Synchronised Hyperedge Replacement and Service Oriented Computing

Notes for BPESO: PhD School on Theory and Practice of Business Process Execution and Service Orientation

Emilio Tuosto

Computer Science Department,
University of Leicester, UK
et52@mcs.le.ac.uk

Abstract These notes are rather technical and only aims to give a written, formal account of a subset of the concepts that will be illustated more intuitivelly and informally during the lectures.
Definitions and examples are borrowed and adapted from [5] the authors of which deserve credit.

1 Hypergraphs

Notation 1.1 *Let V be a set, V^* is the set of tuples on V. We denote a tuple as $\mathbf{v} = \langle v_1, \ldots, v_n \rangle$, the empty tuple as $\langle\rangle$, the i-th element of $\mathbf{v}$ as $\mathbf{v}[i]$, and write $|\mathbf{v}|$ for the length of $\mathbf{v}$.*

Given a function f, $\mathrm{dom}(f)$ is its domain, and function $f|_S$ is the restriction of f to S, namely $f|_S(x) = f(x)$ if $x \in S$, $f|_S(x)$ is undefined otherwise. As usual $f \circ g$ is the composition of f and g befined by $(f \circ g)(x) = f(g(x))$.

For a syntactic structure s with names and binders, $\mathrm{fn}(s)$ is the set of its free names.

Let $\mathcal{N}$ be a countable infinite set of (names of) nodes and $\mathcal{L}$ be the set of edge labels. A label $L \in \mathcal{L}$ is assigned a *rank*, i.e., a natural number (denoted as $\mathrm{rank}(L)$). An edge labelled by L connects $\mathrm{rank}(L)$ nodes and a node connected to an edge is said to be an *attachment node* of that edge.

A *syntactic judgment* specifies a graph along with its interface, i.e., its *free nodes*.

Definition 1.1 (Graphs as judgements). *A* judgment *has form $\Gamma \vdash G$ where:*

1. *$\Gamma \subseteq \mathcal{N}$ is a finite set of names (the free nodes of the graph);*
2. *G is a* graph term *generated by the grammar*

$$G ::= L(\mathbf{x}) \quad | \quad G|G \quad | \quad \nu y\, G \quad | \quad nil$$

where $\mathbf{x} \in \mathcal{N}^*$ is a tuple of names, $L \in \mathcal{L}$, $\mathrm{rank}(L) = |\mathbf{x}|$ and $y \in \mathcal{N}$.

In $\nu y\, G$, restriction operator ν binds y in G, $\mathrm{fn}(G)$ and $\mathrm{bn}(()G)$ are defined as per Table 1. We demand that $\mathrm{fn}(G) \subseteq \Gamma$.

	$L(x_1,\ldots,x_n)$	$G\|G'$	νG	nil
fn(_)	$\{x_1,\ldots,x_n\}$	$\mathrm{fn}G \cup \mathrm{fn}G'$	$\mathrm{fn}G \setminus \{y\}$	$\emptyset$
bn(()_)	$\emptyset$	$\mathrm{bn}(G) \cup \mathrm{bn}(G')$	$\mathrm{fn}G \cup \{y\}$	$\emptyset$

Table 1. Free and bound nodes

Graph *nil* is the empty graph, | is the parallel composition operator of graphs (merging nodes with the same name) and νy is the restriction operator of nodes; free/bound nodes correspond to free/bound names. Edges are terms of the form $L(x_1,\ldots,x_n)$, where the x_i are arbitrary names and $\mathrm{rank}(L) = n$.

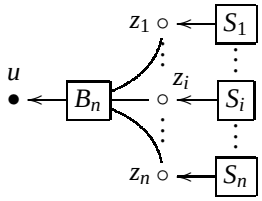
Exercise 1.2. Why Definition 1.1 requires $\mathrm{fn}(G) \subseteq \Gamma$? Give the judgment for the graph consisting only of node x. ◇

We assume that restriction has lower priority than parallel composition. For conciseness, curly brackets are dropped from interfaces Γ in judgements and Γ_1,Γ_2 denotes $\Gamma_1 \cup \Gamma_2$, provided that $\Gamma_1 \cap \Gamma_2 = \emptyset$ (e.g., $\Gamma,x = \Gamma \cup \{x\}$, if $x \notin \Gamma$).

