Independent Composition

**Dependent Composition**  \( P.Q \) (sequential execution)

\( P \) and \( Q \) must have exactly the same state variables

**Independent Composition**  \( P||Q \) (parallel execution)

\( P \) and \( Q \) must have completely different state variables

and the state variables of the composition are those of both \( P \) and \( Q \)

Ignoring time and space variables

\[ P||Q = P \land Q \]
**Independent Composition**

**example** in integer variables \( x, y, \) and \( z \)

\[
x := x + 1 \parallel y := y + 2
\]

partition the variables:

put \( x \) in left part, put \( y \) and \( z \) in right part

\[
x' = x + 1 \parallel y' = y + 2 \land z' = z
\]

reasonable partition rule

If either \( x' \) or \( x := \) appears in a process specification, then \( x \) belongs to that process

(then neither \( x' \) nor \( x := \) can appear in the other process specification).

If neither \( x' \) nor \( x := \) appears at all, then \( x \) can be placed on either side of the partition.
Independent Composition

example in variables \( x, y, \) and \( z \)

\[
x := y \parallel y := x
\]

partition: put \( x \) in left, \( y \) in right, \( z \) in either

\[
x' = y \land y' = x \land z' = z
\]

implementation of a process makes a private copy of the initial value of a variable belonging to the other process if the other process contains an assignment to that variable
Independent Composition

**example** in binary variable $b$ and integer variable $x$

$$
b := x = x \ || \ x := x + 1
$$

= 

$$
b := T \ || \ x := x + 1
$$

replace $x = x$ by $T$

**example** in integer variables $x$ and $y$

$$(x := x + 1, \ x := x - 1) \ || \ y := x$$

= 

$$
ok \ || \ y := x
$$

= 

$$y := x$$
Independent Composition

\[(x := x + y. \ x := x \times y) \ |\ | \ (y := x - y. \ y := x/y)\]

You should have written

\[(x := x + y \ |\ | \ y := x - y). \ (x := x \times y \ |\ | \ y := x/y)\]
Independent Composition

\[ P \| Q = \exists tP, tQ \cdot \]  
\[ \land (\text{substitute } tP \text{ for } t' \text{ in } P) \]  
\[ \land (\text{substitute } tQ \text{ for } t' \text{ in } Q) \]  
\[ \land t' = \text{max } tP \text{ tQ} \]

**laws**

\( (x := e \| y := f). \, P = (\text{for } x \text{ substitute } e \text{ and independently for } y \text{ substitute } f \text{ in } P) \)

\[ P \| Q = Q \| P \]  
\text{symmetry}

\[ P \| (Q \| R) = (P \| Q) \| R \]  
\text{associativity}

\[ P \| Q \lor R = (P \| Q) \lor (P \| R) \]  
\text{distributivity}

\[ P \| \text{if b then } Q \text{ else } R \, \text{fi} = \text{if b then } P \| Q \text{ else } P \| R \, \text{fi} \]  
\text{distributivity}

\[ \text{if b then } P \| Q \text{ else } R \| S \, \text{fi} = \text{if b then } P \text{ else } R \, \text{fi} \| \text{if b then } Q \text{ else } S \, \text{fi} \]  
\text{distributivity}
List Concurrency

\[ L_i := e \equiv L'_i = e \land (\forall j: 0,..\#L \cdot j+i \Rightarrow L'_j = L_j) \land x' = x \land y' = y \land \ldots \]

\[ L_i := e \equiv L'_i = e \land (\forall j: (\text{this part}) \cdot j+i \Rightarrow L'_j = L_j) \land x' = x \land \ldots \]

**example** find the maximum item in a nonempty list

\[ \text{findmax } 0 \ (\#L) \text{ where } \]

\[ \text{findmax } = \langle i, j \rightarrow i < j \Rightarrow L'_i = \text{MAX } L[i;..j] \rangle \]

\[ \text{findmax } i \ j \ \Leftarrow \quad \text{if } j-i = 1 \text{ then ok } \]

\[ \text{else } \ (\text{findmax } i \ (\text{div } (i+j) \ 2) \ | | \ \text{findmax } (\text{div } (i+j) \ 2) \ j). \]

\[ L_i := \text{max } (L \ i) \ (L \ (\text{div } (i+j) \ 2)) \text{ fi} \]

recursive time \( = \text{ceil } (\log \ (j-i)) \)
Sequential to Parallel Transformation

\[ x := y. \quad x := x + 1. \quad z := y \]

\[ = \quad x := y. \quad (x := x + 1 \parallel z := y) \]

\[ = \quad (x := y. \quad x := x + 1) \parallel z := y \]
Sequential to Parallel Transformation

rules

Whenever two programs occur in sequence, and neither assigns to a variable appearing in the other, they can be placed in parallel.

example $x := z. \ y := z$ becomes $x := z \ || \ y := z$

Whenever two programs occur in sequence, and neither assigns to a variable assigned in the other, and no variable assigned in the first appears in the second, they can be placed in parallel; a copy must be made of the initial value of any variable appearing in the first and assigned in the second.

