COWS: a Calculus for Orchestration of Web Services

Alessandro Lapadula

DSIUF, Università di Firenze

Braga, Portugal - March 28, 2007

. . . with the contribution of Rosario Pugliese and Francesco Tiezzi

Outline

Outline

Outline

Outline

Outline

Service-oriented computing & Web services

- *Service-oriented computing* (SOC)
  - an emerging paradigm for developing loosely coupled, interoperable, evolvable systems and applications
  - exploits the pervasiveness of the Internet and its related technologies
- *Web services* are a successful instantiation of SOC
  - autonomous, stateless, platform-independent and composable software entities
  - can be published, located and invoked through the Web via XML messages

  These very features foster a programming style based on service *composition* and *reusability*
- Many new XML-based languages have been designed, e.g.
  - business coordination languages (WS-BPEL, WSFL, WSCI, WS-CDL and XLANG)
  - contract languages (WSDL and SWS)
  - query languages (XPath and XQuery)

Service-oriented computing & Web services

- *Service-oriented computing* (SOC)
  - an emerging paradigm for developing loosely coupled, interoperable, evolvable systems and applications
  - exploits the pervasiveness of the Internet and its related technologies
- *Web services* are a successful instantiation of SOC
  - autonomous, stateless, platform-independent and composable software entities
  - can be published, located and invoked through the Web via XML messages

  These very features foster a programming style based on service *composition* and *reusability*
- Many new XML-based languages have been designed, e.g.
  - business coordination languages (WS-BPEL, WSFL, WSCI, WS-CDL and XLANG)
  - contract languages (WSDL and SWS)
  - query languages (XPath and XQuery)

Service-oriented computing & Web services

- *Service-oriented computing* (SOC)
  - an emerging paradigm for developing loosely coupled, interoperable, evolvable systems and applications
  - exploits the pervasiveness of the Internet and its related technologies
- *Web services* are a successful instantiation of SOC
  - autonomous, stateless, platform-independent and composable software entities
  - can be published, located and invoked through the Web via XML messages

  These very features foster a programming style based on service *composition* and *reusability*
- Many new XML-based languages have been designed, e.g.
  - business coordination languages (WS-BPEL, WSFL, WSCI, WS-CDL and XLANG)
  - contract languages (WSDL and SWS)
  - query languages (XPath and XQuery)

Services & Business processes

- A *business process* is an *orchestration* of services, namely "a collection of services, invoked in a particular sequence with a particular set of rules, to meet a business requirement [IBM]"
  - It could be considered a service in its own right
  - which leads to the idea that business processes may be composed of services of different granularities
- A same business process can be identified by means of different logic names
  - To play more than one partner role
  - To capture interdependencies with more than one partner role
- Business data are exploited to correlate interactions among instances

Services & Business processes

- A *business process* is an *orchestration* of services, namely "a collection of services, invoked in a particular sequence with a particular set of rules, to meet a business requirement [IBM]"
  - It could be considered a service in its own right
  - which leads to the idea that business processes may be composed of services of different granularities
- A same business process can be identified by means of different logic names
  - To play more than one partner role
  - To capture interdependencies with more than one partner role
- Business data are exploited to correlate interactions among instances

Services & Business processes

- A *business process* is an *orchestration* of services, namely "a collection of services, invoked in a particular sequence with a particular set of rules, to meet a business requirement [IBM]"
  - It could be considered a service in its own right
  - which leads to the idea that business processes may be composed of services of different granularities
- A same business process can be identified by means of different logic names
  - To play more than one partner role
  - To capture interdependencies with more than one partner role
- Business data are exploited to correlate interactions among instances

Correlation in SOC

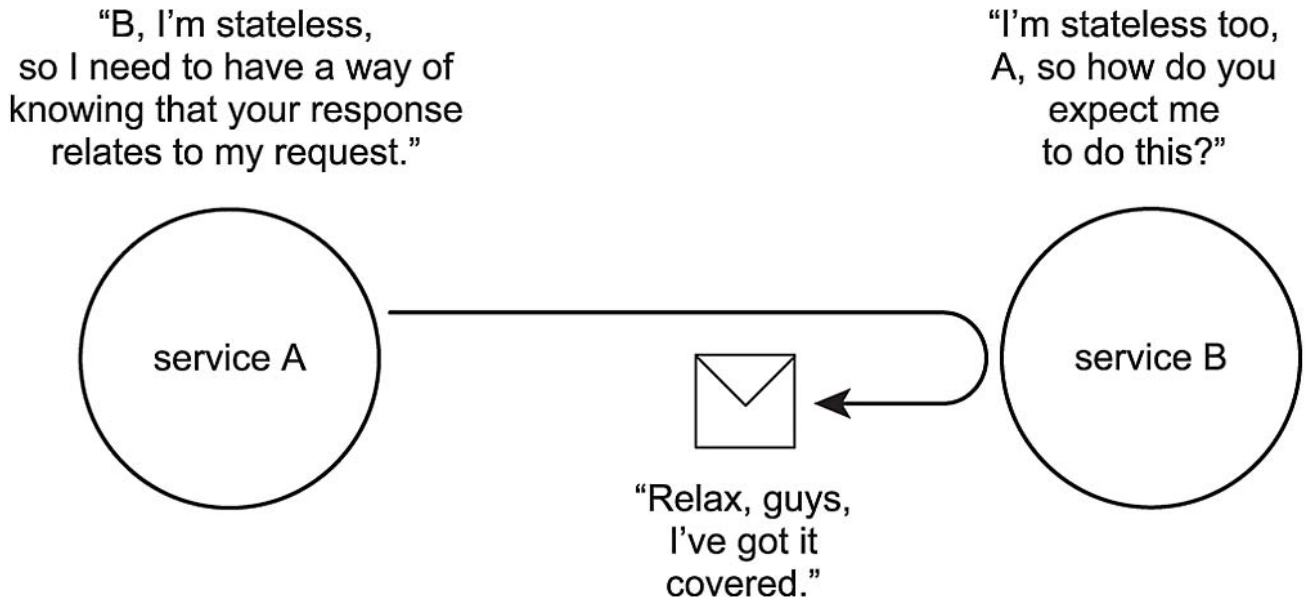
- An essential part of messaging within SOC
  - as it enables the persistence of activity context and state across multiple message exchanges

  while preserving
  - service statelessness and autonomy, and
  - the loosely coupled nature of service-oriented solutions
- Function of correlation: tie messages together without requiring that services somehow manage this association

Correlation in SOC

- An essential part of messaging within SOC
  - as it enables the persistence of activity context and state across multiple message exchanges

  while preserving
  - service statelessness and autonomy, and
  - the loosely coupled nature of service-oriented solutions
- Function of correlation: tie messages together without requiring that services somehow manage this association



Correlation in Web service orchestration

- Added complexity of addressing specific process instances within the correlation data
- Further complicating this scenario is the fact that a single message may participate in multiple contexts, each identified by a separate correlation value
- To support these requirements, the WS-BPEL (*Web Services Business Process Execution Language*) specification defines specific syntax that allows for the creation of extensible *correlation sets*
- These message properties can be dynamically added, deleted, and altered to reflect a wide variety of message exchange scenarios and environments

Correlation in Web service orchestration

- Added complexity of addressing specific process instances within the correlation data
- Further complicating this scenario is the fact that a single message may participate in multiple contexts, each identified by a separate correlation value
- To support these requirements, the WS-BPEL (*Web Services Business Process Execution Language*) specification defines specific syntax that allows for the creation of extensible *correlation sets*
- These message properties can be dynamically added, deleted, and altered to reflect a wide variety of message exchange scenarios and environments

Correlation in Web service orchestration

- Added complexity of addressing specific process instances within the correlation data
- Further complicating this scenario is the fact that a single message may participate in multiple contexts, each identified by a separate correlation value
- To support these requirements, the WS-BPEL (*Web Services Business Process Execution Language*) specification defines specific syntax that allows for the creation of extensible *correlation sets*
- These message properties can be dynamically added, deleted, and altered to reflect a wide variety of message exchange scenarios and environments

Correlation in Web service orchestration

- Added complexity of addressing specific process instances within the correlation data
- Further complicating this scenario is the fact that a single message may participate in multiple contexts, each identified by a separate correlation value
- To support these requirements, the WS-BPEL (*Web Services Business Process Execution Language*) specification defines specific syntax that allows for the creation of extensible *correlation sets*
- These message properties can be dynamically added, deleted, and altered to reflect a wide variety of message exchange scenarios and environments

Goals

- Exploit WS-BPEL, the ‘de facto’ standard language for orchestration of Web services, to drive the design of COWS
  - threads of a same service instance *can share* the store
  - a same process can play *more than one partner role*, and
  - stateful sessions can be programmed 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 an exact counterpart in COWS, rather they are expressed by exploiting COWS more primitive operators
- 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$_i$)

Goals

- Exploit WS-BPEL, the 'de facto' standard language for orchestration of Web services, to drive the design of COWS
  - threads of a same service instance *can share* the store
  - a same process can play *more than one partner role*, and
  - stateful sessions can be programmed 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 an exact counterpart in COWS, rather they are expressed by exploiting COWS more primitive operators
- 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$_i$)

Goals

- Exploit WS-BPEL, the 'de facto' standard language for orchestration of Web services, to drive the design of COWS
  - threads of a same service instance *can share* the store
  - a same process can play *more than one partner role*, and
  - stateful sessions can be programmed 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 an exact counterpart in COWS, rather they are expressed by exploiting COWS more primitive operators
- 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$_i$)

Naming, communication & binding

- Basic elements: *partner* and *operation* names
- *Communication endpoint*: 'partner name' *plus* 'operation name' written $p \bullet o$
- Very flexible naming mechanism that allows
  - a given service to be identified by means of different logical names (i.e. to play more than one partner role)

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

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

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

- Partners and operations can be exchanged in communication
  - but dynamically received names cannot form the communication endpoints used to receive further invocations (in other words, only 'send capability' is transmitted)

Naming, communication & binding

- Basic elements: *partner* and *operation* names
- *Communication endpoint*: 'partner name' *plus* 'operation name' written $p \bullet o$
- Very flexible naming mechanism that allows
  - a given service to be identified by means of different logical names (i.e. to play more than one partner role)

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

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

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

- Partners and operations can be exchanged in communication
  - but dynamically received names cannot form the communication endpoints used to receive further invocations
    (in other words, only 'send capability' is transmitted)

Naming, communication & binding

- Basic elements: *partner* and *operation* names
- *Communication endpoint*: 'partner name' *plus* 'operation name' written $p \bullet o$
- Very flexible naming mechanism that allows
  - a given service to be identified by means of different logical names (i.e. to play more than one partner role)