Example 1.3. Consider the judgment

$$u \vdash \nu z_1,\ldots,z_n\ B_n(u,z_1,\ldots,z_n)|S_1(z_1)|\ldots|S_n(z_n)$$

which describes a system where many servers S_i are connected to the network via a manager B_n and can be graphically represented as:



Edges are drawn as rectangles and nodes are bullets (empty for bound nodes and solid for free nodes). A connection between a node and an edge is represented by a line, called *tentacle*; an arrowed tentacle indicates the first attachment node of the edge. The other nodes are determined by numbering tentacles clockwise (e.g., for B_n, u is the first attachment node, z_1 is the second and so on). ◊

Exercise 1.4. Draw a graph for

1. $x \vdash L(x) \mid \nu y M(x,y)$
2. $x \vdash L(x) \mid \nu x M(x,x)$
3. $x,y \vdash L(x) \mid \nu x M(x,x)$

◇

Definition 1.5 (Structural congruence on graph judgements). *Graph terms are considered up to axioms* $(\mathtt{AG1} \div 7)$ *below:*

$$(\mathtt{AG1})\ (G_1|G_2)|G_3 \equiv G_1|(G_2|G_3) \quad (\mathtt{AG2})\ G_1|G_2 \equiv G_2|G_1 \quad (\mathtt{AG3})\ G|nil \equiv G$$
$$(\mathtt{AG4})\ \nu x\ \nu y\ G \equiv \nu y\ \nu x\ G \quad (\mathtt{AG5})\ \nu x\ G \equiv G \text{ if } x \notin \mathrm{fn}(G)$$
$$(\mathtt{AG6})\ \nu x\ G \equiv \nu y\ G\{y/x\},\ \text{if } y \notin \mathrm{fn}(G) \quad (\mathtt{AG7})\ \nu x\ G_1|G_2 \equiv G_1|\nu x\ G_2,\ \text{if } x \notin \mathrm{fn}(G_1)$$

For judgments, we define $\Gamma_1 \vdash G_1 \equiv \Gamma_2 \vdash G_2$ *iff* $\Gamma_1 = \Gamma_2$ *and* $G_1 \equiv G_2$.

We consider judgements for graphs up to structural congruence which amounts to consider graphs up to graph isomorphisms that preserve free nodes, labels of edges, and tentacles [6].

2 Basic Milner SHR

In a synchronisation *a-la* Milner only two-parties interact: synchronisation actions are such that

- they are partitioned into "normal" actions a and co-actions $\overline{a}$
- $\overline{\overline{a}} = a$
- there is a special action ε standing for "not taking part to the synchronisation"
- there is an action τ representing a complete synchronisation.

Edges attached to a node of an hypergraphs can be synchronised with a Milner synchronisation if two of them "exhibit" two complementary actions while all the others stay idle (i.e., they all exhibit action ε). The result of the synchronisation is τ.

Notation 2.1 *A* renaming *is a function* $\sigma : \mathcal{N} \to \mathcal{N}$, $x\sigma$ *is the application of* σ *to* $x \in \mathrm{dom}(\sigma)$ *and yields* $\sigma(x)$. *Moreover,*

- *if* $\sigma \circ \sigma = \sigma$, *the renaming is said* idempotent
- *a renaming* σ *is* injective *when* σ *is injective*
- *renaming* $\{x/y\}$ *is defined such that* $\{x/y\}(y) = x$ *and* $\{x/y\}(z) = z$ *for all* $z \neq y$ *in the domain of* $\{x/y\}$
- *as usual,* $\{(x,y) \mid x \in \mathrm{dom}(f)\ \wedge\ y = f(x)\}$ *is the* graph *of a function* f.

Definition 2.1 (SHR transitions and productions). *A relation* $\Gamma \vdash G \xrightarrow{\Lambda} \Gamma \vdash G'$ *is an* SHR transition *if* $\Gamma \vdash G$ *and* $\Gamma \vdash G'$ *are judgments for graphs, and* $\Lambda : \Gamma \to \mathfrak{A}$ *is a total function, where* $\mathfrak{A}$ *is a set of actions.*

A production *is an SHR transition of the form:*

$$x_1,\ldots,x_n \vdash L(x_1,\ldots,x_n) \xrightarrow{\Lambda} x_1,\ldots,x_n \vdash G \tag{1}$$

where $\mathrm{rank}(L) = n$ *and* $x_1,\ldots,x_n$ *are all distinct. Production (1) is* idle *iff* $\Lambda(x_i) = \varepsilon$ *for each* i *and* G *is* $L(x_1,\ldots,x_n)$.

A transition is obtained by composing productions in a set $\mathcal{P}$ that contains any idle production and is closed under all injective renamings (that is, the application of an injective renaming to a productions in $\mathcal{P}$ yields productions in $\mathcal{P}$).

Composition is performed by merging nodes and thus connecting the edges. Synchronisation conditions as specified in productions must be satisfied.

