COWS

a Calculus for Orchestration of Web Services

Alessandro Lapadula

Dipartimento di Sistemi e Informatica, Università di Firenze

Joint work with Rosario Pugliese & Francesco Tiezzi

COWS Web Site `http://rap.dsi.unifi.it/cows/`

Outline

Outline

Outline

Outline

Outline

Our approach

- Exploit WS-BPEL, the 'de facto' standard language for orchestration of Web services, to drive the design of COWS (*Calculus for Orchestration of Web Services*)
- Similarly to WS-BPEL, COWS allows
  - *shared states* among threads of a same service instance
  - a same process to play *more than one partner role*, and
  - programming stateful sessions by *correlating different interactions*
- COWS intends to be a foundational model not specifically tight to Web services' current technology
  - Some WS-BPEL constructs, e.g. fault and compensation handlers and flow graphs, do not have a precise counterpart in COWS
- COWS combines in an original way a number of constructs and features borrowed from well-known process calculi
  - asynchronous communication
  - pattern matching
  - polyadic synchronization (π-calculus)
  - localized receiving (Lπ)
  - delimited killing activities & protection (StAC$_i$)

Our approach

- Exploit WS-BPEL, the 'de facto' standard language for orchestration of Web services, to drive the design of COWS (*Calculus for Orchestration of Web Services*)
- Similarly to WS-BPEL, COWS allows
  - *shared states* among threads of a same service instance
  - a same process to play *more than one partner role*, and
  - programming stateful sessions by *correlating different interactions*
- COWS intends to be a foundational model not specifically tight to Web services' current technology
  - Some WS-BPEL constructs, e.g. fault and compensation handlers and flow graphs, do not have a precise counterpart in COWS
- COWS combines in an original way a number of constructs and features borrowed from well-known process calculi
  - asynchronous communication
  - pattern matching
  - polyadic synchronization (π-calculus)
  - localized receiving (Lπ)
  - delimited killing activities & protection (StAC$_i$)

Our approach

- Exploit WS-BPEL, the 'de facto' standard language for orchestration of Web services, to drive the design of COWS (*Calculus for Orchestration of Web Services*)
- Similarly to WS-BPEL, COWS allows
  - *shared states* among threads of a same service instance
  - a same process to play *more than one partner role*, and
  - programming stateful sessions by *correlating different interactions*
- COWS intends to be a foundational model not specifically tight to Web services' current technology
  - Some WS-BPEL constructs, e.g. fault and compensation handlers and flow graphs, do not have a precise counterpart in COWS
- COWS combines in an original way a number of constructs and features borrowed from well-known process calculi
  - asynchronous communication
  - pattern matching
  - polyadic synchronization (π-calculus)
  - localized receiving (Lπ)
  - delimited killing activities & protection (StAC$_i$)

Our approach

- Exploit WS-BPEL, the 'de facto' standard language for orchestration of Web services, to drive the design of COWS (*Calculus for Orchestration of Web Services*)
- Similarly to WS-BPEL, COWS allows
  - *shared states* among threads of a same service instance
  - a same process to play *more than one partner role*, and
  - programming stateful sessions by *correlating different interactions*
- COWS intends to be a foundational model not specifically tight to Web services' current technology
  - Some WS-BPEL constructs, e.g. fault and compensation handlers and flow graphs, do not have a precise counterpart in COWS
- COWS combines in an original way a number of constructs and features borrowed from well-known process calculi
  - asynchronous communication
  - pattern matching
  - polyadic synchronization (π-calculus)
  - localized receiving (Lπ)
  - delimited killing activities & protection (StAC$_i$)

Outline

COWS basic features

- Basic elements: *partners* and *operations*
- *Communication endpoint*: partner name *plus* operation name
- Very flexible naming mechanism that allows
  - a same service to be identified by means of different logic names (i.e. to play more than one partner role)

$$p_{slow} \cdot o?\bar{w}.s_{slow} + p_{fast} \cdot o?\bar{w}.s_{fast}$$

  - the names composing an endpoint to be dealt with separately as in a request-response interaction

$$p \cdot o_{req}?\langle x \rangle.x \cdot o_{res}!\langle \text{"I'm alive"} \rangle$$

- Partners and operations can be exchanged in communication
  - but dynamically received names cannot form the communication endpoints used to receive further invocations

COWS basic features

- Basic elements: *partners* and *operations*
- *Communication endpoint*: partner name *plus* operation name
- Very flexible naming mechanism that allows
  - a same service to be identified by means of different logic names (i.e. to play more than one partner role)

$$p_{slow} \cdot o?\bar{w}.s_{slow} + p_{fast} \cdot o?\bar{w}.s_{fast}$$

  - the names composing an endpoint to be dealt with separately as in a request-response interaction

$$p \cdot o_{req}?\langle x \rangle.x \cdot o_{res}!\langle \text{"I'm alive"} \rangle$$

- Partners and operations can be exchanged in communication
  - but dynamically received names cannot form the communication endpoints used to receive further invocations

COWS basic features

- Basic elements: *partners* and *operations*
- *Communication endpoint*: partner name *plus* operation name
- Very flexible naming mechanism that allows
  - a same service to be identified by means of different logic names (i.e. to play more than one partner role)

$$p_{slow} \cdot o?\bar{w}.s_{slow} + p_{fast} \cdot o?\bar{w}.s_{fast}$$

  - the names composing an endpoint to be dealt with separately as in a request-response interaction

$$p \cdot o_{req}?\langle x \rangle.x \cdot o_{res}!\langle \text{''I'm alive''} \rangle$$

- Partners and operations can be exchanged in communication
  - but dynamically received names cannot form the communication endpoints used to receive further invocations

COWS basic features

- Computational entities are called *services*
  - One specific instance to serve each received request
  - An instance contains concurrent threads (with a shared state) that may offer a choice among alternative receive activities
- *Pattern-matching* is used for
  - correlating, *by means of their same contents*, different service interactions logically forming a same 'session'
  - isolating those data that are important to
    - identify service instances for the routing of messages, and
    - program stateful multi-partners sessions
- The sets of correlation data can be dynamically manipulated
- A single message may participate in multiple interaction sessions, each identified by separate correlation values

COWS basic features

- Computational entities are called *services*
  - One specific instance to serve each received request
  - An instance contains concurrent threads (with a shared state) that may offer a choice among alternative receive activities
- *Pattern-matching* is used for
  - correlating, *by means of their same contents*, different service interactions logically forming a same ‘session’
  - isolating those data that are important to
    - ★ identify service instances for the routing of messages, and
    - ★ program stateful multi-partners sessions
- The sets of correlation data can be dynamically manipulated
- A single message may participate in multiple interaction sessions, each identified by separate correlation values

COWS basic features

- Computational entities are called *services*
  - One specific instance to serve each received request
  - An instance contains concurrent threads (with a shared state) that may offer a choice among alternative receive activities
- *Pattern-matching* is used for
  - correlating, *by means of their same contents*, different service interactions logically forming a same ‘session’
  - isolating those data that are important to
    - ★ identify service instances for the routing of messages, and
    - ★ program stateful multi-partners sessions
- The sets of correlation data can be dynamically manipulated
- A single message may participate in multiple interaction sessions, each identified by separate correlation values

COWS basic features

- The *delimitation* operator is the only binder of the calculus
  - Differently from most process calculi, *receive activities* bind neither names nor variables
  - Differently from the *fusion calculus*, inter-service communication give rise to substitutions of variables with values (*update calculus*)
- Delimitation is used to:
  - regulate the range of application of substitutions generated by communication
  - generate fresh names (as the restriction operator of the π-calculus)
  - delimit the field of action of the *kill* activity, that can be used to force termination of whole service instances
- The *protection* operator can be used to protect sensitive code from the effect of a forced termination

COWS basic features

- The *delimitation* operator is the only binder of the calculus
  - Differently from most process calculi, *receive activities* bind neither names nor variables
  - Differently from the *fusion calculus*, inter-service communication give rise to substitutions of variables with values (*update calculus*)
- Delimitation is used to:
  - regulate the range of application of substitutions generated by communication
  - generate fresh names (as the restriction operator of the π-calculus)
  - delimit the field of action of the *kill* activity, that can be used to force termination of whole service instances
- The *protection* operator can be used to protect sensitive code from the effect of a forced termination

COWS basic features

- The *delimitation* operator is the only binder of the calculus
  - Differently from most process calculi, *receive activities* bind neither names nor variables
  - Differently from the *fusion calculus*, inter-service communication give rise to substitutions of variables with values (*update calculus*)
- Delimitation is used to:
  - regulate the range of application of substitutions generated by communication
  - generate fresh names (as the restriction operator of the π-calculus)
  - delimit the field of action of the *kill* activity, that can be used to force termination of whole service instances
- The *protection* operator can be used to protect sensitive code from the effect of a forced termination

Outline

Syntax of COWS

s ::=		(services)	(notations)
	$\mathbf{kill}(k)$	(kill)	k *(killer) labels*
$\mid$	$u \cdot u'!\bar{e}$	(invoke)	e *expressions*
$\mid$	g	(input-guarded choice)	x *variables*
$\mid$	$s \mid s$	(parallel composition)	v *values*
$\mid$	$\{\mid s \mid\}$	(protection)	n, p, o *names*
$\mid$	$[d]\, s$	(delimitation)	d: labels|names|vars
$\mid$	$* s$	(replication)	

g ::=		(input-guarded choice)	
	$\mathbf{0}$	(nil)	
$\mid$	$p \cdot o?\bar{w}.s$	(request processing)	w: values|variables
$\mid$	$g + g$	(choice)	u: names|variables

$\bar{\cdot}$ denotes tuples of objects, e.g. $\bar{w}$ is a tuple of parameters

One *binding* construct: $[d]\, s$ binds d in the scope s
(free/bound name/variable/label defined accordingly)

Operational semantics

- Only defined for *closed* services
  i.e. services without free killer labels/variables
- Defined in terms of:
  - a labelled transition relation $\xrightarrow{\alpha}$
  - a structural congruence $\equiv$ that identifies syntactically different services that intuitively represent the same service
    E.g. (scope extension rule)

$$s_1 \mid [d]\, s_2 \;\equiv\; [d]\,(s_1 \mid s_2) \quad \text{if } d \notin \mathrm{fd}(s_1) \cup \mathrm{fk}(s_2)$$

Operational semantics

- Only defined for *closed* services
  i.e. services without free killer labels/variables
- Defined in terms of:
  - a labelled transition relation $\xrightarrow{\alpha}$
  - a structural congruence $\equiv$ that identifies syntactically different services that intuitively represent the same service
    E.g. (scope extension rule)

$$s_1 \mid [d]\, s_2 \;\equiv\; [d]\,(s_1 \mid s_2) \quad \text{if } d \notin \mathrm{fd}(s_1) \cup \mathrm{fk}(s_2)$$

Outline

Communication of private names & global scope of variables

- Exploits scope extension as in π-calculus
- To enable communication of private names, besides their scopes, one has to possibly extend the scopes of some variables
- Interacting receive and invoke activities must be in the scopes of the delimitations that bind the variables argument of the receive

$$[x]\,(p \cdot o?\langle x \rangle.s \mid s') \mid [n]\,p \cdot o!\langle n \rangle \quad \equiv \quad (n \text{ fresh})$$
$$[n]\,([x]\,(p \cdot o?\langle x \rangle.s \mid s') \mid p \cdot o!\langle n \rangle) \quad \equiv$$
$$[n]\,[x]\,(p \cdot o?\langle x \rangle.s \mid s' \mid p \cdot o!\langle n \rangle) \quad \xrightarrow{p \cdot o\,[\emptyset]\,\langle x \rangle\,\langle n \rangle}$$
$$[n]\,(s \mid s') \cdot \{x \mapsto n\}$$

- Substitution $\{x \mapsto n\}$ is applied to all terms delimited by $[x]$, not only to the continuation s of the service performing the receive
- Differs from most process calculi and accounts for the global scope of variables (e.g. *delayed input* of fusion calculus)

Communication of private names & global scope of variables

- Exploits scope extension as in π-calculus
- To enable communication of private names, besides their scopes, one has to possibly extend the scopes of some variables
- Interacting receive and invoke activities must be in the scopes of the delimitations that bind the variables argument of the receive

$$[x]\,(p \cdot o?\langle x \rangle.s \mid s') \mid [n]\,p \cdot o!\langle n \rangle \quad \equiv \quad (n \text{ fresh})$$
$$[n]\,([x]\,(p \cdot o?\langle x \rangle.s \mid s') \mid p \cdot o!\langle n \rangle) \quad \equiv$$
$$[n]\,[x]\,(p \cdot o?\langle x \rangle.s \mid s' \mid p \cdot o!\langle n \rangle) \quad \xrightarrow{p \cdot o\,[\emptyset]\,\langle x \rangle\,\langle n \rangle}$$
$$[n]\,(s \mid s') \cdot \{x \mapsto n\}$$

- Substitution $\{x \mapsto n\}$ is applied to all terms delimited by $[x]$, not only to the continuation s of the service performing the receive
- Differs from most process calculi and accounts for the global scope of variables (e.g. *delayed input* of fusion calculus)

Communication of private names & global scope of variables

- Exploits scope extension as in π-calculus
- To enable communication of private names, besides their scopes, one has to possibly extend the scopes of some variables
- Interacting receive and invoke activities must be in the scopes of the delimitations that bind the variables argument of the receive

$$
\begin{array}{lll}
[x]\,(p \cdot o?\langle x \rangle.s \mid s') \mid [n]\,p \cdot o!\langle n \rangle & \equiv & (n \text{ fresh}) \\
[n]\,([x]\,(p \cdot o?\langle x \rangle.s \mid s') \mid p \cdot o!\langle n \rangle) & \equiv & \\
[n]\,[x]\,(p \cdot o?\langle x \rangle.s \mid s' \mid p \cdot o!\langle n \rangle) & \xrightarrow{p \cdot o\,[\emptyset]\,\langle x \rangle\,\langle n \rangle} & \\
[n]\,(s \mid s') \cdot \{x \mapsto n\} & &
\end{array}
$$

- Substitution $\{x \mapsto n\}$ is applied to all terms delimited by $[x]$, not only to the continuation s of the service performing the receive
- Differs from most process calculi and accounts for the global scope of variables (e.g. *delayed input* of fusion calculus)

Communication of private names & global scope of variables

- Exploits scope extension as in π-calculus
- To enable communication of private names, besides their scopes, one has to possibly extend the scopes of some variables
- Interacting receive and invoke activities must be in the scopes of the delimitations that bind the variables argument of the receive

$$
\begin{array}{lcl}
[x]\,(p \cdot o?\langle x\rangle.s \mid s') \mid [n]\,p \cdot o!\langle n\rangle & \equiv & (n \text{ fresh}) \\
[n]\,([x]\,(p \cdot o?\langle x\rangle.s \mid s') \mid p \cdot o!\langle n\rangle) & \equiv & \\
[n]\,[x]\,(p \cdot o?\langle x\rangle.s \mid s' \mid p \cdot o!\langle n\rangle) & \xrightarrow{p \cdot o\ \lfloor\emptyset\rfloor\ \langle x\rangle\ \langle n\rangle} & \\
[n]\,(s \mid s') \cdot \{x \mapsto n\} & &
\end{array}
$$