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

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

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

- Partners and operations can be exchanged in communication
  - but dynamically received names cannot form the communication endpoints used to receive further invocations (in other words, only 'send capability' is transmitted)

Delimitation, forced termination & protection

- The *delimitation* operator $[d]\_$ is the only binder of the calculus (*receive activities* do **not** bind names or variables)
- Delimitation is used to:
  - generate fresh names (like the restriction operator of the π-calculus)
  - regulate the range of application of substitutions generated by communication
  - delimit the field of action of the *kill* activity kill(k), that can be used to force termination of parallel threads
- The *protection* operator $\{|\_|\}$ can be used to protect sensitive code from the effect of a forced termination

Delimitation, forced termination & protection

- The *delimitation* operator $[d]\_$ is the only binder of the calculus
- Delimitation is used to:
  - generate fresh names (like the restriction operator of the π-calculus)
  - regulate the range of application of substitutions generated by communication
  - delimit the field of action of the *kill* activity $\mathbf{kill}(k)$, that can be used to force termination of parallel threads
- The *protection* operator $\{\_\}$ can be used to protect sensitive code from the effect of a forced termination

Delimitation, forced termination & protection

- The *delimitation* operator $[d]\_$ is the only binder of the calculus
- Delimitation is used to:
  - generate fresh names (like the restriction operator of the π-calculus)
  - regulate the range of application of substitutions generated by communication
  - delimit the field of action of the *kill* activity $\mathbf{kill}(k)$, that can be used to force termination of parallel threads
- The *protection* operator $\{\!|\_|\!\}$ can be used to protect sensitive code from the effect of a forced termination

Communication of private names & global scope of variables

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

- Interacting receive and invoke activities must be in the scopes of the delimitations that bind the variables argument of the receive
- Substitution $\{x \mapsto n\}$ is applied to all terms delimited by $[x]$, not only to the continuation s' of the service performing the receive

Delimitation, forced termination & protection

- The *delimitation* operator $[d]\_$ is the only binder of the calculus
- Delimitation is used to:
  - generate fresh names (like the restriction operator of the π-calculus)
  - regulate the range of application of substitutions generated by communication
  - delimit the field of action of the *kill* activity $\mathbf{kill}(k)$, that can be used to force termination of parallel threads
- The *protection* operator $\{\!|\_|\!\}$ can be used to protect sensitive code from the effect of a forced termination

Interplay between communication and kill activity

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

- Kill activities can break communication
- This is the only possible evolution (kill activities are executed *eagerly*)

Delimitation, forced termination & protection

- The *delimitation* operator $[d]\_$ is the only binder of the calculus
- Delimitation is used to:
  - generate fresh names (like the restriction operator of the π-calculus)
  - regulate the range of application of substitutions generated by communication
  - delimit the field of action of the *kill* activity $\mathbf{kill}(k)$, that can be used to force termination of parallel threads
- The *protection* operator $\{|\_|\}$ can be used to protect sensitive code from the effect of a forced termination

Interplay between communication and kill activity

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

- Kill activities can break communication
- This is the only possible evolution (kill activities are executed *eagerly*)

Delimitation, forced termination & protection

- The *delimitation* operator $[d]\_$ is the only binder of the calculus
- Delimitation is used to:
  - generate fresh names (like the restriction operator of the π-calculus)
  - regulate the range of application of substitutions generated by communication
  - delimit the field of action of the *kill* activity $\mathbf{kill}(k)$, that can be used to force termination of parallel threads
- The *protection* operator $\{\!|\_|\!\}$ can be used to protect sensitive code from the effect of a forced termination

Interplay between communication and kill activity

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

- Kill activities can break communication
- This is the only possible evolution (kill activities are executed *eagerly*)
- Communication can be guaranteed by protecting the receive

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

Service instances & correlation

- 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 (*multiple start activities* in WS-BPEL terminology)
- *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
- Flexible mechanism of identifying 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

Service instances & correlation

- 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 (*multiple start activities* in WS-BPEL terminology)
- *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
- Flexible mechanism of identifying 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

Service instances & correlation

- 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 (*multiple start activities* in WS-BPEL terminology)
- *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
- Flexible mechanism of identifying 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

Service instances & correlation

$$* [x, y] \, (\, p \bullet o_{order\_garage} ? \langle x, y \rangle . s_1 \mid p \bullet o_{order\_rental\_car} ? \langle x \rangle . s_2 \,)$$

$$\mid \; p \bullet o_{order\_garage} ! \langle id, gps \rangle \mid p \bullet o_{order\_rental\_car} ! \langle id \rangle$$

$$\longrightarrow$$

$$* [x, y] \, (\, p \bullet o_{order\_garage} ? \langle x, y \rangle . s_1 \mid p \bullet o_{order\_rental\_car} ? \langle x \rangle . s_2 \,)$$

$$\mid \; p \bullet o_{order\_rental\_car} ! \langle id \rangle$$

$$\mid \; (s_1 \{ x \mapsto id, y \mapsto gps \}) \mid (p \bullet o_{order\_rental\_car} ? \langle id \rangle . s_2 \{ x \mapsto id, y \mapsto gps \})$$

The service and the created instance *compete* for processing the request $p \bullet o_{order\_rental\_car} ! \langle id \rangle$

However, by exploiting the correlation data, our (prioritized) semantics for "|" only allows the existing instance to process the request because its input parameter is 'more defined' (and, thus, prevents creation of a new instance)

$$\longrightarrow$$

$$* [x, y] \, (\, p \bullet o_{order\_garage} ? \langle x, y \rangle . s_1 \mid p \bullet o_{order\_rental\_car} ? \langle x \rangle . s_2 \,)$$

$$\mid \; (s_1 \{ x \mapsto id, y \mapsto gps \}) \mid (s_2 \{ x \mapsto id, y \mapsto gps \})$$

Service instances & correlation

$$* [x, y] \, (\, p \bullet o_{order\_garage} ? \langle x, y \rangle . s_1 \mid p \bullet o_{order\_rental\_car} ? \langle x \rangle . s_2 \,)$$

$$\mid \; p \bullet o_{order\_garage} ! \langle id, gps \rangle \mid p \bullet o_{order\_rental\_car} ! \langle id \rangle$$

$$\longrightarrow$$

$$* [x, y] \, (\, p \bullet o_{order\_garage} ? \langle x, y \rangle . s_1 \mid p \bullet o_{order\_rental\_car} ? \langle x \rangle . s_2 \,)$$

$$\mid \; p \bullet o_{order\_rental\_car} ! \langle id \rangle$$

$$\mid \; (s_1 \{ x \mapsto id, y \mapsto gps \}) \mid (p \bullet o_{order\_rental\_car} ? \langle id \rangle . s_2 \{ x \mapsto id, y \mapsto gps \})$$

The service and the created instance *compete* for processing the request $p \bullet o_{order\_rental\_car} ! \langle id \rangle$

However, by exploiting the correlation data, our (prioritized) semantics for "|" only allows the existing instance to process the request because its input parameter is 'more defined' (and, thus, prevents creation of a new instance)

$$\longrightarrow$$

$$* [x, y] \, (\, p \bullet o_{order\_garage} ? \langle x, y \rangle . s_1 \mid p \bullet o_{order\_rental\_car} ? \langle x \rangle . s_2 \,)$$

$$\mid \; (s_1 \{ x \mapsto id, y \mapsto gps \}) \mid (s_2 \{ x \mapsto id, y \mapsto gps \})$$

Service instances & correlation

$$* [x, y] \left(p \bullet o_{order\_garage}?\langle x, y \rangle . s_1 \mid p \bullet o_{order\_rental\_car}?\langle x \rangle . s_2 \right)$$
$$\mid p \bullet o_{order\_garage}!\langle id, gps \rangle \mid p \bullet o_{order\_rental\_car}!\langle id \rangle$$
$$\longrightarrow$$
$$* [x, y] \left(p \bullet o_{order\_garage}?\langle x, y \rangle . s_1 \mid p \bullet o_{order\_rental\_car}?\langle x \rangle . s_2 \right)$$
$$\mid p \bullet o_{order\_rental\_car}!\langle id \rangle$$
$$\mid (s_1\{x \mapsto id, y \mapsto gps\}) \mid (p \bullet o_{order\_rental\_car}?\langle id \rangle . s_2\{x \mapsto id, y \mapsto gps\})$$

The service and the created instance *compete* for processing the request $p \bullet o_{order\_rental\_car}!\langle id \rangle$

However, by exploiting the correlation data, our (prioritized) semantics for "|" only allows the existing instance to process the request because its input parameter is 'more defined' (and, thus, prevents creation of a new instance)

$$\longrightarrow$$
$$* [x, y] \left(p \bullet o_{order\_garage}?\langle x, y \rangle . s_1 \mid p \bullet o_{order\_rental\_car}?\langle x \rangle . s_2 \right)$$
$$\mid (s_1\{x \mapsto id, y \mapsto gps\}) \mid (s_2\{x \mapsto id, y \mapsto gps\})$$

Service instances & correlation

$$*\,[x,y]\,(\,p{\bullet}o_{order\_garage}?\langle x,y\rangle.s_1 \mid p{\bullet}o_{order\_rental\_car}?\langle x\rangle.s_2\,)$$

$$\mid\; p{\bullet}o_{order\_garage}!\langle id,gps\rangle \mid p{\bullet}o_{order\_rental\_car}!\langle id\rangle$$

$$\longrightarrow$$

$$*\,[x,y]\,(\,p{\bullet}o_{order\_garage}?\langle x,y\rangle.s_1 \mid p{\bullet}o_{order\_rental\_car}?\langle x\rangle.s_2\,)$$

$$\mid\; p{\bullet}o_{order\_rental\_car}!\langle id\rangle$$

$$\mid\; (s_1\{x\mapsto id, y\mapsto gps\}) \mid (p{\bullet}o_{order\_rental\_car}?\langle id\rangle.s_2\{x\mapsto id, y\mapsto gps\})$$

The service and the created instance *compete* for processing the request $p{\bullet}o_{order\_rental\_car}!\langle id\rangle$

However, by exploiting the correlation data, our (prioritized) semantics for "|" only allows the existing instance to process the request because its input parameter is 'more defined' (and, thus, prevents creation of a new instance)

$$\longrightarrow$$

$$*\,[x,y]\,(\,p{\bullet}o_{order\_garage}?\langle x,y\rangle.s_1 \mid p{\bullet}o_{order\_rental\_car}?\langle x\rangle.s_2\,)$$

$$\mid\; (s_1\{x\mapsto id, y\mapsto gps\}) \mid (s_2\{x\mapsto id, y\mapsto gps\})$$