Definition 2.2 (Inference rules for bMSHR). *The admissible behaviours of bMSHR are defined by the following inference rules.*

$$\textit{(par-b)}\quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda} \Gamma \vdash G_2 \qquad \Gamma' \vdash G_1' \xrightarrow{\Lambda'} \Gamma' \vdash G_2' \qquad \Gamma \cap \Gamma' = \emptyset}{\Gamma,\Gamma' \vdash G_1 | G_1' \xrightarrow{\Lambda \cup \Lambda'} \Gamma,\Gamma' \vdash G_2 | G_2'}$$

$$\textit{(merge-b)}\quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda} \Gamma \vdash G_2}{\Gamma\sigma \vdash G_1\sigma \xrightarrow{\Lambda'} \Gamma\sigma \vdash G_2\sigma}$$

where $\sigma : \Gamma \to \Gamma$ *is an idempotent renaming and:*

1. *for all* $x,y \in \Gamma$ *such that* $x \neq y$, *if* $x\sigma = y\sigma$, $\Lambda(x) \neq \varepsilon$ *and* $\Lambda(y) \neq \varepsilon$ *then* $(\forall z \in \Gamma \setminus \{x,y\}. z\sigma = x\sigma \Rightarrow \Lambda(z) = \varepsilon) \ \wedge\ \Lambda(x) = a \ \wedge\ \Lambda(y) = \overline{a} \ \wedge\ a \neq \tau$
2. $\Lambda'(z) = \begin{cases} \tau & \text{if } x\sigma = y\sigma = z \ \wedge\ x \neq y \ \wedge\ \Lambda(x) \neq \varepsilon \ \wedge\ \Lambda(y) \neq \varepsilon \\ \Lambda(x) & \text{if } x\sigma = z \ \wedge\ \Lambda(x) \neq \varepsilon \\ \varepsilon & \text{otherwise} \end{cases}$

$$\textit{(res-b)}\quad \frac{\Gamma, x \vdash G_1 \xrightarrow{\Lambda} \Gamma, x \vdash G_2 \qquad \Lambda(x) = \varepsilon \vee \Lambda(x) = \tau}{\Gamma \vdash \nu x\ G_1 \xrightarrow{\Lambda|_\Gamma} \Gamma \vdash \nu x\ G_2}$$

$$\textit{(new-b)}\quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda} \Gamma \vdash G_2 \qquad x \notin \Gamma}{\Gamma, x \vdash G_1 \xrightarrow{\Lambda \cup \{(x,\varepsilon)\}} \Gamma, x \vdash G_2}$$

Rule (par-b) deals with the composition of transitions which have disjoint sets of nodes and rule (merge-b) allows to merge nodes. Condition 1 requires that at most two non ε actions are performed on nodes to be merged. If they are exactly two then they have to be complementary, and the resulting action is τ (condition 2). Since σ is required to be idempotent, it yields an equivalence relation on Γ and a choice of a standard representative. In fact, $x,y \in \Gamma$ are equivalent under σ iff $x\sigma = y\sigma$; the representative element of the equivalence class of x is $x\sigma$. Rule (res-b) binds node x. This is allowed only if either τ or ε actions are performed on x, forcing either a complete synchronisation (τ) or no synchronisation (ε). Rule (new-b) allows to add to the source graph an isolated free node where an action ε is performed.

Example 2.3. Consider an instance of the system in Example 1.3 where edge $B_2(u,z_1,z_2)$ takes requests on node u and broadcasts them to $S_1(z_1)$ and $S_2(z_2)$ by synchronising on nodes z_1 and z_2, respectively. The productions for B_2 and S_i ($i \in \{1,2\}$) are:

$$u, z_1', z_2' \vdash B_2(u,z_1',z_2') \xrightarrow{(u,\mathrm{req}),(z_1',\overline{\mathrm{req}}),(z_2',\overline{\mathrm{req}})} u, z_1', z_2' \vdash B_2(u,z_1',z_2') \tag{2}$$

$$z_i \vdash S_i(z_i) \xrightarrow{(z_i,\mathrm{req})} z_i \vdash S_i'(z_i) \tag{3}$$

The inference rules for bMSHR can be used to derive transition

$$u, z_1, z_2 \vdash B_2(u,z_1,z_2) | S_1(z_1) | S_2(z_2) \xrightarrow{(u,\mathrm{req})} u \vdash \nu z_1, z_2\ B_2(u,z_1,z_2) | S_1'(z_1) | S_2'(z_2)$$

$\Diamond$

Exercise 2.4. Give a derivation of the transition of Example 2.3. $\diamond$

3 Action Signatures

Mobility is added to SHR transitions (Definition 2.1), according to the approach of [4], which allows existing and newly created nodes to be merged.

Definition 3.1 (Action signature). *An* action signature *is a triple* $(\mathfrak{A}, \mathrm{ar}, \varepsilon)$ *where* $\mathfrak{A}$ *is the set of actions,* $\varepsilon \in \mathfrak{A}$, *and* $\mathrm{ar} : \mathfrak{A} \to \mathbb{N}$ *is the arity function satisfying* $\mathrm{ar}(\varepsilon) = 0$.