- Substitution $\{x \mapsto n\}$ is applied to all terms delimited by $[x]$, not only to the continuation s of the service performing the receive
- Differs from most process calculi and accounts for the global scope of variables (e.g. *delayed input* of fusion calculus)

Communication of private names & global scope of variables

- Exploits scope extension as in π-calculus
- To enable communication of private names, besides their scopes, one has to possibly extend the scopes of some variables
- Interacting receive and invoke activities must be in the scopes of the delimitations that bind the variables argument of the receive

$$
\begin{array}{lcl}
[x]\,(p \cdot o?\langle x\rangle.s \mid s') \mid [n]\,p \cdot o!\langle n\rangle & \equiv & (n \text{ fresh}) \\
[n]\,([x]\,(p \cdot o?\langle x\rangle.s \mid s') \mid p \cdot o!\langle n\rangle) & \equiv & \\
[n]\,[x]\,(p \cdot o?\langle x\rangle.s \mid s' \mid p \cdot o!\langle n\rangle) & \xrightarrow{p \cdot o\ \lfloor\emptyset\rfloor\ \langle x\rangle\ \langle n\rangle} & \\
[n]\,(s \mid s') \cdot \{x \mapsto n\} & &
\end{array}
$$

- Substitution $\{x \mapsto n\}$ is applied to all terms delimited by $[x]$, not only to the continuation s of the service performing the receive
- Differs from most process calculi and accounts for the global scope of variables (e.g. *delayed input* of fusion calculus)

Communication of private names & global scope of variables

- Exploits scope extension as in π-calculus
- To enable communication of private names, besides their scopes, one has to possibly extend the scopes of some variables
- Interacting receive and invoke activities must be in the scopes of the delimitations that bind the variables argument of the receive

$$
\begin{array}{lcl}
[x]\,(p \cdot o?\langle x\rangle.s \mid s') \mid [n]\,p \cdot o!\langle n\rangle & \equiv & (n \text{ fresh}) \\
[n]\,([x]\,(p \cdot o?\langle x\rangle.s \mid s') \mid p \cdot o!\langle n\rangle) & \equiv & \\
[n]\,[x]\,(p \cdot o?\langle x\rangle.s \mid s' \mid p \cdot o!\langle n\rangle) & \xrightarrow{p \cdot o\,\lfloor\emptyset\rfloor\,\langle x\rangle\,\langle n\rangle} & \\
[n]\,(s \mid s') \cdot \{x \mapsto n\} & &
\end{array}
$$

- Substitution $\{x \mapsto n\}$ is applied to all terms delimited by $[x]$, not only to the continuation s of the service performing the receive
- Differs from most process calculi and accounts for the global scope of variables (e.g. *delayed input* of fusion calculus)

Interplay between communication and kill activity

- Kill activities can break communication

$$p \cdot o!\langle n \rangle \mid [k]\,([x]\,p \cdot o?\langle x \rangle.s \mid \mathbf{kill}(k)) \xrightarrow{\dagger} p \cdot o!\langle n \rangle \mid [k]\,[x]\,\mathbf{0}$$

Note: this is the only possible evolution (kill activities are executed *eagerly*)

- Communication can however be guaranteed by protecting the receive activity

$$p \cdot o!\langle n \rangle \mid [k]\,([x]\,\{\!|p \cdot o?\langle x \rangle.s|\!\} \mid \mathbf{kill}(k)) \xrightarrow{\dagger}$$

$$p \cdot o!\langle n \rangle \mid [k]\,[x]\,\{\!|p \cdot o?\langle x \rangle.s|\!\} \equiv$$

$$[x]\,(p \cdot o!\langle n \rangle \mid [k]\,\{\!|p \cdot o?\langle x \rangle.s|\!\}) \xrightarrow{p \cdot o\,\lfloor\emptyset\rfloor\,\langle x \rangle\,\langle n \rangle}$$

$$[k]\,\{\!|s \cdot \{x \mapsto n\}|\!\}$$

Interplay between communication and kill activity

- Kill activities can break communication

$$p \cdot o!\langle n \rangle \mid [k]\,([x]\,p \cdot o?\langle x \rangle.s \mid \mathbf{kill}(k)) \xrightarrow{\dagger} p \cdot o!\langle n \rangle \mid [k]\,[x]\,\mathbf{0}$$

Note: this is the only possible evolution (kill activities are executed *eagerly*)

- Communication can however be guaranteed by protecting the receive activity

$$p \cdot o!\langle n \rangle \mid [k]\,([x]\,\{\!|p \cdot o?\langle x \rangle.s|\!\} \mid \mathbf{kill}(k)) \xrightarrow{\dagger}$$

$$p \cdot o!\langle n \rangle \mid [k]\,[x]\,\{\!|p \cdot o?\langle x \rangle.s|\!\} \equiv$$

$$[x]\,(p \cdot o!\langle n \rangle \mid [k]\,\{\!|p \cdot o?\langle x \rangle.s|\!\}) \xrightarrow{p \cdot o\,\lfloor\emptyset\rfloor\,\langle x \rangle\,\langle n \rangle}$$

$$[k]\,\{\!|s \cdot \{x \mapsto n\}|\!\}$$

Interplay between communication and kill activity

- Kill activities can break communication

$$p \cdot o!\langle n \rangle \mid [k]\,([x]\,p \cdot o?\langle x \rangle.s \mid \mathbf{kill}(k)) \xrightarrow{\dagger} p \cdot o!\langle n \rangle \mid [k]\,[x]\,\mathbf{0}$$

Note: this is the only possible evolution (kill activities are executed *eagerly*)

- Communication can however be guaranteed by protecting the receive activity

$$p \cdot o!\langle n \rangle \mid [k]\,([x]\,\{\!|p \cdot o?\langle x \rangle.s|\!\}) \mid \mathbf{kill}(k)) \xrightarrow{\dagger}$$

$$p \cdot o!\langle n \rangle \mid [k]\,[x]\,\{\!|p \cdot o?\langle x \rangle.s|\!\} \equiv$$

$$[x]\,(p \cdot o!\langle n \rangle \mid [k]\,\{\!|p \cdot o?\langle x \rangle.s|\!\}) \xrightarrow{p \cdot o\,\lfloor\emptyset\rfloor\,\langle x \rangle\,\langle n \rangle}$$

$$[k]\,\{\!|s \cdot \{x \mapsto n\}|\!\}$$

Conflicting receive activities

- Consider a *persistent service* that, once instantiated, enables two conflicting receives

$$* [x] (p_1 \cdot o?\langle x \rangle . s_1 \mid p_2 \cdot o?\langle x \rangle . s_2) \mid p_1 \cdot o!\langle v \rangle \mid p_2 \cdot o!\langle v \rangle$$

$$\xrightarrow{p_1 \cdot o \lfloor \emptyset \rfloor \langle x \rangle \langle v \rangle}$$

$$* [x] (p_1 \cdot o?\langle x \rangle . s_1 \mid p_2 \cdot o?\langle x \rangle . s_2) \mid p_2 \cdot o!\langle v \rangle \mid s_1 \cdot \{x \mapsto v\} \mid p_2 \cdot o?\langle v \rangle . s_2 \cdot \{x \mapsto v\}$$

- Now, the service and the created instance, being both able to receive the same tuple $\langle v \rangle$ along the endpoint $p_2 \cdot o$, *compete* for the request $p_2 \cdot o!\langle v \rangle$
- However, our (prioritized) semantics for "|" allows only the existing instance to evolve (and, thus, avoids creation of a new instance)

$$* [x] (p_1 \cdot o?\langle x \rangle . s_1 \mid p_2 \cdot o?\langle x \rangle . s_2) \mid s_1 \cdot \{x \mapsto v\} \mid s_2 \cdot \{x \mapsto v\}$$

Conflicting receive activities

- Consider a *persistent service* that, once instantiated, enables two conflicting receives

$$* [x] (p_1 \cdot o?\langle x \rangle . s_1 \mid p_2 \cdot o?\langle x \rangle . s_2) \mid p_1 \cdot o!\langle v \rangle \mid p_2 \cdot o!\langle v \rangle$$

$$\xrightarrow{p_1 \cdot o \lfloor \emptyset \rfloor \langle x \rangle \langle v \rangle}$$

$$* [x] (p_1 \cdot o?\langle x \rangle . s_1 \mid p_2 \cdot o?\langle x \rangle . s_2) \mid p_2 \cdot o!\langle v \rangle \mid s_1 \cdot \{x \mapsto v\} \mid p_2 \cdot o?\langle v \rangle . s_2 \cdot \{x \mapsto v\}$$

- Now, the service and the created instance, being both able to receive the same tuple $\langle v \rangle$ along the endpoint $p_2 \cdot o$, *compete* for the request $p_2 \cdot o!\langle v \rangle$
- However, our (prioritized) semantics for "|" allows only the existing instance to evolve (and, thus, avoids creation of a new instance)

$$* [x] (p_1 \cdot o?\langle x \rangle . s_1 \mid p_2 \cdot o?\langle x \rangle . s_2) \mid s_1 \cdot \{x \mapsto v\} \mid s_2 \cdot \{x \mapsto v\}$$

Conflicting receive activities

- Consider a *persistent service* that, once instantiated, enables two conflicting receives

$$* [x] (p_1 \cdot o?\langle x \rangle . s_1 \mid p_2 \cdot o?\langle x \rangle . s_2) \mid p_1 \cdot o!\langle v \rangle \mid p_2 \cdot o!\langle v \rangle$$

$$\xrightarrow{p_1 \cdot o \lfloor \emptyset \rfloor \langle x \rangle \langle v \rangle}$$

$$* [x] (p_1 \cdot o?\langle x \rangle . s_1 \mid p_2 \cdot o?\langle x \rangle . s_2) \mid p_2 \cdot o!\langle v \rangle \mid s_1 \cdot \{x \mapsto v\} \mid p_2 \cdot o?\langle v \rangle . s_2 \cdot \{x \mapsto v\}$$

- Now, the service and the created instance, being both able to receive the same tuple $\langle v \rangle$ along the endpoint $p_2 \cdot o$, *compete* for the request $p_2 \cdot o!\langle v \rangle$
- However, our (prioritized) semantics for "|" allows only the existing instance to evolve (and, thus, avoids creation of a new instance)

$$* [x] (p_1 \cdot o?\langle x \rangle . s_1 \mid p_2 \cdot o?\langle x \rangle . s_2) \mid s_1 \cdot \{x \mapsto v\} \mid s_2 \cdot \{x \mapsto v\}$$

Message correlation

- Consider un-correlated receives executed by a same instance
  E.g.

$$* [x]\, p_1 \cdot o_1?\langle x \rangle.[y]\, p_2 \cdot o_2?\langle y \rangle.s$$

- Since messages for operations o_1 and o_2 are un-correlated, if there are concurrent instances then successive invocations for a same instance can mix up and be delivered, instead, to different instances
- This behaviour can be avoided simply by correlating successive messages by means of some correlation data
  E.g. the first received value, as in the following example:

$$* [x]\, p_1 \cdot o_1?\langle x \rangle.[y]\, p_2 \cdot o_2?\langle y, x \rangle.s$$

Message correlation

- Consider un-correlated receives executed by a same instance
  E.g.
  $$* [x]\, p_1 \cdot o_1?\langle x \rangle.[y]\, p_2 \cdot o_2?\langle y \rangle.s$$
- Since messages for operations o_1 and o_2 are un-correlated, if there are concurrent instances then successive invocations for a same instance can mix up and be delivered, instead, to different instances
- This behaviour can be avoided simply by correlating successive messages by means of some correlation data
  E.g. the first received value, as in the following example:
  $$* [x]\, p_1 \cdot o_1?\langle x \rangle.[y]\, p_2 \cdot o_2?\langle y, x \rangle.s$$

Message correlation

- Consider un-correlated receives executed by a same instance
  E.g.
  $$* [x]\, p_1 \cdot o_1?\langle x \rangle.[y]\, p_2 \cdot o_2?\langle y \rangle.s$$
- Since messages for operations o_1 and o_2 are un-correlated, if there are concurrent instances then successive invocations for a same instance can mix up and be delivered, instead, to different instances
- This behaviour can be avoided simply by correlating successive messages by means of some correlation data
  E.g. the first received value, as in the following example:
  $$* [x]\, p_1 \cdot o_1?\langle x \rangle.[y]\, p_2 \cdot o_2?\langle y, x \rangle.s$$

Outline

COWS encodings

- Encoding other orchestration languages
  - localised π-calculus (Lπ) $\Rightarrow$ COWS (without **kill**(_) and $\{|\_|\}$)
  - SCC $\Rightarrow$ COWS (without $\{|\_|\}$)
  - Orc $\Rightarrow$ COWS
  - WS-CALCULUS $\Rightarrow$ COWS

  Encodings enjoy an operational correspondence
- Modelling imperative and orchestration constructs
  - Assignment, conditional choice, sequential composition,...
  - Fault and compensation handlers, flow graphs

COWS encodings

- Encoding other orchestration languages
  - localised π-calculus (Lπ) $\Rightarrow$ COWS (without **kill**(_) and $\{|\_|\}$)
  - SCC $\Rightarrow$ COWS (without $\{|\_|\}$)
  - Orc $\Rightarrow$ COWS
  - WS-CALCULUS $\Rightarrow$ COWS

  Encodings enjoy an operational correspondence
- Modelling imperative and orchestration constructs
  - Assignment, conditional choice, sequential composition,...
  - Fault and compensation handlers, flow graphs

Fault and compensation handlers: syntax

- *Compensation*: execution of specific activities (attempting) to reverse the effects of previously executed activities

Syntax for compesation

s	::=	...	(*services*)
	|	**throw**(ϕ)	(*fault generator*)
	|	**undo**$(\imath)$	(*compensate*)
	|	$[s : \textbf{catch}(\phi_1)\{s_1\} : \ldots : \textbf{catch}(\phi_n)\{s_n\} : s_c]_\imath$	(*scope*)

- **throw**(ϕ): rises a fault signal ϕ that triggers execution of s if a construct **catch**$(\phi)\{s\}$ exists within the same scope
- **undo**$(\imath)$: invokes a compensation handler of an inner scope $\imath$ that has already completed normally (i.e. without faulting)
- $[s : \textbf{catch}(\phi_1)\{s_1\} : \ldots : \textbf{catch}(\phi_n)\{s_n\} : s_c]_\imath$: is uniquely identified by $\imath$ and groups together a service s (the normal behaviour), an optional list of fault handlers, and a compensation handler s_c

