Labeling, Hiding, Structure Diagrams
CS 2SD3

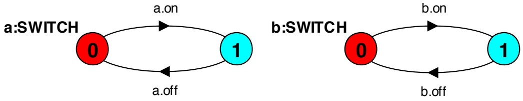
Ryszard Janicki

Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada

Process Instances and Labeling

- $a : P$ prefixes each action label in the alphabet of P with a
- Two instances of a switch process:

$$SWITCH = on \rightarrow off \rightarrow SWITCH$$
$$\| \ TWO\_SWITCH = a : SWITCH \ \| \ b : SWITCH$$



- An array of *instances* of the switch process:

$$\| \ SWITCHES(N = 3) = (forall[i : 1..N] s[i] : SWITCH)$$
$$\| \ SWITCHES(N = 3) = (s[i : 1..N] : SWITCH)$$

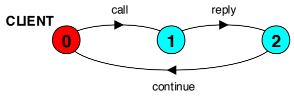
Action Relabeling

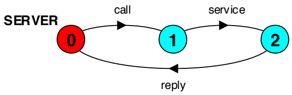
- Relabeling functions are applied to processes to change the names of action labels. The general form of the relabeling function is:

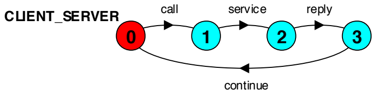
$$/\{newlabel_1/oldlabel_1, \ldots, newlabel_n/oldlabel_n\}.$$

- Relabeling is used to ensure that composed processes synchronize on particular actions.

$$CLIENT = call \to wait \to continue \to CLIENT$$
$$SERVER = request \to service \to reply \to SERVER$$
$$\| CLIENT\_SERVER = (CLIENT \parallel SERVER)/\{call/request, reply/wait\}$$
$$\Downarrow$$
$$CLIENT = call \to reply \to continue \to CLIENT$$
$$SERVER = call \to service \to reply \to SERVER$$







Process labeling by a set of prefix labels

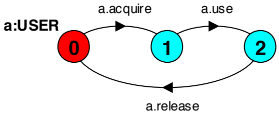
- $\{a1, \ldots, ax\} :: P$ replaces every action label n in the alphabet of P with the labels $a1.n, \ldots, ax.n$. Thus, every transition $(n \rightarrow X)$ in the definition of P is replaced with the transitions $(\{a1.n, \ldots, ax.n\} \rightarrow X)$.
$$\Updownarrow$$
$(a1.n \rightarrow X \mid a2.n \rightarrow X \mid \ldots \mid ax.n \rightarrow X)$
- Process prefixing is useful for modeling shared resources:

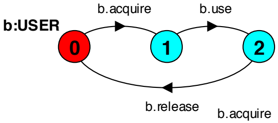
$$RESOURCE = acquire \rightarrow release \rightarrow RESOURCE$$
$$USER = acquire \rightarrow use \rightarrow release \rightarrow USER$$

$$\| RESOURCE\_SHARE = a : USER \parallel b : USER \parallel \{a, b\} :: RESOURCE$$

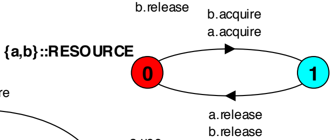
Process prefix labels for shared resources

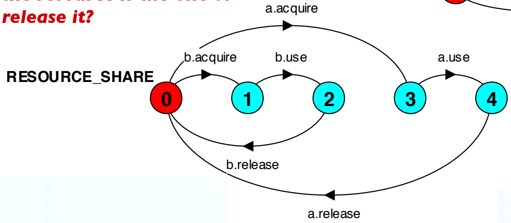
$$RESOURCE = acquire \rightarrow release \rightarrow RESOURCE$$
$$USER = acquire \rightarrow use \rightarrow release \rightarrow USER$$
$$\| RESOURCE\_SHARE = a : USER \| b : USER \| \{a, b\} :: RESOURCE$$





How does the model ensure that the user that acquires the resource is the one to release it?





Can this be achieved using relabelling rather than sharing? How?

Action relabeling - prefix labels

An alternative formulation of the client server system is described below using qualified or prefixed labels:

```
SERVERv2 = (accept.request
             ->service->accept.reply->SERVERv2).
CLIENTv2 = (call.request
             ->call.reply->continue->CLIENTv2).


||CLIENT_SERVERv2 = (CLIENTv2 || SERVERv2)
                    /{call/accept}.
```

Action Hiding

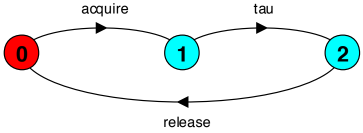
- When applied to a process P, the hiding operator $\backslash\{a1 \ldots ax\}$ removes the action names $a1 \ldots ax$ from the alphabet of P and makes these concealed actions "*silent*". These silent actions are labeled τ. Silent actions in different processes are not shared.
- Sometimes it is more convenient to specify the set of labels to be exposed:

  When applied to a process P, the interface operator $@\{a1 \ldots ax\}$ *hides* all actions in the alphabet of P *not labeled* in the set $\{a1 \ldots ax\}$.