Mobility is modelled by letting function Λ in transitions to carry tuples of nodes. Hereafter, $\Lambda : \Gamma \to \mathfrak{A} \times \mathcal{N}^*$ is a total function assigning, to each node $x \in \Gamma$, an action $a \in \mathfrak{A}$ and a tuple $\mathbf{y}$ of nodes such that $\mathrm{ar}(a) = |\mathbf{y}|$. We let $\mathrm{act}_\Lambda(x) = a$ and $\mathrm{n}_\Lambda(x) = \mathbf{y}$ when $\Lambda(x) = (a,\mathbf{y})$. Finally, the *set of communicated (resp. fresh) names* of Λ is $\mathrm{n}(\Lambda) = \{z \mid \exists x. z \in \mathrm{n}_\Lambda(x)\}$ (resp. $\Gamma_\Lambda = \mathrm{n}(\Lambda) \setminus \Gamma$).

Definition 3.2 (SHR transitions with mobility). *Given an action signature* $(\mathfrak{A}, \mathrm{ar}, \varepsilon)$*, an* SHR transition with mobility *is a relation of the form:*

$$\Gamma \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'$$

where $\pi : \Gamma \to \Gamma$ *is an idempotent renaming accounting for node merging such that* $\forall x \in \mathrm{n}(\Lambda).\ x\pi = x$*. Finally,* $\Phi = \Gamma\pi \cup \Gamma_\Lambda$.

Condition $\forall x \in \mathrm{n}(\Lambda).\ x\pi = x$ states that only representatives nodes can be communicated while $\Phi = \Gamma\pi \cup \Gamma_\Lambda$ states that free nodes are never erased ($\supseteq$) and new nodes are bound unless communicated ($\subseteq$).

Remark 3.3. Φ is fully determined by Λ and π (since $\Gamma = \mathrm{dom}(\Lambda)$) and, unlike in bMSHR, it might be $\Phi \neq \Gamma$. •

The definition of productions is extended as follows.

Definition 3.4 (Productions). *A* production *is now an SHR transition of the form:*

$$x_1, \ldots, x_n \vdash L(x_1, \ldots, x_n) \xrightarrow{\Lambda,\pi} \Phi \vdash G \tag{4}$$

where $\mathrm{rank}(L) = n$ *and* $x_1, \ldots, x_n$ *are all distinct. Production (4) is idle if* $\Lambda(x_i) = (\varepsilon, \langle\rangle)$ *for each* i*,* $\pi = id$ *and* $\Phi \vdash G = x_1, \ldots, x_n \vdash L(x_1, \ldots, x_n)$.

As before, sets of productions include all the idle productions and are closed under injective renamings.

4 Milner SHR

Milner SHR is presented below and extends bMSHR with the machinery to deal with mobility. The action signature (with mobility) $(\mathfrak{A}_\mathfrak{M}, \mathrm{ar}, \varepsilon)$ for Milner synchronisation has further structure wrt Definition 3.1.

Definition 4.1. *An* action signature for Milner synchronisation *is an action signature* $\mathfrak{A}_\mathfrak{M} = \mathcal{A} \cup \overline{\mathcal{A}} \cup \{\tau, \varepsilon\}$ *where*

- $\mathcal{A}$ *is the set of (input) actions and* $\overline{\mathcal{A}} = \{\overline{a} \mid a \in \mathcal{A}\}$ *is the set of* co-actions*;*
- $a = \overline{\overline{a}}$*;*
- τ *is a special action such that* $\mathrm{ar}(\tau) = 0$*;*
- *for each* $a \in \mathcal{A}$*,* $\mathrm{ar}(a) = \mathrm{ar}(\overline{a})$.

MSHR semantics (and the other SHR extensions) exploits a *most general unifier (mgu)* accounting for name fusions. The result of the application of the mgu is the fusion of nodes (new and old ones) changing the topology of graph (i.e. mobility).

Definition 4.2 (Inference rules for MSHR). *The admissible behaviours of MSHR are defined by the following inference rules.*

$$\textit{(par-M)}\quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2 \qquad \Gamma' \vdash G_1' \xrightarrow{\Lambda',\pi'} \Phi' \vdash G_2' \qquad (\Gamma \cup \Phi) \cap (\Gamma' \cup \Phi') = \emptyset}{\Gamma, \Gamma' \vdash G_1 | G_1' \xrightarrow{\Lambda \cup \Lambda', \pi \cup \pi'} \Phi, \Phi' \vdash G_2 | G_2'}$$

$$\textit{(merge-M)}\quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma\sigma \vdash G_1\sigma \xrightarrow{\Lambda',\pi'} \Phi' \vdash \nu U\ G_2\sigma\rho}$$