Fault and compensation handlers: syntax

- *Compensation*: execution of specific activities (attempting) to reverse the effects of previously executed activities

Syntax for compesation

s	::=	$\ldots$	(*services*)
	$\mid$	$\textbf{throw}(\phi)$	(*fault generator*)
	$\mid$	$\textbf{undo}(\imath)$	(*compensate*)
	$\mid$	$[s : \textbf{catch}(\phi_1)\{s_1\} : \ldots : \textbf{catch}(\phi_n)\{s_n\} : s_c]_\imath$	(*scope*)

- $\textbf{throw}(\phi)$: rises a fault signal ϕ that triggers execution of s if a construct $\textbf{catch}(\phi)\{s\}$ exists within the same scope
- $\textbf{undo}(\imath)$: invokes a compensation handler of an inner scope $\imath$ that has already completed normally (i.e. without faulting)
- $[s : \textbf{catch}(\phi_1)\{s_1\} : \ldots : \textbf{catch}(\phi_n)\{s_n\} : s_c]_\imath$: is uniquely identified by $\imath$ and groups together a service s (the normal behaviour), an optional list of fault handlers, and a compensation handler s_c

Fault and compensation handlers: syntax

- *Compensation*: execution of specific activities (attempting) to reverse the effects of previously executed activities

Syntax for compesation

s ::=	$\ldots$	(*services*)
$\mid$	**throw**(ϕ)	(*fault generator*)
$\mid$	**undo**$(\imath)$	(*compensate*)
$\mid$	$[s : \mathbf{catch}(\phi_1)\{s_1\} : \ldots : \mathbf{catch}(\phi_n)\{s_n\} : s_c]_\imath$	(*scope*)

- **throw**(ϕ): rises a fault signal ϕ that triggers execution of s if a construct **catch**$(\phi)\{s\}$ exists within the same scope
- **undo**$(\imath)$: invokes a compensation handler of an inner scope $\imath$ that has already completed normally (i.e. without faulting)
- $[s : \mathbf{catch}(\phi_1)\{s_1\} : \ldots : \mathbf{catch}(\phi_n)\{s_n\} : s_c]_\imath$: is uniquely identified by $\imath$ and groups together a service s (the normal behaviour), an optional list of fault handlers, and a compensation handler s_c

Fault and compensation handlers: syntax

- *Compensation*: execution of specific activities (attempting) to reverse the effects of previously executed activities

Syntax for compesation

$$
\begin{array}{lll}
s ::= & \ldots & (\textit{services}) \\
\mid & \mathbf{throw}(\phi) & (\textit{fault generator}) \\
\mid & \mathbf{undo}(\imath) & (\textit{compensate}) \\
\mid & [s : \mathbf{catch}(\phi_1)\{s_1\} : \ldots : \mathbf{catch}(\phi_n)\{s_n\} : s_c]_\imath & (\textit{scope})
\end{array}
$$

- $\mathbf{throw}(\phi)$: rises a fault signal ϕ that triggers execution of s if a construct $\mathbf{catch}(\phi)\{s\}$ exists within the same scope
- $\mathbf{undo}(\imath)$: invokes a compensation handler of an inner scope $\imath$ that has already completed normally (i.e. without faulting)
- $[s : \mathbf{catch}(\phi_1)\{s_1\} : \ldots : \mathbf{catch}(\phi_n)\{s_n\} : s_c]_\imath$: is uniquely identified by $\imath$ and groups together a service s (the normal behaviour), an optional list of fault handlers, and a compensation handler s_c

Fault and compensation handlers: syntax

- *Compensation*: execution of specific activities (attempting) to reverse the effects of previously executed activities

Syntax for compesation

s	::=	$\ldots$	(*services*)
	$\mid$	**throw**(ϕ)	(*fault generator*)
	$\mid$	**undo**$(\imath)$	(*compensate*)
	$\mid$	$[s : \textbf{catch}(\phi_1)\{s_1\} : \ldots : \textbf{catch}(\phi_n)\{s_n\} : s_c]_\imath$	(*scope*)

- **throw**(ϕ): rises a fault signal ϕ that triggers execution of s if a construct **catch**$(\phi)\{s\}$ exists within the same scope
- **undo**$(\imath)$: invokes a compensation handler of an inner scope $\imath$ that has already completed normally (i.e. without faulting)
- $[s : \textbf{catch}(\phi_1)\{s_1\} : \ldots : \textbf{catch}(\phi_n)\{s_n\} : s_c]_\imath$: is uniquely identified by $\imath$ and groups together a service s (the normal behaviour), an optional list of fault handlers, and a compensation handler s_c

Fault and compensation handlers: encoding

$$\langle\!\langle [s : \mathbf{catch}(\phi_1)\{s_1\} : \ldots : \mathbf{catch}(\phi_n)\{s_n\} : s_c]_i \rangle\!\rangle_k = [p_{\phi_1}, \ldots, p_{\phi_n}] \; (\langle\!\langle \mathbf{catch}(\phi_1)\{s_1\} \rangle\!\rangle_k \mid \ldots \mid \langle\!\langle \mathbf{catch}(\phi_n)\{s_n\} \rangle\!\rangle_k \mid [k_i] (\langle\!\langle s \rangle\!\rangle_{k_i} ; (x_{done} \cdot o_{done}!\langle\rangle \mid \{\!| p_i \cdot o_{comp}?\langle\rangle . \langle\!\langle s_c \rangle\!\rangle_{k_i} |\!\})))$$

$$\langle\!\langle \mathbf{catch}(\phi)\{s\} \rangle\!\rangle_k = p_\phi \cdot o_{fault}?\langle\rangle . [k'] \langle\!\langle s \rangle\!\rangle_{k'}$$

$$\langle\!\langle \mathbf{undo}(i) \rangle\!\rangle_k = p_i \cdot o_{comp}!\langle\rangle \mid x_{done} \cdot o_{done}!\langle\rangle$$

$$\langle\!\langle \mathbf{throw}(\phi) \rangle\!\rangle_k = \{\!| p_\phi \cdot o_{fault}!\langle\rangle \mid x_{done} \cdot o_{done}!\langle\rangle |\!\} \mid \mathbf{kill}(k)$$

- s_c is installed when the normal behaviour s successfully completes, but it is activated only when signal $p_i \cdot o_{comp}!\langle\rangle$ occurs
- Whenever a fault ϕ occurs, the remaining activities of the scope are terminated by the $\mathbf{kill}(k)$ generated by the encoding $\langle\!\langle \mathbf{throw}(\phi) \rangle\!\rangle_k$ and a signal $p_\phi \cdot o_{fault}!\langle\rangle$ triggers execution of the corresponding fault handler (if any)

Fault and compensation handlers: encoding

$$\langle\!\langle [s : \mathbf{catch}(\phi_1)\{s_1\} : \ldots : \mathbf{catch}(\phi_n)\{s_n\} : s_c]_\imath \rangle\!\rangle_k \;=\; [p_{\phi_1}, \ldots, p_{\phi_n}]\ (\ \langle\!\langle \mathbf{catch}(\phi_1)\{s_1\} \rangle\!\rangle_k \mid \ \ldots\ \mid \langle\!\langle \mathbf{catch}(\phi_n)\{s_n\} \rangle\!\rangle_k \mid [k_\imath]\,(\ \langle\!\langle s \rangle\!\rangle_{k_\imath} ; (x_{done} \cdot o_{done}!\langle\rangle \mid \{\!|\, p_\imath \cdot o_{comp}?\langle\rangle . \langle\!\langle s_c \rangle\!\rangle_{k_\imath} \,|\!\}\ )\ )\ )$$

$$\langle\!\langle \mathbf{catch}(\phi)\{s\} \rangle\!\rangle_k \;=\; p_\phi \cdot o_{fault}?\langle\rangle . [k']\, \langle\!\langle s \rangle\!\rangle_{k'}$$

$$\langle\!\langle \mathbf{undo}(\imath) \rangle\!\rangle_k \;=\; p_\imath \cdot o_{comp}!\langle\rangle \mid x_{done} \cdot o_{done}!\langle\rangle$$

$$\langle\!\langle \mathbf{throw}(\phi) \rangle\!\rangle_k \;=\; \{\!|\, p_\phi \cdot o_{fault}!\langle\rangle \mid x_{done} \cdot o_{done}!\langle\rangle \,|\!\} \mid \mathbf{kill}(k)$$

- s_c is installed when the normal behaviour s successfully completes, but it is activated only when signal $p_\imath \cdot o_{comp}!\langle\rangle$ occurs
- Whenever a fault ϕ occurs, the remaining activities of the scope are terminated by the $\mathbf{kill}(k)$ generated by the encoding $\langle\!\langle \mathbf{throw}(\phi) \rangle\!\rangle_k$ and a signal $p_\phi \cdot o_{fault}!\langle\rangle$ triggers execution of the corresponding fault handler (if any)

Fault and compensation handlers: encoding

$$\langle\!\langle [s : \mathbf{catch}(\phi_1)\{s_1\} : \ldots : \mathbf{catch}(\phi_n)\{s_n\} : s_c]_i \rangle\!\rangle_k \;=\; [p_{\phi_1}, \ldots, p_{\phi_n}]\ (\ \langle\!\langle \mathbf{catch}(\phi_1)\{s_1\} \rangle\!\rangle_k \mid \ \ldots\ \mid \langle\!\langle \mathbf{catch}(\phi_n)\{s_n\} \rangle\!\rangle_k \mid [k_i]\,(\ \langle\!\langle s \rangle\!\rangle_{k_i} ; (x_{done} \cdot o_{done}!\langle\rangle \mid \{\!|\, p_i \cdot o_{comp}?\langle\rangle . \langle\!\langle s_c \rangle\!\rangle_{k_i} \,|\!\}\ )\ )\ )$$

$$\langle\!\langle \mathbf{catch}(\phi)\{s\} \rangle\!\rangle_k \;=\; p_\phi \cdot o_{fault}?\langle\rangle . [k']\, \langle\!\langle s \rangle\!\rangle_{k'}$$

$$\langle\!\langle \mathbf{undo}(i) \rangle\!\rangle_k \;=\; p_i \cdot o_{comp}!\langle\rangle \mid x_{done} \cdot o_{done}!\langle\rangle$$

$$\langle\!\langle \mathbf{throw}(\phi) \rangle\!\rangle_k \;=\; \{\!|\, p_\phi \cdot o_{fault}!\langle\rangle \mid x_{done} \cdot o_{done}!\langle\rangle \,|\!\} \;\mid \mathbf{kill}(k)$$

- s_c is installed when the normal behaviour s successfully completes, but it is activated only when signal $p_i \cdot o_{comp}!\langle\rangle$ occurs
- Whenever a fault ϕ occurs, the remaining activities of the scope are terminated by the $\mathbf{kill}(k)$ generated by the encoding $\langle\!\langle \mathbf{throw}(\phi) \rangle\!\rangle_k$ and a signal $p_\phi \cdot o_{fault}!\langle\rangle$ triggers execution of the corresponding fault handler (if any)

Fault and compensation handlers: encoding

$$\langle\!\langle [s : \mathbf{catch}(\phi_1)\{s_1\} : \ldots : \mathbf{catch}(\phi_n)\{s_n\} : s_c]_i \rangle\!\rangle_k = [p_{\phi_1}, \ldots, p_{\phi_n}] \; (\langle\!\langle \mathbf{catch}(\phi_1)\{s_1\} \rangle\!\rangle_k \mid \ldots \mid \langle\!\langle \mathbf{catch}(\phi_n)\{s_n\} \rangle\!\rangle_k \mid [k_i] (\langle\!\langle s \rangle\!\rangle_{k_i} ; (x_{done} \cdot o_{done}!\langle\rangle \mid \{\!| p_i \cdot o_{comp}?\langle\rangle . \langle\!\langle s_c \rangle\!\rangle_{k_i} |\!\})))$$

$$\langle\!\langle \mathbf{catch}(\phi)\{s\} \rangle\!\rangle_k = p_\phi \cdot o_{fault}?\langle\rangle . [k'] \langle\!\langle s \rangle\!\rangle_{k'}$$

$$\langle\!\langle \mathbf{undo}(i) \rangle\!\rangle_k = p_i \cdot o_{comp}!\langle\rangle \mid x_{done} \cdot o_{done}!\langle\rangle$$

$$\langle\!\langle \mathbf{throw}(\phi) \rangle\!\rangle_k = \{\!| p_\phi \cdot o_{fault}!\langle\rangle \mid x_{done} \cdot o_{done}!\langle\rangle |\!\} \mid \mathbf{kill}(k)$$

- s_c is installed when the normal behaviour s successfully completes, but it is activated only when signal $p_i \cdot o_{comp}!\langle\rangle$ occurs
- Whenever a fault ϕ occurs, the remaining activities of the scope are terminated by the $\mathbf{kill}(k)$ generated by the encoding $\langle\!\langle \mathbf{throw}(\phi) \rangle\!\rangle_k$ and a signal $p_\phi \cdot o_{fault}!\langle\rangle$ triggers execution of the corresponding fault handler (if any)

Outline

SENSORIA Car repair scenario

Story

A car manufacturer offers a service that once a users car breaks down the on-board computer system tries to locate garages, car rentals and towing truck services in the car's vicinity, so that

- the car is towed to the garage and repaired
- the car owner may continue its travel

Orchestration logic

- Before orders, the credit card is charged with a security amount
- Before looking for a tow truck, a garage must be found
- If finding a tow truck fails, the garage appointment must be revoked
- If renting a car succeeds and finding either a tow truck or a garage fails, the rental car must be redirected to the broken down car's actual location
- If the car rental fails, it should not affect the tow truck and garage orders

SENSORIA Car repair scenario

Story

A car manufacturer offers a service that once a users car breaks down the on-board computer system tries to locate garages, car rentals and towing truck services in the car's vicinity, so that

- the car is towed to the garage and repaired
- the car owner may continue its travel

Orchestration logic

- Before orders, the credit card is charged with a security amount
- Before looking for a tow truck, a garage must be found
- If finding a tow truck fails, the garage appointment must be revoked
- If renting a car succeeds and finding either a tow truck or a garage fails, the rental car must be redirected to the broken down car's actual location
- If the car rental fails, it should not affect the tow truck and garage orders

Recovery system

The on-board recovery system (s_{sys}) receives failure signals

$$[\bar{x}_e, \bar{x}_b, \ldots](\ldots + p_{sys} \cdot o_{engine}?\bar{x}_e.s_{engine} + p_{sys} \cdot o_{breakes}?\bar{x}_b.s_{breaks} + \ldots)$$

