \Omega \otimes x} s \in \text{Sync} \end{cases}$$

Basically, $\Omega(x)$ yields the synchronisation of requirements in $\Omega \otimes x$ according to the c-semiring product.

The weighting function induced by $\Gamma$ and $\Omega$ is

$$\Gamma_{\Omega} : \text{dom}(\Gamma) \rightarrow S,$$

$$\Gamma_{\Omega}(x) = \begin{cases} 1, & x \in \text{new}(\Omega) \\ \Gamma(x), & |\Omega \otimes x| = 1 \\ \Gamma_{\Omega}(x) = \Omega(x) \downarrow 1, & \text{otherwise} \end{cases}$$

The weighting function computes the new weights of graphs after the synchronisations induced by $\Omega$. 
A network of rings consists of “rings” of different sizes connected by gates.

\( l \)-edges avoid new gates to be attached on the node they insist on. (e.g., above, only the 2-rings can create (gates to) new rings).

The nodes with no \( l \)-edges, can be used to generate new rings and will be weighted by the amount of available resource.
C-semirings for the ring case study

The c-semiring for networks of rings is $\mathcal{H}$ given by the cartesian product of the Hoare synchronisations c-semiring

$$\mathcal{H} = \langle \mathcal{H}, +_H, \ast_H, 0_H, 1_H \rangle,$$

where

$$\mathcal{H} = \{ a, b, c, 1_H, 0_H, \bot \}$$

and the $\mathcal{R} = \langle \omega_\infty, \max, \min, 0, +\infty \rangle$.

The idea is that

- $\mathcal{H}$ coordinates the network rewritings
- $\mathcal{R}$ handles resource availability
- the initial graph is a ring
- non-limited nodes have weights $(1_H, u)$ where value $u$ is the maximal amount of available resource
- the limited nodes created during ring evolution are weighted with $(b, +\infty)$ which is constantly maintained.
Productions for the ring case study

**Create Brother** \((n < u)\)

\[
x \bullet (a,n) \quad \rightarrow \quad R \quad \rightarrow \quad z \bullet l
\]

\[
x : (0_H, u), y : 0 \gg R(x,y) \quad \rightarrow \quad R(x,z) \mid R(z,y) \mid l(z)
\]

**Accept Syncrhonisation** \(R\)

\[
x \bullet (b, +\infty) \quad \rightarrow \quad R
\]

\[
x : (0_H, +\infty), y : 0 \gg R(x,y) \quad \rightarrow \quad R(x,y)
\]

**Accept Syncrhonisation** \(L\)

\[
x \bullet (b, +\infty) \quad \rightarrow \quad l
\]

\[
x : 0 \gg l(x) \quad \rightarrow \quad l(x)
\]

where \(\alpha \in \{a, b\}\)
Productions for the ring case study

Create Gate \((r > 0)\)

\[
x \cdot (a, +\infty) \\
\]

\[
y \cdot (b, +\infty) \\
\]

\[
x : (0_H, u), y : 0 \triangleright R(x, y) \xrightarrow{x \mapsto (a, +\infty), y \mapsto (b, +\infty)} R(x, y) | l(x) | G(z, x) | R_r^u(z, z) \\
\]

Init Ring

\[
(r > 0) \\
\]

\[
x \cdot (c, u) \\
\]

\[
x \mapsto (c, u) \\
\]

\[
R_r^u(x, y) \xrightarrow{x \mapsto (c, u), y \mapsto (c, +\infty)} R(x, z) | R_r^u(z, y) \\
\]

where \(\beta \in \{b, c\}\)
Productions for the ring case study

Accept Synchronisation Init

\[
\begin{align*}
\[x \bullet (c, +\infty) \quad & \quad x \bullet \quad \quad R \quad \rightarrow \quad R \quad y \bullet (c, +\infty) \quad & \quad y \bullet
\end{align*}
\]

\[
x : 0, y : 0 \triangleright R(x, y) \quad \xleftarrow{(c, +\infty)} \quad y \leftarrow (c, +\infty)
\]

where \( \beta \in \{b, c\} \)