where $\sigma : \Gamma \to \Gamma$ *is an idempotent renaming and:*

1. *for all* $x, y \in \Gamma$ *such that* $x \neq y$*, if* $x\sigma = y\sigma \ \wedge\ \Lambda(x) \neq \varepsilon \ \wedge\ \Lambda(y) \neq \varepsilon$ *then* $(\forall z \in \Gamma \setminus \{x, y\}.z\sigma = x\sigma \Rightarrow \Lambda(z) = \varepsilon) \ \wedge\ \Lambda(x) = a \ \wedge\ \Lambda(y) = \overline{a} \ \wedge\ a \neq \tau$
2. $S = \{x = y \mid x\pi = y\pi\} \cup \{\mathrm{n}_\Lambda(x) = \mathrm{n}_\Lambda(y) \mid x\sigma = y\sigma\}$
3. $\rho = \mathrm{mgu}(S)\sigma$ *and* ρ *maps names to representatives in* $\Gamma\sigma$ *whenever possible*
4. $\Lambda'(z) = \begin{cases} (\tau, \langle\rangle) & \text{if } x\sigma = y\sigma = z \ \wedge\ x \neq y \ \wedge\ \mathrm{act}_\Lambda(x) \neq \varepsilon \ \wedge\ \mathrm{act}_\Lambda(y) \neq \varepsilon \\ (\Lambda(x))\sigma\rho & \text{if } x\sigma = z \ \wedge\ \mathrm{act}_\Lambda(x) \neq \varepsilon \\ (\varepsilon, \langle\rangle) & \text{otherwise} \end{cases}$
5. $\pi' = \rho\lfloor_{\Gamma\sigma}$
6. $U = (\Phi\sigma\rho) \setminus \Phi'$

$$\textit{(res-M)}\quad \frac{\Gamma, x \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma \vdash \nu x\ G_1 \xrightarrow{\Lambda\lfloor_\Gamma, \pi\lfloor_\Gamma} \Phi' \vdash \nu Z\ G_2}$$

where:

- $(\exists y \in \Gamma . x\pi = y\pi) \Rightarrow x\pi \neq x$
- $\mathrm{act}_\Lambda(x) = \varepsilon \vee \mathrm{act}_\Lambda(x) = \tau$
- $Z = \{x\}$ *if* $x \notin \mathrm{n}(\Lambda|_\Gamma), Z = \emptyset$ *otherwise*

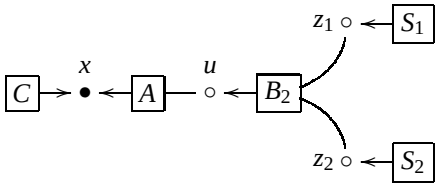
$$\textit{(new-M)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2 \qquad x \notin \Gamma \cup \Phi}{\Gamma, x \vdash G_1 \xrightarrow{\Lambda \cup \{(x,\varepsilon,\langle\rangle)\},\pi} \Phi, x \vdash G_2}$$

Rules (par-M) and (new-M) are essentially as before. In rule (merge-M) now mobility must be handled; indeed, when actions and co-actions synchronise, parameters in corresponding positions are merged. This set of merges is computed in S (condition 2). Condition 3 updates the equations with σ and then chooses a representative for each equivalence class using an mgu; among the possible equivalent mgus we choose one of those where nodes in $\Gamma\sigma$ are chosen as representatives (if they are in the equivalence class). This is necessary to avoid unexpected renamings of nodes because of fusions with new nodes which may then disappear.

Remark 4.3. Λ is updated with the merges specified by ρ (condition 4) and, π' is ρ restricted to the nodes of the graph which is the source of the transition (condition 5). •

Restrictions should be reintroduced (condition 6) when nodes are extruded by the synchronised actions, since they will no more appear in the label. In rule (res-M) the bound node x must not be a representative if it belongs to a non trivial equivalence class.

Example 4.4. Consider the system in Example 2.3 with two servers S_1 and S_2, but where a client C must be first authenticated by an authority A. The graph representing the system is as follows:



We can model the fact that C is allowed to access the services by letting it move from node x to node u, namely by extruding the private node u to C. The productions for C and A are as follows:

$$x \vdash C(x) \xrightarrow{(x,\overline{\mathrm{auth}},\langle y\rangle)} x, y \vdash C'(y) \qquad\qquad x, u \vdash A(x,u) \xrightarrow{(x,\mathrm{auth},\langle u\rangle)} x, u \vdash A(x,u)$$

where, in the first production the client becomes attached to the received node y after the transition. In fact, when synchronisation is performed, new node y and node u are merged, with u as representative. Note that the restriction on u is reintroduced. Starting from $x \vdash \nu u\, C(x) \mid A(x,u)$ we will obtain $x \vdash \nu u\, C'(u) \mid A(x,u)$. ◊

Exercise 4.5. Derive the transition from $x \vdash \nu u\, C(x) \mid A(x,u)$ to $x \vdash \nu u\, C'(u) \mid A(x,u)$ using the productions of Example 4.4 and the rules of Definition 4.2. ◇

5 Synchronisation Algebras

Synchronisation Algebras with Mobility (SAMs) allow us to parameterise SHR with respect to different synchronisation models. For example, MSHR will come out as just an instance of the general framework.

SHR can be parameterised with respect to the synchronisation policy by using SAMs Also, SHR for heterogeneous systems where different subsystems exploit different synchronisation protocols can be modelled through SAMs [9,8]; heterogeneity is introduced by labelling nodes with SAMs that specify the synchronisation policy used on them. SAMs labelling can dynamically change as a result of synchronisation among different parties.

