

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

$$P||Q = P \wedge Q$$

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

$$= x' = x+1 \parallel y' = y+2 \wedge z' = z$$

$$= x' = x+1 \wedge y' = y+2 \wedge z' = z$$

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

= $x' = y \wedge y' = x \wedge z' = z$

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 boolean variable b and integer variable x

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

replace $x = x$ by \top

$$= b := \top \parallel x := x + 1$$

example in integer variables x and y

$$(x := x + 1. x := x - 1) \parallel y := x$$

$$= ok \parallel y := x$$

$$= y := x$$

Independent Composition

$(x := x+y. x := x \times y) \parallel (y := x-y. y := x/y)$

Independent Composition

$(x := x + y. x := x \times y) \parallel (y := x - y. y := x / y)$

You should have written

$(x := x + y \parallel y := x - y). (x := x \times y \parallel y := x / y)$

Independent Composition

$P \parallel Q = \exists tP, tQ \cdot$ (substitute tP for t' in P)

\wedge (substitute tQ for t' in Q)

$\wedge t' = \max tP tQ$

laws

$(x := e \parallel y := f) \cdot P =$ (for x substitute e and independently for y substitute f in P)

$P \parallel Q = Q \parallel P$ symmetry

$P \parallel (Q \parallel R) = (P \parallel Q) \parallel R$ associativity

$P \parallel ok = ok \parallel P = P$ identity

$P \parallel Q \vee R = (P \parallel Q) \vee (P \parallel R)$ distributivity

$P \parallel \text{if } b \text{ then } Q \text{ else } R = \text{if } b \text{ then } (P \parallel Q) \text{ else } (P \parallel R)$ distributivity

$\text{if } b \text{ then } (P \parallel Q) \text{ else } (R \parallel S) = \text{if } b \text{ then } P \text{ else } R \parallel \text{if } b \text{ then } Q \text{ else } S$ distributivity

List Concurrency

$$Li := e = L'i = e \wedge (\forall j: 0, \dots, \#L. j \neq i \Rightarrow L'j = Lj) \wedge x' = x \wedge y' = y \wedge \dots$$

$$Li := e = L'i = e \wedge (\forall j: (\text{this part}). j \neq i \Rightarrow L'j = Lj) \wedge x' = x \wedge \dots$$

example find the maximum item in a nonempty list

findmax 0 (#L) where

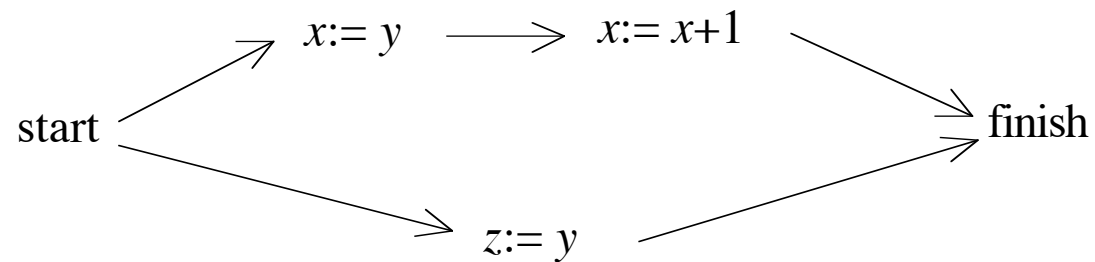
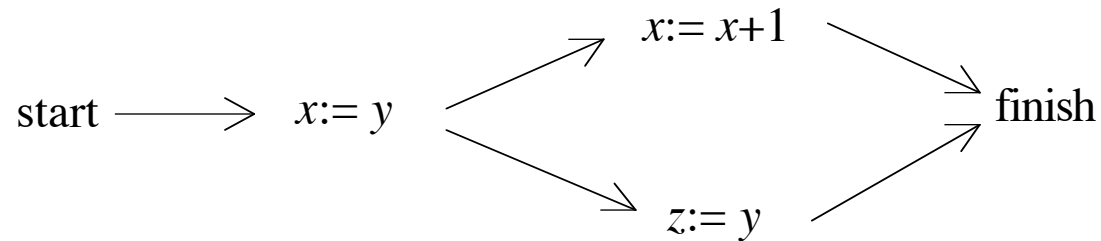
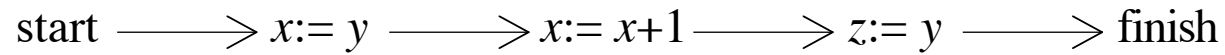
findmax = $\langle i, j \rightarrow i < j \Rightarrow L' i = \text{MAX } L [i;..j] \rangle$

findmax i j \Leftarrow **if** $j - i = 1$ **then** *ok*
else (*findmax* i (div (i+j) 2) || *findmax* (div (i+j) 2) j).
 $L i := \text{max} (L i) (L (\text{div} (i+j) 2))$)

recursive time = *ceil* (log (j-i))

