Parallel Composition in FSP

If $P$ and $Q$ are processes then $(P || Q)$ represents the concurrent execution of $P$ and $Q$. The operator $||$ is the parallel composition operator.

Commutative: $(P || Q) = (Q || P)$
Associative: $(P || (Q || R)) = ((P || Q) || R)$

= $(P || Q || R)$
Parallel Composition in FSP

\[
\text{SCRATCH} = (\text{scratch} \rightarrow \text{STOP}). \\
\text{TALK} = (\text{think} \rightarrow \text{talk} \rightarrow \text{STOP}). \\
\text{TALK} \_ \text{SCRATCH} = (\text{SCRATCH} \parallel \text{TALK}).
\]

Possible traces

- think \(\rightarrow\) talk \(\rightarrow\) scratch
- think \(\rightarrow\) scratch \(\rightarrow\) talk
- scratch \(\rightarrow\) think \(\rightarrow\) talk

Interleaving of Concurrent Actions

The states of the individual processes SCRATCH and ITCH, respectively.
Another Parallel Composition

Clock radio example

\[
\text{CLOCK} = (\text{tick} \rightarrow \text{CLOCK}). \\
\text{RADIO} = (\text{on} \rightarrow \text{off} \rightarrow \text{RADIO}). \\
\mid \mid \text{CLOCK\_RADIO} = (\text{CLOCK} \mid \mid \text{RADIO}).
\]

LTS? Traces? Number of states?

Process Interaction in FSP

If processes in a composition have actions in common, these actions are said to be shared -- shared actions are used to model process interaction.

- How is a shared action different than a non-shared action?
  - The execution of non-shared actions may be arbitrarily interleaved
  - Shared actions must be executed at the same time by all processes that share the action
**Process Interaction in FSP**

Producer/consumer example

\[
\begin{align*}
MAKE & = (\text{make} \rightarrow \text{ready} \rightarrow \text{MAKE}). \\
USE & = (\text{ready} \rightarrow \text{use} \rightarrow \text{USE}). \\
\mid \mid \text{MAKE}_\text{USE} & = (\text{MAKE} \mid \mid \text{USE}).
\end{align*}
\]

MAKE synchronizes with USE when ready.

**Process Handshaking in FSP**

Handshaking to acknowledge an action by another

\[
\begin{align*}
\text{MAKE}_v2 & = (\text{make} \rightarrow \text{ready} \rightarrow \text{used} \rightarrow \text{MAKE}_v2). \\
\text{USE}_v2 & = (\text{ready} \rightarrow \text{use} \rightarrow \text{used} \rightarrow \text{USE}_v2). \\
\mid \mid \text{MAKE}_\text{USE}_v2 & = (\text{MAKE}_v2 \mid \mid \text{USE}_v2).
\end{align*}
\]

Interaction constrains the overall behavior
Process Handshaking in FSP

Multi-party synchronization

\[
\begin{align*}
\text{MAKE}_A &= (\text{make}A \rightarrow \text{ready} \rightarrow \text{used} \rightarrow \text{MAKE}_A). \\
\text{MAKE}_B &= (\text{make}B \rightarrow \text{ready} \rightarrow \text{used} \rightarrow \text{MAKE}_B). \\
\text{ASSEMBLE} &= (\text{ready} \rightarrow \text{assemble} \rightarrow \text{used} \rightarrow \text{ASSEMBLE}). \\
\text{FACTORY} &= (\text{MAKE}_A \ || \ \text{MAKE}_B \ || \ \text{ASSEMBLE}).
\end{align*}
\]

Composite Processes in FSP

A composite process is a parallel composition of primitive processes. These composite processes can be used in the definition of further compositions.

\[
\begin{align*}
\text{MAKERS} &= (\text{MAKE}_A \ || \ \text{MAKE}_B). \\
\text{FACTORY} &= (\text{MAKERS} \ || \ \text{ASSEMBLE}).
\end{align*}
\]

Substituting the definition for \text{MAKERS} in \text{FACTORY} and applying the commutative and associative laws for parallel composition results in the original definition for \text{FACTORY} in terms of primitive processes.

\[
\text{FACTORY} = (\text{MAKE}_A \ || \ \text{MAKE}_B \ || \ \text{ASSEMBLE}).
\]
Using Duplicate Processes in FSP

Modeling two light switch processes

\[
\text{SWITCH} = (\text{on} \rightarrow \text{off} \rightarrow \text{SWITCH}) . \\
\text{||TWO_SWITCH} = (\text{SWITCH} \mid \mid \text{SWITCH}) .
\]

This does not work, why?

