Specification and verification in service oriented computing

Emilio Tuosto

Dipartimento di Informatica
Università di Pisa

Lectureship interview, Leicester University
August 2005
Service Oriented Computing

- Distributed computing is moving toward SOC
- Integration of software and (heterogeneous) networks of (heterogeneous) systems (e.g., Internet & mobile phones, wireless & wired networks)
Service Oriented Computing

- Distributed computing is moving toward SOC
- Integration of software and (heterogeneous) networks of (heterogeneous) systems (e.g., Internet & mobile phones, wireless & wired networks)

- SOC architectures are
  - distributed
  - interconnected
  - based on different communication infrastructures:
    - IP, wireless, satellites...
    - multi-layered: overlay networks
  - Designers, programmers and end-users may ignore the stratification and complexity
Service Oriented Computing

- Distributed computing is moving toward SOC
- Integration of software and (heterogeneous) networks of (heterogeneous) systems (e.g., Internet & mobile phones, wireless & wired networks)

SOC architectures are
- distributed
- interconnected
- based on different communication infrastructures:
  - IP, wireless, satellites...
  - multi-layered: overlay networks
- Designers, programmers and end-users may ignore the stratification and complexity

SOC applications (SOAs) are soups of services
- programmable coordination
- “autonomous”
- independent
- mobile/stationary
- “interconnected” through interfaces and published, searched and binded...
- offline and mostly ad-hoc way
Distributed computing is moving toward SOC

Integration of software and (heterogeneous) networks of (heterogeneous) systems (e.g., Internet & mobile phones, wireless & wired networks)

SOC architectures are
- distributed
- interconnected
- based on IP & more
- multiple networks

Designers, programmers and end-users may ignore the stratification and complexity

How do we program SOC?

Can search/bind be dynamically done at run-time?

What is considered relevant for searching services?

What about verification?
Formal tools for SOC

**Specification**
- Process calculi
  - Klaim
  - cIP
  - CCS\parallel
  - Symbolic Fusion calculus
- SHR
- QoS as c-semirings
- Spatial logics wrt
  - HD-automata
  - SHR

**Verification**
- HD-automata
- operational model
- minimisation
- cIP & PL
- Spatial model checking

**Implementation**
- Klaim
- Aspaya
- SHR: Gredy
- Mihda
- JTWS & JSAGA
- Long running transactions

Works with several people:

**Inside Pisa:**
- Pisa Department: Andrea Bracciali, Roberto Bruni, Gianluigi Ferrari, Dan Hirsch, Ivan Lanese, Hernán Melgratti, Ugo Montanari, Daniele Strollo
- Pisa CNR: Stefania Gnesi

**Outside Pisa:**
- Florence: Rocco De Nicola and Rosario Pugliese
- Uppsala: Kidane Yemane & Björn Victor
- Lisbon: Hugo Torres Viera
Formal tools for SOC

**Specification**
- Process calculi
  - Klaim...KoS
  - cIP
  - CCS
  - Symbolic Fusion calculus
- SHR
- QoS as c-semirings
- Spatial logics wrt
  - HD-automata
  - SHR

**Verification**
- HD-automata operational model
- minimisation
- cIP & PL
- Spatial model checking

**Implementation**
- Klaim
- Aspasya
- SHR: Gredy
- Mihda
- JTWS & JSAGA
- Long running transactions
KoS

Joint works with

Oath of Hippocrates
Lifting QoS issues to applications

- with programmable application level QoS
- QoS in designing and implementing SOAs
Lifting QoS issues to applications
- with programmable application level QoS
- QoS in designing and implementing SOAs

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 the application level

First steps (extending Klaim) in [DFM+03] and recently in [DFM+05]
Lifting QoS issues to applications

- with programmable application level QoS
- QoS in designing and implementing SOAs

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 the application level

Application level QoS abstracted as **constraint-semiring** [BMR95, BMR97]

- for coordinating mobility

- and synchronisations

**C-semiring** are particularly indicated because

- they have an implicitly defined partial order
- their structure is preserved by many mathematical operations...
- ...in particular cartesian product of c-semirings is a c-semiring
- hence c-semirings are suitable for multi-criteria
**KoS** aims at being a **minimal** calculus for SOC

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

- ’cause it naturally supports a **peer-to-peer** programming model

- **KoS** primitives handle QoS values as first class entities

- **KoS** semantics ensures that the QoS values are respected during execution

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

- Link construction primitives

- Only one remote action

- Which relies on link topology

- Semantic transitions report the “cost” of the execution
Remote actions in $\mathcal{KoS}$

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

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

$\mathcal{KoS}$ ... graphically
Remote actions in \( \text{KoS} \)

\( \text{KoS} \) ... graphically
Remote actions in $\textit{KoS}$

$\textit{KoS}$ ... graphically
SHR & SOC

Joint works with
G. Ferrari, D. Hirsch, I. Lanese, A.
LLuch-lafuente, U. Montanari
Hypergraphs Programming model

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
Synchronised Replacement of Hyperedges
Synchronised Replacement of Hyperedges

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

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

Benefits:
- Uniform framework for $\pi$, $\pi$-l, fusion
- LTS for Ambient ...
- ... for Klaim ...
- ... and path reservation for KAOS
- expressive for distributed coordination
- wireless networks
Synchronised Replacement of Hyperedges

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

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

SHR can combine QoS & sophisticated synchronisations (details in [HT05, LT05, HLT])
Clients $C_1, \ldots, C_m$ invoke a service from a remote servers $S_1, \ldots, S_n$ provided that they are authorised.