Service instances & correlation

$$*\,[x,y]\,(\,p{\bullet}o_{order\_garage}?\langle x,y\rangle.s_1 \mid p{\bullet}o_{order\_rental\_car}?\langle x\rangle.s_2\,)$$
$$\mid\; p{\bullet}o_{order\_garage}!\langle id,gps\rangle \mid p{\bullet}o_{order\_rental\_car}!\langle id\rangle$$
$$\longrightarrow$$
$$*\,[x,y]\,(\,p{\bullet}o_{order\_garage}?\langle x,y\rangle.s_1 \mid p{\bullet}o_{order\_rental\_car}?\langle x\rangle.s_2\,)$$
$$\mid\; p{\bullet}o_{order\_rental\_car}!\langle id\rangle$$
$$\mid\; (s_1\{x\mapsto id, y\mapsto gps\}) \mid (p{\bullet}o_{order\_rental\_car}?\langle id\rangle.s_2\{x\mapsto id, y\mapsto gps\})$$

The service and the created instance *compete* for processing the request $p{\bullet}o_{order\_rental\_car}!\langle id\rangle$

However, by exploiting the correlation data, our (prioritized) semantics for "|" only allows the existing instance to process the request because its input parameter is 'more defined' (and, thus, prevents creation of a new instance)

$$\longrightarrow$$
$$*\,[x,y]\,(\,p{\bullet}o_{order\_garage}?\langle x,y\rangle.s_1 \mid p{\bullet}o_{order\_rental\_car}?\langle x\rangle.s_2\,)$$
$$\mid\; (s_1\{x\mapsto id, y\mapsto gps\}) \mid (s_2\{x\mapsto id, y\mapsto gps\})$$

Syntax of COWS

s ::=	(services)	(notations)
$\mathbf{kill}(k)$	(kill)	k *(killer) labels*
$\mid u \bullet u'!\bar{e}$	(invoke)	e *expressions*
$\mid \sum_{i=0}^{r} p_i \bullet o_i?\bar{w}_i.s_i$	(receive-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)	u: names$\mid$vars
$\mid * s$	(replication)	w: values$\mid$vars
		d: labels$\mid$names$\mid$vars

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

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

Abbrev. $\hat{n}$ stands for a (fresh) endpoint $n_p \cdot n_o$

Syntax of COWS

s ::=		(services)	(notations)
	$\mathbf{kill}(k)$	(kill)	k *(killer) labels*
|	$u \bullet u'!\bar{e}$	(invoke)	e *expressions*
|	$\sum_{i=0}^{r} p_i \bullet o_i?\bar{w}_i.s_i$	(receive-guarded choice)	x *variables*
|	$s \mid s$	(parallel composition)	v *values*
|	$\{\!\|s\|\!\}$	(protection)	n, p, o *names*
|	$[d]\, s$	(delimitation)	u: names|vars
|	$* s$	(replication)	w: values|vars
			d: labels|names|vars

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

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

Abbrev. $\hat{n}$ stands for a (fresh) endpoint $n_p \cdot n_o$

Syntax of COWS

s ::=		(services)	(notations)
	$\mathbf{kill}(k)$	(kill)	k *(killer) labels*
$\mid$	$u \bullet u'!\bar{e}$	(invoke)	e *expressions*
$\mid$	$\sum_{i=0}^{r} p_i \bullet o_i?\bar{w}_i.s_i$	(receive-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)	u: names$\mid$vars
$\mid$	$* s$	(replication)	w: values$\mid$vars
			d: labels$\mid$names$\mid$vars

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

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

Abbrev. $\hat{n}$ stands for a (fresh) endpoint $n_p \cdot n_o$

Syntax of COWS

s ::=		(services)	(notations)
	$\mathbf{kill}(k)$	(kill)	k *(killer) labels*
$\mid$	$u \bullet u'!\bar{e}$	(invoke)	e *expressions*
$\mid$	$\sum_{i=0}^{r} p_i \bullet o_i?\bar{w}_i.s_i$	(receive-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)	u: names$\mid$vars
$\mid$	$* s$	(replication)	w: values$\mid$vars
			d: labels$\mid$names$\mid$vars

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

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

Abbrev. $\hat{n}$ stands for a (fresh) endpoint $n_p \cdot n_o$

Syntax of COWS

s ::=		(services)	(notations)
	$\mathbf{kill}(k)$	(kill)	k *(killer) labels*
$\mid$	$u \bullet u'!\bar{e}$	(invoke)	e *expressions*
$\mid$	$\sum_{i=0}^{r} p_i \bullet o_i?\bar{w}_i.s_i$	(receive-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)	u: names$\mid$vars
$\mid$	$* s$	(replication)	w: values$\mid$vars
			d: labels$\mid$names$\mid$vars

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

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

Abbrev. $\hat{n}$ stands for a (fresh) endpoint $n_p \cdot n_o$

COWS: a Calculus for Orchestration of Web Services

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*)
	$\mid$	$\mathbf{throw}(\phi)$	(*fault generator*)
	$\mid$	$\mathbf{undo}(\imath)$	(*compensate*)
	$\mid$	$\langle\!\mid s : \mathbf{catch}(\phi_1)\{s_1\} : \ldots : \mathbf{catch}(\phi_n)\{s_n\} : s_c \mid\!\rangle_{\imath}$	(*scope*)

- $\mathbf{throw}(\phi)$: 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)
- $\langle\!\mid s : \mathbf{catch}(\phi_1)\{s_1\} : \ldots : \mathbf{catch}(\phi_n)\{s_n\} : s_c \mid\!\rangle_{\imath}$: is uniquely identified by $\imath$ and groups together a service s representing 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*)
	$\mid$	$\mathbf{throw}(\phi)$	(*fault generator*)
	$\mid$	$\mathbf{undo}(\imath)$	(*compensate*)
	$\mid$	$\langle\!\mid s : \mathbf{catch}(\phi_1)\{s_1\} : \ldots : \mathbf{catch}(\phi_n)\{s_n\} : s_c \mid\!\rangle_\imath$	(*scope*)

- $\mathbf{throw}(\phi)$: 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)
- $\langle\!\mid s : \mathbf{catch}(\phi_1)\{s_1\} : \ldots : \mathbf{catch}(\phi_n)\{s_n\} : s_c \mid\!\rangle_\imath$: is uniquely identified by $\imath$ and groups together a service s representing 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*)
	$\mid$	$\mathbf{throw}(\phi)$	(*fault generator*)
	$\mid$	$\mathbf{undo}(\imath)$	(*compensate*)
	$\mid$	$\{\!\vert\, s : \mathbf{catch}(\phi_1)\{s_1\} : \ldots : \mathbf{catch}(\phi_n)\{s_n\} : s_c \,\vert\!\}_\imath$	(*scope*)

- $\mathbf{throw}(\phi)$: 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)
- $\{\!\vert\, s : \mathbf{catch}(\phi_1)\{s_1\} : \ldots : \mathbf{catch}(\phi_n)\{s_n\} : s_c \,\vert\!\}_\imath$: is uniquely identified by $\imath$ and groups together a service s representing 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

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

- **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)
- $\langle\!|\, s : \mathbf{catch}(\phi_1)\{s_1\} : \ldots : \mathbf{catch}(\phi_n)\{s_n\} : s_c \,|\!\rangle_{\imath}$: is uniquely identified by $\imath$ and groups together a service s representing 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

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

- **throw**(ϕ): rises a fault signal ϕ that triggers execution of s if a construct **catch**(ϕ)$\{s\}$ 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 representing the normal behaviour, an optional list of fault handlers, and a compensation handler s_c

Fault and compensation handlers: encoding

$$\langle\!\langle \{\!| s : \mathbf{catch}(\phi_1)\{s_1\} : \ldots : \mathbf{catch}(\phi_n)\{s_n\} : s_c \,|\!\}_\imath \rangle\!\rangle_k \;=$$
$$[\widehat{throw}]\ (\ \langle\!\langle \mathbf{catch}(\phi_1)\{s_1\} \rangle\!\rangle_k \mid \ \ldots\ \mid \langle\!\langle \mathbf{catch}(\phi_n)\{s_n\} \rangle\!\rangle_k \mid$$
$$[k]\,(\ \langle\!\langle s \rangle\!\rangle_k \,;\, (\ x_{done} \bullet o_{done}!\langle\rangle \mid [k]\,\{\!|\widehat{undo}?\langle\imath\rangle.\langle\!\langle s_c \rangle\!\rangle_k |\!\}\ )\ )\ )$$

$$\langle\!\langle \mathbf{catch}(\phi)\{s\} \rangle\!\rangle_k \;=\; \widehat{throw}?\langle\phi\rangle.[k]\,\langle\!\langle s \rangle\!\rangle_k$$

$$\langle\!\langle \mathbf{undo}(\imath) \rangle\!\rangle_k \;=\; \widehat{undo}!\langle\imath\rangle \mid x_{done} \bullet o_{done}!\langle\rangle$$

$$\langle\!\langle \mathbf{throw}(\phi) \rangle\!\rangle_k \;=\; \{\!|\widehat{throw}!\langle\phi\rangle|\!\} \mid \mathbf{kill}(k)$$

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

Fault and compensation handlers: encoding

$$\langle\!\langle \{\!| s : \mathbf{catch}(\phi_1)\{s_1\} : \ldots : \mathbf{catch}(\phi_n)\{s_n\} : s_c \,|\!\}_i \rangle\!\rangle_k \;=$$
$$[\widehat{throw}]\ (\ \langle\!\langle \mathbf{catch}(\phi_1)\{s_1\} \rangle\!\rangle_k \mid \ \ldots\ \mid \langle\!\langle \mathbf{catch}(\phi_n)\{s_n\} \rangle\!\rangle_k \mid$$
$$[k]\,(\ \langle\!\langle s \rangle\!\rangle_k \,;\, (\ x_{done} \bullet o_{done}!\langle\rangle \mid [k]\,\{\!|\widehat{undo}?\langle i\rangle.\langle\!\langle s_c \rangle\!\rangle_k |\!\}\ )\ )\ )$$

$$\langle\!\langle \mathbf{catch}(\phi)\{s\} \rangle\!\rangle_k \;=\; \widehat{throw}?\langle\phi\rangle.[k]\,\langle\!\langle s \rangle\!\rangle_k$$

$$\langle\!\langle \mathbf{undo}(i) \rangle\!\rangle_k \;=\; \widehat{undo}!\langle i\rangle \mid x_{done} \bullet o_{done}!\langle\rangle$$

$$\langle\!\langle \mathbf{throw}(\phi) \rangle\!\rangle_k \;=\; \{\!|\widehat{throw}!\langle\phi\rangle|\!\} \mid \mathbf{kill}(k)$$