example $x := y. \ y := z$ becomes $c := y. \ (x := c \ || \ y := z)$
Buffer

\[produce = \cdots b := e \cdots\]

\[consume = \cdots x := b \cdots\]

\[control = produce \cdot consume \cdot control\]

\[P \rightarrow C \rightarrow P \rightarrow C \rightarrow P \rightarrow C \rightarrow P \rightarrow C \rightarrow\]
Buffer

\[
\text{produce} = \cdots b := e \cdots
\]

\[
\text{consume} = \cdots x := b \cdots
\]

\[
\text{control} = \text{produce}. \text{newcontrol}
\]

\[
\text{newcontrol} = \text{consume}. \text{produce}. \text{newcontrol}
\]

\[
\text{produce} = \cdots b := e \cdots
\]

\[
\text{consume} = \cdots x := b \cdots
\]

\[
\text{control} = \text{produce}. \text{newcontrol}
\]

\[
\text{newcontrol} = (\text{consume} \, || \, \text{produce}). \text{newcontrol}
\]

11/19
Buffer

\[produce = \cdots b := e \cdots\]

\[consume = \cdots x := c \cdots\]

\[control = produce . newcontrol\]

\[newcontrol = c := b . (consume \parallel produce) . newcontrol\]
Buffer

\[
\begin{align*}
\text{produce} &= \cdots b \ w := e. \ w := w + 1 \cdots \\
\text{consume} &= \cdots x := b. \ r := r + 1 \cdots \\
\text{control} &= w := 0. \ r := 0. \ \text{newcontrol} \\
\text{newcontrol} &= \text{produce. consume. newcontrol}
\end{align*}
\]
Buffer

\[ \text{produce} = \ldots b \ w := e. \ w := \text{mod} (w+1) \ n \ldots \]

\[ \text{consume} = \ldots x := b \ r. \ r := \text{mod} (r+1) \ n \ldots \]

\[ \text{control} = w := 0. \ r := 0. \ \text{newcontrol} \]

\[ \text{newcontrol} = \text{produce. consume. newcontrol} \]
**Insertion Sort**

define

\[ sort = \langle n \rightarrow \forall i, j: 0,..n \cdot i \leq j \Rightarrow L_i \leq L_j \rangle \]

\[ swap = \langle i, j: 0,..\#L \rightarrow L_i := L_j \parallel L_j := L_i \rangle \]

\[ sort' (\#L) \iff sort 0 \Rightarrow sort' (\#L) \]

\[ sort 0 \Rightarrow sort' (\#L) \iff \text{for } n := 0,..\#L \text{ do } sort n \Rightarrow sort' (n+1) \text{ od} \]

\[ sort n \Rightarrow sort' (n+1) \iff \]

\hspace{1em} if \( n=0 \) then ok

\hspace{1em} else if \( L(n-1) \leq L n \) then ok

\hspace{1em} else swap \((n-1)\ n. \quad sort (n-1) \Rightarrow sort' n \fi \fi

\[
\begin{bmatrix}
L_0 & L_1 & L_2 & L_3 & L_4
\end{bmatrix}
\]

0 1 2 3 4 5
Insertion Sort

If $|i-j| > 1$ then $S_i$ and $S_j$ in parallel

If $|i-j| > 1$ then $S_i$ and $C_j$ in parallel

$C_i$ and $C_j$ in parallel
Dining Philosophers
Dining Philosophers

\[
\begin{align*}
\text{life} & = (P_0 \lor P_1 \lor P_2 \lor P_3 \lor P_4). \text{life} \\
P_i & = \text{up } i \text{. up}(i+1) \text{. eat } i \text{. down } i \text{. down}(i+1) \\
\text{up } i & = \text{chopstick } i := T \\
\text{down } i & = \text{chopstick } i := \perp \\
\text{eat } i & = \cdots \text{chopstick } i \cdots \text{chopstick}(i+1) \cdots
\end{align*}
\]

If \( i \neq j \), (up \( i \). up \( j \)) becomes (up \( i \) \| up \( j \)).
If \( i \neq j \), (up \( i \). down \( j \)) becomes (up \( i \) \| down \( j \)).
If \( i \neq j \), (down \( i \). up \( j \)) becomes (down \( i \) \| up \( j \)).
If \( i \neq j \), (down \( i \). down \( j \)) becomes (down \( i \) \| down \( j \)).
If \( i \neq j \land i+1 \neq j \), (eat \( i \). up \( j \)) becomes (eat \( i \) \| up \( j \)).
If \( i \neq j \land i+1 \neq j \), (up \( i \). eat \( j \)) becomes (up \( i \) \| eat \( j \)).
If \( i \neq j \land i+1 \neq j \), (eat \( i \). down \( j \)) becomes (eat \( i \) \| down \( j \)).
If \( i \neq j \land i+1 \neq j \land i \neq j+1 \), (eat \( i \). eat \( j \)) becomes (eat \( i \) \| eat \( j \)).
Dining Philosophers

\[
\begin{align*}
\text{life} &= (P_0 \lor P_1 \lor P_2 \lor P_3 \lor P_4). \text{ life} \\
P_i &= \text{up } i. \text{ up}(i+1). \text{ eat } i. \text{ down } i. \text{ down}(i+1) \\
\text{up } i &= \text{chopstick } i:= \top \\
\text{down } i &= \text{chopstick } i:= \bot \\
\text{eat } i &= \cdots \text{chopstick } i \cdots \text{chopstick}(i+1) \cdots
\end{align*}
\]

\[
\begin{align*}
\text{life} &= P_0 \parallel P_1 \parallel P_2 \parallel P_3 \parallel P_4 \quad \times \\
P_i &= (\text{up } i \parallel \text{up}(i+1)). \text{ eat } i. (\text{down } i \parallel \text{down}(i+1)). \text{ P } i
\end{align*}
\]