Example 5.1. The system of Example 4.4 can be now more accurately modelled by simultaneously using a SAM for Milner synchronisation on actions for authorisation, and one for broadcast of requests. Thus on each node only the desired actions are available. This avoids undesired executions caused by malicious clients. Available synchronisations are exploited by the authority to ensure that clients can issue only authorised requests. Also, actions can specify the synchronisation policy (e.g, Milner or broadcast synchronisation) so that clients dynamically choose what protocol to use. ◊

See [9,5,8] for details.

6 SHReQ: Coordinating Application Level QoS

Awareness of *Quality of Service* (QoS) is an emergent exigency in SOC which is no longer considered only as a low-level aspect of systems. The ability of formally specifying and programming QoS requirements may represent a significant added-value of the SOC paradigm. Moreover, QoS information can drive the design and development of programming interfaces and languages for QoS-aware middlewares as well as to drive the search-bind cycle of SOC.

In SHReQ, a calculus based on SHR, abstract high-level QoS requirements are expressed as *constraint-semiring* [1] and embedded in the rewriting mechanism which is parameterised with respect to a given c-semiring. Basically, values of c-semirings are synchronisation actions so that synchronising corresponds to the product operation of c-semirings that can be regarded as the simultaneous satisfaction of the QoS requirements of the participants to the synchronisation.

Definition 6.1 (C-semiring). *An algebraic structure* $\langle S, +, \cdot, \mathbf{0}, \mathbf{1}\rangle$ *is a* constraint semiring *if* S *is a set with* $\mathbf{0}, \mathbf{1} \in S$*, and* $+$ *and* $\cdot$ *are binary operations on* S *such that:*

- $+$ *is commutative, associative, idempotent,* $\mathbf{0}$ *is its unit element and* $\mathbf{1}$ *is its absorbing element (i.e.,* $a + \mathbf{1} = \mathbf{1}$*, for any* $a \in S$*);*
- $\cdot$ *is commutative, associative, distributes over* $+$*,* $\mathbf{1}$ *is its unit element, and* $\mathbf{0}$ *is its absorbing element (i.e.,* $a \cdot \mathbf{0} = \mathbf{0}$*, for any* $a \in S$*).*

The additive operation $(+)$ of a c-semiring induces a partial order on S defined as $a \leq_S b \iff \exists c : a + c = b$. The minimum is thus $\mathbf{0}$ and the maximum is $\mathbf{1}$. C-semirings have two distinguished features that result very useful for modelling abstract QoS. First, the cartesian product of c-semirings is still a c-semiring, hence we can uniformly deal with many different quantities simultaneously. Second, partial order $\leq_S$ provides a mechanism of choice. These features make c-semirings suitable for reasoning about multi-criteria QoS issues [2,3]. The fact that c-semiring structure is preserved by cartesian product is here exploited to compose synchronisation policies.

Example 6.2. The following examples introduce some c-semirings together with their intended application to model QoS attributes. A more complete list can be found in [1].

- The boolean c-semiring $\langle \{true, false\}, \vee, \wedge, false, true\rangle$ can be used to model network and service availability.
- The optimisation c-semiring $\langle \text{Real}, min, +, +\infty, 0\rangle$ applies to a wide range of cases, like prices or propagation delay.
- The max/min c-semiring $\langle \text{Real}, max, min, 0, +\infty\rangle$ can be used to formalise bandwidth, while the corresponding c-semiring over the naturals $\langle \mathbb{N}, max, min, 0, +\infty\rangle$ can be applied for resource availability.
- Performance can be represented by the probabilistic c-semiring $\langle [0,1], max, \cdot, 0, 1\rangle$.
- Security degrees are modelled via the c-semiring $\langle [0, 1, \ldots, n], max, min, 0, n\rangle$, where n is the maximal security level (unknown) and 0 is the minimal one (public).

$\Diamond$

Exercise 6.3. Prove that the structures listed in the Example 6.2 are c-semirings. $\diamond$

Hereafter, given a c-semiring $\langle S, +, \cdot, \mathbf{0}, \mathbf{1}\rangle$, $\text{ar}_S : S \to \mathbb{N}$ is an arity function assigning arities to values in S. Graphs in SHReQ are called *weighted graphs* because values is S are used as weights and record quantitative information on the computation of the system. We write $x_1 : s_1, \ldots, x_n : s_n \vdash G$ for the weighted graph whose weighting function maps x_i to s_i, for $i \in \{1, \ldots, n\}$.

SHReQ rewriting mechanism relies on c-semirings where additional structure is defined. More precisely, we assume sets *Sync*, *Fin* and *NoSync* such that

- $Sync \subseteq Fin \subseteq S$, $\mathbf{1} \in Sync$ and $\text{ar}_S(s) = 0$ if $s \in Sync$;
- $NoSync \subseteq S \setminus Fin$, $\mathbf{0} \in NoSync$ and $\forall s \in S. \forall t \in NoSync. s \cdot t \in NoSync$.