- Recovery services ($s_{engine}, s_{breaks}, \ldots$) are activated and variables ($\bar{x}_e, \bar{x}_b, \ldots$) are initialized with sensor diagnostic data

E.g., the diagnostic system reports an engine failure

$$s_{sys} \mid p_{sys} \cdot o_{engine}!\bar{d} \xrightarrow{p_{sys} \cdot o_{engine}[\emptyset]\bar{x}_e\bar{d}} s_{engine} \mid [\bar{x}_e := \bar{d}]$$

Recovery system

The on-board recovery system (s_{sys}) receives failure signals

$$[\bar{x}_e, \bar{x}_b, \ldots]\,(\ldots + p_{sys} \cdot o_{engine}?\bar{x}_e.s_{engine} + p_{sys} \cdot o_{breakes}?\bar{x}_b.s_{breaks} + \ldots)$$

- Recovery services ($s_{engine}, s_{breaks}, \ldots$) are activated and variables ($\bar{x}_e, \bar{x}_b, \ldots$) are initialized with sensor diagnostic data

E.g., the diagnostic system reports an engine failure

$$s_{sys} \mid p_{sys} \cdot o_{engine}!\bar{d} \xrightarrow{p_{sys} \cdot o_{engine}\,[\emptyset]\,\bar{x}_e\,\bar{d}} \ldots$$

Recovery system

The on-board recovery system (s_{sys}) receives failure signals

$$[\bar{x}_e, \bar{x}_b, \ldots]\,(\ldots + p_{sys} \cdot o_{engine}?\bar{x}_e.s_{engine} + p_{sys} \cdot o_{breakes}?\bar{x}_b.s_{breaks} + \ldots)$$

- Recovery services ($s_{engine}, s_{breaks}, \ldots$) are activated and variables ($\bar{x}_e, \bar{x}_b, \ldots$) are initialized with sensor diagnostic data

E.g., the diagnostic system reports an engine failure

$$s_{sys} \mid p_{sys} \cdot o_{engine}!\bar{d} \xrightarrow{p_{sys} \cdot o_{engine}\,\lfloor\emptyset\rfloor\,\bar{x}_e\,\bar{d}} s_{engine} \cdot \{\bar{x}_e \mapsto \bar{d}\}$$

Recovery system

The on-board recovery system (s_{sys}) receives failure signals

$$[\bar{x}_e, \bar{x}_b, \ldots]\,(\ \ldots + p_{sys} \cdot o_{engine}?\bar{x}_e.s_{engine} + p_{sys} \cdot o_{breakes}?\bar{x}_b.s_{breaks} + \ldots\ )$$

- Recovery services ($s_{engine}, s_{breaks}, \ldots$) are activated and variables ($\bar{x}_e, \bar{x}_b, \ldots$) are initialized with sensor diagnostic data

E.g., the diagnostic system reports an engine failure

$$s_{sys} \mid p_{sys} \cdot o_{engine}!\bar{d} \xrightarrow{p_{sys} \cdot o_{engine}\ \lfloor\emptyset\rfloor\ \bar{x}_e\ \bar{d}} s_{engine} \cdot \{\bar{x}_e \mapsto \bar{d}\}$$

Recovery system

The on-board recovery system (s_{sys}) receives failure signals

$$[\bar{x}_e, \bar{x}_b, \ldots]\,(\ \ldots + p_{sys} \cdot o_{engine}?\bar{x}_e.s_{engine} + p_{sys} \cdot o_{breakes}?\bar{x}_b.s_{breaks} + \ldots\ )$$

- Recovery services ($s_{engine}, s_{breaks}, \ldots$) are activated and variables ($\bar{x}_e, \bar{x}_b, \ldots$) are initialized with sensor diagnostic data

E.g., the diagnostic system reports an engine failure

$$s_{sys} \mid p_{sys} \cdot o_{engine}!\bar{d} \xrightarrow{p_{sys} \cdot o_{engine}\ \lfloor\emptyset\rfloor\ \bar{x}_e\ \bar{d}} s_{engine} \cdot \{\bar{x}_e \mapsto \bar{d}\}$$

The engine recovery: service booking

$$s_{engine} = [\, chargeCC \,;\, booking \,:\, FH \,:\, nil \,]_{cc}$$
$$booking = [\, orderGA \,;\, (orderTT \mid [\, orderRC(y_{gps}) \,:\, redirect \,]_{rc}) \,:\, cancelGA \,]_{ga}$$

y_{gps} = garage's GPS position (initialized by the *orderGA*)

The engine recovery: service booking

$$s_{engine} = [\, chargeCC \,;\, booking \;:\; FH \;:\; nil \,]_{cc}$$
$$booking = [\, orderGA \,;\, (orderTT \mid [\, orderRC(y_{gps}) \;:\; redirect \,]_{rc}) \;:\; cancelGA \,]_{ga}$$

y_{gps} = garage's GPS position (initialized by the *orderGA*)

Successfull computations

$$chargeCC \,;\, orderGA \,;\, orderTT \,;\, orderRC(g_{gps})$$
$$chargeCC \,;\, orderGA \,;\, orderRC(g_{gps}) \,;\, orderTT$$

The engine recovery: service booking

$$s_{engine} = [\, chargeCC \,;\, booking \;:\; FH \;:\; nil \,]_{cc}$$
$$booking = [\, orderGA \,;\, (orderTT \mid [\, orderRC(y_{gps}) \;:\; redirect \,]_{rc}) \;:\; cancelGA \,]_{ga}$$

y_{gps} = garage's GPS position (initialized by the *orderGA*)

Successfull computations

$$chargeCC \,;\, orderGA \,;\, orderTT \,;\, orderRC(g_{gps})$$
$$chargeCC \,;\, orderGA \,;\, orderRC(g_{gps}) \,;\, orderTT$$

Computations if *orderTT* fails and *orderRC* succeeds

$$chargeCC \,;\, orderGA \,;\, orderRC(g_{gps}) \,;\, (redirect \mid cancelGA)$$

The engine recovery: service booking

$$s_{engine} = [\, chargeCC \,;\, booking \;:\; FH \;:\; nil \,]_{cc}$$
$$booking = [\, orderGA \,;\, (orderTT \mid [\, orderRC(y_{gps}) \;:\; redirect \,]_{rc}) \;:\; cancelGA \,]_{ga}$$

y_{gps} = garage's GPS position (initialized by the *orderGA*)

Successfull computations

$$chargeCC \,;\, orderGA \,;\, orderTT \,;\, orderRC(g_{gps})$$
$$chargeCC \,;\, orderGA \,;\, orderRC(g_{gps}) \,;\, orderTT$$

Computations if *orderTT* fails and *orderRC* succeeds

$$chargeCC \,;\, orderGA \,;\, orderRC(g_{gps}) \,;\, (redirect \mid cancelGA)$$

The failure of both *orderGA* and *orderRC* means that *chargeCC* must be revoked by the fault handler *FH*

The engine recovery: service booking

$$s_{engine} = [\, chargeCC \,;\, booking \;:\; FH \;:\; nil \,]_{cc}$$
$$booking = [\, orderGA \,;\, (orderTT \mid [\, orderRC(y_{gps}) \;:\; redirect \,]_{rc}) \;:\; cancelGA \,]_{ga}$$

y_{gps} = garage's GPS position (initialized by the *orderGA*)

Fault handlers

$$FH \;=\; \mathbf{catch}(\phi_{ga})\{failGA\} \;:\; \mathbf{catch}(\phi_{tt})\{failTT\}$$

The engine recovery: service booking

$$s_{engine} = [\, chargeCC \,;\, booking \;:\; FH \;:\; nil \,]_{cc}$$
$$booking = [\, orderGA \,;\, (orderTT \mid [\, orderRC(y_{gps}) \;:\; redirect \,]_{rc}) \;:\; cancelGA \,]_{ga}$$

y_{gps} = garage's GPS position (initialized by the *orderGA*)

Fault handlers

$$FH \;=\; \mathbf{catch}(\phi_{ga})\{failGA\} \;:\; \mathbf{catch}(\phi_{tt})\{failTT\}$$

If finding a garage appointment fails

$$failGA \;=\; [x_{res}]\, orderRC(x_{gps}); \mathbf{if}\ (\neg x_{res})\ \mathbf{then}\ \{revokeCharge\}$$

x_{gps} = broken down car's GPS position (initialized by the system)

x_{res} = success of car rental order (initialized by $orderRC(x_{gps})$)

The engine recovery: service booking

$$s_{engine} = [\, chargeCC \,;\, booking \;:\; FH \;:\; nil \,]_{cc}$$
$$booking = [\, orderGA \,;\, (orderTT \mid [\, orderRC(y_{gps}) \;:\; redirect \,]_{rc}) \;:\; cancelGA \,]_{ga}$$

y_{gps} = garage's GPS position (initialized by the *orderGA*)

Fault handlers

$$FH = \mathbf{catch}(\phi_{ga})\{failGA\} \;:\; \mathbf{catch}(\phi_{tt})\{failTT\}$$

If finding a tow truck fails

$$failTT = \mathbf{undo}(ga) \mid \mathbf{if}\ (x_{orderRC})\ \mathbf{then}\ \{\mathbf{undo}(rc)\}\ \mathbf{else}\ \{revokeCharge\}$$

$x_{orderRC}$ = success of car rental order (initialized by $orderRC(y_{gps})$)

P.S. **undo**(*rc*) triggers the compensation activity *redirect*

Charge credit card & Order garage appointment

The credit card can be charged with a security amount

$$\begin{aligned} \mathit{chargeCC} &= p_{bank} \cdot o_{charge}!\langle cc, amount\rangle \mid [z]\, p_{rs} \cdot o_{resp}?\langle z\rangle \\ \mathit{revokeCharge} &= p_{bank} \cdot o_{revoke}!\langle cc, amount\rangle \end{aligned}$$

$p_{bank} \cdot o_{charge}$ = bank service, cc = credit card data, $amount$ = security amount

p_{rs} = private partner name used for the reply

Finding a garage appointment exploits the proxy service $p_{proxy} \cdot o_{find\_ga}$

$$\begin{aligned} \mathit{orderGA} &= p_{proxy} \cdot o_{find\_ga}!\langle p_{rs}, x_{gps}, x_{pref}\rangle \\ &\quad \mid [z_g]\, p_{rs} \cdot o_{find\_ga\_res}?\langle z_g, y_{gps}\rangle.z_g \cdot o_{order\_ga}!\langle p_{rs}, x_{gps}\rangle \\ &\quad \mid [x_{res}]\, p_{rs} \cdot o_{order\_ga\_res}?\langle x_{res}\rangle.\mathbf{if}\ (\neg x_{res})\ \mathbf{then}\ \{\mathbf{throw}(\phi_{ga})\} \\ \mathit{cancelGA} &= z_g \cdot o_{cancel}!\langle p_{rs}\rangle \end{aligned}$$

x_{gps} = car's GPS position (initialized by the system), x_{pref} = user preferences, y_{gps} = garage's GPS position

Charge credit card & Order garage appointment

The credit card can be charged with a security amount

$$\begin{aligned} \mathit{chargeCC} &= p_{bank} \cdot o_{charge}!\langle cc, amount\rangle \mid [z]\, p_{rs} \cdot o_{resp}?\langle z\rangle \\ \mathit{revokeCharge} &= p_{bank} \cdot o_{revoke}!\langle cc, amount\rangle \end{aligned}$$

$p_{bank} \cdot o_{charge}$ = bank service, cc = credit card data, $amount$ = security amount

p_{rs} = private partner name used for the reply

Finding a garage appointment exploits the proxy service $p_{proxy} \cdot o_{find\_ga}$

$$\begin{aligned} \mathit{orderGA} &= p_{proxy} \cdot o_{find\_ga}!\langle p_{rs}, x_{gps}, x_{pref}\rangle \\ &\quad \mid [z_g]\, p_{rs} \cdot o_{find\_ga\_res}?\langle z_g, y_{gps}\rangle.z_g \cdot o_{order\_ga}!\langle p_{rs}, x_{gps}\rangle \\ &\quad \mid [x_{res}]\, p_{rs} \cdot o_{order\_ga\_res}?\langle x_{res}\rangle.\mathbf{if}\ (\neg x_{res})\ \mathbf{then}\ \{\mathbf{throw}(\phi_{ga})\} \\ \mathit{cancelGA} &= z_g \cdot o_{cancel}!\langle p_{rs}\rangle \end{aligned}$$

x_{gps} = car's GPS position (initialized by the system), x_{pref} = user preferences, y_{gps} = garage's GPS position

Order tow truck & Order car rental

Finding a tow truck exploits the proxy service $p_{proxy} \cdot o_{find\_tt}$

$$\begin{aligned} orderTT \;=\; & p_{proxy} \cdot o_{find\_tt}!\langle p_{rs}, x_{gps}, y_{gps}, y_{pref}\rangle \\ & \mid [z_{tt}]\, p_{rs} \cdot o_{find\_tt\_res}?\langle z_{tt}\rangle . z_g \cdot o_{order\_tt}!\langle p_{rs}\rangle \\ & \mid [x_{res}]\, p_{rs} \cdot o_{order\_tt\_res}?\langle x_{res}\rangle . \textbf{if } (\neg x_{res}) \textbf{ then } \{\textbf{throw}(\phi_{tt})\} \end{aligned}$$

p_{rs} = private partner name used for the reply, x_{gps} = car's GPS position (initialized by the system), y_{pref} = user preferences, y_{gps} = garage's GPS position

Finding a car rental exploits the proxy service $p_{proxy} \cdot o_{find\_rc}$

$$\begin{aligned} orderRC(w_{gps}) \;=\; & p_{proxy} \cdot o_{find\_rc}!\langle p_{rs}, w_{gps}, z_{pref}\rangle \\ & \mid [z_g]\, p_{rs} \cdot o_{find\_rc\_res}?\langle z_g\rangle . z_g \cdot o_{order\_rc}!\langle p_{rs}, w_{gps}\rangle \\ & \mid p_{rs} \cdot o_{order\_rc\_res}?\langle x_{orderRC}\rangle \\ redirect \;=\; & z_g \cdot o_{redirect}!\langle p_{rs}, x_{gps}\rangle \end{aligned}$$

w_{gps} = car's GPS position or garage's GPS position, z_{pref} = user preferences

Order tow truck & Order car rental

Finding a tow truck exploits the proxy service $p_{proxy} \cdot o_{find\_tt}$

$$
\begin{array}{rcl}
orderTT & = & p_{proxy} \cdot o_{find\_tt}!\langle p_{rs}, x_{gps}, y_{gps}, y_{pref} \rangle \\
 & & \mid [z_{tt}]\, p_{rs} \cdot o_{find\_tt\_res}?\langle z_{tt} \rangle . z_g \cdot o_{order\_tt}!\langle p_{rs} \rangle \\
 & & \mid [x_{res}]\, p_{rs} \cdot o_{order\_tt\_res}?\langle x_{res} \rangle . \textbf{if } (\neg x_{res}) \textbf{ then } \{\textbf{throw}(\phi_{tt})\}
\end{array}
$$

p_{rs} = private partner name used for the reply, x_{gps} = car's GPS position (initialized by the system), y_{pref} = user preferences, y_{gps} = garage's GPS position