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

Fault and compensation handlers: encoding

$$\langle\!\langle \{\!| s : \mathbf{catch}(\phi_1)\{s_1\} : \ldots : \mathbf{catch}(\phi_n)\{s_n\} : s_c \,|\!\}_i \rangle\!\rangle_k \;=$$
$$[\widehat{throw}]\ (\ \langle\!\langle \mathbf{catch}(\phi_1)\{s_1\} \rangle\!\rangle_k \mid \ \ldots\ \mid \langle\!\langle \mathbf{catch}(\phi_n)\{s_n\} \rangle\!\rangle_k \mid$$
$$[k]\,(\ \langle\!\langle s \rangle\!\rangle_k \,;\, (\ x_{done} \bullet o_{done}! \langle\rangle \mid [k]\,\{\!|\widehat{undo}?\langle i \rangle . \langle\!\langle s_c \rangle\!\rangle_k |\!\}\ )\ )\ )$$

$$\langle\!\langle \mathbf{catch}(\phi)\{s\} \rangle\!\rangle_k \;=\; \widehat{throw}?\langle\phi\rangle.[k]\,\langle\!\langle s \rangle\!\rangle_k$$

$$\langle\!\langle \mathbf{undo}(i) \rangle\!\rangle_k \;=\; \widehat{undo}!\langle i \rangle \mid x_{done} \bullet o_{done}! \langle\rangle$$

$$\langle\!\langle \mathbf{throw}(\phi) \rangle\!\rangle_k \;=\; \{\!|\widehat{throw}!\langle\phi\rangle|\!\} \mid \mathbf{kill}(k)$$

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

Fault and compensation handlers: encoding

$$\langle\!\langle \{\!| s : \mathbf{catch}(\phi_1)\{s_1\} : \ldots : \mathbf{catch}(\phi_n)\{s_n\} : s_c \,|\!\}_\iota \rangle\!\rangle_k \;=$$
$$[\widehat{throw}]\ (\ \langle\!\langle \mathbf{catch}(\phi_1)\{s_1\} \rangle\!\rangle_k \mid \ \ldots\ \mid \langle\!\langle \mathbf{catch}(\phi_n)\{s_n\} \rangle\!\rangle_k \mid$$
$$[k]\,(\ \langle\!\langle s \rangle\!\rangle_k \,;\, (\ x_{done} \bullet o_{done}!\langle\rangle \mid [k]\,\{\!|\widehat{undo}?\langle\iota\rangle.\langle\!\langle s_c \rangle\!\rangle_k |\!\}\ )\ )\ )$$

$$\langle\!\langle \mathbf{catch}(\phi)\{s\} \rangle\!\rangle_k \;=\; \widehat{throw}?\langle\phi\rangle.[k]\,\langle\!\langle s \rangle\!\rangle_k$$

$$\langle\!\langle \mathbf{undo}(\iota) \rangle\!\rangle_k \;=\; \widehat{undo}!\langle\iota\rangle \mid x_{done} \bullet o_{done}!\langle\rangle$$

$$\langle\!\langle \mathbf{throw}(\phi) \rangle\!\rangle_k \;=\; \{\!|\widehat{throw}!\langle\phi\rangle|\!\} \mid \mathbf{kill}(k)$$

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

Achievements

- 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 can encode other process and orchestration languages
  - Orc $\Rightarrow$ COWS (without $\{\!|\_|\!\}$)
  - WS-BPEL$_{\ell ight}$ $\Rightarrow$ COWS
- Accommodation of timed activities
- COWS type system permits [FSEN'07] specifying and forcing policies for constraining the partners, and hence the services, that can safely access any given datum
- COWS interpreter

  `http://fmt.isti.cnr.it/cows/V0.1/cows.html`

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

On-going and future work

- Java-based implementation of COWS and its type system
- Development of more powerful type systems based on, e.g., *resource usage types*
  - 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 given protocols
  - need non trivial adaptations to deal with all COWS peculiar features (e.g. killing and protection activities)
- Definition of behavioural equivalences for COWS services
  - 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
- Development of static analysis techniques and logics

Thank you!

Similarities with localised π-calculus (Lπ)

- Variant of π-calculus closest to COWS

Syntax of Lπ processes

$$P ::= \mathbf{0} \mid a(b).P \mid \bar{a}b \mid P \mid P \mid (\nu a)P \mid !a(b).P$$

with the constraint that in processes $a(b).P$ and $!a(b).P$ name b may not occur free in P in input position

-
- All Lπ constructs have a direct counterpart in COWS
- Our encoding enjoys *operational correspondence*

Similarities with localised π-calculus (Lπ)

- Variant of π-calculus closest to COWS

Syntax of Lπ processes

$$P ::= \mathbf{0} \mid a(b).P \mid \bar{a}b \mid P \mid P \mid (\nu a)P \mid !a(b).P$$

with the constraint that in processes $a(b).P$ and $!a(b).P$ name b may not occur free in P in input position

- All Lπ constructs have a direct counterpart in COWS
- Our encoding enjoys *operational correspondence*

Similarities with localised π-calculus (Lπ)

- Variant of π-calculus closest to COWS

Syntax of Lπ processes

$$P ::= \mathbf{0} \mid a(b).P \mid \bar{a}b \mid P \mid P \mid (\nu a)P \mid !a(b).P$$

with the constraint that in processes $a(b).P$ and $!a(b).P$ name b may not occur free in P in input position

- All Lπ constructs have a direct counterpart in COWS
- Our encoding enjoys *operational correspondence*

Similarities with localised π-calculus

- Lπ channel name = COWS communication endpoint consisting of
  - variables, if the channel name is bound by an input prefix
  - names, otherwise
- Encoding of process P = service $\langle\langle P \rangle\rangle_S$ with $S = \emptyset$

  As the encoding proceeds, S records the names that have been freed and were initially bound by an input prefix

$$\langle\langle a \rangle\rangle_{S \cup \{a\}} = x_a \cdot y_a \qquad \langle\langle a \rangle\rangle_S = p_a \cdot o_a \quad \text{if } a \notin S$$

$$\langle\langle \mathbf{0} \rangle\rangle_S = \mathbf{0} \qquad \langle\langle a(b).P \rangle\rangle_S = [\langle\langle b \rangle\rangle_{S'}]\, \langle\langle a \rangle\rangle_S ? \langle\langle b \rangle\rangle_{S'} . \langle\langle P \rangle\rangle_{S'} \quad S' = S \cup \{b\}$$

$$\langle\langle \bar{a}b \rangle\rangle_S = \langle\langle a \rangle\rangle_S ! \langle\langle b \rangle\rangle_S \qquad \langle\langle (\nu a)P \rangle\rangle_S = [\langle\langle a \rangle\rangle_S]\, \langle\langle P \rangle\rangle_S$$

$$\langle\langle P_1 \mid P_2 \rangle\rangle_S = \langle\langle P_1 \rangle\rangle_S \mid \langle\langle P_2 \rangle\rangle_S \qquad \langle\langle !a(b).P \rangle\rangle_S = * \langle\langle a(b).P \rangle\rangle_S$$

Similarities with localised π-calculus

- Lπ channel name = COWS communication endpoint consisting of
  - variables, if the channel name is bound by an input prefix
  - names, otherwise
- Encoding of process P = service $\langle\!\langle P \rangle\!\rangle_S$ with $S = \emptyset$

  As the encoding proceeds, S records the names that have been freed and were initially bound by an input prefix

$$\langle\!\langle a \rangle\!\rangle_{S \cup \{a\}} = x_a \cdot y_a \qquad \langle\!\langle a \rangle\!\rangle_S = p_a \cdot o_a \quad \text{if } a \notin S$$

$$\langle\!\langle \mathbf{0} \rangle\!\rangle_S = \mathbf{0} \qquad \langle\!\langle a(b).P \rangle\!\rangle_S = [\langle\!\langle b \rangle\!\rangle_{S'}]\, \langle\!\langle a \rangle\!\rangle_{S'} ? \langle\!\langle b \rangle\!\rangle_{S'} . \langle\!\langle P \rangle\!\rangle_{S'} \quad S' = S \cup \{b\}$$

$$\langle\!\langle \bar{a}b \rangle\!\rangle_S = \langle\!\langle a \rangle\!\rangle_S ! \langle\!\langle b \rangle\!\rangle_S \qquad \langle\!\langle (\nu a)P \rangle\!\rangle_S = [\langle\!\langle a \rangle\!\rangle_S]\, \langle\!\langle P \rangle\!\rangle_S$$

$$\langle\!\langle P_1 \mid P_2 \rangle\!\rangle_S = \langle\!\langle P_1 \rangle\!\rangle_S \mid \langle\!\langle P_2 \rangle\!\rangle_S \qquad \langle\!\langle !a(b).P \rangle\!\rangle_S = * \langle\!\langle a(b).P \rangle\!\rangle_S$$

Similarities with localised π-calculus

- Lπ channel name = COWS communication endpoint consisting of
  - variables, if the channel name is bound by an input prefix
  - names, otherwise
- Encoding of process P = service $\langle\!\langle P \rangle\!\rangle_S$ with $S = \emptyset$

  As the encoding proceeds, S records the names that have been freed and were initially bound by an input prefix

$$\langle\!\langle a \rangle\!\rangle_{S \cup \{a\}} = x_a \cdot y_a \qquad \langle\!\langle a \rangle\!\rangle_S = p_a \cdot o_a \quad \text{if } a \notin S$$

$$\langle\!\langle \mathbf{0} \rangle\!\rangle_S = \mathbf{0} \qquad \langle\!\langle a(b).P \rangle\!\rangle_S = [\langle\!\langle b \rangle\!\rangle_{S'}]\, \langle\!\langle a \rangle\!\rangle_{S'} ? \langle\!\langle b \rangle\!\rangle_{S'} . \langle\!\langle P \rangle\!\rangle_{S'} \quad S' = S \cup \{b\}$$

$$\langle\!\langle \bar{a}b \rangle\!\rangle_S = \langle\!\langle a \rangle\!\rangle_S ! \langle\!\langle b \rangle\!\rangle_S \qquad \langle\!\langle (\nu a)P \rangle\!\rangle_S = [\langle\!\langle a \rangle\!\rangle_S]\, \langle\!\langle P \rangle\!\rangle_S$$

$$\langle\!\langle P_1 \mid P_2 \rangle\!\rangle_S = \langle\!\langle P_1 \rangle\!\rangle_S \mid \langle\!\langle P_2 \rangle\!\rangle_S \qquad \langle\!\langle !a(b).P \rangle\!\rangle_S = * \langle\!\langle a(b).P \rangle\!\rangle_S$$