All actions are shared, thus the resulting process composition reduces to a single light switch.

---

Process Labeling in FSP

\( a : P \) prefixes each action label in the alphabet of \( P \) with \( a \).

Correctly modeling two light switch processes

\[
\text{SWITCH} = (\text{on} \rightarrow \text{off} \rightarrow \text{SWITCH}) . \\
\text{||TWO_SWITCH} = (a : \text{SWITCH} \mid \mid b : \text{SWITCH}) .
\]

The actions of the process are relabeled with the prefix

\( a : \text{SWITCH} \)

\( b : \text{SWITCH} \)
### Process Labeling in FSP

**LTS graph of two light switches**

![LTS graph of two light switches](image)

An array of light switches

```
||SWITCHES(N=3) = (s[i:1..N]:SWITCH).
```

### Labeling with a Set of Prefixes

A set of prefixes are useful for modeling shared resources in a program.

- {a1, ..., ax} replaces every action label n in the alphabet of P with the labels a1.n, ..., ax.n.
- Further, every transition (n→X) in the definition of P is replaced with the transitions ({a1.n, ..., ax.n}→X).

Labeling with a set of prefixes are useful for modeling *shared* resources in a program.

```
RESOURCE = (acquire->release->RESOURCE).
USER = (acquire->use->release->USER).
||RESOURCE_SHARE = (a:USER || b:USER || {a,b}:RESOURCE).
```
Labeling with a Set of Prefixes

How does the model ensure that the user that acquires the resource is the same one that releases it?

Relabeling Actions in FSP

Relabel functions are applied to processes to change the names of action labels. The general form of the relabel function is:
\[/\{\text{newlabel1/oldlabel1, \ldots, newlabeln/oldlabeln}\}\].

Relabeling to ensure that composed processes synchronize on particular actions.

\[
\text{CLIENT} = (\text{call} \rightarrow \text{wait} \rightarrow \text{continue} \rightarrow \text{CLIENT}).
\]
\[
\text{SERVER} = (\text{request} \rightarrow \text{service} \rightarrow \text{reply} \rightarrow \text{SERVER}).
\]
\[
\text{CLIENT SERVER} = (\text{CLIENT} \mid \mid \text{SERVER})
\]
\[
\{/\{\text{call/request, reply/wait}\}\}.
\]
**Relabeling Actions in FSP**

\[
|| \text{CLIENT\_SERVER} = (\text{CLIENT} || \text{SERVER}) \\
/{\text{call/request, reply/wait}}. \\
\]

**Action Hiding in FSP**

When applied to a process \( P \), the hiding operator \( \{a_1..a_x\} \) removes the action names \( a_1..a_x \) from the alphabet of \( P \) and makes these concealed actions "silent". These silent actions are labeled \( \tau \). 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 \( @\{a_1..a_x\} \) hides all actions in the alphabet of \( P \) not labeled in the set \( a_1..a_x \).
**Action Hiding in FSP**

These processes are equivalent

\[
\text{USER} = (\text{acquire} \rightarrow \text{use} \rightarrow \text{release} \rightarrow \text{USER}) \setminus \{\text{use}\}.
\]

\[
\text{USER} = (\text{acquire} \rightarrow \text{use} \rightarrow \text{release} \rightarrow \text{USER}) @\{\text{acquire}, \text{release}\}.
\]

Minimization removes hidden \textit{tau} actions to produce an LTS with equivalent observable behavior.

Why might we do this?

---

**Structure Diagrams**

- State machine diagrams to depict the dynamic behavior of processes

- Structure diagrams capture the structure of a model expressed by the static combination operators: \textit{parallel composition}, \textit{relabeling}, and \textit{hiding}
Structure Diagrams

Process $P$ with alphabet $\{a, b\}$

Parallel Composition $P \parallel Q$

$/\{m/a, m/b, c/d\}$

Composite process $\parallel S = (P \parallel Q)$

@$\{x, y\}$

range $T = 0..3$

BUFF = (in[i:T]→out[i]→BUFF).

$\parallel$TWOBUFF = ?

It can be easier to understand relabeling with structure diagrams.

$\parallel$TWOBUFF = $(a:BUFF \parallel b:BUFF)$

$/\{in/a.in, a.out/b.in, out/b.out\}$

@$\{in, out\}$. 
Structure Diagrams

Structure diagram for client/server example

Structure Diagrams

Shared resources are depicted as a rounded rectangle.