The intuition is that *Fin* contains those values of S representing events of complete synchronisations. Among the actions in *Fin* we can select a subset of "pure" synchronisation actions, namely complete synchronisations that do not expose nodes. Set *NoSync*, on the contrary, contains the values that represent "impossible" synchronisations.

SHReQ productions follow the lines of Definition 3.4 and 4.2, but have a slightly different interpretation.

Remark 6.4. For simplicity, we avoid the π component in SHReQ transitions and require that free nodes cannot be merged. Technically, this is obtained by considering undefined the most general unifier operation when it yields the fusion of two free nodes. In [7] the general unification is defined for SHReQ. •

Definition 6.5 (SHReQ productions). *Let S be a c-semiring $\langle S,+,\cdot,\mathbf{0},\mathbf{1}\rangle$. A* SHReQ production *is a production*

$$\Gamma \vdash L(x_1,\ldots,x_n) \xrightarrow{\Lambda} \Phi \vdash G \tag{5}$$

built on top of the action signature $(S,\mathrm{ar}_S,\mathbf{1})$ where Γ maps nodes in $\{x_1,\ldots,x_n\}$ to S.

Production (5) states that, in order to replace L with G in a graph H, applicability conditions expressed by the function Γ on the attachment nodes of L must be satisfied in H and, henceforth, L "contributes" to the rewriting by offering Λ in the synchronisation with adjacent edges. Function Γ expresses the *minimal* QoS requirements on the environment in order to apply the production, i.e., given $x \in \mathrm{dom}(\Gamma)$, the weight w on the node corresponding to x must satisfy $\Gamma(x) \leq w$. As before, function Φ is fully determined by Γ and Λ, where the weight of new nodes is set to $\mathbf{1}$ (i.e., $\Phi(y) = \mathbf{1}$ if $y \in \Gamma_\Lambda$), while for old nodes it traces the result of the synchronisation performed on them.

In production (5), c-semiring values play different roles in Γ and Λ: in Γ, they are interpreted as the minimal requirements to be fulfilled by the environment; in Λ they are the "contribution" that L yields to the synchronisation with the surrounding edges.

We only give the inference rule (merge-s) for merging nodes, the other rules being a simple rephrasing of those seen in previous sections. Rule (merge-s) is defined as:

$$\text{(merge-s)}\quad \frac{\Gamma, x:r, y:s \vdash G_1 \xrightarrow{\Lambda\cup\{(x,s_1,\mathbf{v}_1),(y,s_2,\mathbf{v}_2)\}} \Phi \vdash G_2}{\Gamma, x:r+s \vdash G_1\sigma \xrightarrow{\Lambda'} \Phi' \vdash \nu U\ G_2\sigma\rho}$$

with $\sigma = \{x/y\}$ and

1. $S = \{x = y\} \cup \{\mathbf{v}_1[1] = \mathbf{v}_2[1], \ldots, \mathbf{v}_1[n] = \mathbf{v}_2[n] \mid n = |\mathbf{v_1}| = |\mathbf{v_2}|\}$
2. $\rho = \mathrm{mgu}(S\sigma)$ and ρ maps names to representatives in Γ, x whenever possible
3. $\Lambda'(z) = \begin{cases} (s_1 \cdot s_2, \mathbf{w}) \text{ if } z = x \\ \Lambda(z)\sigma\rho \text{ for each } z \in \Gamma \end{cases}$
4. $\pi' = \rho|_{\Gamma,x}$
5. $U = \Phi\sigma\rho \setminus \Phi'$

In order to ensure applicability of productions also when there are more resources available than required, the following rule is introduced.

$$\text{(order-s)}\quad \frac{\Gamma, x:r \vdash G_1 \xrightarrow{\Lambda} \Phi \vdash G_2 \qquad r \leq t}{\Gamma, x:t \vdash G_1 \xrightarrow{\Lambda} \Phi \vdash G_2}$$

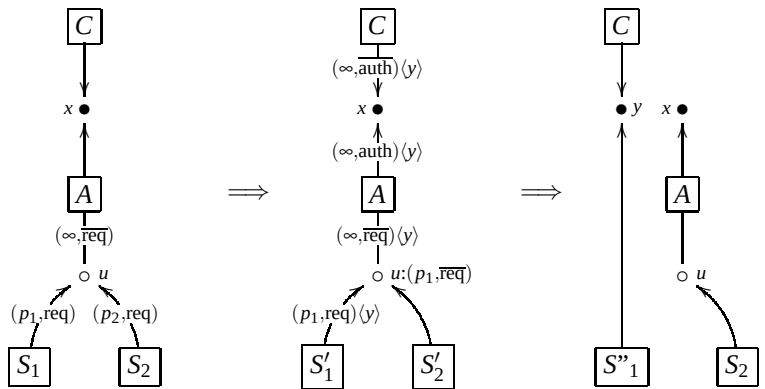
The other rules are similar to the ones in Definition 4.2.