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: asynchronous operational semantics

$S(v) \overset{!v'}{\hookrightarrow} \mathbf{0}$ *(SiteCall)*

$$\frac{E(x) \triangleq f}{E(w) \overset{\tau}{\hookrightarrow} f \cdot \{x \mapsto w\}} \textit{(Def)}$$

$$\frac{f \overset{l}{\hookrightarrow} f'}{f \mid g \overset{l}{\hookrightarrow} f' \mid g} \textit{(Sym1)} \qquad \frac{g \overset{l}{\hookrightarrow} g'}{f \mid g \overset{l}{\hookrightarrow} f \mid g'} \textit{(Sym2)}$$

$$\frac{f \overset{\tau}{\hookrightarrow} f'}{f > x > g \overset{\tau}{\hookrightarrow} f' > x > g} \textit{(S1)} \qquad \frac{f \overset{!v}{\hookrightarrow} f'}{f > x > g \overset{\tau}{\hookrightarrow} (f' > x > g) \mid g \cdot \{x \mapsto v\}} \textit{(S2)}$$

$$\frac{g \overset{l}{\hookrightarrow} g'}{g \textbf{ where } x :\in f \overset{l}{\hookrightarrow} g' \textbf{ where } x :\in f} \textit{(A1)} \qquad \frac{f \overset{\tau}{\hookrightarrow} f'}{g \textbf{ where } x :\in f \overset{\tau}{\hookrightarrow} g \textbf{ where } x :\in f'} \textit{(A2)}$$

$$\frac{f \overset{!v}{\hookrightarrow} f'}{g \textbf{ where } x :\in f \overset{\tau}{\hookrightarrow} g \cdot \{x \mapsto v\}} \textit{(A3)}$$

Orc: asynchronous operational semantics

$$S(v) \overset{!v'}{\hookrightarrow} \mathbf{0} \quad \textit{(SiteCall)}$$

$$\frac{E(x) \triangleq f}{E(w) \overset{\tau}{\hookrightarrow} f \cdot \{x \mapsto w\}} \quad \textit{(Def)}$$

$$\frac{f \overset{l}{\hookrightarrow} f'}{f \mid g \overset{l}{\hookrightarrow} f' \mid g} \quad \textit{(Sym1)}$$

$$\frac{g \overset{l}{\hookrightarrow} g'}{f \mid g \overset{l}{\hookrightarrow} f \mid g'} \quad \textit{(Sym2)}$$

$$\frac{f \overset{\tau}{\hookrightarrow} f'}{f > x > g \overset{\tau}{\hookrightarrow} f' > x > g} \quad \textit{(S1)}$$

$$\frac{f \overset{!v}{\hookrightarrow} f'}{f > x > g \overset{\tau}{\hookrightarrow} (f' > x > g) \mid g \cdot \{x \mapsto v\}} \quad \textit{(S2)}$$

$$\frac{g \overset{l}{\hookrightarrow} g'}{g \textbf{ where } x :\in f \overset{l}{\hookrightarrow} g' \textbf{ where } x :\in f} \quad \textit{(A1)}$$

$$\frac{f \overset{\tau}{\hookrightarrow} f'}{g \textbf{ where } x :\in f \overset{\tau}{\hookrightarrow} g \textbf{ where } x :\in f'} \quad \textit{(A2)}$$

$$\frac{f \overset{!v}{\hookrightarrow} f'}{g \textbf{ where } x :\in f \overset{\tau}{\hookrightarrow} g \cdot \{x \mapsto v\}} \quad \textit{(A3)}$$

Orc: asynchronous operational semantics

$$S(v) \overset{!v'}{\hookrightarrow} \mathbf{0} \quad \textit{(SiteCall)}$$

$$\frac{E(x) \triangleq f}{E(w) \overset{\tau}{\hookrightarrow} f \cdot \{x \mapsto w\}} \quad \textit{(Def)}$$

$$\frac{f \overset{l}{\hookrightarrow} f'}{f \mid g \overset{l}{\hookrightarrow} f' \mid g} \quad \textit{(Sym1)}$$

$$\frac{g \overset{l}{\hookrightarrow} g'}{f \mid g \overset{l}{\hookrightarrow} f \mid g'} \quad \textit{(Sym2)}$$

$$\frac{f \overset{\tau}{\hookrightarrow} f'}{f > x > g \overset{\tau}{\hookrightarrow} f' > x > g} \quad \textit{(S1)}$$

$$\frac{f \overset{!v}{\hookrightarrow} f'}{f > x > g \overset{\tau}{\hookrightarrow} (f' > x > g) \mid g \cdot \{x \mapsto v\}} \quad \textit{(S2)}$$

$$\frac{g \overset{l}{\hookrightarrow} g'}{g \textbf{ where } x :\in f \overset{l}{\hookrightarrow} g' \textbf{ where } x :\in f} \quad \textit{(A1)}$$

$$\frac{f \overset{\tau}{\hookrightarrow} f'}{g \textbf{ where } x :\in f \overset{\tau}{\hookrightarrow} g \textbf{ where } x :\in f'} \quad \textit{(A2)}$$

$$\frac{f \overset{!v}{\hookrightarrow} f'}{g \textbf{ where } x :\in f \overset{\tau}{\hookrightarrow} g \cdot \{x \mapsto v\}} \quad \textit{(A3)}$$

Orc: asynchronous operational semantics

$$S(v) \overset{!v'}{\hookrightarrow} \mathbf{0} \quad \textit{(SiteCall)}$$

$$\frac{E(x) \triangleq f}{E(w) \overset{\tau}{\hookrightarrow} f \cdot \{x \mapsto w\}} \quad \textit{(Def)}$$

$$\frac{f \overset{l}{\hookrightarrow} f'}{f \mid g \overset{l}{\hookrightarrow} f' \mid g} \quad \textit{(Sym1)}$$

$$\frac{g \overset{l}{\hookrightarrow} g'}{f \mid g \overset{l}{\hookrightarrow} f \mid g'} \quad \textit{(Sym2)}$$

$$\frac{f \overset{\tau}{\hookrightarrow} f'}{f > x > g \overset{\tau}{\hookrightarrow} f' > x > g} \quad \textit{(S1)}$$

$$\frac{f \overset{!v}{\hookrightarrow} f'}{f > x > g \overset{\tau}{\hookrightarrow} (f' > x > g) \mid g \cdot \{x \mapsto v\}} \quad \textit{(S2)}$$

$$\frac{g \overset{l}{\hookrightarrow} g'}{g \textbf{ where } x :\in f \overset{l}{\hookrightarrow} g' \textbf{ where } x :\in f} \quad \textit{(A1)}$$

$$\frac{f \overset{\tau}{\hookrightarrow} f'}{g \textbf{ where } x :\in f \overset{\tau}{\hookrightarrow} g \textbf{ where } x :\in f'} \quad \textit{(A2)}$$

$$\frac{f \overset{!v}{\hookrightarrow} f'}{g \textbf{ where } x :\in f \overset{\tau}{\hookrightarrow} g \cdot \{x \mapsto v\}} \quad \textit{(A3)}$$

Orc: asynchronous operational semantics

$$S(v) \overset{!v'}{\hookrightarrow} \mathbf{0} \quad \textit{(SiteCall)}$$

$$\frac{E(x) \triangleq f}{E(w) \overset{\tau}{\hookrightarrow} f \cdot \{x \mapsto w\}} \quad \textit{(Def)}$$

$$\frac{f \overset{l}{\hookrightarrow} f'}{f \mid g \overset{l}{\hookrightarrow} f' \mid g} \quad \textit{(Sym1)}$$

$$\frac{g \overset{l}{\hookrightarrow} g'}{f \mid g \overset{l}{\hookrightarrow} f \mid g'} \quad \textit{(Sym2)}$$

$$\frac{f \overset{\tau}{\hookrightarrow} f'}{f > x > g \overset{\tau}{\hookrightarrow} f' > x > g} \quad \textit{(S1)}$$

$$\frac{f \overset{!v}{\hookrightarrow} f'}{f > x > g \overset{\tau}{\hookrightarrow} (f' > x > g) \mid g \cdot \{x \mapsto v\}} \quad \textit{(S2)}$$

$$\frac{g \overset{l}{\hookrightarrow} g'}{g \textbf{ where } x :\in f \overset{l}{\hookrightarrow} g' \textbf{ where } x :\in f} \quad \textit{(A1)}$$

$$\frac{f \overset{\tau}{\hookrightarrow} f'}{g \textbf{ where } x :\in f \overset{\tau}{\hookrightarrow} g \textbf{ where } x :\in f'} \quad \textit{(A2)}$$

$$\frac{f \overset{!v}{\hookrightarrow} f'}{g \textbf{ where } x :\in f \overset{\tau}{\hookrightarrow} g \cdot \{x \mapsto v\}} \quad \textit{(A3)}$$

Orc: asynchronous operational semantics

$$S(v) \overset{!v'}{\hookrightarrow} \mathbf{0} \quad \textit{(SiteCall)}$$

$$\frac{E(x) \triangleq f}{E(w) \overset{\tau}{\hookrightarrow} f \cdot \{x \mapsto w\}} \quad \textit{(Def)}$$

$$\frac{f \overset{l}{\hookrightarrow} f'}{f \mid g \overset{l}{\hookrightarrow} f' \mid g} \quad \textit{(Sym1)}$$

$$\frac{g \overset{l}{\hookrightarrow} g'}{f \mid g \overset{l}{\hookrightarrow} f \mid g'} \quad \textit{(Sym2)}$$

$$\frac{f \overset{\tau}{\hookrightarrow} f'}{f > x > g \overset{\tau}{\hookrightarrow} f' > x > g} \quad \textit{(S1)}$$

$$\frac{f \overset{!v}{\hookrightarrow} f'}{f > x > g \overset{\tau}{\hookrightarrow} (f' > x > g) \mid g \cdot \{x \mapsto v\}} \quad \textit{(S2)}$$

$$\frac{g \overset{l}{\hookrightarrow} g'}{g \textbf{ where } x :\in f \overset{l}{\hookrightarrow} g' \textbf{ where } x :\in f} \quad \textit{(A1)}$$

$$\frac{f \overset{\tau}{\hookrightarrow} f'}{g \textbf{ where } x :\in f \overset{\tau}{\hookrightarrow} g \textbf{ where } x :\in f'} \quad \textit{(A2)}$$

$$\frac{f \overset{!v}{\hookrightarrow} f'}{g \textbf{ where } x :\in f \overset{\tau}{\hookrightarrow} g \cdot \{x \mapsto v\}} \quad \textit{(A3)}$$