A trusted authority $Au$ checks for the authorisation.

Clients are connected to $Au$ on a “public” node $x$ while servers are connected on a “private” (i.e., restricted) ones. $B_2$ will simply acquire the requests from clients and forward them to each server.

Notice that:

- synchronisations $C_i \leftarrow Au$ are “Milner” (e.g., in PPP connections)
- $B_2$ is required when broadcast is not primitive
- then broadcast must be “encoded”
Clients $C_1, \ldots, C_m$ invoke a service from a remote servers $S_1, \ldots, S_n$ provided that they are authorised

A trusted authority $Au$ checks for the authorisation

Clients are connected to $Au$ on a “public” node $x$ while servers are connected on a “private” (i.e., restricted) ones.

In SHR we can simply specify

$$x : Mil \vdash C_i(x) \xrightarrow{(x, auth_i, \langle y \rangle)} x : Mil, y : Bdc \vdash C'_i(y)$$

$$x : Mil, u : Bdc \vdash Au(x, u) \xrightarrow{(x, auth_i, \langle u \rangle)} x : Mil, u : Bdc \vdash Au(x, u)$$

where $Bdc$ is the broadcast SAM.
A spatio-temporal logic for SHR interpreted over c-semirings has been defined in [HLT].

\[
\phi \ ::= \ nil \mid \Gamma(\xi) \mid \mathcal{L}(\tilde{\xi}) \mid \phi \& \phi \mid \phi \parallel \phi
\]

\[
\mid f(\phi, \ldots, \phi)
\mid \sum_u \phi \mid \prod_u \phi
\mid [\sum] \phi \mid [\prod] \phi
\mid u = v
\mid r(\tilde{\xi}) \mid (\mu r(\tilde{u}).\phi)\tilde{\xi} \mid (\nu r(\tilde{u}).\phi)\tilde{\xi}
\]

where \( \langle S, +, \cdot, 0, 1 \rangle \) is a c-semiring, \( \mathcal{L} \subseteq \mathcal{L} \) 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.
A spatio-temporal logic for SHR interpreted over c-semirings has been defined in \cite{HLT}.

\[
\phi ::= \quad \text{nil} \mid \Gamma(\xi) \mid \mathcal{L}(\tilde{\xi}) \mid \phi\phi \mid \phi\|\phi \\
\mid f(\phi, \ldots, \phi) \\
\mid \sum_u \phi \mid \prod_u \phi \\
\mid [\sum] \phi \mid [\prod] \phi \\
\mid u = v \\
\mid r(\tilde{\xi}) \mid (\mu r(\tilde{u}).\phi)\tilde{\xi} \mid (\nu r(\tilde{u}).\phi)\tilde{\xi}
\]

where \( \langle S, +, \ast, 0, 1 \rangle \) is a c-semiring, \( \mathcal{L} \subseteq \mathcal{L} \) 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.
A logic for SHR & QoS

A spatio-temporal logic for SHR interpreted over c-semirings has been defined in [HLT].

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

where \( \langle S, +, *, 0, 1 \rangle \) is a c-semiring, \( \mathcal{L} \subseteq \mathcal{L} \) 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.

Let \( G \) be the set of weighted graphs: we interpret formulae as maps \( G \to S \).
Let \( \sigma \) be a map from node variables to nodes and \( \rho \) be a map from recursion variables to functions \( G \to S \).

\[
\begin{align*}
\phi_1\phi_2]_{\sigma;\rho}(\Gamma \vdash G) &= \sum_{(G_1, G_2) \in \Theta(G)} \{ \phi_1\phi_2]_{\sigma;\rho}(\Gamma \vdash G_1) * \phi_2\phi_1]_{\sigma;\rho}(\Gamma \vdash G_2) \\
\phi_1\phi_2]_{\sigma;\rho}(\Gamma \vdash G) &= \prod_{(G_1, G_2) \in \Theta(G)} \{ \phi_1\phi_2]_{\sigma;\rho}(\Gamma \vdash G_1) + \phi_2\phi_1]_{\sigma;\rho}(\Gamma \vdash G_2) \\
\sum_u \phi]_{\sigma;\rho}(\Gamma \vdash G) &= \sum_{x \in n(G)} \{ \phi]_{\sigma[x/u];\rho}(\Gamma \vdash G) \\
\prod_u \phi]_{\sigma;\rho}(\Gamma \vdash G) &= \prod_{x \in n(G)} \{ \phi]_{\sigma[x/u];\rho}(\Gamma \vdash G) \\
[\sum] \phi](\Gamma \vdash G) &= \sum_{\Gamma \vdash G \prec \Gamma' \vdash G'} \phi](\Gamma' \vdash G') \\
[\prod] \phi](\Gamma \vdash G) &= \prod_{\Gamma \vdash G \prec \Gamma' \vdash G'} \phi](\Gamma' \vdash G')
\end{align*}
\]
HD-automata