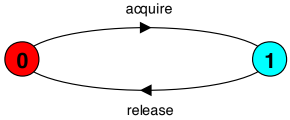
$$USER = (acquire \rightarrow use \rightarrow release \rightarrow USER) \setminus \{use\}$$

$$\Updownarrow$$

$$USER = (acquire \rightarrow use \rightarrow release \rightarrow USER)@\{acquire, release\}$$

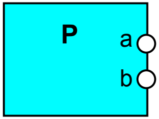


$\Longleftrightarrow$

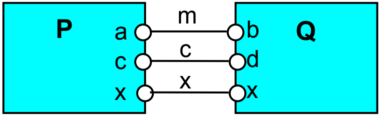


- The above $\Longleftrightarrow$ follows form the standard procedure of removing ε-moves (λ/τ-moves) in automata theory. This is **NOT** minimization as the textbook claims!

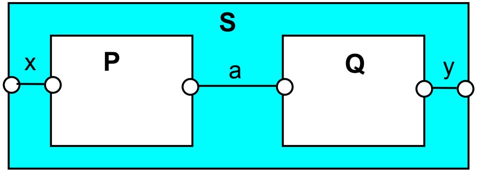
Structure Diagrams - Systems as Interacting Processes



Process P with alphabet {a,b}.



Parallel Composition (P||Q) / {m/a,m/b,c/d}



Composite process ||S = (P||Q) @ {x,y}

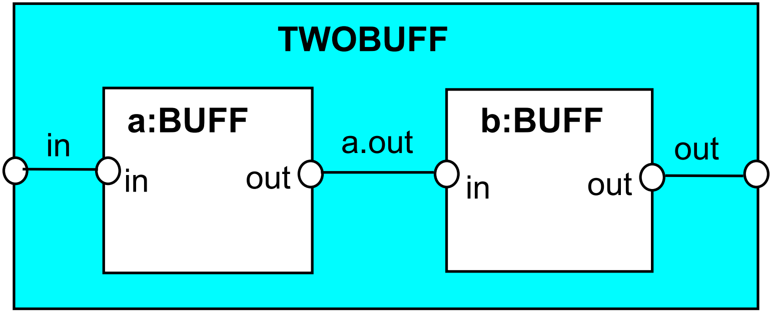
Structure Diagrams

- We use structure diagrams to capture the structure of a model expressed by the static combinators: *parallel composition*, *relabeling* and *hiding*.

  $rangeT = 0..3$

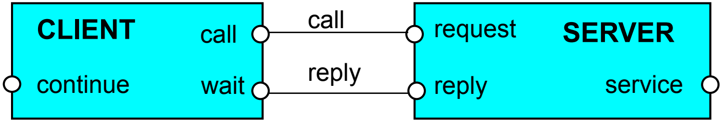
  $BUFF = (in[i : T] \rightarrow out[i] \rightarrow BUFF)$

  $\| \; TWOBUFF = ((a : BUFF \; \| \; b : BUFF)/\{a.out/b.in\})@\{in, out\}$

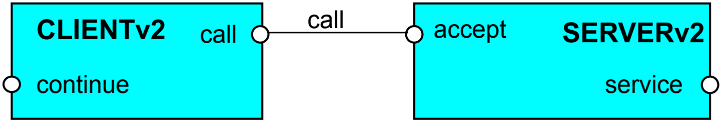


Structure Diagrams

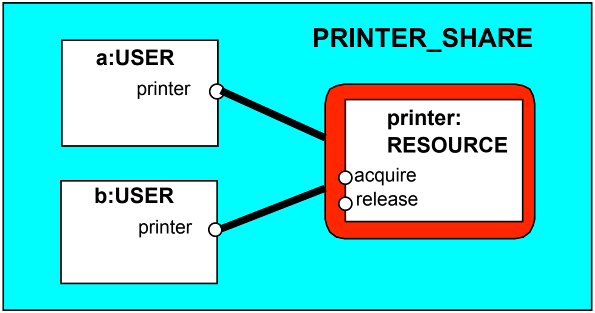
Structure diagram for **`CLIENT_SERVER`**



Structure diagram for **`CLIENT_SERVERv2`**



Structure Diagrams - Resource Sharing



```
RESOURCE = (acquire->release->RESOURCE).
USER =     (printer.acquire->use
            ->printer.release->USER)\{use}.

||PRINTER_SHARE
  = (a:USER||b:USER||{a,b}::printer:RESOURCE).
```

$\{a, b\} :: printer : RESOURCE =$
$(a.printer.acquire \to a.printer.release \to RESOURCE$
$\mid b.printer.acquire \to b.printer.release \to RESOURCE)$