Finding a car rental exploits the proxy service $p_{proxy} \cdot o_{find\_rc}$

$$
\begin{array}{rcl}
orderRC(w_{gps}) & = & p_{proxy} \cdot o_{find\_rc}!\langle p_{rs}, w_{gps}, z_{pref} \rangle \\
 & & \mid [z_g]\, p_{rs} \cdot o_{find\_rc\_res}?\langle z_g \rangle . z_g \cdot o_{order\_rc}!\langle p_{rs}, w_{gps} \rangle \\
 & & \mid p_{rs} \cdot o_{order\_rc\_res}?\langle x_{orderRC} \rangle \\
redirect & = & z_g \cdot o_{redirect}!\langle p_{rs}, x_{gps} \rangle
\end{array}
$$

w_{gps} = car's GPS position or garage's GPS position, z_{pref} = user preferences

Conclusions

- COWS permits modelling different and typical aspects of services and Web services technologies, such as
  - multiple start activities, receive conflicts, routing of correlated messages, service instances and interactions among them
- COWS can express the most common workflow patterns and encode other process and orchestration languages

COWS Web Site `http://rap.dsi.unifi.it/cows/`

Conclusions

- COWS permits modelling different and typical aspects of services and Web services technologies, such as
  - multiple start activities, receive conflicts, routing of correlated messages, service instances and interactions among them
- COWS can express the most common workflow patterns and encode other process and orchestration languages

COWS Web Site `http://rap.dsi.unifi.it/cows/`

Conclusions

- COWS permits modelling different and typical aspects of services and Web services technologies, such as
  - multiple start activities, receive conflicts, routing of correlated messages, service instances and interactions among them
- COWS can express the most common workflow patterns and encode other process and orchestration languages

COWS Web Site `http://rap.dsi.unifi.it/cows/`

Ongoing work

Programme: To lay rigorous methodological foundations for specification and validation of SOC middleware and applications

- We have defined a type system for specifying and forcing policies for constraining the partners, and hence the services, that can safely access any given datum (*SENSORIA TASK T3.2*)
- We are developing more powerful type systems by exploiting *resource usage types* (N. Kobayashi et Al.)
  - permit to express and enforce policies for , e.g., disciplining partners usage, constraining the sequences of messages accepted by services, checking that interaction between partners obeys a given protocol
  - need non trivial adaptations to deal with all COWS peculiar features (e.g. killing and protection activities)
- We also plan to define behavioural equivalences
  - They could provide a means to establish formal correspondences between different views (abstraction levels) of a service, e.g. the contract it has to honour and its true implementation

Ongoing work

Programme: To lay rigorous methodological foundations for specification and validation of SOC middleware and applications

- We have defined a type system for specifying and forcing policies for constraining the partners, and hence the services, that can safely access any given datum (*SENSORIA TASK T3.2*)
- We are developing more powerful type systems by exploiting *resource usage types* (N. Kobayashi et Al.)
  - permit to express and enforce policies for , e.g., disciplining partners usage, constraining the sequences of messages accepted by services, checking that interaction between partners obeys a given protocol
  - need non trivial adaptations to deal with all COWS peculiar features (e.g. killing and protection activities)
- We also plan to define behavioural equivalences
  - They could provide a means to establish formal correspondences between different views (abstraction levels) of a service, e.g. the contract it has to honour and its true implementation

Ongoing work

Programme: To lay rigorous methodological foundations for specification and validation of SOC middleware and applications

- We have defined a type system for specifying and forcing policies for constraining the partners, and hence the services, that can safely access any given datum (*SENSORIA TASK T3.2*)
- We are developing more powerful type systems by exploiting *resource usage types* (N. Kobayashi et Al.)
  - permit to express and enforce policies for , e.g., disciplining partners usage, constraining the sequences of messages accepted by services, checking that interaction between partners obeys a given protocol
  - need non trivial adaptations to deal with all COWS peculiar features (e.g. killing and protection activities)
- We also plan to define behavioural equivalences
  - They could provide a means to establish formal correspondences between different views (abstraction levels) of a service, e.g. the contract it has to honour and its true implementation

Ongoing work

Programme: To lay rigorous methodological foundations for specification and validation of SOC middleware and applications

- We have defined a type system for specifying and forcing policies for constraining the partners, and hence the services, that can safely access any given datum (*SENSORIA TASK T3.2*)
- We are developing more powerful type systems by exploiting *resource usage types* (N. Kobayashi et Al.)
  - permit to express and enforce policies for , e.g., disciplining partners usage, constraining the sequences of messages accepted by services, checking that interaction between partners obeys a given protocol
  - need non trivial adaptations to deal with all COWS peculiar features (e.g. killing and protection activities)
- We also plan to define behavioural equivalences
  - They could provide a means to establish formal correspondences between different views (abstraction levels) of a service, e.g. the contract it has to honour and its true implementation

Ongoing work

Programme: To lay rigorous methodological foundations for specification and validation of SOC middleware and applications

- We have defined a type system for specifying and forcing policies for constraining the partners, and hence the services, that can safely access any given datum (*SENSORIA TASK T3.2*)
- We are developing more powerful type systems by exploiting *resource usage types* (N. Kobayashi et Al.)
  - permit to express and enforce policies for , e.g., disciplining partners usage, constraining the sequences of messages accepted by services, checking that interaction between partners obeys a given protocol
  - need non trivial adaptations to deal with all COWS peculiar features (e.g. killing and protection activities)
- We also plan to define behavioural equivalences
  - They could provide a means to establish formal correspondences between different views (abstraction levels) of a service, e.g. the contract it has to honour and its true implementation

Ongoing work

Programme: To lay rigorous methodological foundations for specification and validation of SOC middleware and applications

- We have defined a type system for specifying and forcing policies for constraining the partners, and hence the services, that can safely access any given datum (*SENSORIA TASK T3.2*)
- We are developing more powerful type systems by exploiting *resource usage types* (N. Kobayashi et Al.)
  - permit to express and enforce policies for , e.g., disciplining partners usage, constraining the sequences of messages accepted by services, checking that interaction between partners obeys a given protocol
  - need non trivial adaptations to deal with all COWS peculiar features (e.g. killing and protection activities)
- We also plan to define behavioural equivalences
  - They could provide a means to establish formal correspondences between different views (abstraction levels) of a service, e.g. the contract it has to honour and its true implementation

Thank you!

Structural congruence

$$
\begin{array}{rcll}
* \mathbf{0} & \equiv & \mathbf{0} & (repl_1) \\
* s & \equiv & s \mid * s & (repl_2) \\
\{\!|\mathbf{0}|\!\} & \equiv & \mathbf{0} & (prot_1) \\
\{\!|\, \{\!|s|\!\} \,|\!\} & \equiv & \{\!|s|\!\} & (prot_2) \\
\{\!|[d]\, s|\!\} & \equiv & [d]\, \{\!|s|\!\} & (prot_3) \\
[d]\, \mathbf{0} & \equiv & \mathbf{0} & (delim_1) \\
[d_1]\, [d_2]\, s & \equiv & [d_2]\, [d_1]\, s & (delim_2) \\
s_1 \mid [d]\, s_2 & \equiv & [d]\, (s_1 \mid s_2) \quad \text{if } d \notin \mathrm{fd}(s_1) \cup \mathrm{fk}(s_2) & (delim_3)
\end{array}
$$

$$
\frac{s \equiv s_1 \quad s_1 \xrightarrow{\alpha} s_2 \quad s_2 \equiv s'}{s \xrightarrow{\alpha} s'} \; (cong)
$$

Structural congruence

$$
\begin{array}{rcll}
* \mathbf{0} & \equiv & \mathbf{0} & (repl_1) \\
* s & \equiv & s \mid * s & (repl_2) \\
\{\!|\mathbf{0}|\!\} & \equiv & \mathbf{0} & (prot_1) \\
\{\!|\, \{\!|s|\!\} \,|\!\} & \equiv & \{\!|s|\!\} & (prot_2) \\
\{\!|[d]\, s|\!\} & \equiv & [d]\, \{\!|s|\!\} & (prot_3) \\
[d]\, \mathbf{0} & \equiv & \mathbf{0} & (delim_1) \\
[d_1]\, [d_2]\, s & \equiv & [d_2]\, [d_1]\, s & (delim_2) \\
s_1 \mid [d]\, s_2 & \equiv & [d]\, (s_1 \mid s_2) \quad \text{if } d \notin \mathrm{fd}(s_1) \cup \mathrm{fk}(s_2) & (delim_3)
\end{array}
$$

$$
\frac{s \equiv s_1 \quad s_1 \xrightarrow{\alpha} s_2 \quad s_2 \equiv s'}{s \xrightarrow{\alpha} s'} \; (cong)
$$

Invoke/receive activities & Choice

$$\frac{[\![\bar{e}]\!] = \bar{v}}{p \cdot o!\bar{e} \xrightarrow{(p \cdot o) \lhd \bar{v}} \mathbf{0}} \ (inv) \qquad p \cdot o?\bar{w}.s \xrightarrow{(p \cdot o) \rhd \bar{w}} s \ (rec)$$

$$\frac{g_1 \xrightarrow{\alpha} s}{g_1 + g_2 \xrightarrow{\alpha} s} \ (choice)$$

Function $[\![\_]\!]$ evaluates *closed* expressions (i.e. expressions without variables): it takes a closed expression and returns a value

Invoke/receive activities & Choice

$$\frac{[\![\bar{e}]\!] = \bar{v}}{p \cdot o!\bar{e} \xrightarrow{(p \cdot o) \lhd \bar{v}} \mathbf{0}} \ (inv) \qquad p \cdot o?\bar{w}.s \xrightarrow{(p \cdot o) \rhd \bar{w}} s \ (rec)$$

$$\frac{g_1 \xrightarrow{\alpha} s}{g_1 + g_2 \xrightarrow{\alpha} s} \ (choice)$$

Function $[\![\_]\!]$ evaluates *closed* expressions (i.e. expressions without variables): it takes a closed expression and returns a value

Kill activity & Protection

$$\mathbf{kill}(k) \xrightarrow{\dagger k} \mathbf{0} \quad (kill) \qquad \frac{s_1 \xrightarrow{\dagger k} s_1'}{s_1 \mid s_2 \xrightarrow{\dagger k} s_1' \mid halt(s_2)} \quad (par_{kill})$$

$$\frac{s \xrightarrow{\dagger k} s'}{[k]\, s \xrightarrow{\dagger} [k]\, s'} \quad (del_{kill}) \qquad \frac{s \xrightarrow{\alpha} s'}{\{\!|s|\!\} \xrightarrow{\alpha} \{\!|s'|\!\}} \quad (prot)$$

Function $halt(\_)$, given a service s, returns the service obtained by only retaining the protected activities inside s

$$halt(\mathbf{kill}(k)) = halt(u_1 \cdot u_2!\bar{e}) = halt(g) = \mathbf{0}$$

$$halt(\{\!|s|\!\}) = \{\!|s|\!\} \qquad halt(s_1 \mid s_2) = halt(s_1) \mid halt(s_2)$$

$$halt([d]\, s) = [d]\, halt(s) \qquad halt(*\, s) = *\, halt(s)$$

Kill activity & Protection

$$\mathbf{kill}(k) \xrightarrow{\dagger k} \mathbf{0} \quad (kill) \qquad \frac{s_1 \xrightarrow{\dagger k} s_1'}{s_1 \mid s_2 \xrightarrow{\dagger k} s_1' \mid halt(s_2)} \quad (par_{kill})$$

$$\frac{s \xrightarrow{\dagger k} s'}{[k]\, s \xrightarrow{\dagger} [k]\, s'} \quad (del_{kill}) \qquad \frac{s \xrightarrow{\alpha} s'}{\{\!|s|\!\} \xrightarrow{\alpha} \{\!|s'|\!\}} \quad (prot)$$

Function *halt*(_), given a service s, returns the service obtained by only retaining the protected activities inside s

$$halt(\mathbf{kill}(k)) = halt(u_1 \cdot u_2!\bar{e}) = halt(g) = \mathbf{0}$$

$halt(\{\!|s|\!\}) = \{\!|s|\!\}$ $\qquad$ $halt(s_1 \mid s_2) = halt(s_1) \mid halt(s_2)$

$halt([d]\, s) = [d]\, halt(s)$ $\qquad$ $halt(* s) = * halt(s)$

Kill activity & Protection

$$\mathbf{kill}(k) \xrightarrow{\dagger k} \mathbf{0} \quad (kill) \qquad \frac{s_1 \xrightarrow{\dagger k} s_1'}{s_1 \mid s_2 \xrightarrow{\dagger k} s_1' \mid halt(s_2)} \quad (par_{kill})$$

$$\frac{s \xrightarrow{\dagger k} s'}{[k]\, s \xrightarrow{\dagger} [k]\, s'} \quad (del_{kill}) \qquad \frac{s \xrightarrow{\alpha} s'}{\{\!|s|\!\} \xrightarrow{\alpha} \{\!|s'|\!\}} \quad (prot)$$

Function $halt(\_)$, given a service s, returns the service obtained by only retaining the protected activities inside s

$$halt(\mathbf{kill}(k)) = halt(u_1 \cdot u_2!\bar{e}) = halt(g) = \mathbf{0}$$

$$halt(\{\!|s|\!\}) = \{\!|s|\!\} \qquad halt(s_1 \mid s_2) = halt(s_1) \mid halt(s_2)$$

$$halt([d]\, s) = [d]\, halt(s) \qquad halt(*\, s) = *\, halt(s)$$

Kill activity & Protection

$$\mathbf{kill}(k) \xrightarrow{\dagger k} \mathbf{0} \quad (kill) \qquad \frac{s_1 \xrightarrow{\dagger k} s_1'}{s_1 \mid s_2 \xrightarrow{\dagger k} s_1' \mid halt(s_2)} \quad (par_{kill})$$

$$\frac{s \xrightarrow{\dagger k} s'}{[k]\, s \xrightarrow{\dagger} [k]\, s'} \quad (del_{kill}) \qquad \frac{s \xrightarrow{\alpha} s'}{\{\!|s|\!\} \xrightarrow{\alpha} \{\!|s'|\!\}} \quad (prot)$$