Joint works with
HD-automata...intuitively

- HD-automata as an operational model of history-dependent calculi [Pis99, MP98]
- allow a finite representation of classes of infinite LTS
HD-automata...intuitively

- HD-automata as an operational model of history-dependent calculi [Pis99, MP98]
- allow a finite representation of classes of infinite LTS

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...intuitively

- HD-automata as an operational model of history-dependent calculi \[\text{[Pis99, MP98]}\]
- allow a finite representation of classes of infinite LTS

- states and transitions have local names:
  - names explicit in the operational model
  - so that HD-automata model name creation/deallocation or extrusion
- State $s$ has three names: 1, 2 and 3
- State $d$ has two names: 4 and 5
- The transition is labelled by $lab$ and exposes names 2 (of $s$) and a fresh name 0
HD-automata...intuitively

- HD-automata as an operational model of history-dependent calculi [Pis99, MP98]
- allow a finite representation of classes of infinite LTS

- states and transitions have local names:
  - names explicit in the operational model
  - so that HD-automata model name creation/deallocation or extrusion
- State $s$ has three names: 1, 2 and 3
- State $d$ has two names: 4 and 5
- The transition is labelled by $lab$ and exposes names 2 (of $s$) and a fresh name 0
- $\sigma : 4 \mapsto 1$ and $\sigma : 5 \mapsto 0$, the new name
- 3 is “discharged”
Minimising History Dependent Automata:

- Co-algebraic specification
- Partition Refinement Algorithm based on co-algebraic specification [FMP02]
- Mihda: Ocaml implementation (refining $\lambda \to, \Pi, \Sigma$ spec. [FMT05a])

<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>

- Adherent to specs
- Highly modular
- Easily extendible
The main step
The main step
The main step
The main step
The main step
**Theorem**  Any block at a generic iteration $i$ collects those states that cannot be distinguished in $i$-steps.

**Theorem**  The algorithm converges on finite HD-automata (for $\pi$-calculus [FMT05a])

**Theorem**  The iterative partition refinement algorithm is convergent on finite HD-automata whenever the normalisation functor is monotone on nfs (for $\pi$-calculus [FMT+05b])
References


[DFM+05] Rocco De Nicola, Gianluigi Ferrari, Ugo Montanari, Rosario Pugliese, and Emilio Tuosto. A Basic Calculus for Modelling Service Level Agreements. In Jean-Marie Jacquet and Gian Pietro


Application level QoS abstracted as constraint-semiring \[\text{[BMR95, BMR97]}\]

- for coordinating mobility
- and synchronisations

An algebraic structure \(\langle S, +, *, 0, 1 \rangle\) is a c-semiring iff

- \(0, 1 \in S\) and \(0 \neq 1\)

\[
\begin{align*}
+ & : S \times S \rightarrow S \\
& x + y = y + x \\
& (x + y) + z = x + (y + z) \\
& x + 0 = x \\
& x + 1 = 1 \\
& x + x = x \\

* & : S \times S \rightarrow S \\
& x * y = y * x \\
& (x * y) * z = x * (y * z) \\
& x * 1 = x \\
& x * 0 = 0 \\
& (x + y) * z = (x * z) + (y * z)
\end{align*}
\]

and

Implicit partial order:

\[a \leq b \iff a + b = b\]

"b is better than a"

The cartesian product of c-semirings is a c-semiring
Let $C$ be the $c$-semiring of *QoS values* (ranged over by $\kappa$):

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

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

\[
\begin{align*}
\gamma & ::= (T) \\
& | \quad \langle v_1, \ldots, v_n \rangle \\
& | \quad \varepsilon^\kappa[P]@t \\
& | \quad con_{\kappa}(t) \\
& | \quad acc_{\kappa}(t) \\
& | \quad node_{\kappa}(t)
\end{align*}
\]

\[
\begin{align*}
T & ::= \varepsilon \\
& | \quad v \\
& | \quad ?x \\
& | \quad \neg v \\
& | \quad T, T
\end{align*}
\]
Let $C$ be the $c$-semiring of QoS values (ranged over by $\kappa$).

\[ N, M ::= \\
0 \\
s :: P \\
(\nu s)N \\
N \parallel M \\
s \overset{\kappa}{\rightarrow} t \]

\[ P, Q ::= \\
inl \\
\gamma . P \\
(\nu s)P \\
P \mid Q \\
!P \\
\ldots \]

\[ \gamma ::= \\
(T) \\
\langle v_1, \ldots, v_n \rangle \\
\varepsilon \kappa [P]@t \\
con_{\kappa} \langle t \rangle \\
acc_{\kappa} \langle t \rangle \\
node_{\kappa} \langle t \rangle \]

\[ T ::= \\
\varepsilon \\
v \\
?x \\
\neg v \\
T, T \]

$N \overset{\alpha}_{\kappa} \rightarrow M$ states that $N$ performs $\alpha$ with a cost $\kappa$ and becomes $M$. 
Let $C$ be the c-semiring of QoS values (ranged over by $\kappa$).