Orc: asynchronous operational semantics

$$S(v) \overset{!v'}{\hookrightarrow} \mathbf{0} \quad \textit{(SiteCall)}$$

$$\frac{E(x) \triangleq f}{E(w) \overset{\tau}{\hookrightarrow} f \cdot \{x \mapsto w\}} \quad \textit{(Def)}$$

$$\frac{f \overset{l}{\hookrightarrow} f'}{f \mid g \overset{l}{\hookrightarrow} f' \mid g} \quad \textit{(Sym1)}$$

$$\frac{g \overset{l}{\hookrightarrow} g'}{f \mid g \overset{l}{\hookrightarrow} f \mid g'} \quad \textit{(Sym2)}$$

$$\frac{f \overset{\tau}{\hookrightarrow} f'}{f > x > g \overset{\tau}{\hookrightarrow} f' > x > g} \quad \textit{(S1)}$$

$$\frac{f \overset{!v}{\hookrightarrow} f'}{f > x > g \overset{\tau}{\hookrightarrow} (f' > x > g) \mid g \cdot \{x \mapsto v\}} \quad \textit{(S2)}$$

$$\frac{g \overset{l}{\hookrightarrow} g'}{g \textbf{ where } x :\in f \overset{l}{\hookrightarrow} g' \textbf{ where } x :\in f} \quad \textit{(A1)}$$

$$\frac{f \overset{\tau}{\hookrightarrow} f'}{g \textbf{ where } x :\in f \overset{\tau}{\hookrightarrow} g \textbf{ where } x :\in f'} \quad \textit{(A2)}$$

$$\frac{f \overset{!v}{\hookrightarrow} f'}{g \textbf{ where } x :\in f \overset{\tau}{\hookrightarrow} g \cdot \{x \mapsto v\}} \quad \textit{(A3)}$$

Orc: intended semantics

- Evaluation of expressions may call a number of sites and returns a (possibly empty) stream of values
- A site call can progress only when the actual parameter is a value (rule *(SiteCall)*); it elicits one response
- Expression calls are evaluated using call-by-name (rule *(Def)*)

  namely the actual parameter replaces the formal one and then the corresponding expression is evaluated
- Symmetric parallel composition $f \mid g$ consists of concurrent evaluations of f and g (rules *(Sym1)* & *(Sym2)*)
- Sequential composition $f > x > g$ activates a concurrent copy of g with x replaced by v, for each value v returned by f (rules *(S1)* & *(S2)*)
- Asymmetric parallel composition g **where** $x :\in f$ prunes threads selectively. It starts in parallel both f and the part of g that does not need x (rules *(A2)* and *(A1)*)

  The first value returned by f is assigned to x and the continuation of f and all its descendants are then terminated (rule *(A3)*)

Orc: intended semantics

- Evaluation of expressions may call a number of sites and returns a (possibly empty) stream of values
- A site call can progress only when the actual parameter is a value (rule *(SiteCall)*); it elicits one response
- Expression calls are evaluated using call-by-name (rule *(Def)*)

  namely the actual parameter replaces the formal one and then the corresponding expression is evaluated
- Symmetric parallel composition $f \mid g$ consists of concurrent evaluations of f and g (rules *(Sym1)* & *(Sym2)*)
- Sequential composition $f > x > g$ activates a concurrent copy of g with x replaced by v, for each value v returned by f (rules *(S1)* & *(S2)*)
- Asymmetric parallel composition g **where** $x :\in f$ prunes threads selectively. It starts in parallel both f and the part of g that does not need x (rules *(A2)* and *(A1)*)

  The first value returned by f is assigned to x and the continuation of f and all its descendants are then terminated (rule *(A3)*)

Orc: intended semantics

- Evaluation of expressions may call a number of sites and returns a (possibly empty) stream of values
- A site call can progress only when the actual parameter is a value (rule *(SiteCall)*); it elicits one response
- Expression calls are evaluated using call-by-name (rule *(Def)*)

  namely the actual parameter replaces the formal one and then the corresponding expression is evaluated
- Symmetric parallel composition $f \mid g$ consists of concurrent evaluations of f and g (rules *(Sym1)* & *(Sym2)*)
- Sequential composition $f > x > g$ activates a concurrent copy of g with x replaced by v, for each value v returned by f (rules *(S1)* & *(S2)*)
- Asymmetric parallel composition g **where** $x :\in f$ prunes threads selectively. It starts in parallel both f and the part of g that does not need x (rules *(A2)* and *(A1)*)

  The first value returned by f is assigned to x and the continuation of f and all its descendants are then terminated (rule *(A3)*)

Orc: intended semantics

- Evaluation of expressions may call a number of sites and returns a (possibly empty) stream of values
- A site call can progress only when the actual parameter is a value (rule *(SiteCall)*); it elicits one response
- Expression calls are evaluated using call-by-name (rule *(Def)*)

  namely the actual parameter replaces the formal one and then the corresponding expression is evaluated
- Symmetric parallel composition $f \mid g$ consists of concurrent evaluations of f and g (rules *(Sym1)* & *(Sym2)*)
- Sequential composition $f > x > g$ activates a concurrent copy of g with x replaced by v, for each value v returned by f (rules *(S1)* & *(S2)*)
- Asymmetric parallel composition g **where** $x :\in f$ prunes threads selectively. It starts in parallel both f and the part of g that does not need x (rules *(A2)* and *(A1)*)

  The first value returned by f is assigned to x and the continuation of f and all its descendants are then terminated (rule *(A3)*)

Orc: intended semantics

- Evaluation of expressions may call a number of sites and returns a (possibly empty) stream of values
- A site call can progress only when the actual parameter is a value (rule *(SiteCall)*); it elicits one response
- Expression calls are evaluated using call-by-name (rule *(Def)*)

  namely the actual parameter replaces the formal one and then the corresponding expression is evaluated
- Symmetric parallel composition $f \mid g$ consists of concurrent evaluations of f and g (rules *(Sym1)* & *(Sym2)*)
- Sequential composition $f > x > g$ activates a concurrent copy of g with x replaced by v, for each value v returned by f (rules *(S1)* & *(S2)*)
- Asymmetric parallel composition g **where** $x :\in f$ prunes threads selectively. It starts in parallel both f and the part of g that does not need x (rules *(A2)* and *(A1)*)

  The first value returned by f is assigned to x and the continuation of f and all its descendants are then terminated (rule *(A3)*)

Orc: intended semantics

- Evaluation of expressions may call a number of sites and returns a (possibly empty) stream of values
- A site call can progress only when the actual parameter is a value (rule *(SiteCall)*); it elicits one response
- Expression calls are evaluated using call-by-name (rule *(Def)*)

  namely the actual parameter replaces the formal one and then the corresponding expression is evaluated
- Symmetric parallel composition $f \mid g$ consists of concurrent evaluations of f and g (rules *(Sym1)* & *(Sym2)*)
- Sequential composition $f > x > g$ activates a concurrent copy of g with x replaced by v, for each value v returned by f (rules *(S1)* & *(S2)*)
- Asymmetric parallel composition g **where** $x :\in f$ prunes threads selectively. It starts in parallel both f and the part of g that does not need x (rules *(A2)* and *(A1)*)

  The first value returned by f is assigned to x and the continuation of f and all its descendants are then terminated (rule *(A3)*)

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

$$\langle\!\langle \mathbf{0} \rangle\!\rangle_{\hat{r}} = \mathbf{0}$$

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

$$\langle\!\langle S(w) \rangle\!\rangle_{\hat{r}} = \hat{S}!\langle w, \hat{r} \rangle$$

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

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

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

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

Orc: auxiliary function for the encoding

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

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

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

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

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

Orc: auxiliary function for the encoding

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

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

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

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

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

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

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

$[\![f]\!]_{\hat{f}}$ is the parallel composition of $\langle\!\langle f\rangle\!\rangle_{\hat{f}}$, 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: operational correspondence

There is a formal correspondence, based on the operational semantics, between Orc expressions and the COWS services resulting from their encoding

Operational correspondence

Given an Orc expression f and a communication endpoint $\hat{r}$

$$f \overset{l}{\hookrightarrow} f' \text{ implies } [\![f]\!]_{\hat{r}} \equiv \langle\!\langle f \rangle\!\rangle_{\hat{r}} \mid s \overset{\alpha}{\Longrightarrow} \langle\!\langle f' \rangle\!\rangle_{\hat{r}} \mid s$$

where $\alpha = \hat{r} \lhd v$ if $l = !v$, and $\alpha = p \cdot o \lfloor \emptyset \rfloor \bar{w}\, \bar{v}$ if $l = \tau$

Proof: by induction on the length of the inference of $f \overset{l}{\hookrightarrow} f'$

$s \overset{\alpha}{\Longrightarrow} s'$ denotes that there exist two services, s_1 and s_2, such that s_1 is a reduct of s, $s_1 \overset{\alpha}{\longrightarrow} s_2$ and s' is a reduct of s_2

Orc: operational correspondence

There is a formal correspondence, based on the operational semantics, between Orc expressions and the COWS services resulting from their encoding

Operational correspondence

Given an Orc expression f and a communication endpoint $\hat{r}$

$$f \overset{l}{\hookrightarrow} f' \text{ implies } [\![f]\!]_{\hat{r}} \equiv \langle\!\langle f \rangle\!\rangle_{\hat{r}} \mid s \overset{\alpha}{\Longrightarrow} \langle\!\langle f' \rangle\!\rangle_{\hat{r}} \mid s$$

where $\alpha = \hat{r} \lhd v$ if $l = !v$, and $\alpha = p \cdot o \lfloor \emptyset \rfloor \bar{w}\, \bar{v}$ if $l = \tau$

Proof: by induction on the length of the inference of $f \overset{l}{\hookrightarrow} f'$

$s \overset{\alpha}{\Longrightarrow} s'$ denotes that there exist two services, s_1 and s_2, such that s_1 is a reduct of s, $s_1 \overset{\alpha}{\longrightarrow} s_2$ and s' is a reduct of s_2