Accept Synchronisation Gate

\[
\begin{align*}
\[x \bullet (\beta, +\infty) \quad & \quad x \bullet \quad \quad G \quad \rightarrow \quad G \quad y \bullet (b, +\infty) \quad & \quad y \bullet
\end{align*}
\]

\[
x : 0, y : 0 \triangleright G(x, y) \quad \xleftarrow{(\beta, +\infty)} \quad y \leftarrow (b, +\infty)
\]
SHReQ for the ring case study

The derivation starts from a 2-ring components with resource value 5.

\[ \text{CreatBrother}(u = 5, n = 2) \times \text{CreatBrother}(u = 4, n = 3) \]

\( R \) chooses production \textbf{Create Brother} \( u = 5 \) (satisfying condition \( 5 \leq 5 \)) and \( n = 2 \) while \( R \) chooses \( u = 4 \) (satisfying condition \( 4 \leq 5 \)) and \( n = 3 \). The resulting synchronisation produces the new weights for the nodes as,

\[
(a, 2) = (a, 2) \star (a, +\infty) = (a * _H a, \min(2, +\infty))
\]

\[
(a, 3) = (a, 3) \star (a, +\infty) = (a * _H a, \min(3, +\infty))
\]
**SHReQ for the ring case study**

\[
\text{CreatGate}(r = 1, u = 2) \times \text{CreatGate}(r = 1, u = 3) \times \text{Accept}^*
\]

Only \( R \) and \( R \) can create brothers or gates and they use the remaining resources to create gates to two 2-rings \( (r = 1) \); the other edges apply the \text{Accept} productions.
Note that . and . now are internal.
InitRing($r = 1, u = 2$) $\times$ InitRing($r = 1, u = 3$) $\times$ Accept*
History Dependent Automata
HD-automata

HD-automata as an operational model of *history-dependent* calculi \cite{Pis99, MP98, MP00, FMP02}

allow a finite representation of classes of infinite LTS states and transitions equipped with names:

- names no longer dealt as syntactic components: they become *explicit* in the operational model
- as a consequence...HD-automata model name creation/deallocation or name extrusion

names of HD-automata are *local*...

...hence a mechanism for describing how names correspond each other along transitions is required

so that, a “history” of names in the computation can be determined
**HD-automata: an intuition**

- The transition is labelled by $lab$ and exposes names 2 (of $s$) and a fresh name 0.
- State $s$ has three names: 1, 2 and 3.
- State $d$ has two names: 4 and 5.
- 4 correspond to 1 and 5 to the new name 0.
- Notice that names 3 in $s$ is “discharged” along such transition.

A HD-automaton associates a “history” to names of the states appearing in the computation: it is possible to reconstruct the associations that lead to the state containing the name.

If a state is reached in two different computations, different histories could be assigned to its names.
HD-automata Foundations

- HD-automata with symmetries are based on permutation algebras [MP00]
- In [FMT05a] a type-theoretic definition of HD-automata in terms of a polymorphic lambda calculus ($\lambda \rightarrow, \Pi, \Sigma$) has been given
- Dependent types formally state the relationships between the different components of HD-automata...
- ...and have been exploited for implementing Mihda: a partition refinement algorithm over HD-automata (introduced in [FMP02])
- Mihda minimizes HD-automata representing $\pi$-calculus agents (wrt early bisimulation)
- HD-automata and Mihda have been also used for modelling Fusion calculus [FMT+05c]
Named sets are used for representing the states of HD-automata. Basically, a named set is a set whose elements are equipped with a finite set of names and a symmetry.

**Definition 1** A named set is a structure \( \langle Q, \_|\_, G \rangle \) such that

- \( \eta : Q \to \wp_{\text{fin}}(\mathcal{N}) \)
- \( G \) is a function on \( Q \) such that, for any \( q \in Q \), \( G(q) \) is a group of permutations of \( \eta(q) \)

Given a named set \( A \), we write \( Q_A, \_|\_A \) and \( G_A \) for denoting the components of \( A \)

**Example 4** Consider the \( \pi \)-calculus agent

\[
A(x, y) \triangleq (\nu z)(\bar{x}z.P + \bar{y}z.P).
\]

A state \( q_A \in NS_A \) of a named set representing \( A(x, y) \) has two local names (namely, \( |q_A| = \{x, y\} \)). The symmetry of \( q_A \) consists of the identity permutations and the permutation that exchanges \( x \) with \( y \).
Named functions

