



# *Formal Methods For Web Service Process Modeling*

**Dr. Yuhong Yan  
NRC-IIT**

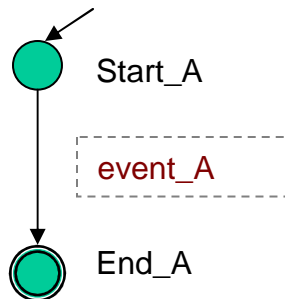


# Modeling Web Service Processes with Process Algebra

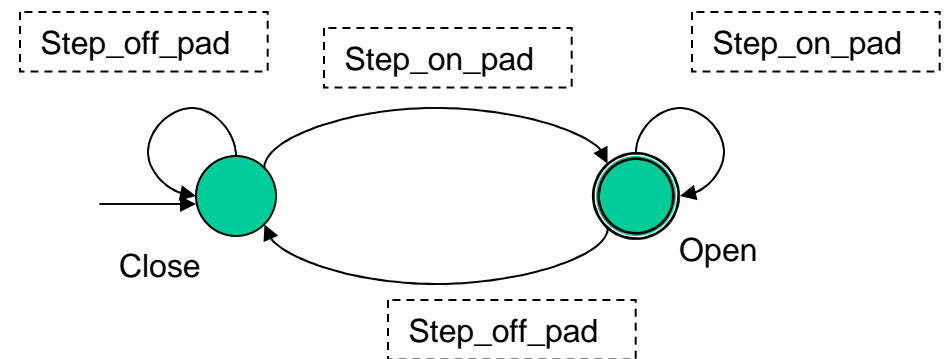
# Automata

A finite automaton  $\Gamma$  is a 5-tuple  $\Gamma = (X, \Sigma, T, x_0, F)$ , where:

- ★  $X$  is a finite set of states;
- ★  $\Sigma$  is a finite set of events;
- ★  $T \subseteq X \times \Sigma \rightarrow X$  is a finite set of transitions;
- ★  $x_0 \in X$  is a initial state;
- ★  $F \subseteq X$  is a finite set of final states.



A Web Service Activity



An Automatic Door



## *Language of Automaton*

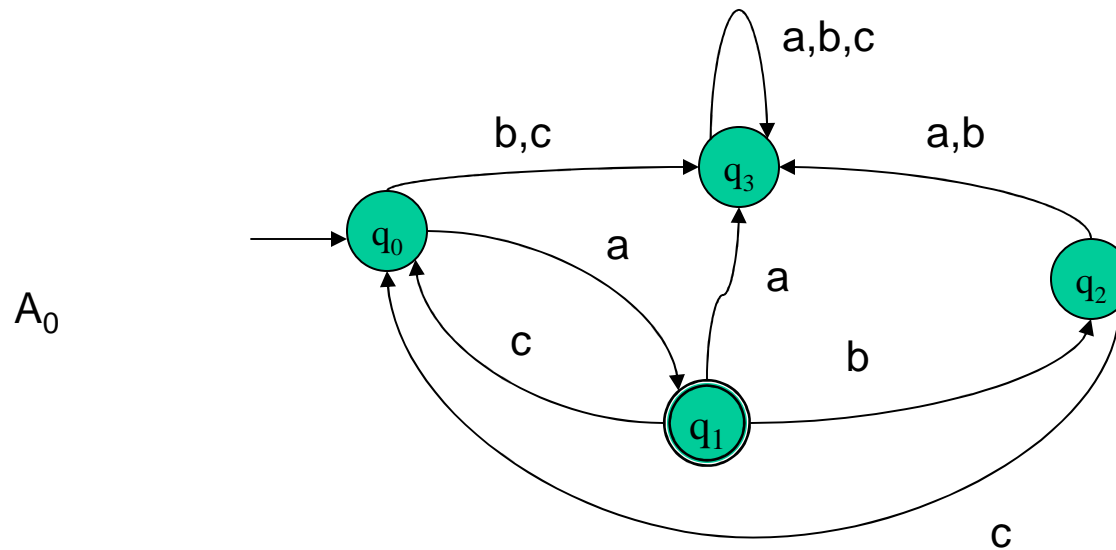
- **A is an automaton,  $s=e_1e_2\dots e_n$  is a string over  $\Sigma$ , then A is said to accept s if there is a path in A, from  $x_0$  to a final state (accepting state), whose arcs are labeled successively  $e_1e_2\dots e_n$ , A is said to accept string s.**
- **The language L of A is the set of strings accepted by A.**