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

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

\[
\begin{align*}
\gamma & ::= (T) \mid \langle v_1, \ldots, v_n \rangle \mid \epsilon \kappa [P] @ t \mid \text{con}_\kappa \langle t \rangle \mid \text{acc}_\kappa \langle t \rangle \mid \text{node}_\kappa \langle t \rangle \\
T & ::= \epsilon \mid v \mid ?x \mid \neg v \mid T, T
\end{align*}
\]

$N \xrightarrow{\alpha}{\kappa} M$ states that $N$ performs $\alpha$ with a cost $\kappa$ and becomes $M$.

\[\text{(ROUTE)}\]

$N \xrightarrow{r, \epsilon^s, (P) @ t}{\kappa'} N' \quad \text{M} \xrightarrow{r, \text{link} r'}{\kappa''} \text{M'} \quad \kappa' \ast \kappa'' \leq \kappa$

$\kappa' \ast \kappa'' \leq \kappa$

$N \parallel M \xrightarrow{r', \epsilon^s, (P) @ t}{\kappa'} N' \parallel M'$

$t \neq r'$
Let $C$ be the $c$-semiring of QoS values (ranged over by $\kappa$).

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

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

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

\[
T ::= \varepsilon \\ &\mid v \\ &\mid ?x \\ &\mid \neg v \\ &\mid T, T
\]

$N \overset{\alpha}{\kappa} \rightarrow M$ states that $N$ performs $\alpha$ with a cost $\kappa$ and becomes $M$.

\[
\begin{align*}
N \overset{r}{\varepsilon} \kappa\langle P \rangle@t &\rightarrow N' \\ M \overset{r}{\text{link}} t &\rightarrow M' \\
\kappa' \ast \kappa'' &\leq \kappa
\end{align*}
\]

(LAND)

\[
\begin{align*}
N \parallel M \overset{\tau}{\kappa'} \ast \kappa'' &\rightarrow N' \parallel M' \parallel t :: P
\end{align*}
\]
A motivating example

Consider a scenario where $n$ servers provide services to $m$ clients and focus on balancing the load of the servers.

- clients ($c_i$) and servers ($s_j$) are located on different nodes
- $c_i$ issues requests to $s_j$ by spawning a process $R$

A generic client is described by the following term:

$$c_i :: \langle s_1, \kappa_1 \rangle \mid \ldots \mid \langle s_n, \kappa_n \rangle \mid !C_\delta$$

- $\langle s_j, \kappa_j \rangle$ represents the load $\kappa_j$ of the server $s_j$ perceived by $c_i$
- $C_\delta$ and $R$ specify the behaviour of $c_i$:

$$C_\delta \triangleq (\exists u, v).\varepsilon_v[R].u.con_{v*\delta}\langle u \rangle.\langle u, v \ast \delta \rangle$$

$$R \triangleq (\exists x).\langle x + 1 \rangle \ldots \text{actual request} \ldots (\exists y).\langle y - 1 \rangle$$

**Remark 1** Remote spawning consumes the traversed links, hence $c_i$ attempts to re-establish a connection with the server!
A motivating example

$s_j$ is described as:

$$s_j :: \langle h \rangle \mid \langle c_1, \kappa'_1 \rangle \mid \ldots \mid \langle c_m, \kappa'_m \rangle \mid !(S \ c_1 \ s_j) \mid \ldots \mid !(S \ c_m \ s_j)$$

- $\langle c_i, \kappa'_i \rangle$ records the QoS value $\kappa'_i$ assigned to the link towards $c_i$
- $\langle h \rangle$ is the current load of $s_j$
- $S \ c_i \ s_j$ is a load manager for $c_i$

\[ S \ c \ s \triangleq (?l).\langle l \rangle . If \ s \ l < max \ then \ (c, ?v).acc_{f(v,l)} \langle c \rangle \cdot \langle c, f(v,l) \rangle \]

$S$ repeatedly acquires $\langle h \rangle$ and depending on the load decides whether to accept requests for new connections coming from $c$. 
\[(\text{PREF}) \quad s :: \gamma . P \xrightarrow{\gamma \circ s} \_ \quad 1 \quad s :: P, \quad \gamma \not\in \{\text{node}_\kappa \langle t \rangle, \text{con}_\kappa \langle s \rangle, \text{acc}_\kappa \langle s \rangle\}\]

\[(\text{CON}) \quad \frac{N \xrightarrow{s \ \text{con}_\kappa \langle t \rangle} N'}{1} \quad \frac{M \xrightarrow{t \ \text{acc}_{\kappa'} \langle s \rangle} M'}{1} \quad 0 < \kappa \leq \kappa' \]

\[\frac{N || M \xrightarrow{\tau} N' || M'}{1} \xrightarrow{s \ \kappa} t \quad (\text{COMM}) \quad \frac{N \xrightarrow{s \ (T)} N'}{1} \quad \frac{M \xrightarrow{s \ t} M'}{1} \quad \triangleq (T, t) = \sigma \]

\[\frac{N || M \xrightarrow{\tau} N' \sigma || M'}{1} \]
(LINK) \[ s \xrightarrow{\kappa} t \quad \frac{s \text{ link } t}{0} \]