Transitions among states are represented by means of named functions:

**Definition 2**  Let $S$ and $D$ be two named sets. A named function is a pair $\langle h : S \rightarrow D, \Sigma \rangle$ such that $h$ is a function from $Q_S$ to $Q_D$ and for all $q \in S$, $\Sigma(q)$ yields a finite set of functions from $|h(q)|_D$ to names in $|q|_S$ of to the distinguished name $\star$ such that

1. $\forall \sigma \in \Sigma(q). G_D(h(q)); \sigma = \Sigma(q),$
2. $\forall \sigma \in \Sigma(q). \sigma; G_S(q) \subseteq \Sigma(q),$
3. any function of $\Sigma(q)$ is injective.

Given a named function $H = \langle h : S \rightarrow D, \Sigma \rangle$ we write

- $\mathbf{dom}(H) = S,$
- $\mathbf{cod}(H) = D,$
- $h_H = h,$
- $\Sigma_H = \Sigma,$

**Definition 3**  Let $H, K$ be two named functions. We say that $H$ and $K$ can be composed iff $\text{cod}(H) = \text{dom}(K)$. In this case, the composition of $H$ and $K$ is the named function $H; K$ such that

- $\mathbf{dom}(H; K) = \mathbf{dom}(H),$
- $\mathbf{cod}(H; K) = \mathbf{cod}(K), h_{H;K} = h_H; h_K$ and
- $\Sigma_{H;K} = \lambda q \in \mathbf{dom}(H; K). \Sigma_K(h_H(q)); \Sigma_H(q)$
Verification techniques for SOC
Q: What is interesting in a SOA computation?
A: No precise answer...our interpretation is

Systems for SOC are obtained by gluing services that are distributed and evolve “together” (causality, parallelism,...)

Systems evolve both “in time” and “in space”. Time evolution is usually interpreted as the dynamic of a system, while spatial evolution corresponds to structural reconfiguration of systems.
A logic for SHReQ

SHReQ has been recently equipped with a spatio-temporal logic interpreted over c-semirings [HLT].

Let \( \langle S, +, *, 0, 1 \rangle \) be a fixed c-semiring and \( \mathcal{G} \) the set of weighted graphs.

\[
\phi ::= \text{nil} \mid \Gamma(\xi) \mid \mathcal{L}(\tilde{\xi}) \mid \phi \vee \phi \mid \phi \land \phi
\]

- spatial operators
- c-semiring operators
- node quantification
- node equality
- temporal operator
- fixpoints

where \( \mathcal{L} \subseteq \mathcal{L} \) is a finite set of labels, \( \xi \) is a metavariable for nodes or node variables and \( f \) is an operation on the fixed c-semiring values.
Fixed a SHReQ rewriting system \( (Q \mathcal{P}, \Gamma \vdash G) \), we interpret formulae as maps \( G \rightarrow S \).
Let \( \sigma \) be a map from node variables to nodes and \( \rho \) be a map from recursion variables to functions \( G \rightarrow S \).