Function $halt(\_)$, given a service s, returns the service obtained by only retaining the protected activities inside s

$$halt(\mathbf{kill}(k)) = halt(u_1 \cdot u_2!\bar{e}) = halt(g) = \mathbf{0}$$

$$halt(\{\!|s|\!\}) = \{\!|s|\!\} \qquad halt(s_1 \mid s_2) = halt(s_1) \mid halt(s_2)$$

$$halt([d]\, s) = [d]\, halt(s) \qquad halt(*\, s) = *\, halt(s)$$

Kill activity & Protection

$$\mathbf{kill}(k) \xrightarrow{\dagger k} \mathbf{0} \quad (kill) \qquad \frac{s_1 \xrightarrow{\dagger k} s_1'}{s_1 \mid s_2 \xrightarrow{\dagger k} s_1' \mid halt(s_2)} \quad (par_{kill})$$

$$\frac{s \xrightarrow{\dagger k} s'}{[k]\, s \xrightarrow{\dagger} [k]\, s'} \quad (del_{kill}) \qquad \frac{s \xrightarrow{\alpha} s'}{\{\!|s|\!\} \xrightarrow{\alpha} \{\!|s'|\!\}} \quad (prot)$$

Function *halt*(_), given a service s, returns the service obtained by only retaining the protected activities inside s

$$halt(\mathbf{kill}(k)) = halt(u_1 \cdot u_2!\bar{e}) = halt(g) = \mathbf{0}$$

$halt(\{\!|s|\!\}) = \{\!|s|\!\}$ $\qquad$ $halt(s_1 \mid s_2) = halt(s_1) \mid halt(s_2)$

$halt([d]\, s) = [d]\, halt(s)$ $\qquad$ $halt(* s) = * halt(s)$

Delimitation

$$\frac{s \xrightarrow{\dagger k} s'}{[k]\, s \xrightarrow{\dagger} [k]\, s'}\ (del_{kill}) \qquad \frac{s \xrightarrow{p \cdot o \lfloor \sigma \uplus \{x \mapsto v'\} \rfloor\ \bar{w}\, \bar{v}} s'}{[x]\, s \xrightarrow{p \cdot o \lfloor \sigma \rfloor\ \bar{w}\, \bar{v}} s' \cdot \{x \mapsto v'\}}\ (del_{sub})$$

$$\frac{s \xrightarrow{\alpha} s' \quad d \notin d(\alpha) \quad s \equiv \mathbb{A}[\![\mathbf{kill}(d)]\!] \Rightarrow \alpha = \dagger, \dagger k}{[d]\, s \xrightarrow{\alpha} [d]\, s'}\ (del_{pass})$$

Substitution σ: function from variables to values (collection of pairs $x \mapsto v$)
$\sigma_1 \uplus \sigma_2$ denotes the union of σ_1 and σ_2 when they have disjoint domains
$|\sigma|$ is the number of pairs in σ

Active context $\mathbb{A}$: service with a hole s.t. if the term $\mathbb{A}[\![s]\!]$ resulting from filling the hole with an s is a service, then it can immediately perform an activity of s

$$\mathbb{A} \quad ::= \quad [\![\cdot]\!] \mid \mathbb{A} + g \mid g + \mathbb{A} \mid \mathbb{A} \mid s \mid s \mid \mathbb{A} \mid \{\!|\mathbb{A}|\!\} \mid [d]\, \mathbb{A} \mid * \mathbb{A}$$

Delimitation

$$\frac{s \xrightarrow{\dagger k} s'}{[k]\, s \xrightarrow{\dagger} [k]\, s'}\ (del_{kill}) \qquad \frac{s \xrightarrow{p \cdot o \lfloor \sigma \uplus \{x \mapsto v'\} \rfloor\ \bar{w}\ \bar{v}} s'}{[x]\, s \xrightarrow{p \cdot o \lfloor \sigma \rfloor\ \bar{w}\ \bar{v}} s' \cdot \{x \mapsto v'\}}\ (del_{sub})$$

$$\frac{s \xrightarrow{\alpha} s' \quad d \notin d(\alpha) \quad s \equiv \mathbb{A}[\![\mathbf{kill}(d)]\!] \Rightarrow \alpha = \dagger, \dagger k}{[d]\, s \xrightarrow{\alpha} [d]\, s'}\ (del_{pass})$$

Substitution σ: function from variables to values (collection of pairs $x \mapsto v$)
$\sigma_1 \uplus \sigma_2$ denotes the union of σ_1 and σ_2 when they have disjoint domains
$|\sigma|$ is the number of pairs in σ

Active context $\mathbb{A}$: service with a hole s.t. if the term $\mathbb{A}[\![s]\!]$ resulting from filling the hole with an s is a service, then it can immediately perform an activity of s

$$\mathbb{A} \ ::= \ [\![\cdot]\!] \mid \mathbb{A} + g \mid g + \mathbb{A} \mid \mathbb{A} \mid s \mid s \mid \mathbb{A} \mid \{\!|\mathbb{A}|\!\} \mid [d]\, \mathbb{A} \mid * \mathbb{A}$$

Delimitation

$$\frac{s \xrightarrow{\dagger k} s'}{[k]\, s \xrightarrow{\dagger} [k]\, s'}\ (del_{kill}) \qquad \frac{s \xrightarrow{p \cdot o \lfloor \sigma \uplus \{x \mapsto v'\} \rfloor\ \bar{w}\, \bar{v}} s'}{[x]\, s \xrightarrow{p \cdot o \lfloor \sigma \rfloor\ \bar{w}\, \bar{v}} s' \cdot \{x \mapsto v'\}}\ (del_{sub})$$

$$\frac{s \xrightarrow{\alpha} s' \quad d \notin d(\alpha) \quad s \equiv \mathbb{A}[\![\mathbf{kill}(d)]\!] \Rightarrow \alpha = \dagger, \dagger k}{[d]\, s \xrightarrow{\alpha} [d]\, s'}\ (del_{pass})$$

Substitution σ: function from variables to values (collection of pairs $x \mapsto v$)
$\sigma_1 \uplus \sigma_2$ denotes the union of σ_1 and σ_2 when they have disjoint domains
$|\sigma|$ is the number of pairs in σ

Active context $\mathbb{A}$: service with a hole s.t. if the term $\mathbb{A}[\![s]\!]$ resulting from filling the hole with an s is a service, then it can immediately perform an activity of s

$$\mathbb{A} \ ::= \ [\![\cdot]\!] \mid \mathbb{A} + g \mid g + \mathbb{A} \mid \mathbb{A} \mid s \mid s \mid \mathbb{A} \mid \{\!|\mathbb{A}|\!\} \mid [d]\, \mathbb{A} \mid * \mathbb{A}$$

Communication

$$\frac{s_1 \xrightarrow{(p\cdot o)\rhd \bar{w}} s_1' \qquad s_2 \xrightarrow{(p\cdot o)\lhd \bar{v}} s_2' \qquad \mathcal{M}(\bar{w},\bar{v}) = \sigma \qquad \mathit{noc}(s_1 \mid s_2, p\cdot o, \bar{w}, \bar{v})}{s_1 \mid s_2 \xrightarrow{p\cdot o \lfloor \sigma \rfloor \bar{w}\,\bar{v}} s_1' \mid s_2'} \; (com)$$

$$\mathcal{M}(x,v) = \{x \mapsto v\} \qquad \mathcal{M}(v,v) = \emptyset \qquad \frac{\mathcal{M}(w_1,v_1) = \sigma_1 \qquad \mathcal{M}(\bar{w}_2,\bar{v}_2) = \sigma_2}{\mathcal{M}((w_1,\bar{w}_2),(v_1,\bar{v}_2)) = \sigma_1 \uplus \sigma_2}$$

$\mathit{noc}(s, p\cdot o, \bar{w}, \bar{v}) \triangleq (\, s \equiv \mathbb{A}[\![p\cdot o?\bar{w}'.s']\!] \wedge \mathcal{M}(\bar{w}',\bar{v}) = \sigma \,) \Rightarrow |\mathcal{M}(\bar{w},\bar{v})| \leqslant |\sigma|$

holds true if

- either there are no receive conflicts

  (namely, s cannot immediately perform a receive activity matching $\bar{v}$ over the endpoint $p\cdot o$ other than $p\cdot o?\bar{w}$)
- or $p\cdot o?\bar{w}$ is the 'most defined' matching receive

Communication

$$\frac{s_1 \xrightarrow{(p\cdot o)\rhd \bar{w}} s_1' \qquad s_2 \xrightarrow{(p\cdot o)\lhd \bar{v}} s_2' \qquad \mathcal{M}(\bar{w},\bar{v}) = \sigma \qquad \mathit{noc}(s_1 \mid s_2, p\cdot o, \bar{w}, \bar{v})}{s_1 \mid s_2 \xrightarrow{p\cdot o \lfloor \sigma \rfloor \bar{w}\,\bar{v}} s_1' \mid s_2'} \; (com)$$

$$\mathcal{M}(x,v) = \{x \mapsto v\} \qquad \mathcal{M}(v,v) = \emptyset \qquad \frac{\mathcal{M}(w_1,v_1) = \sigma_1 \qquad \mathcal{M}(\bar{w}_2,\bar{v}_2) = \sigma_2}{\mathcal{M}((w_1,\bar{w}_2),(v_1,\bar{v}_2)) = \sigma_1 \uplus \sigma_2}$$

$\mathit{noc}(s, p\cdot o, \bar{w}, \bar{v}) \triangleq (\, s \equiv \mathbb{A}[\![p\cdot o?\bar{w}'.s']\!] \wedge \mathcal{M}(\bar{w}',\bar{v}) = \sigma \,) \Rightarrow |\mathcal{M}(\bar{w},\bar{v})| \leqslant |\sigma|$

holds true if

- either there are no receive conflicts

  (namely, s cannot immediately perform a receive activity matching $\bar{v}$ over the endpoint $p\cdot o$ other than $p\cdot o?\bar{w}$)
- or $p\cdot o?\bar{w}$ is the 'most defined' matching receive

Communication

$$\frac{s_1 \xrightarrow{(p\cdot o)\rhd \bar{w}} s_1' \qquad s_2 \xrightarrow{(p\cdot o)\lhd \bar{v}} s_2' \qquad \mathcal{M}(\bar{w},\bar{v}) = \sigma \qquad \mathit{noc}(s_1 \mid s_2, p\cdot o, \bar{w}, \bar{v})}{s_1 \mid s_2 \xrightarrow{p\cdot o \lfloor \sigma \rfloor \ \bar{w}\ \bar{v}} s_1' \mid s_2'} \ (com)$$

$$\mathcal{M}(x,v) = \{x \mapsto v\} \qquad \mathcal{M}(v,v) = \emptyset \qquad \frac{\mathcal{M}(w_1,v_1) = \sigma_1 \qquad \mathcal{M}(\bar{w}_2,\bar{v}_2) = \sigma_2}{\mathcal{M}((w_1,\bar{w}_2),(v_1,\bar{v}_2)) = \sigma_1 \uplus \sigma_2}$$

$$\mathit{noc}(s, p\cdot o, \bar{w}, \bar{v}) \triangleq (\, s \equiv \mathbb{A}[\![p\cdot o?\bar{w}'.s']\!] \wedge \mathcal{M}(\bar{w}',\bar{v}) = \sigma \,) \Rightarrow |\mathcal{M}(\bar{w},\bar{v})| \leqslant |\sigma|$$

holds true if

- either there are no receive conflicts

  (namely, s cannot immediately perform a receive activity matching $\bar{v}$ over the endpoint $p\cdot o$ other than $p\cdot o?\bar{w}$)
- or $p\cdot o?\bar{w}$ is the 'most defined' matching receive

Parallel composition

$$\frac{s_1 \xrightarrow{\dagger k} s_1'}{s_1 \mid s_2 \xrightarrow{\dagger k} s_1' \mid halt(s_2)}\ (par_{kill})$$

$$\frac{s_1 \xrightarrow{p \cdot o \lfloor \sigma \rfloor \bar{w}\, \bar{v}} s_1' \qquad noc(s_2, p \cdot o, \bar{w}, \bar{v})}{s_1 \mid s_2 \xrightarrow{p \cdot o \lfloor \sigma \rfloor \bar{w}\, \bar{v}} s_1' \mid s_2}\ (par_{conf})$$

$$\frac{s_1 \xrightarrow{\alpha} s_1' \qquad \alpha \neq (p \cdot o \lfloor \sigma \rfloor \bar{w}\, \bar{v}), \dagger k}{s_1 \mid s_2 \xrightarrow{\alpha} s_1' \mid s_2}\ (par_{pass})$$

Parallel composition

$$\frac{s_1 \xrightarrow{\dagger k} s_1'}{s_1 \mid s_2 \xrightarrow{\dagger k} s_1' \mid halt(s_2)}\ (par_{kill})$$

$$\frac{s_1 \xrightarrow{p \cdot o \lfloor \sigma \rfloor \bar{w}\, \bar{v}} s_1' \qquad noc(s_2, p \cdot o, \bar{w}, \bar{v})}{s_1 \mid s_2 \xrightarrow{p \cdot o \lfloor \sigma \rfloor \bar{w}\, \bar{v}} s_1' \mid s_2}\ (par_{conf})$$

$$\frac{s_1 \xrightarrow{\alpha} s_1' \qquad \alpha \neq (p \cdot o \lfloor \sigma \rfloor \bar{w}\, \bar{v}), \dagger k}{s_1 \mid s_2 \xrightarrow{\alpha} s_1' \mid s_2}\ (par_{pass})$$

Parallel composition

$$\frac{s_1 \xrightarrow{\dagger k} s_1'}{s_1 \mid s_2 \xrightarrow{\dagger k} s_1' \mid halt(s_2)} \; (par_{kill})$$

$$\frac{s_1 \xrightarrow{p \cdot o \lfloor \sigma \rfloor \bar{w}\, \bar{v}} s_1' \qquad noc(s_2, p \cdot o, \bar{w}, \bar{v})}{s_1 \mid s_2 \xrightarrow{p \cdot o \lfloor \sigma \rfloor \bar{w}\, \bar{v}} s_1' \mid s_2} \; (par_{conf})$$

$$\frac{s_1 \xrightarrow{\alpha} s_1' \qquad \alpha \neq (p \cdot o \lfloor \sigma \rfloor \bar{w}\, \bar{v}), \dagger k}{s_1 \mid s_2 \xrightarrow{\alpha} s_1' \mid s_2} \; (par_{pass})$$