Flow graphs

- Provide a direct and intuitive way to structure workflow processes, where activities executed in parallel can be synchronized by settling dependencies (*flow links*) among them
- Initially, all involved links are inactive and only those activities with no synchronization dependencies can execute
- Once all incoming links of an activity are active (i.e., they have been assigned either a positive or negative state), a guard (*join condition*) is evaluated
- When an activity in the flow graph cannot execute because its join condition fails, a *join failure* fault is emitted, unless attribute *suppress join failure* is set to 'yes' (*Dead-Path Elimination* effect)
- When an activity terminates, the status of each outgoing link is determined through evaluation of a *transition condition*

Flow graphs

- Provide a direct and intuitive way to structure workflow processes, where activities executed in parallel can be synchronized by settling dependencies (*flow links*) among them
- Initially, all involved links are inactive and only those activities with no synchronization dependencies can execute
- Once all incoming links of an activity are active (i.e., they have been assigned either a positive or negative state), a guard (*join condition*) is evaluated
- When an activity in the flow graph cannot execute because its join condition fails, a *join failure* fault is emitted, unless attribute *suppress join failure* is set to 'yes' (*Dead-Path Elimination* effect)
- When an activity terminates, the status of each outgoing link is determined through evaluation of a *transition condition*

Flow graphs

- Provide a direct and intuitive way to structure workflow processes, where activities executed in parallel can be synchronized by settling dependencies (*flow links*) among them
- Initially, all involved links are inactive and only those activities with no synchronization dependencies can execute
- Once all incoming links of an activity are active (i.e., they have been assigned either a positive or negative state), a guard (*join condition*) is evaluated
- When an activity in the flow graph cannot execute because its join condition fails, a *join failure* fault is emitted, unless attribute *suppress join failure* is set to 'yes' (*Dead-Path Elimination* effect)
- When an activity terminates, the status of each outgoing link is determined through evaluation of a *transition condition*

Flow graphs

- Provide a direct and intuitive way to structure workflow processes, where activities executed in parallel can be synchronized by settling dependencies (*flow links*) among them
- Initially, all involved links are inactive and only those activities with no synchronization dependencies can execute
- Once all incoming links of an activity are active (i.e., they have been assigned either a positive or negative state), a guard (*join condition*) is evaluated
- When an activity in the flow graph cannot execute because its join condition fails, a *join failure* fault is emitted, unless attribute *suppress join failure* is set to 'yes' (*Dead-Path Elimination* effect)
- When an activity terminates, the status of each outgoing link is determined through evaluation of a *transition condition*

Flow graphs

- Provide a direct and intuitive way to structure workflow processes, where activities executed in parallel can be synchronized by settling dependencies (*flow links*) among them
- Initially, all involved links are inactive and only those activities with no synchronization dependencies can execute
- Once all incoming links of an activity are active (i.e., they have been assigned either a positive or negative state), a guard (*join condition*) is evaluated
- When an activity in the flow graph cannot execute because its join condition fails, a *join failure* fault is emitted, unless attribute *suppress join failure* is set to 'yes' (*Dead-Path Elimination* effect)
- When an activity terminates, the status of each outgoing link is determined through evaluation of a *transition condition*

Flow graphs: syntax

$$
\begin{array}{rcll}
s & ::= & \ldots \mid [\overline{fl}]\, ls \mid \sum_{i \in I} p_i \bullet o_i ? \bar{w}_i . s_i & (\textit{services}) \\
ls & ::= & (jc) \overset{sjf}{\Rightarrow} s \Rightarrow (\overline{fl}, \bar{e}) \mid s \Rightarrow (\overline{fl}, \bar{e}) \mid ls \mid ls & (\textit{linked services}) \\
jc & ::= & \mathbf{true} \mid \mathbf{false} \mid fl \mid \neg jc \mid jc \vee jc \mid jc \wedge jc & (\textit{join conditions}) \\
sjf & ::= & yes \mid no & (\textit{supp. join failure})
\end{array}
$$

- *Flow graph* $[\overline{fl}]\, ls$

  a delimited *linked service* ls whose internal activities can synchronize by means of the flow links in $\overline{fl}$
- *Flow links* $\overline{fl}$ are (boolean) variables
- $(jc) \overset{sjf}{\Rightarrow}$ denotes a join condition jc with attribute sjf
- $\Rightarrow (\overline{fl}, \bar{e})$ denotes services' outgoing links,

  i.e. pairs (fl_i, e_i) with fl_i flow link and e_i transition (boolean) condition

Flow graphs: syntax

$$
\begin{array}{rcll}
s & ::= & \ldots \mid [\overline{fl}]\, ls \mid \sum_{i \in I} p_i \bullet o_i ? \bar{w}_i . s_i & (\textit{services}) \\
ls & ::= & (jc) \overset{sjf}{\Rightarrow} s \Rightarrow (\overline{fl}, \bar{e}) \mid s \Rightarrow (\overline{fl}, \bar{e}) \mid ls \mid ls & (\textit{linked services}) \\
jc & ::= & \textbf{true} \mid \textbf{false} \mid fl \mid \neg jc \mid jc \vee jc \mid jc \wedge jc & (\textit{join conditions}) \\
sjf & ::= & yes \mid no & (\textit{supp. join failure})
\end{array}
$$

- *Flow graph* $[\overline{fl}]\, ls$

  a delimited *linked service* ls whose internal activities can synchronize by means of the flow links in $\overline{fl}$
- *Flow links* $\overline{fl}$ are (boolean) variables
- $(jc) \overset{sjf}{\Rightarrow}$ denotes a join condition jc with attribute sjf
- $\Rightarrow (\overline{fl}, \bar{e})$ denotes services' outgoing links,

  i.e. pairs (fl_i, e_i) with fl_i flow link and e_i transition (boolean) condition

Flow graphs: syntax

$$
\begin{array}{rcll}
s & ::= & \ldots \mid [\overline{fl}]\, ls \mid \sum_{i \in I} p_i \bullet o_i ? \bar{w}_i . s_i & (\textit{services}) \\
ls & ::= & (jc) \overset{sjf}{\Rightarrow} s \Rightarrow (\overline{fl}, \bar{e}) \mid s \Rightarrow (\overline{fl}, \bar{e}) \mid ls \mid ls & (\textit{linked services}) \\
jc & ::= & \textbf{true} \mid \textbf{false} \mid fl \mid \neg jc \mid jc \vee jc \mid jc \wedge jc & (\textit{join conditions}) \\
sjf & ::= & yes \mid no & (\textit{supp. join failure})
\end{array}
$$

- *Flow graph* $[\overline{fl}]\, ls$

  a delimited *linked service* ls whose internal activities can synchronize by means of the flow links in $\overline{fl}$
- *Flow links* $\overline{fl}$ are (boolean) variables
- $(jc) \overset{sjf}{\Rightarrow}$ denotes a join condition jc with attribute sjf
- $\Rightarrow (\overline{fl}, \bar{e})$ denotes services' outgoing links,

  i.e. pairs (fl_i, e_i) with fl_i flow link and e_i transition (boolean) condition

Flow graphs: encoding

$$\langle\!\langle [\overline{fl}]\ ls \rangle\!\rangle = [\overline{fl}]\ \langle\!\langle ls \rangle\!\rangle$$

$$\langle\!\langle ls_1 \mid ls_2 \rangle\!\rangle = \langle\!\langle ls_1 \rangle\!\rangle \mid \langle\!\langle ls_2 \rangle\!\rangle$$

$$\langle\!\langle s \Rightarrow (\overline{fl}, \bar{e}) \rangle\!\rangle = \langle\!\langle s \rangle\!\rangle ; [\overline{fl} = \bar{e}]$$

$$\langle\!\langle (jc) \overset{yes}{\Rightarrow} s \Rightarrow (\overline{fl}, \bar{e}) \rangle\!\rangle = \textbf{if } (jc) \textbf{ then } \{\langle\!\langle s \rangle\!\rangle ; [\overline{fl} = \bar{e}]\} \textbf{ else } \{[outLinkOf(s) = \overline{\textbf{false}}]\}$$

$$\langle\!\langle (jc) \overset{no}{\Rightarrow} s \Rightarrow (\overline{fl}, \bar{e}) \rangle\!\rangle = \textbf{if } (jc) \textbf{ then } \{\langle\!\langle s \rangle\!\rangle ; [\overline{fl} = \bar{e}]\} \textbf{ else } \{\textbf{throw}(\phi_{join\_f})\}$$

$$\langle\!\langle \textstyle\sum_{i \in \{1..n\}} p_i \bullet o_i ? \bar{w}_i . s_i \rangle\!\rangle = p_1 \bullet o_1 ? \bar{w}_1 . [\bigcup_{j \in \{2..n\}} outLinkOf(s_j) = \overline{\textbf{false}}] . \langle\!\langle s_1 \rangle\!\rangle + \ldots + p_n \bullet o_n ? \bar{w}_n . [\bigcup_{j \in \{1..n-1\}} outLinkOf(s_j) = \overline{\textbf{false}}] . \langle\!\langle s_n \rangle\!\rangle$$

- Join conditions are boolean conditions within conditional constructs, transition conditions are assigned to flow links by means of $[\overline{fl} = \bar{e}]$
- When a branch of a choice among (linked) services is selected, the links outgoing from the activities of the discarded branches are set to **false**
- If *suppress join failure* is set to 'no', a join condition failure produces a fault signal that can be caught by a proper fault handler

Flow graphs: encoding

$$\langle\!\langle [\overline{fl}]\ ls \rangle\!\rangle = [\overline{fl}]\ \langle\!\langle ls \rangle\!\rangle$$

$$\langle\!\langle ls_1 \mid ls_2 \rangle\!\rangle = \langle\!\langle ls_1 \rangle\!\rangle \mid \langle\!\langle ls_2 \rangle\!\rangle$$

$$\langle\!\langle s \Rightarrow (\overline{fl}, \bar{e}) \rangle\!\rangle = \langle\!\langle s \rangle\!\rangle ; [\overline{fl} = \bar{e}]$$

$$\langle\!\langle (jc) \overset{yes}{\Rightarrow} s \Rightarrow (\overline{fl}, \bar{e}) \rangle\!\rangle = \textbf{if } (jc) \textbf{ then } \{\langle\!\langle s \rangle\!\rangle ; [\overline{fl} = \bar{e}]\} \textbf{ else } \{[outLinkOf(s) = \overline{\textbf{false}}]\}$$

$$\langle\!\langle (jc) \overset{no}{\Rightarrow} s \Rightarrow (\overline{fl}, \bar{e}) \rangle\!\rangle = \textbf{if } (jc) \textbf{ then } \{\langle\!\langle s \rangle\!\rangle ; [\overline{fl} = \bar{e}]\} \textbf{ else } \{\textbf{throw}(\phi_{join\_f})\}$$

$$\langle\!\langle \textstyle\sum_{i \in \{1..n\}} p_i \bullet o_i ? \bar{w}_i . s_i \rangle\!\rangle = p_1 \bullet o_1 ? \bar{w}_1 . [\bigcup_{j \in \{2..n\}} outLinkOf(s_j) = \overline{\textbf{false}}] . \langle\!\langle s_1 \rangle\!\rangle + \ldots + p_n \bullet o_n ? \bar{w}_n . [\bigcup_{j \in \{1..n-1\}} outLinkOf(s_j) = \overline{\textbf{false}}] . \langle\!\langle s_n \rangle\!\rangle$$