Example 6.6. Let us consider Example 4.4. We can model the authority choosing the server that offers the cheapest service. To this aim, we use the cartesian product of two c-semirings. The first c-semiring is: $\langle R^+, max, min, 0, \infty\rangle$, for the price of the service. The second c-semiring is used for synchronisation. In this way, we are able to define a general synchronisation policy as a unique c-semiring combining a classical synchronisation algebra with QoS requirements. The second c-semiring corresponds to multicast synchronisation. Assume $W = \{\mathrm{req}, \mathrm{auth}, \overline{\mathrm{req}}, \overline{\mathrm{auth}}, \mathbf{1}_W, \mathbf{0}_W, \bot\}$. Set W can be equipped with a c-semiring structure $\langle W,+,\cdot,\mathbf{0}_W,\mathbf{1}_W\rangle$, where:

$$\mathrm{req}\cdot\overline{\mathrm{req}} = \overline{\mathrm{req}}, \quad \mathrm{auth}\cdot\overline{\mathrm{auth}} = \overline{\mathrm{auth}}, \quad \mathrm{req}\cdot\mathrm{req} = \mathrm{req}, \quad \mathrm{auth}\cdot\mathrm{auth} = \mathrm{auth},$$
$$a,b \in W \setminus \{\mathbf{0}_W, \mathbf{1}_W\} \wedge a \neq b \wedge b \neq \overline{a} \implies a \cdot b = \bot$$
plus rules obtained by commutativity and the ones for $\mathbf{0}_W$ and $\mathbf{1}_W$.

The operation $+$ is obtained by extending the c-semiring axioms for the additive operation with $a + a = a$ and $a, b \notin \{\mathbf{0}_W, \mathbf{1}_W\} \wedge a \neq b \implies a + b = \bot$, for all $a, b \in W$.

Below we show a graphical representation of a two steps derivation. Instead of reporting productions for each rewriting step, tentacles are decorated with actions. For the sake of clarity, in each step we only write actions and weights of the relevant nodes.



The first step selects the server with the lowest price where p_i is the price for S_i (in this step no names are communicated). This is obtained as the result of the synchronisation in u, i.e., $((\overline{\text{req}} \cdot \text{req}) \cdot \text{req},\ min(\infty, p_1, p_2))$. Assuming p_1 less than p_2 the new weight of u is $(\overline{\text{req}}, p_1)$. The second step shows the client connecting to the cheapest server S_1 (informed by A) by connecting to a new node y. After the first synchronisation, the cheapest server is identified by the authority using the new weight on node u. This guides the behaviour of S_1 and of the authority to produce the new connection to the client. In particular, the applicability condition of server rule requires its price to be less than or equal to the price on the node, and this can be satisfied only by the cheapest one (we suppose for simplicity that server costs are unique). ◊

Exercise 6.7. Show how the transitions above are derived. ◇

References

1. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44(2):201–236, 1997.
2. R. De Nicola, G. Ferrari, U. Montanari, R. Pugliese, and E. Tuosto. A Formal Basis for Reasoning on Programmable QoS. In *International Symposium on Verification – Theory and Practice*, volume 2772 of *LNCS*, pages 436–479. Springer, 2003.
3. R. De Nicola, G. Ferrari, U. Montanari, R. Pugliese, and E. Tuosto. A process calculus for qos-aware applications. In *Proc. of Coordination'05*, volume 3454 of *LNCS*, pages 33–48. Springer, 2003.
4. G. Ferrari, U. Montanari, and E. Tuosto. A LTS semantics of ambients via graph synchronization with mobility. In *ICTCS'01*, volume 2202 of *LNCS*, pages 1–16. Springer, 2001.
5. Gianluigi Ferrari, Dan Hirsch, Ivan Lanese, Ugo Montanari, and Emilio Tuosto. LaTeXSynchronised Hyperedge Replacement as a Model for Service Oriented Computing. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *Formal Methods for Components and Objects: 4th International Symposium, FMCO*, volume 4111 of *LNCS*, Amsterdam, The Netherlands, November 2006. Springer-Verlag. Revised Lectures.
6. D. Hirsch. *Graph Transformation Models for Software Architecture Styles*. PhD thesis, Departamento de Computación, Facultad de Ciencias Exactas y Naturales, U.B.A., 2003.
7. D. Hirsch and E. Tuosto. Coordinating Application Level QoS with SHReQ. *Journal of Software and Systems Modelling*, 2006. Submitted.
8. I. Lanese. *Synchronization Strategies for Global Computing Models*. PhD thesis, Computer Science Department, University of Pisa, Pisa, Italy, 2006. Forthcoming.
9. I. Lanese and E. Tuosto. Synchronized hyperedge replacement for heterogeneous systems. In *Proc. of Coordination'05*, volume 3454 of *LNCS*, pages 220–235. Springer, 2005.