Labelled transition rules

$$\mathbf{kill}(k) \xrightarrow{\dagger k} \mathbf{0} \quad (kill) \qquad p \cdot o?\bar{w}.s \xrightarrow{(p\cdot o)\rhd \bar{w}} s \quad (rec) \qquad \frac{[\![\bar{e}]\!] = \bar{v}}{p \cdot o!\bar{e} \xrightarrow{(p\cdot o)\lhd \bar{v}} \mathbf{0}} \quad (inv)$$

$$\frac{g_1 \xrightarrow{\alpha} s}{g_1 + g_2 \xrightarrow{\alpha} s} \quad (choice) \qquad \frac{s \xrightarrow{p\cdot o\, \lfloor \sigma \uplus \{x \mapsto v'\} \rfloor\, \bar{w}\, \bar{v}} s'}{[x]\, s \xrightarrow{p\cdot o\, \lfloor \sigma \rfloor\, \bar{w}\, \bar{v}} s' \cdot \{x \mapsto v'\}} \quad (del_{sub}) \qquad \frac{s \xrightarrow{\dagger k} s'}{[k]\, s \xrightarrow{\dagger} [k]\, s'} \quad (del_{kill})$$

$$\frac{s \xrightarrow{\alpha} s' \quad d \notin d(\alpha) \quad s \equiv \mathbb{A}[\![\mathbf{kill}(d)]\!] \Rightarrow \alpha = \dagger, \dagger k}{[d]\, s \xrightarrow{\alpha} [d]\, s'} \quad (del_{pass}) \qquad \frac{s \xrightarrow{\alpha} s'}{\{\!|s|\!\} \xrightarrow{\alpha} \{\!|s'|\!\}} \quad (prot)$$

$$\frac{s_1 \xrightarrow{(p\cdot o)\rhd \bar{w}} s_1' \qquad s_2 \xrightarrow{(p\cdot o)\lhd \bar{v}} s_2' \qquad \mathcal{M}(\bar{w}, \bar{v}) = \sigma \qquad noc(s_1 \mid s_2, p \cdot o, \bar{w}, \bar{v})}{s_1 \mid s_2 \xrightarrow{p\cdot o\, \lfloor \sigma \rfloor\, \bar{w}\, \bar{v}} s_1' \mid s_2'} \quad (com)$$

$$\frac{s_1 \xrightarrow{p\cdot o\, \lfloor \sigma \rfloor\, \bar{w}\, \bar{v}} s_1' \qquad noc(s_2, p \cdot o, \bar{w}, \bar{v})}{s_1 \mid s_2 \xrightarrow{p\cdot o\, \lfloor \sigma \rfloor\, \bar{w}\, \bar{v}} s_1' \mid s_2} \quad (par_{conf}) \qquad \frac{s_1 \xrightarrow{\dagger k} s_1'}{s_1 \mid s_2 \xrightarrow{\dagger k} s_1' \mid halt(s_2)} \quad (par_{kill})$$

$$\frac{s_1 \xrightarrow{\alpha} s_1' \qquad \alpha \neq (p \cdot o\, \lfloor \sigma \rfloor\, \bar{w}\, \bar{v}), \dagger k}{s_1 \mid s_2 \xrightarrow{\alpha} s_1' \mid s_2} \quad (par_{pass}) \qquad \frac{s \equiv s_1 \quad s_1 \xrightarrow{\alpha} s_2 \quad s_2 \equiv s'}{s \xrightarrow{\alpha} s'} \quad (cong)$$

Orc: syntax

Orc is a recently proposed task orchestration language with applications in workflow, business process management, and Web service orchestration

$$
\begin{array}{rcll}
f, g & ::= & \mathbf{0} \mid S(w) \mid E(w) & (\textit{Expressions}) \\
& \mid & f > x > g & (\textit{sequential composition}) \\
& \mid & f \mid g & (\textit{symmetric parallel composition}) \\
& \mid & g \textbf{ where } x :\in f & (\textit{asymmetric parallel composition}) \\
w & ::= & x \mid v & (\textit{Parameters})
\end{array}
$$

S ranges over *site* names, x over *variables*, and v over *values*

Elementary expressions: $\mathbf{0}$, $S(w)$ (site call) and $E(w)$ (expression call)

Each *expression name* E has a unique declaration $E(x) \triangleq f$

x is *bound* in g in expressions $f > x > g$ and g **where** $x :\in f$

Orc: syntax

Orc is a recently proposed task orchestration language with applications in workflow, business process management, and Web service orchestration

$$
\begin{array}{rcll}
f, g & ::= & \mathbf{0} \mid S(w) \mid E(w) & (\textit{Expressions}) \\
 & \mid & f > x > g & (\textit{sequential composition}) \\
 & \mid & f \mid g & (\textit{symmetric parallel composition}) \\
 & \mid & g \textbf{ where } x :\in f & (\textit{asymmetric parallel composition}) \\
w & ::= & x \mid v & (\textit{Parameters})
\end{array}
$$

S ranges over *site* names, x over *variables*, and v over *values*

Elementary expressions: $\mathbf{0}$, $S(w)$ (site call) and $E(w)$ (expression call)

Each *expression name* E has a unique declaration $E(x) \triangleq f$

x is *bound* in g in expressions $f > x > g$ and g **where** $x :\in f$

Orc: syntax

Orc is a recently proposed task orchestration language with applications in workflow, business process management, and Web service orchestration

$$
\begin{array}{rcll}
f, g & ::= & \mathbf{0} \mid S(w) \mid E(w) & (\textit{Expressions}) \\
 & \mid & f > x > g & (\textit{sequential composition}) \\
 & \mid & f \mid g & (\textit{symmetric parallel composition}) \\
 & \mid & g \textbf{ where } x :\in f & (\textit{asymmetric parallel composition}) \\
w & ::= & x \mid v & (\textit{Parameters})
\end{array}
$$

S ranges over *site* names, x over *variables*, and v over *values*

Elementary expressions: $\mathbf{0}$, $S(w)$ (site call) and $E(w)$ (expression call)

Each *expression name* E has a unique declaration $E(x) \triangleq f$

x is *bound* in g in expressions $f > x > g$ and g **where** $x :\in f$

Orc: syntax

Orc is a recently proposed task orchestration language with applications in workflow, business process management, and Web service orchestration

$$
\begin{array}{rcll}
f, g & ::= & \mathbf{0} \mid S(w) \mid E(w) & (\textit{Expressions}) \\
 & \mid & f > x > g & (\textit{sequential composition}) \\
 & \mid & f \mid g & (\textit{symmetric parallel composition}) \\
 & \mid & g \textbf{ where } x :\in f & (\textit{asymmetric parallel composition}) \\
w & ::= & x \mid v & (\textit{Parameters})
\end{array}
$$

S ranges over *site* names, x over *variables*, and v over *values*

Elementary expressions: $\mathbf{0}$, $S(w)$ (site call) and $E(w)$ (expression call)

Each *expression name* E has a unique declaration $E(x) \triangleq f$

x is *bound* in g in expressions $f > x > g$ and g **where** $x :\in f$

Orc: encoding

For each site S, we define the service

$$*\,[x, y]\,\hat{S}?\langle x, y\rangle.y!e_x^S \qquad (1)$$

For each expression declaration $E(x) \triangleq f$ we define the service

$$*\,[y, z]\,\hat{E}?\langle y, z\rangle.[\hat{r}]\,(z!\hat{r} \mid [x]\,(\hat{r}?\langle x\rangle \mid \langle\!\langle f\rangle\!\rangle_y)\,) \qquad (2)$$

$[\![f]\!]_{\hat{r}}$ is the parallel composition of $\langle\!\langle f\rangle\!\rangle_{\hat{r}}$, of a service of the form (1) or (2) for each site or expression called in f, in any of the expressions called in f, and so on recursively

Orc: encoding

For each site S, we define the service

$$* [x, y] \, \hat{S}?\langle x, y \rangle . y! e_x^S \qquad (1)$$

For each expression declaration $E(x) \triangleq f$ we define the service

$$* [y, z] \, \hat{E}?\langle y, z \rangle . [\hat{r}] \, (z! \hat{r} \mid [x] \, (\hat{r}?\langle x \rangle \mid \langle\!\langle f \rangle\!\rangle_y)) \qquad (2)$$

$[\![f]\!]_{\hat{r}}$ is the parallel composition of $\langle\!\langle f \rangle\!\rangle_{\hat{r}}$, of a service of the form (1) or (2) for each site or expression called in f, in any of the expressions called in f, and so on recursively

Orc: encoding

For each site S, we define the service

$$* [x, y] \, \hat{S}?\langle x, y \rangle . y! e_x^S \qquad (1)$$

For each expression declaration $E(x) \triangleq f$ we define the service

$$* [y, z] \, \hat{E}?\langle y, z \rangle . [\hat{r}] \, (z! \hat{r} \mid [x] \, (\hat{r}?\langle x \rangle \mid \langle\!\langle f \rangle\!\rangle_y) \,) \qquad (2)$$

$[\![f]\!]_{\hat{r}}$ is the parallel composition of $\langle\!\langle f \rangle\!\rangle_{\hat{r}}$, of a service of the form (1) or (2) for each site or expression called in f, in any of the expressions called in f, and so on recursively

Orc: auxiliary function for the encoding

$$\langle\!\langle \mathbf{0} \rangle\!\rangle_{\hat{r}} = \mathbf{0} \qquad \langle\!\langle f \mid g \rangle\!\rangle_{\hat{r}} = \langle\!\langle f \rangle\!\rangle_{\hat{r}} \mid \langle\!\langle g \rangle\!\rangle_{\hat{r}}$$

$$\langle\!\langle S(w) \rangle\!\rangle_{\hat{r}} = \hat{S}!\langle w, \hat{r} \rangle \qquad \langle\!\langle E(w) \rangle\!\rangle_{\hat{r}} = [\hat{r}'] \, (\hat{E}!\langle \hat{r}, \hat{r}' \rangle \mid [z] \, \hat{r}'?\langle z \rangle . z!\langle w \rangle)$$

$$\langle\!\langle f > x > g \rangle\!\rangle_{\hat{r}} = [\hat{r}_f] \, (\langle\!\langle f \rangle\!\rangle_{\hat{r}_f} \mid \ * [x] \, \hat{r}_f?x . \langle\!\langle g \rangle\!\rangle_{\hat{r}})$$

$$\langle\!\langle g \ \mathbf{where} \ x :\in f \rangle\!\rangle_{\hat{r}} = [\hat{r}_f, x] \ (\ \langle\!\langle g \rangle\!\rangle_{\hat{r}} \mid [k] \, (\ \langle\!\langle f \rangle\!\rangle_{\hat{r}_f} \mid \hat{r}_f?x . \mathbf{kill}(k) \ ) \ )$$

Endpoint $\hat{r}$ returns the result of expressions evaluation

Orc: auxiliary function for the encoding

$$\langle\!\langle \mathbf{0} \rangle\!\rangle_{\hat{r}} = \mathbf{0} \qquad \langle\!\langle f \mid g \rangle\!\rangle_{\hat{r}} = \langle\!\langle f \rangle\!\rangle_{\hat{r}} \mid \langle\!\langle g \rangle\!\rangle_{\hat{r}}$$

$$\langle\!\langle S(w) \rangle\!\rangle_{\hat{r}} = \hat{S}!\langle w, \hat{r} \rangle \qquad \langle\!\langle E(w) \rangle\!\rangle_{\hat{r}} = [\hat{r}'] \, (\hat{E}!\langle \hat{r}, \hat{r}' \rangle \mid [z] \, \hat{r}'?\langle z \rangle . z!\langle w \rangle)$$

$$\langle\!\langle f > x > g \rangle\!\rangle_{\hat{r}} = [\hat{r}_f] \, (\langle\!\langle f \rangle\!\rangle_{\hat{r}_f} \mid \ * [x] \, \hat{r}_f?x . \langle\!\langle g \rangle\!\rangle_{\hat{r}})$$

$$\langle\!\langle g \ \mathbf{where} \ x :\in f \rangle\!\rangle_{\hat{r}} = [\hat{r}_f, x] \ (\ \langle\!\langle g \rangle\!\rangle_{\hat{r}} \mid [k] \, (\ \langle\!\langle f \rangle\!\rangle_{\hat{r}_f} \mid \hat{r}_f?x . \mathbf{kill}(k) \ ) \ )$$

Endpoint $\hat{r}$ returns the result of expressions evaluation

Orc: auxiliary function for the encoding

$$\langle\!\langle \mathbf{0} \rangle\!\rangle_{\hat{r}} = \mathbf{0} \qquad \langle\!\langle f \mid g \rangle\!\rangle_{\hat{r}} = \langle\!\langle f \rangle\!\rangle_{\hat{r}} \mid \langle\!\langle g \rangle\!\rangle_{\hat{r}}$$

$$\langle\!\langle S(w) \rangle\!\rangle_{\hat{r}} = \hat{S}!\langle w, \hat{r} \rangle \qquad \langle\!\langle E(w) \rangle\!\rangle_{\hat{r}} = [\hat{r}'] \, (\hat{E}!\langle \hat{r}, \hat{r}' \rangle \mid [z] \, \hat{r}'?\langle z \rangle . z!\langle w \rangle)$$

$$\langle\!\langle f > x > g \rangle\!\rangle_{\hat{r}} = [\hat{r}_f] \, (\langle\!\langle f \rangle\!\rangle_{\hat{r}_f} \mid \ * [x] \, \hat{r}_f?x . \langle\!\langle g \rangle\!\rangle_{\hat{r}})$$

$$\langle\!\langle g \ \mathbf{where} \ x :\in f \rangle\!\rangle_{\hat{r}} = [\hat{r}_f, x] \ (\ \langle\!\langle g \rangle\!\rangle_{\hat{r}} \mid [k] \, (\ \langle\!\langle f \rangle\!\rangle_{\hat{r}_f} \mid \hat{r}_f?x . \mathbf{kill}(k) \ ) \ )$$

Endpoint $\hat{r}$ returns the result of expressions evaluation

Orc: auxiliary function for the encoding

$$\langle\!\langle \mathbf{0} \rangle\!\rangle_{\hat{r}} = \mathbf{0} \qquad \langle\!\langle f \mid g \rangle\!\rangle_{\hat{r}} = \langle\!\langle f \rangle\!\rangle_{\hat{r}} \mid \langle\!\langle g \rangle\!\rangle_{\hat{r}}$$

$$\langle\!\langle S(w) \rangle\!\rangle_{\hat{r}} = \hat{S}!\langle w, \hat{r} \rangle \qquad \langle\!\langle E(w) \rangle\!\rangle_{\hat{r}} = [\hat{r}'] \, (\hat{E}!\langle \hat{r}, \hat{r}' \rangle \mid [z] \, \hat{r}'?\langle z \rangle . z!\langle w \rangle)$$

$$\langle\!\langle f > x > g \rangle\!\rangle_{\hat{r}} = [\hat{r}_f] \, (\langle\!\langle f \rangle\!\rangle_{\hat{r}_f} \mid \ * [x] \, \hat{r}_f?x . \langle\!\langle g \rangle\!\rangle_{\hat{r}})$$

$$\langle\!\langle g \textbf{ where } x :\in f \rangle\!\rangle_{\hat{r}} = [\hat{r}_f, x] \ (\langle\!\langle g \rangle\!\rangle_{\hat{r}} \mid [k] \, (\langle\!\langle f \rangle\!\rangle_{\hat{r}_f} \mid \hat{r}_f?x . \textbf{kill}(k) \,) \,)$$