- Join conditions are boolean conditions within conditional constructs, transition conditions are assigned to flow links by means of $[\overline{fl} = \bar{e}]$
- When a branch of a choice among (linked) services is selected, the links outgoing from the activities of the discarded branches are set to **false**
- If *suppress join failure* is set to 'no', a join condition failure produces a fault signal that can be caught by a proper fault handler

Flow graphs: encoding

$$\langle\!\langle [\overline{fl}]\ ls \rangle\!\rangle = [\overline{fl}]\ \langle\!\langle ls \rangle\!\rangle$$

$$\langle\!\langle ls_1 \mid ls_2 \rangle\!\rangle = \langle\!\langle ls_1 \rangle\!\rangle \mid \langle\!\langle ls_2 \rangle\!\rangle$$

$$\langle\!\langle s \Rightarrow (\overline{fl}, \bar{e}) \rangle\!\rangle = \langle\!\langle s \rangle\!\rangle ; [\overline{fl} = \bar{e}]$$

$$\langle\!\langle (jc) \overset{yes}{\Rightarrow} s \Rightarrow (\overline{fl}, \bar{e}) \rangle\!\rangle = \textbf{if } (jc) \textbf{ then } \{\langle\!\langle s \rangle\!\rangle ; [\overline{fl} = \bar{e}]\} \textbf{ else } \{[outLinkOf(s) = \overline{\textbf{false}}]\}$$

$$\langle\!\langle (jc) \overset{no}{\Rightarrow} s \Rightarrow (\overline{fl}, \bar{e}) \rangle\!\rangle = \textbf{if } (jc) \textbf{ then } \{\langle\!\langle s \rangle\!\rangle ; [\overline{fl} = \bar{e}]\} \textbf{ else } \{\textbf{throw}(\phi_{join\_f})\}$$

$$\langle\!\langle \textstyle\sum_{i \in \{1..n\}} p_i \bullet o_i ? \bar{w}_i . s_i \rangle\!\rangle = p_1 \bullet o_1 ? \bar{w}_1 . [\bigcup_{j \in \{2..n\}} outLinkOf(s_j) = \overline{\textbf{false}}] . \langle\!\langle s_1 \rangle\!\rangle + \ldots + p_n \bullet o_n ? \bar{w}_n . [\bigcup_{j \in \{1..n-1\}} outLinkOf(s_j) = \overline{\textbf{false}}] . \langle\!\langle s_n \rangle\!\rangle$$

- Join conditions are boolean conditions within conditional constructs, transition conditions are assigned to flow links by means of $[\overline{fl} = \bar{e}]$
- When a branch of a choice among (linked) services is selected, the links outgoing from the activities of the discarded branches are set to **false**
- If *suppress join failure* is set to 'no', a join condition failure produces a fault signal that can be caught by a proper fault handler

Flow graphs: encoding

$$\langle\!\langle [\overline{fl}]\ ls \rangle\!\rangle = [\overline{fl}]\ \langle\!\langle ls \rangle\!\rangle$$

$$\langle\!\langle ls_1 \mid ls_2 \rangle\!\rangle = \langle\!\langle ls_1 \rangle\!\rangle \mid \langle\!\langle ls_2 \rangle\!\rangle$$

$$\langle\!\langle s \Rightarrow (\overline{fl}, \bar{e}) \rangle\!\rangle = \langle\!\langle s \rangle\!\rangle ; [\overline{fl} = \bar{e}]$$

$$\langle\!\langle (jc) \overset{yes}{\Rightarrow} s \Rightarrow (\overline{fl}, \bar{e}) \rangle\!\rangle = \textbf{if } (jc) \textbf{ then } \{\langle\!\langle s \rangle\!\rangle ; [\overline{fl} = \bar{e}]\} \textbf{ else } \{[outLinkOf(s) = \overline{\textbf{false}}]\}$$

$$\langle\!\langle (jc) \overset{no}{\Rightarrow} s \Rightarrow (\overline{fl}, \bar{e}) \rangle\!\rangle = \textbf{if } (jc) \textbf{ then } \{\langle\!\langle s \rangle\!\rangle ; [\overline{fl} = \bar{e}]\} \textbf{ else } \{\textbf{throw}(\phi_{join\_f})\}$$

$$\langle\!\langle \textstyle\sum_{i \in \{1..n\}} p_i \bullet o_i ? \bar{w}_i . s_i \rangle\!\rangle = p_1 \bullet o_1 ? \bar{w}_1 . [\bigcup_{j \in \{2..n\}} outLinkOf(s_j) = \overline{\textbf{false}}] . \langle\!\langle s_1 \rangle\!\rangle + \ldots + p_n \bullet o_n ? \bar{w}_n . [\bigcup_{j \in \{1..n-1\}} outLinkOf(s_j) = \overline{\textbf{false}}] . \langle\!\langle s_n \rangle\!\rangle$$

- Join conditions are boolean conditions within conditional constructs, transition conditions are assigned to flow links by means of $[\overline{fl} = \bar{e}]$
- When a branch of a choice among (linked) services is selected, the links outgoing from the activities of the discarded branches are set to **false**
- If *suppress join failure* is set to 'no', a join condition failure produces a fault signal that can be caught by a proper fault handler

Flow graphs: encoding

$$\langle\!\langle [\overline{fl}]\ ls \rangle\!\rangle = [\overline{fl}]\ \langle\!\langle ls \rangle\!\rangle$$

$$\langle\!\langle ls_1 \mid ls_2 \rangle\!\rangle = \langle\!\langle ls_1 \rangle\!\rangle \mid \langle\!\langle ls_2 \rangle\!\rangle$$

$$\langle\!\langle s \Rightarrow (\overline{fl}, \bar{e}) \rangle\!\rangle = \langle\!\langle s \rangle\!\rangle ; [\overline{fl} = \bar{e}]$$

$$\langle\!\langle (jc) \overset{yes}{\Rightarrow} s \Rightarrow (\overline{fl}, \bar{e}) \rangle\!\rangle = \textbf{if } (jc) \textbf{ then } \{\langle\!\langle s \rangle\!\rangle ; [\overline{fl} = \bar{e}]\} \textbf{ else } \{[outLinkOf(s) = \overline{\textbf{false}}]\}$$

$$\langle\!\langle (jc) \overset{no}{\Rightarrow} s \Rightarrow (\overline{fl}, \bar{e}) \rangle\!\rangle = \textbf{if } (jc) \textbf{ then } \{\langle\!\langle s \rangle\!\rangle ; [\overline{fl} = \bar{e}]\} \textbf{ else } \{\textbf{throw}(\phi_{join\_f})\}$$

$$\langle\!\langle \textstyle\sum_{i \in \{1..n\}} p_i \bullet o_i ? \bar{w}_i . s_i \rangle\!\rangle = p_1 \bullet o_1 ? \bar{w}_1 . [\bigcup_{j \in \{2..n\}} outLinkOf(s_j) = \overline{\textbf{false}}] . \langle\!\langle s_1 \rangle\!\rangle + \ldots + p_n \bullet o_n ? \bar{w}_n . [\bigcup_{j \in \{1..n-1\}} outLinkOf(s_j) = \overline{\textbf{false}}] . \langle\!\langle s_n \rangle\!\rangle$$

- Join conditions are boolean conditions within conditional constructs, transition conditions are assigned to flow links by means of $[\overline{fl} = \bar{e}]$
- When a branch of a choice among (linked) services is selected, the links outgoing from the activities of the discarded branches are set to **false**
- If *suppress join failure* is set to 'no', a join condition failure produces a fault signal that can be caught by a proper fault handler

Flow graphs: encoding

$$\langle\!\langle [\overline{fl}]\ ls \rangle\!\rangle = [\overline{fl}]\ \langle\!\langle ls \rangle\!\rangle$$

$$\langle\!\langle ls_1 \mid ls_2 \rangle\!\rangle = \langle\!\langle ls_1 \rangle\!\rangle \mid \langle\!\langle ls_2 \rangle\!\rangle$$

$$\langle\!\langle s \Rightarrow (\overline{fl}, \bar{e}) \rangle\!\rangle = \langle\!\langle s \rangle\!\rangle ; [\overline{fl} = \bar{e}]$$

$$\langle\!\langle (jc) \overset{yes}{\Rightarrow} s \Rightarrow (\overline{fl}, \bar{e}) \rangle\!\rangle = \textbf{if } (jc) \textbf{ then } \{\langle\!\langle s \rangle\!\rangle ; [\overline{fl} = \bar{e}]\} \textbf{ else } \{[outLinkOf(s) = \overline{\textbf{false}}]\}$$

$$\langle\!\langle (jc) \overset{no}{\Rightarrow} s \Rightarrow (\overline{fl}, \bar{e}) \rangle\!\rangle = \textbf{if } (jc) \textbf{ then } \{\langle\!\langle s \rangle\!\rangle ; [\overline{fl} = \bar{e}]\} \textbf{ else } \{\textbf{throw}(\phi_{join\_f})\}$$

$$\langle\!\langle \textstyle\sum_{i \in \{1..n\}} p_i \bullet o_i ? \bar{w}_i . s_i \rangle\!\rangle = p_1 \bullet o_1 ? \bar{w}_1 . [\bigcup_{j \in \{2..n\}} outLinkOf(s_j) = \overline{\textbf{false}}] . \langle\!\langle s_1 \rangle\!\rangle$$
$$+ \ \ldots \ + p_n \bullet o_n ? \bar{w}_n . [\bigcup_{j \in \{1..n-1\}} outLinkOf(s_j) = \overline{\textbf{false}}] . \langle\!\langle s_n \rangle\!\rangle$$

- Join conditions are boolean conditions within conditional constructs, transition conditions are assigned to flow links by means of $[\overline{fl} = \bar{e}]$
- When a branch of a choice among (linked) services is selected, the links outgoing from the activities of the discarded branches are set to **false**
- If *suppress join failure* is set to 'no', a join condition failure produces a fault signal that can be caught by a proper fault handler