\[
\begin{align*}
[\text{nil}]_{\sigma;\rho}(\Gamma \vdash G) & = G \equiv \text{nil} \\
[\Gamma(\xi)]_{\sigma;\rho}(\Gamma \vdash G) & = (\xi \sigma \in n(G)) \ast \Gamma(\xi \sigma) \\
[\mathcal{L}(\tilde{\xi})]_{\sigma;\rho}(\Gamma \vdash G) & = (G \equiv \mathcal{L}(\xi \sigma)) \ast (L \in \mathcal{L}) \\
[\phi_1|\phi_2]_{\sigma;\rho}(\Gamma \vdash G) & = \sum_{(G_1,G_2) \in \Theta(G)} \{ [\phi_1]_{\sigma;\rho}(\Gamma \vdash G_1) \ast [\phi_2]_{\sigma;\rho}(\Gamma \vdash G_2) \} \\
[\phi_1||\phi_2]_{\sigma;\rho}(\Gamma \vdash G) & = \Pi_{(G_1,G_2) \in \Theta(G)} \{ [\phi_1]_{\sigma;\rho}(\Gamma \vdash G_1) + [\phi_2]_{\sigma;\rho}(\Gamma \vdash G_2) \} \\
[\text{f}(\phi_1, \ldots, \phi_n)]_{\sigma;\rho}(\Gamma \vdash G) & = \text{f}([\phi_1]_{\sigma;\rho}(\Gamma \vdash G), \ldots, [\phi_n]_{\sigma;\rho}(\Gamma \vdash G)) \\
[\sum u \phi]_{\sigma;\rho}(\Gamma \vdash G) & = \sum_{x \in n(G)} [\phi]_{\sigma[x/u];\rho}(\Gamma \vdash G) \\
[\prod u \phi]_{\sigma;\rho}(\Gamma \vdash G) & = \prod_{x \in n(G)} [\phi]_{\sigma[x/u];\rho}(\Gamma \vdash G) \\
[\xi = \xi']_{\sigma;\rho}(\Gamma \vdash G) & = \xi \sigma = \xi' \sigma \\
[\sum \phi]_{\sigma;\rho}(\Gamma \vdash G) & = \sum_{\Gamma \vdash G} \Delta_{\Gamma' \vdash G'} [\phi]_{\Gamma' \vdash G'} \\
[\prod \phi]_{\sigma;\rho}(\Gamma \vdash G) & = \prod_{\Gamma \vdash G} \Delta_{\Gamma' \vdash G'} [\phi]_{\Gamma' \vdash G'} \\
[\text{r}(\tilde{\xi})]_{\sigma;\rho}(\Gamma \vdash G) & = \text{r}(\xi \sigma) \\
[\mu r(\tilde{\nu}).\phi]_{\sigma;\rho}(\Gamma \vdash G) & = \text{lf}(\lambda r'.\lambda \tilde{\nu}. [\phi]_{\sigma[\tilde{\nu}/\tilde{\nu}],\rho[r'/r']})(\tilde{\xi} \sigma)(\Gamma \vdash G) \\
[\nu r(\tilde{\nu}).\phi]_{\sigma;\rho}(\Gamma \vdash G) & = \text{g}(\lambda r'.\lambda \tilde{\nu}. [\phi]_{\sigma[\tilde{\nu}/\tilde{\nu}],\rho[r'/r']})(\tilde{\xi} \sigma)(\Gamma \vdash G)
\end{align*}
\]
A logic for SHReQ

Fixed a SHReQ rewriting system \((\mathcal{QP}, \Gamma \vdash G)\), we interpret formulae as maps \(G \rightarrow S\).

Let \(\sigma\) be a map from node variables to nodes and \(\rho\) be a map from recursion variables to functions \(G \rightarrow S\).