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

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

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

\[(\text{LEVEL}) \quad s :: \varepsilon_\kappa[Q]@s.P \xrightarrow{\tau} s :: P \parallel s :: Q\]

\[(\text{ROUTE}) \quad N \xrightarrow{r \varepsilon_\kappa^s\langle P \rangle@t} N' \quad M \xrightarrow{r \text{ link } r'} M' \quad \kappa' \ast \kappa'' \leq \kappa \quad t \neq r'\]

\[
\frac{N \parallel M \xrightarrow{r' \varepsilon_\kappa^s\langle P \rangle@t} N' \parallel M'}{\kappa' \ast \kappa'' \leq \kappa}
\]

\[(\text{LAND}) \quad N \xrightarrow{r \varepsilon_\kappa^s\langle P \rangle@t} N' \quad M \xrightarrow{r \text{ link } t} M' \quad \kappa' \ast \kappa'' \leq \kappa\]

\[
\frac{N \parallel M \xrightarrow{\tau} N' \parallel M' \parallel t :: P}{\kappa' \ast \kappa'' \leq \kappa}
\]

Local spawning is always enabled while \(\varepsilon_\kappa[Q]@t\) 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:

$$N \triangleq \begin{array}{c} s :: \varepsilon_3[P]@t \ | \ s \overset{1}{\rightarrow} r \ | \ r :: \text{con}_2(t).\varepsilon_2[Q]@t \ | \ t :: \text{acc}_2(r), \end{array}$$

- $s$ 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 ($\text{CON}$) can be applied:

$$N' \triangleq \begin{array}{c} s :: \varepsilon_3[P]@t \ | \ s \overset{1}{\rightarrow} r \ | \ r :: \varepsilon_2[Q]@t \ | \ r \overset{2}{\rightarrow} t \ | \ t :: \text{nil}. \end{array}$$

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

$$N' \xrightarrow{\tau} \begin{array}{c} s :: \text{nil} \ | \ r :: \varepsilon_2[Q]@t \ | \ t :: P. \end{array}$$

Noteworthy, the migration of $P$ prevents $Q$ to be spawned because the link created by $r$ has been used by $P$. 
Private links can be traversed only by those processes having the appropriate “rights”. Access rights are (particular) names.

\[ N \trianglerighteq s :: \varepsilon_{\{r,s\}}[P] @ t \parallel s \{ r \} \quad s' \]

\[ M \trianglerighteq s :: \varepsilon_{\{r,s\}}[P] @ t \parallel s \{ r,u \} \quad s' \]

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

Access rights c-semiring:

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

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

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

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

\(\mathcal{Ko}_S\) links are vanishing but permanent links can be easily encoded:

\[ s :: !\text{con}_\kappa(t) \parallel t :: !\text{acc}_\kappa'(s) \]

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

\[ \text{Stable}_s \ G \ t \triangleq !\text{con}_\kappa(t) \mid \varepsilon[\text{While } G \text{ do } \text{acc}_\kappa(s) \text{ od } \text{nil}] \odot t \]
Hypergraphs Syntax

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

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

\[
\begin{aligned}
x & \xrightarrow{3} L \\
\end{aligned}
\]

\[
\begin{aligned}
y & \xrightarrow{1} \\
\end{aligned}
\]

\[
\begin{aligned}
z & \xrightarrow{2} \\
\end{aligned}
\]

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

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)

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

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

\begin{center}
\begin{tikzpicture}
  \node (y) at (0,0) [circle,fill] {y};
  \node (x) at (-1,-1) [circle,fill] {x};
  \node (z) at (1,-1) [circle,fill] {z};
  \node (L) at (0,-1.5) [draw] {L};
  \draw[->] (x) -- (L) node[midway,above] {3};
  \draw[->] (L) -- (z) node[midway,above] {2};
  \draw[->] (y) -- (L) node[midway,above] {1};
\end{tikzpicture}
\end{center}

**Syntactic Judgement**

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

An example:

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

\[ x : 1, y : 0 \vdash \nu z. (L(y, z, x)|M(y, z)) \]
Productions

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

\[
\text{production} \quad \chi \triangleright L(\tilde{x}) \xrightarrow{\Lambda} G
\]

- \( \tilde{x} \) 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
- \( G \) is a graph s.t. \( \text{fn}(G) \subseteq \{\tilde{x}\} \cup n(\Lambda) \)
Productions based on requirements: $\mathcal{R} = S \times \mathcal{N}^*$ (where $\langle S, +, \star, 0, 1 \rangle$ is a c-semiring)

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

- $\tilde{x}$ 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
- $G$ is a graph s.t. $\text{fn}(G) \subseteq \{|\tilde{x}|\} \cup n(\Lambda)$

Replacing $L$ with $G$ in $H$ 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

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.

- \( S \ast \text{NoSync} \subseteq \text{NoSync} \)
- \( 0 \in \text{NoSync} \)
Synchronised Rewriting

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.