Endpoint $\hat{r}$ returns the result of expressions evaluation

Orc: auxiliary function for the encoding

$$\langle\!\langle \mathbf{0} \rangle\!\rangle_{\hat{r}} = \mathbf{0} \qquad \langle\!\langle f \mid g \rangle\!\rangle_{\hat{r}} = \langle\!\langle f \rangle\!\rangle_{\hat{r}} \mid \langle\!\langle g \rangle\!\rangle_{\hat{r}}$$

$$\langle\!\langle S(w) \rangle\!\rangle_{\hat{r}} = \hat{S}!\langle w, \hat{r} \rangle \qquad \langle\!\langle E(w) \rangle\!\rangle_{\hat{r}} = [\hat{r}'] \, (\hat{E}!\langle \hat{r}, \hat{r}' \rangle \mid [z] \, \hat{r}'?\langle z \rangle . z!\langle w \rangle)$$

$$\langle\!\langle f > x > g \rangle\!\rangle_{\hat{r}} = [\hat{r}_f] \, (\langle\!\langle f \rangle\!\rangle_{\hat{r}_f} \mid \ * [x] \, \hat{r}_f?x . \langle\!\langle g \rangle\!\rangle_{\hat{r}})$$

$$\langle\!\langle g \ \mathbf{where} \ x :\in f \rangle\!\rangle_{\hat{r}} = [\hat{r}_f, x] \ (\ \langle\!\langle g \rangle\!\rangle_{\hat{r}} \mid [k] \, (\ \langle\!\langle f \rangle\!\rangle_{\hat{r}_f} \mid \hat{r}_f?x . \mathbf{kill}(k) \ ) \ )$$

Endpoint $\hat{r}$ returns the result of expressions evaluation

SCC syntax

$$
\begin{array}{lcll}
P, Q & ::= & \mathbf{0} & (\textit{Nil}) \\
 & | & a.P & (\textit{Concretion}) \\
 & | & (x)P & (\textit{Abstraction}) \\
 & | & \text{return } a.P & (\textit{Return Value}) \\
 & | & a \Rightarrow (x)P : (y)T & (\textit{Service Definition}) \\
 & | & a\{(x)P\} \Leftarrow_{\ell} Q & (\textit{Service Invocation}) \\
 & | & a \triangleright_{\ell} P & (\textit{Session}) \\
 & | & P \mid Q & (\textit{Parallel Composition}) \\
 & | & (\nu a)P & (\textit{New Name})
\end{array}
$$

SCC: auxiliary encoding & function

Synchronous COWS

$$\begin{aligned} \langle\!\langle u \cdot u'!\langle e_1, \ldots e_n\rangle.s\rangle\!\rangle &= [\hat{r}]\,(u \cdot u'!\langle e_1, \ldots e_n, \hat{r}\rangle \mid \hat{r}?\langle\rangle.\langle\!\langle s\rangle\!\rangle) \\ \langle\!\langle p \cdot o?\langle w_1, \ldots, w_n\rangle.s\rangle\!\rangle &= [x]\,(p \cdot o?\langle w_1, \ldots, w_n, x\rangle.(\hat{x}!\langle\rangle \mid \langle\!\langle s\rangle\!\rangle)\,) \end{aligned}$$

A partial function from SCC termination names to COWS killer labels

$$K : \{\text{SCC } \textit{names}\} \rightarrow \{\text{COWS } \textit{killer labels}\}$$

SCC: auxiliary encoding & function

Synchronous COWS

$$\begin{array}{rcl} \langle\!\langle u \cdot u'!\langle e_1, \ldots e_n\rangle.s\rangle\!\rangle & = & [\hat{r}]\,(u \cdot u'!\langle e_1, \ldots e_n, \hat{r}\rangle \mid \hat{r}?\langle\rangle.\langle\!\langle s\rangle\!\rangle) \\ \langle\!\langle p \cdot o?\langle w_1, \ldots, w_n\rangle.s\rangle\!\rangle & = & [x]\,(p \cdot o?\langle w_1, \ldots, w_n, x\rangle.(\hat{x}!\langle\rangle \mid \langle\!\langle s\rangle\!\rangle)\,) \end{array}$$

A partial function from SCC termination names to COWS killer labels

$$K : \{\text{SCC } \textit{names}\} \rightarrow \{\text{COWS } \textit{killer labels}\}$$

SCC encoding

Concretion

$$[\![a.P]\!]^K_{\hat{in},\hat{out},\hat{ret}} = \hat{out}!\langle a\rangle.[\![P]\!]^K_{\hat{in},\hat{out},\hat{ret}}$$

Abstraction

$$[\![(x)P]\!]^K_{\hat{in},\hat{out},\hat{ret}} = [x]\,\hat{in}?\langle x\rangle.[\![P]\!]^K_{\hat{in},\hat{out},\hat{ret}}$$

Return value

$$[\![\text{return } a.P]\!]^K_{\hat{in},\hat{out},\hat{ret}} = \hat{ret}!\langle a\rangle.[\![P]\!]^K_{\hat{in},\hat{out},\hat{ret}}$$

Session

$$[\![a \triangleright_\ell P]\!]^K_{\hat{in},\hat{out},\hat{ret}} = [k]\,[\![P]\!]^{K,\{\ell\mapsto k\}}_{\hat{a}_1,\hat{a}_2,\hat{out}} \qquad [\![\tilde{a} \triangleright_\ell P]\!]^K_{\hat{in},\hat{out},\hat{ret}} = [k]\,[\![P]\!]^{K,\{\ell\mapsto k\}}_{\hat{a}_2,\hat{a}_1,\hat{out}}$$

SCC encoding

Concretion

$$[\![a.P]\!]^K_{\hat{in},\hat{out},\hat{ret}} = \hat{out}!\langle a\rangle.[\![P]\!]^K_{\hat{in},\hat{out},\hat{ret}}$$

Abstraction

$$[\![(x)P]\!]^K_{\hat{in},\hat{out},\hat{ret}} = [x]\,\hat{in}?\langle x\rangle.[\![P]\!]^K_{\hat{in},\hat{out},\hat{ret}}$$

Return value

$$[\![\text{return } a.P]\!]^K_{\hat{in},\hat{out},\hat{ret}} = \hat{ret}!\langle a\rangle.[\![P]\!]^K_{\hat{in},\hat{out},\hat{ret}}$$

Session

$$[\![a \triangleright_\ell P]\!]^K_{\hat{in},\hat{out},\hat{ret}} = [k]\,[\![P]\!]^{K,\{\ell\mapsto k\}}_{\hat{a}_1,\hat{a}_2,\hat{out}} \qquad [\![\tilde{a} \triangleright_\ell P]\!]^K_{\hat{in},\hat{out},\hat{ret}} = [k]\,[\![P]\!]^{K,\{\ell\mapsto k\}}_{\hat{a}_2,\hat{a}_1,\hat{out}}$$

SCC encoding

Concretion

$$[\![a.P]\!]^K_{\hat{in},\hat{out},\hat{ret}} = \hat{out}!\langle a\rangle.[\![P]\!]^K_{\hat{in},\hat{out},\hat{ret}}$$

Abstraction

$$[\![(x)P]\!]^K_{\hat{in},\hat{out},\hat{ret}} = [x]\,\hat{in}?\langle x\rangle.[\![P]\!]^K_{\hat{in},\hat{out},\hat{ret}}$$

Return value

$$[\![\text{return } a.P]\!]^K_{\hat{in},\hat{out},\hat{ret}} = \hat{ret}!\langle a\rangle.[\![P]\!]^K_{\hat{in},\hat{out},\hat{ret}}$$

Session

$$[\![a \triangleright_\ell P]\!]^K_{\hat{in},\hat{out},\hat{ret}} = [k]\,[\![P]\!]^{K,\{\ell\mapsto k\}}_{\hat{a}_1,\hat{a}_2,\hat{out}} \qquad [\![\tilde{a} \triangleright_\ell P]\!]^K_{\hat{in},\hat{out},\hat{ret}} = [k]\,[\![P]\!]^{K,\{\ell\mapsto k\}}_{\hat{a}_2,\hat{a}_1,\hat{out}}$$

SCC encoding

Concretion

$$[\![a.P]\!]^K_{\hat{in},\hat{out},\hat{ret}} = \hat{out}!\langle a\rangle.[\![P]\!]^K_{\hat{in},\hat{out},\hat{ret}}$$

Abstraction

$$[\![(x)P]\!]^K_{\hat{in},\hat{out},\hat{ret}} = [x]\,\hat{in}?\langle x\rangle.[\![P]\!]^K_{\hat{in},\hat{out},\hat{ret}}$$

Return value

$$[\![\text{return } a.P]\!]^K_{\hat{in},\hat{out},\hat{ret}} = \hat{ret}!\langle a\rangle.[\![P]\!]^K_{\hat{in},\hat{out},\hat{ret}}$$

Session

$$[\![a \triangleright_\ell P]\!]^K_{\hat{in},\hat{out},\hat{ret}} = [k]\,[\![P]\!]^{K,\{\ell\mapsto k\}}_{\hat{a}_1,\hat{a}_2,\hat{out}} \qquad [\![\tilde{a} \triangleright_\ell P]\!]^K_{\hat{in},\hat{out},\hat{ret}} = [k]\,[\![P]\!]^{K,\{\ell\mapsto k\}}_{\hat{a}_2,\hat{a}_1,\hat{out}}$$

SCC encoding

Service invocation (if $a \notin dom(K)$)

$$[\![a\{(x)P\} \Leftarrow_\ell Q]\!]^K_{\hat{in},\hat{out},\hat{ret}} = [\hat{r}]\,([\![Q]\!]^K_{\hat{in},\hat{r},\hat{ret}} \mid *[x]\,\hat{r}?\langle x\rangle.[\ell', \hat{i}, \hat{o}]\ \hat{scc}!\langle a, \hat{i}, \hat{o}, \ell, \ell', x\rangle.[k]\,[\![P[\ell'/\mathrm{close}]]\!]^{K,\{\ell' \mapsto k\}}_{\hat{i},\hat{o},\hat{out}})$$

Service invocation (if $a \in dom(K)$)

$$[\![a\{(x)P\} \Leftarrow_\ell Q]\!]^K_{\hat{in},\hat{out},\hat{ret}} = [\hat{r}]\,([\![Q]\!]^K_{\hat{in},\hat{r},\hat{ret}} \mid *[x]\,\hat{r}?\langle x\rangle.\ \hat{th}!\langle a, x\rangle.\mathbf{kill}(K(a)))$$

Service definition

$$[\![a \Rightarrow (x)P : (z)T]\!]^K_{\hat{in},\hat{out},\hat{ret}} = *[x_1, x_2, y_1, y_2, x]\,\hat{scc}?\langle a, x_1, x_2, y_1, y_2, x\rangle.\ [k]\,([\![P[y_1/\mathrm{close}]]\!]^{K,\{y_1 \mapsto k\}}_{x_2,x_1,\hat{out}} \mid *[z]\,\hat{th}?\langle y_2, z\rangle.[\![T[y_1/\mathrm{close}]]\!]^{K,\{y_1 \mapsto k\}}_{\_,\_,\hat{out}})$$

SCC encoding

Service invocation (if $a \notin dom(K)$)

$$[\![a\{(x)P\} \Leftarrow_\ell Q]\!]^K_{\hat{in},\hat{out},\hat{ret}} = [\hat{r}]\,([\![Q]\!]^K_{\hat{in},\hat{r},\hat{ret}} \mid *[x]\,\hat{r}?\langle x\rangle.[\ell', \hat{i}, \hat{o}]\ \hat{scc}!\langle a, \hat{i}, \hat{o}, \ell, \ell', x\rangle.[k]\,[\![P[\ell'/\mathrm{close}]]\!]^{K,\{\ell' \mapsto k\}}_{\hat{i},\hat{o},\hat{out}})$$

Service invocation (if $a \in dom(K)$)

$$[\![a\{(x)P\} \Leftarrow_\ell Q]\!]^K_{\hat{in},\hat{out},\hat{ret}} = [\hat{r}]\,([\![Q]\!]^K_{\hat{in},\hat{r},\hat{ret}} \mid *[x]\,\hat{r}?\langle x\rangle.\ \hat{th}!\langle a, x\rangle.\mathbf{kill}(K(a)))$$

Service definition

$$[\![a \Rightarrow (x)P : (z)T]\!]^K_{\hat{in},\hat{out},\hat{ret}} = *[x_1, x_2, y_1, y_2, x]\,\hat{scc}?\langle a, x_1, x_2, y_1, y_2, x\rangle.\ [k]\,([\![P[y_1/\mathrm{close}]]\!]^{K,\{y_1 \mapsto k\}}_{x_2,x_1,\hat{out}} \mid *[z]\,\hat{th}?\langle y_2, z\rangle.[\![T[y_1/\mathrm{close}]]\!]^{K,\{y_1 \mapsto k\}}_{\_,\_,\hat{out}})$$

SCC encoding

Service invocation (if $a \notin dom(K)$)

$$[\![a\{(x)P\} \Leftarrow_\ell Q]\!]^K_{\hat{in},\hat{out},\hat{ret}} = [\hat{r}]\,([\![Q]\!]^K_{\hat{in},\hat{r},\hat{ret}} \mid *[x]\,\hat{r}?\langle x\rangle.[\ell', \hat{i}, \hat{o}]\ \hat{scc}!\langle a, \hat{i}, \hat{o}, \ell, \ell', x\rangle.[k]\,[\![P[\ell'/\mathrm{close}]]\!]^{K,\{\ell' \mapsto k\}}_{\hat{i},\hat{o},\hat{out}})$$

Service invocation (if $a \in dom(K)$)

$$[\![a\{(x)P\} \Leftarrow_\ell Q]\!]^K_{\hat{in},\hat{out},\hat{ret}} = [\hat{r}]\,([\![Q]\!]^K_{\hat{in},\hat{r},\hat{ret}} \mid *[x]\,\hat{r}?\langle x\rangle.\ \hat{th}!\langle a, x\rangle.\mathbf{kill}(K(a)))$$

Service definition

$$[\![a \Rightarrow (x)P : (z)T]\!]^K_{\hat{in},\hat{out},\hat{ret}} = *[x_1, x_2, y_1, y_2, x]\,\hat{scc}?\langle a, x_1, x_2, y_1, y_2, x\rangle.\ [k]\,([\![P[y_1/\mathrm{close}]]\!]^{K,\{y_1 \mapsto k\}}_{x_2,x_1,\hat{out}} \mid *[z]\,\hat{th}?\langle y_2, z\rangle.[\![T[y_1/\mathrm{close}]]\!]^{K,\{y_1 \mapsto k\}}_{\_,\_,\hat{out}})$$