\[
\begin{align*}
\left[\text{nil}\right]_{\sigma;\rho}(\Gamma \vdash G) &= G \equiv \text{nil} \\
\left[\Gamma(\xi)\right]_{\sigma;\rho}(\Gamma \vdash G) &= (\xi \sigma \in n(G)) \ast \Gamma(\xi \sigma) \\
\left[\xi(\tilde{\xi})\right]_{\sigma;\rho}(\Gamma \vdash G) &= (G \equiv L(\tilde{\xi} \sigma)) \ast (L \in \xi) \\
\left[\phi_1|\phi_2\right]_{\sigma;\rho}(\Gamma \vdash G) &= \sum_{(G_1,G_2) \in \Theta(G)} \{ \left[\phi_1\right]_{\sigma;\rho}(\Gamma \vdash G_1) \ast \left[\phi_2\right]_{\sigma;\rho}(\Gamma \vdash G_2) \} \\
\left[\phi_1 \parallel \phi_2\right]_{\sigma;\rho}(\Gamma \vdash G) &= \prod_{(G_1,G_2) \in \Theta(G)} \{ \left[\phi_1\right]_{\sigma;\rho}(\Gamma \vdash G_1) + \left[\phi_2\right]_{\sigma;\rho}(\Gamma \vdash G_2) \} \\
\left[f(\phi_1, \ldots, \phi_n)\right]_{\sigma;\rho}(\Gamma \vdash G) &= f([\left[\phi_1\right]_{\sigma;\rho}(\Gamma \vdash G), \ldots, [\left[\phi_n\right]_{\sigma;\rho}(\Gamma \vdash G)]] \\
\left[\sum_u \phi\right]_{\sigma;\rho}(\Gamma \vdash G) &= \sum_{x \in n(G)} [\phi]_{\sigma[x/u];\rho}(\Gamma \vdash G) \\
\left[\prod_u \phi\right]_{\sigma;\rho}(\Gamma \vdash G) &= \prod_{x \in n(G)} [\phi]_{\sigma[x/u];\rho}(\Gamma \vdash G) \\
\left[\xi = \xi'\right]_{\sigma;\rho}(\Gamma \vdash G) &= \xi \sigma = \xi' \sigma \\
\left[\sum\phi\right](\Gamma \vdash G) &= \sum_{\Gamma \vdash G, \Delta, \Gamma' \vdash G'} [\phi](\Gamma' \vdash G') \\
\left[\prod\phi\right](\Gamma \vdash G) &= \prod_{\Gamma \vdash G, \Delta, \Gamma' \vdash G'} [\phi](\Gamma' \vdash G') \\
\left[r(\tilde{\xi})\right]_{\sigma;\rho}(\Gamma \vdash G) &= r(\xi \sigma) \\
\left[(\mu r(\tilde{\xi}).\phi)\xi\right]_{\sigma;\rho}(\Gamma \vdash G) &= \operatorname{lfp}(\lambda \Gamma'. \lambda \tilde{v}. \left[\phi\right]_{r[\tilde{v}/\tilde{u}],\rho[\Gamma'/\Gamma]})(\xi \sigma)(\Gamma \vdash G) \\
\left[(\nu r(\tilde{\xi}).\phi)\tilde{\xi}\right]_{\sigma;\rho}(\Gamma \vdash G) &= \operatorname{gfp}(\lambda \Gamma'. \lambda \tilde{v}. \left[\phi\right]_{r[\tilde{v}/\tilde{u}],\rho[\Gamma'/\Gamma]})(\xi \sigma)(\Gamma \vdash G)
\end{align*}
\]

\text{nil} characterises graphs with no edges, \(\Gamma(\xi)\) yields the weight of \(\xi\), \(L(\tilde{\xi})\) states that an edge \(L\) attached to \(\tilde{\xi}\) exists s.t. \(L \in \xi\). \(\phi_1|\phi_2\) sums up all the values of \(\phi_i\) on all decompositions. The temporal operator \([\sum]\phi\) sums the values of \(\phi\) after one transition (and similarly for \([\prod]\phi\).
Applying **SHReQ** logic

**Path existence:**

\[ \text{path}(u, v, \mathcal{L}) \equiv \mu r(u, v). (u = v) + \sum_{w} \mathcal{L}(u, w) \mid r(w, v) \]

**Ring membership:**

\[ \text{ring}(u, v) \equiv \text{path}(u, v, \{R\}) \mid \text{path}(v, u, \{R\}) \]

**Highest availability:**

\[ \sum u.\Gamma(u) \ast \neg(\{l\}(u) \mid 1) \]

(on a ring evaluates to the maximum over the weights of non limited nodes)

**Inspecting new rings:**

\[ \text{resource} \equiv \sum_{w,v} (\neg(\{R_0^u\}(w, v) \mid 1)) + ((\{R_0^u\}(w, v) \mid 1) \ast \sum (\{R\}(w, v) \mid 1) \ast \Gamma(w))) \ast (0_H, +\infty)) \]

(4) looks for \(R_0^u\)-edge, and, after the next rewriting step, (5) selects resource of the newly introduce \(R\)-edge.
Verification with HD-automata

- SHReQ is amenable of model checking (forthcoming)
- However, properties of systems can be also stated in terms of equivalences: see [TV04] for spatial properties and [FMT05b] for a more general discussion
- HD-automata can be minimised through a partition refinement algorithm
- The co-algebraic specification of HD-automata and the minimisation algorithm is independent of the chosen language or equivalence
  - $\pi$-calculus & early bisimulation
  - Fusion calculus & hyperbisimulation
Minimising HD-automata

The minimisation algorithm builds the minimal realisation \( \bar{H} \) of (finite) HD-automata by constructing (approximations of) the final coalgebra morphism. The active names of each state \( q \) are those in the ranges of \( \Sigma_{\bar{H}}(q) \).