- \( S \ast NoSync \subseteq NoSync \)
- \( 0 \in NoSync \)

\( \text{mgu} \) accounting for node fusions; let \( \Omega \) be a finite multiset over \( \mathcal{N} \times \mathcal{R} \).

\[
\text{mgu}(\Omega) = \{\tilde{u}_i = \tilde{v}_i \mid \forall s, t \in S : (x, s, \tilde{u}), (x, t, \tilde{v}) \in \Omega \land 1 \leq i \leq |\tilde{u}|\}
\]

is an idempotent substitution defined iff

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

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.
- \( S \ast NoSync \subseteq NoSync \)
- \( 0 \in NoSync \)

\( \text{mgu accounting for node fusions; let } \Omega \text{ be a finite multiset over } \mathcal{N} \times \mathcal{R}. \)

\[
\text{mgu}(\Omega) = \{ \tilde{u}_i = \tilde{v}_i \mid \exists s, t \in S : (x, s, \tilde{u}), (x, t, \tilde{v}) \in \Omega \wedge 1 \leq i \leq |\tilde{u}| \}
\]

is an idempotent substitution defined iff

\[
|\Omega @ x| > 1 \implies \prod_{(x, s, \tilde{y}) \in \Omega @ x} s \notin NoSync
\]

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

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

\( \rho = \text{mgu}(\Lambda_1 \cup \Lambda_2) \)
The set $QP$ of quasi-productions on $P$ is the smallest set s.t. $P \subseteq QP$ and

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

$$\downarrow$$

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

where

$$\chi' : \{\tilde{x}\} \setminus \{x\} \cup \{y\} \rightarrow S \quad \chi'(z) = \begin{cases} 
\chi(z), & z \in \{\tilde{x}\} \setminus \{x, y\} \\
\chi(x) + \chi(y), & z = y \wedge y \in \tilde{x} \\
\chi(x), & z = y \wedge y \notin \{\tilde{x}\}
\end{cases}$$

rewriting system: $(QP, \Gamma \vdash G)$
(REN)

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

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

(COM)

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

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

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

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

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

**COM**
\[
\Gamma_1 \vdash G_1 \xrightarrow{\Lambda_1} \Gamma'_1 \vdash G'_1 \quad \Gamma_2 \vdash G_2 \xrightarrow{\Lambda_2} \Gamma'_2 \vdash G'_2 \quad \rho = \text{mgu}(\Lambda_1 \uplus \Lambda_2)
\]

\[
\bigwedge_{x \in \text{dom}(\Gamma_1) \cap \text{dom}(\Gamma_2)} \Gamma_1(x) = \Gamma_2(x)
\]

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

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

\[
\Omega(x) = \begin{cases} 
(t, \tilde{y} \rho), & t = \prod_{(x, s, \tilde{y}) \in \Omega \atop \in \text{Sync}} s \notin \text{Sync} \\
(t, \emptyset), & t = \prod_{(x, s, \tilde{y}) \in \Omega \atop \in \text{Sync}} s \in \text{Sync}
\end{cases}
\]

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

The weighting function induced by \( \Gamma \) and \( \Omega \) is

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

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

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 $\mathcal{H} \mathcal{R}$ given by the cartesian product of the Hoare synchronisations c-semiring $\mathcal{H} = \langle \mathcal{H}, +_H, \star_H, 0_H, 1_H \rangle$, where

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

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

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 \((n < u)\)

\[
x \bullet \rightarrow R \rightarrow y \bullet (\alpha, +\infty)
\]

\[
x : (0_H, u), y : 0 \triangleright R(x, y) \xrightarrow{x \rightarrow (a, n)} R(x, z) \upharpoonright R(z, y) \upharpoonright l(z)
\]

**Accept Synchronisation R**

\[
x \bullet (b, +\infty) \xrightarrow{x \rightarrow (b, +\infty)} R(x, y)
\]

\[
x : (0_H, +\infty), y : 0 \triangleright R(x, y) \xrightarrow{x \rightarrow (b, +\infty)} R(x, y)
\]

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

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

\[
x : 0, y : 0 \triangleright R_{r}^{u}(x, y) \quad \xrightarrow{x \mapsto (c, u), y \mapsto (c, +\infty)} \quad R(x, z) \mid R_{r-1}^{u}(z, y)
\]

\[
x : 0, y : 0 \triangleright R_{0}^{u}(x, y) \quad \xrightarrow{x \mapsto (c, u), y \mapsto (c, +\infty)} \quad R(x, y) \mid l(y)
\]

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

Accept Synchronisation Init

\[
\begin{align*}
R & \quad \rightarrow \quad R \\
y \cdot (c, +\infty) & \quad x \cdot (c, +\infty) \\
x : 0, y : 0 & \triangleright R(x, y) \quad x \mapsto (c, +\infty) \\
y \mapsto (c, +\infty) & \quad R(x, y)
\end{align*}
\]