## *Regular Set*

- The operations for building sets of strings
  - Union:  $S_1 \cup S_2$
  - Concatenation:  $S_1 \bullet S_2 = \{s_1 s_2 \mid s_1 \in S_1, s_2 \in S_2\}$
  - Iteration:  $S^* = \{\varepsilon\} \cup S \cup S \bullet S \cup S \bullet S \bullet S \cup \dots$

## *The algebra for automaton*



$$(1) X_0 = aX_1 + bX_3 + cX_3$$

$$(2) X_1 = aX_3 + bX_2 + cX_0 + \varepsilon$$

$$(3) X_2 = aX_3 + bX_3 + cX_0$$

$$(4) X_3 = aX_3 + bX_3 + cX_3$$

$$L = X_0 = a((bc+c)a)^*$$

# Process Algebra

- The processes are given by

$$P ::= a.P \mid P + P' \mid P \parallel_S P' \mid P \setminus \{a\} \mid P[f] \mid !P \mid \mathbf{0}$$

- ★ the *inaction*  $\mathbf{0}$ : a process that does nothing;
- ★ the *prefix*  $a.P$ :  $a \in \mathcal{Act} = \{a, b, c, \dots\} \cup \{\tau\}$ .  $\tau$  is non-observable action.  
 $a$ : receive a message.  $'a$ : emit a message.
- ★ *nondeterministic choice*  $P + P'$ .
- ★ *parallel composition*  $P \parallel_S P'$ .  $S \subseteq \mathcal{Act}$ : the synchronization set.
- ★ *restriction*  $P \setminus \{a\}$ : action  $a$  is within the scope of  $P$ . The external environment observes a  $\tau$  action.
- ★  $P[f]$  behaves like  $P$ , but with the actions relabeled by the function  $f : \mathcal{Act} \rightarrow \mathcal{Act}$ .
- ★ *replication*  $!P$ : an infinite composition of  $P|P|...$



## *Actions in Pi-calculus*

$a ::= c\langle x \rangle$

*send the name  $x$  via the channel  $c$*

$c(x)$

*receive any name via channel  $c$ ,*

*binding the name received to  $x$*

$\tau$

*unobservable action*



# *Process Algebra for WS Process Modeling*

- **Manipulate variables**

$$\bar{w} \mapsto \bar{u}$$

- **Basic activities:**

$\langle \text{receive } \underline{a} \ \underline{v} \rangle \langle \text{corr. } \underline{w} \ \langle \backslash \text{corr. } \rangle \langle \backslash \text{receive} \rangle : \text{rcv}(\bar{l}, \bar{v}, \bar{w})$

- **Structured activities**

$\langle \text{sequence} \rangle : ;$

$\langle \text{flow} \rangle : \parallel$

$\langle \text{while condition} = \underline{e} \ \underline{a} \ \langle \backslash \text{while} \rangle : \text{while}(e) \{a\}$



# *Reaction Rules*

$$\tau.P + M \rightarrow P$$

$$(a.P + M) | (\bar{a}.Q + N) \rightarrow P | Q$$

$$\frac{P \rightarrow Q}{P | Q \rightarrow P' | Q}$$

$$\frac{P \rightarrow P}{Q \rightarrow Q}$$

If  $P \equiv P'$ ,  $Q \equiv Q'$

# *Process Algebra for WS Process Modeling*

- **Operational Semantics**

<flow >:

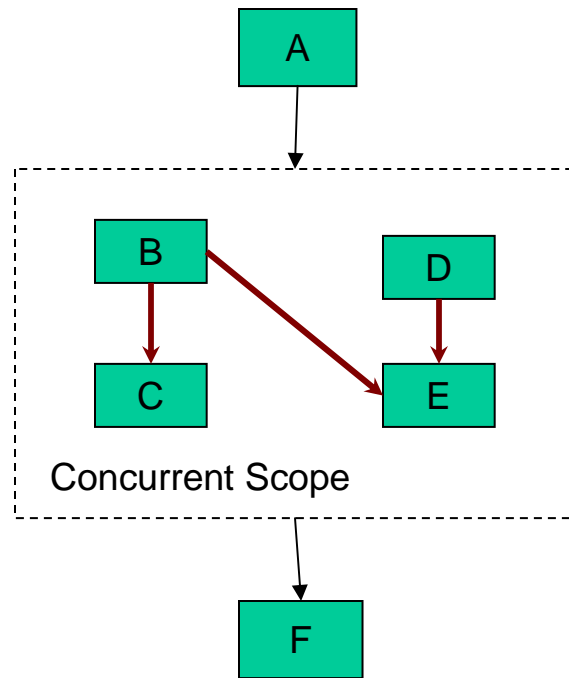
$$(\bar{w} \mapsto \bar{u})(s_0 \parallel s_1) \xrightarrow{\alpha} (\bar{w}' \mapsto \bar{u}')(s'_0 \parallel s_1) \text{ if } (\bar{w} \mapsto \bar{u})s_0 \xrightarrow{\alpha} (\bar{w}' \mapsto \bar{u}')s'_0$$

<link >

$$(\bar{w} \mapsto \bar{u})(source(\lambda); s \parallel target(\lambda); s') \xrightarrow{\tau} (\bar{w} \mapsto \bar{u})(s \parallel s')$$

# Process Algebra for the Example WS Process

## A process



```

recv( $\bar{a}, \bar{v}, \bar{p}$ );
{ { invoke( $\bar{b}, \bar{b}_{in}, \bar{b}_{out}$ ); ((source(b_to_c); target(b_to_c));
  invoke( $\bar{c}, \bar{c}_{in}, \bar{c}_{out}$ )) || source(b_to_e)) } ||
  { ((invoke( $\bar{d}, \bar{d}_{in}, \bar{d}_{out}$ ); source(d_to_e));
    target(d_to_e)) || target(b_to_e));
    invoke( $\bar{e}, \bar{e}_{in}, \bar{e}_{out}$ ) } };
send( $\bar{f}, \bar{w}, \bar{p}$ )
  
```

Simplified to :

```

recv( $\bar{a}, \bar{v}, \bar{p}$ );
{ { invoke( $\bar{b}, \bar{b}_{in}, \bar{b}_{out}$ ); (
  invoke( $\bar{c}, \bar{c}_{in}, \bar{c}_{out}$ ) || source(b_to_e)) } ||
  { (invoke( $\bar{d}, \bar{d}_{in}, \bar{d}_{out}$ ) || target(b_to_e));
    invoke( $\bar{e}, \bar{e}_{in}, \bar{e}_{out}$ ) } };
send( $\bar{f}, \bar{w}, \bar{p}$ )
  
```



## *Control flow verification*

- **Temporal logic model checking to prove properties of services: liveness, safety, and request-response (a request is always satisfied, also for infinite behaviors). If the property is not satisfied, a counterexample is returned.**
- **Bisimulation, to check whether the behaviors of two services or two versions of the same service are equivalent; or, if they are different.**
- **Simulation, to check whether the behavior of a service is included within the behavior of other interacting services.**
- **Execution traces of the service, to understand the behavior of the service.**



## *Data flow verification*

- **data type checking, in the case of LOTOS and other process algebras allowing data handling.**



## *Reconfiguration*

- **Change business partners or business process at run time**
- **Model the end points in the model**
- **Model the operations to forward the end points?**
- **BPEL has to use the forwarded end points?**
- **Replanning?**