Given a \( T \)-coalgebra \( K : A \to T_1(A) \) (i.e., a HD-automata) on named set \( A \), the minimisation algorithm is specified in a declarative way by the equations

\[
\text{Initial approximation:} \quad H_0 : \langle q \mapsto \bot, \Sigma : q \mapsto \emptyset \rangle \tag{6}
\]

\[
\text{Iterative construction:} \quad H_{i+1} \triangleq K \circ T_2(H_i). \tag{7}
\]

Intuitively, in the starting phase of the algorithm, all the states of automaton \( K \) are considered equivalent. At the \((i + 1)\)-th iteration, the image through \( T_2 \) of the \( i \)-th iteration is composed with \( K \) as prescribed in (7).

At each iteration, two cases can arise:

- a class is split because the states that it contains are no longer considered equivalent or
- a new active name is discovered.

The algorithm terminates when both these two cases do not occur. This is equivalent to saying that there \( H_{n+1} \) is equal to \( H_n \), for some \( n \).
The main step

Diagram showing a graph with nodes labeled as \( q \), \( \theta \), and \( x \) connected by arrows and angles indicated by \( \theta_q \).
The main step

let bundle hd q =
  List.sort compare
  (List.filter (fun h -> (Arrow.source h) = q) (arrows hd))
The main step

List.map $h_n$ bundle
The main step

\[ h_{n+1} = \text{norm}\langle\text{states, } \{\langle l, \pi, h_n(q'), \sigma'; \sigma\rangle | \xrightarrow{q} l \pi \sigma q' \land \sigma' \in \Sigma_n(q')\rangle}\]  

At each iteration, redundant transitions decrease and, when the iterative construction terminates, only the really redundant free inputs are removed.
Let \( an = \text{active_names_bundle} \) (red bundle) in

\[
\begin{align*}
\text{let } \text{remove_in } ar &= \text{match } ar \text{ with } \\
| \text{Arrow}(\_\_, \_\_, \text{In}(\_\_, \_\_)) &\rightarrow \text{not } (\text{List.mem (obj ar) an}) \\
| \_ \rightarrow \text{false } \text{in} \\
\text{list_diff } \text{bundle } (\text{List.filter remove_in } \text{bundle})
\end{align*}
\]

The main step
\[ \Sigma_{n+1}(q) = (\text{compute\_group} (\text{norm bundle})); \quad \theta_q^{-1} \]
The main step

\[ \Sigma_{n+1}(q) = (\text{compute}_\text{group} \ (\text{norm} \ \text{bundle})) \ ; \ \theta_q^{-1} \]

**Theorem** At the end of each iteration \( i \) blocks corresponds to \( h_{H_i} \).
Minimizing History Dependent Automata:

- HD-automata for history dependent calculi
- Co-algebraic specification
- Partition Refinement Algorithm based on co-algebraic specification [FMP02]
- Mihda: Ocaml implementation (refining \(\lambda \to \Pi, \Sigma\) spec.)

<table>
<thead>
<tr>
<th></th>
<th>Comp. Time</th>
<th>States</th>
<th>Trans.</th>
<th>Min. Time</th>
<th>States</th>
<th>Trans.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSM small</td>
<td>0m 0.931s</td>
<td>211</td>
<td>398</td>
<td>0m 4.193s</td>
<td>105</td>
<td>197</td>
</tr>
<tr>
<td>GSM full</td>
<td>0m 8.186s</td>
<td>964</td>
<td>1778</td>
<td>0m 54.690s</td>
<td>137</td>
<td>253</td>
</tr>
</tbody>
</table>
Mihda Architecture

- Adherent to specs
- Highly modular
- Easily extendible

Diagram:

- Domination
- Bundle
- Automaton
- Transitions
- Labels
- States
- Block

Graph illustrating the relationships between Domination, Bundle, Automaton, Transitions, Labels, States, and Block.
References


56-3


[WSC] Web Services Conversation Language (WSCL) 1.0. [http://www.w3.org/TR/wscl10/](http://www.w3.org/TR/wscl10/).