Accept Synchronisation Gate

\[
\begin{align*}
G & \quad \rightarrow \quad G \\
y \cdot (b, +\infty) & \quad x \cdot (\beta, +\infty) \\
x : 0, y : 0 & \triangleright G(x, y) \quad x \mapsto (\beta, +\infty) \\
y \mapsto (b, +\infty) & \quad G(x, y)
\end{align*}
\]

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

\[
\begin{align*}
(a, 2) &= (a, 2) \times (a, +\infty) = (a \ast_H a, \min(2, +\infty)) \\
(a, 3) &= (a, 3) \times (a, +\infty) = (a \ast_H a, \min(3, +\infty)).
\end{align*}
\]
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.
The ring case study

Note that • and • now are internal.
The ring case study

\[ \text{InitRing}(r = 1, u = 2) \times \text{InitRing}(r = 1, u = 3) \times \text{Accept}^* \]
Named sets & named functions

Let \( \mathcal{N} \) be a set of names, and \( \text{sym}(N) \overset{\Delta}{=} \{ \rho \in \text{Aut}(\mathcal{N}) \mid \forall x \not\in N. \rho(x) = x \} \), if \( N \subseteq \mathcal{N} \),

States of HD-automata are defined by named sets

**Definition 1** Permutation algebra \( \langle S, O \rangle \)
1. the carrier \( S \) is a set and
2. \( O \subseteq \{ \hat{\rho} : S \rightarrow S \mid \rho \in \text{Aut}(\mathcal{N}) \} \) are s.t.
   - \( \hat{id} \in O \) and, for all \( x \in S \)
   - \( x \cdot \hat{id} = x \)
   - \( \forall \rho_1, \rho_2 \in \text{Aut}(\mathcal{N}). \ x \rho_1 \rho_2 = (x \rho_1) \rho_2 \)

**Definition 2** Named set (ns) \( \langle Q, g \rangle \)
1. \( Q \) is a permutation algebra;
2. \( g : Q \rightarrow \bigcup_{N \in \varphi_{\text{fin}}(\mathcal{N})} \{ \text{sym}(N) \} \) s.t.
   \[ \forall \rho \in g(q). q = q \hat{\rho}. \]
   \[ |q| = \text{dom}(\rho) \in g(q) \text{ are the names of } q. \]
   \[ \|q\| \text{ is the cardinality of } |q|. \]

Transitions among states are represented by means of named functions

**Definition 3** A named function \( \langle h : S \rightarrow D, \Sigma \rangle \) is s.t. \( S \) and \( D \) are ns, \( h \) is a function from \( Q_S \) to \( Q_D \) and \( \forall q \in S. \Sigma(q) \in \varphi_{\text{fin}}(\{ \|q\|_S \cup \{\ast\} \}^{h(q)|D}) \) s.t.
1. \( \forall \sigma \in \Sigma(q). gD(h(q)); \sigma = \Sigma(q), \)
2. \( \forall \sigma \in \Sigma(q). \sigma; gS(q) \subseteq \Sigma(q), \)
3. any function of \( \Sigma(q) \) is injective.
The category **NS** has named sets as objects and named functions and
1. \( \bot = \langle \emptyset, \emptyset \rangle \) is initial object, \( I = \langle \{\ast\}, \ast \mapsto \emptyset \rangle \) is the terminal object and
2. the covariant powerset functor on **Set** is \( \wp_{\text{fin}}(D) = \langle \wp_{\text{fin}}(Q_D), g \rangle \), where, given
   \( Q \subseteq Q_D \), \( g(Q) = \{ \rho \mid \rho \) is a permutation over \( \bigcup_{q \in Q} |q| \} \land Q \rho = Q \).

**Definition 4** Given a ns \( L \), a **HD-automaton** over \( L \) is a coalgebra for

\[
T_L(D) = \wp_{\text{fin}}(L \otimes D)
\]

where the pairing operation \( D \otimes E = \langle Q_D \times Q_E, g \rangle \) is s.t.
\[
g : Q_D \times Q_E \to \bigcup_{N,M \in \wp_{\text{fin}}(N)} \{ \text{sym}(N) + \text{sym}(M) \} \text{ where}
\]
\[
g(d, e) = \{ \rho_1 + \rho_2 \mid \rho_1 \in g_D(d) \land \rho_2 \in g_E(e) \}
\]

(formally, \( D \otimes E \) is not a ns but \( g(d, e) \) is a symmetry on \( |d| + |e| \))
Minimising HD-automata

Fixed a HD-automaton $K$ on the functor $T_L(D) = \varphi_{\text{fin}}(L \otimes D)$, $T : \textbf{NS} \rightarrow \textbf{NS}$ is the functor s.t.

$$T(D) = \begin{cases} T_L(D) & D \in \text{obj}\,(\textbf{NS}) \\ \langle h, \Sigma \rangle & D = \langle h_D, \Sigma_D \rangle \in \textbf{NS}(E, F) \text{ for } E, F \in \text{obj}\,(\textbf{NS}) \end{cases}$$

where,

$$\begin{align*}
\text{h}(B) &= \{ \langle l, \text{h}_D(q) \rangle \mid \langle l, q \rangle \in B \} \\
\Sigma(B) &= \{ \langle l, \text{h}_D(q), \sigma; \sigma' \rangle \mid \langle l, q, \sigma \rangle \in B \land \langle l, q', \sigma' \rangle \in \Sigma_D(q) \}
\end{align*}$$

$$B \in \varphi_{\text{fin}}(L \otimes E),$$
Fixed a HD-automaton $K$ on the functor $T_L(D) = \varphi_{\text{fin}}(L \otimes D)$, $T : \textbf{NS} \to \textbf{NS}$ is the functor s.t.

$$T(D) = \begin{cases} T_L(D) & D \in \text{obj}(\textbf{NS}) \\ \langle h, \Sigma \rangle & D = \langle h_D, \Sigma_D \rangle \in \textbf{NS}(E, F) \text{ for } E, F \in \text{obj}(\textbf{NS}) \end{cases}$$

where,

$$h(B) = \{ \langle l, h_D(q) \rangle \mid \langle l, q \rangle \in B \} \quad \B \in \varphi_{\text{fin}}(L \otimes E),$$

$$\Sigma(B) = \{ \langle l, h_D(q), \sigma; \sigma' \rangle \mid \langle l, q, \sigma' \rangle \in B \wedge \langle l, q', \sigma' \rangle \in \Sigma_D(q) \}$$

A normalisation functor $N$ is any functor s.t. $N(D)$ is isomorphic to a subset of $D$. 
Minimising HD-automata

Fixed a HD-automaton \( K \) on the functor \( T_L(D) = \varphi_{\text{fin}}(L \otimes D) \), \( T : \mathbf{NS} \to \mathbf{NS} \) is the functor s.t.

\[
T(D) = \begin{cases} 
T_L(D) & D \in \text{obj}(\mathbf{NS}) \\
\langle h, \Sigma \rangle & D = \langle h_D, \Sigma_D \rangle \in \mathbf{NS}(E, F) \text{ for } E, F \in \text{obj}(\mathbf{NS})
\end{cases}
\]

where,

\[
\begin{align*}
    h(B) &= \{ \langle l, h_D(q) \rangle \mid \langle l, q \rangle \in B \} \\
\Sigma(B) &= \{ \langle l, h_D(q), \sigma; \sigma' \rangle \mid \langle l, q, \sigma' \rangle \in B \land \langle l, q', \sigma' \rangle \in \Sigma_D(q) \}
\end{align*}
\]

A normalisation functor \( N \) is any functor s.t. \( N(D) \) is isomorphic to a subset of \( D \).

The minimisation algorithm on a \( T_L \) coalgebra \((D, K : D \to T_L(D))\) is

\[
\begin{align*}
    H_0 &= (q \mapsto \bot, q \mapsto \emptyset), \text{ where } \text{dom}(H_0) = D \\
    H_{i+1} &= K ; N(T(H_i)),
\end{align*}
\]

where \( N \) is a normalisation functor.
Fixed a HD-automaton $K$ on the functor $T_L(D) = \phi_{\text{fin}}(L \otimes D)$, $T : \text{NS} \to \text{NS}$ is the functor s.t.

$$T(D) = \begin{cases} T_L(D) & D \in \text{obj}(\text{NS}) \\ \langle h, \Sigma \rangle & D = \langle h_D, \Sigma_D \rangle \in \text{NS}(E, F) \text{ for } E, F \in \text{obj}(\text{NS}) \end{cases}$$

where,

$$h(B) = \{ \langle l, h_D(q) \rangle \mid \langle l, q \rangle \in B \}$$

$$\Sigma(B) = \{ \langle l, h_D(q), \sigma; \sigma' \rangle \mid \langle l, q, \sigma' \rangle \in B \land \langle l, q', \sigma' \rangle \in \Sigma_D(q) \}$$

A normalisation functor $N$ is any functor s.t. $N(D)$ is isomorphic to a subset of $D$.

The minimisation algorithm on a $T_L$ coalgebra $(D, K : D \to T_L(D))$ is

$$H(0) \triangleq \langle q \mapsto \bot, q \mapsto \emptyset \rangle, \text{ where } \text{dom}(H(0)) = D$$

$$H(i+1) \triangleq K; N(T(H(i)))$$

where $N$ is a normalisation functor

- All the states of $K$ are initially considered equivalent
- At the $(i + 1)$-th step, $H(i)$ through $T$ is first normalised and then mapped through $K$
- at the end, the kernel yields the equivalence classes grouping equivalent states.
Theorem  The iterative partition refinement algorithm is convergent on finite HD-automata whenever the normalisation functor is monotone on nfs.

Proof.

By construction, $\varphi_{\text{fin}}(\_)$ is monotone, hence $T$ is monotone because it is the composition of two monotone functors. Therefore, $\mathcal{M} : H \mapsto K ; T(H)$ is monotone and finite. Finally, all nfs chains having finite domain are finite, hence, the iterative algorithm converges to the maximal fix-point of $\mathcal{M}$.

This proof mimics that in [FMT05a] with the difference that there only the case of the early semantics of $\pi$-calculus is dealt with, while here, the result is extended to the general case of finite HD-automata (with the only additional assumption that the normalisation functor is monotone).