Sequential to Parallel Transformation

$$\begin{aligned} & x:=y. x:=x+1. z:=y \\ = & x:=y. (x:=x+1 \parallel z:=y) \\ = & (x:=y. x:=x+1) \parallel z:=y \end{aligned}$$



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 $x := z \parallel y := z$

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 $c := y. (x := c \parallel y := z)$

Buffer

produce = $b := e$

consume = $x := b$

control = *produce*. *consume*. *control*

$P \longrightarrow C \longrightarrow P \longrightarrow C \longrightarrow P \longrightarrow C \longrightarrow P \longrightarrow C \longrightarrow$

Buffer

produce =*b:= e*.....

consume =*x:= b*.....

control = *produce. newcontrol*

newcontrol = *consume. produce. newcontrol*

Buffer

produce =*b:= e*.....

consume =*x:= b*.....

control = *produce*. *newcontrol*

newcontrol = (*consume* || *produce*). *newcontrol*

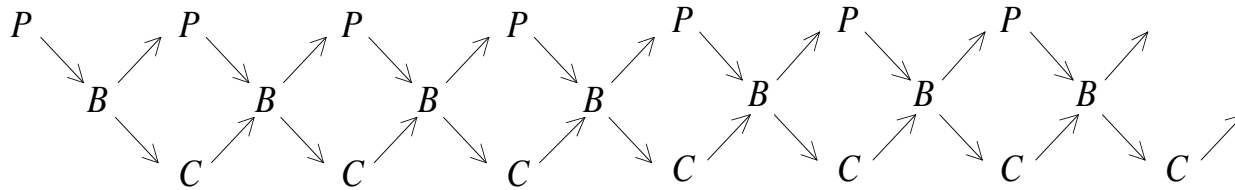
Buffer

produce =*b:= e*.....

consume =*x:= c*.....

control = *produce*. *newcontrol*

newcontrol = *c:= b*. (*consume* || *produce*). *newcontrol*



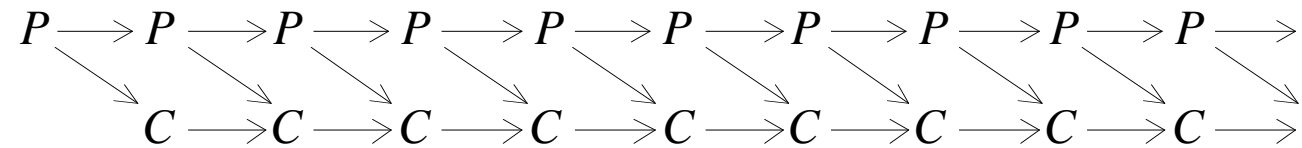
Buffer

produce =*b* *w*:= *e*. *w*:= *w*+1.....

consume =*x*:= *b* *r*. *r*:= *r*+1.....

control = *w*:= 0. *r*:= 0. *newcontrol*

newcontrol = *produce*. *consume*. *newcontrol*



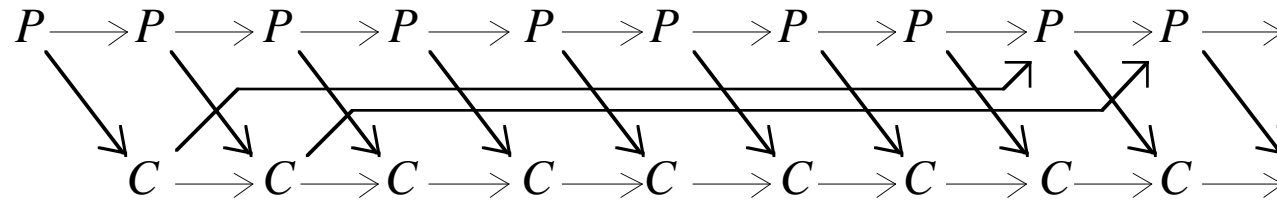
Buffer

produce =*b* *w* := *e*. *w* := *mod* (*w*+1) *n*.....

consume =*x* := *b* *r*. *r* := *mod* (*r*+1) *n*.....

control = *w* := 0. *r* := 0. *newcontrol*

newcontrol = *produce*. *consume*. *newcontrol*



Insertion Sort

define

$$\text{sort} = \langle n \rightarrow \forall i, j: 0, \dots, n \cdot i \leq j \Rightarrow L\ i \leq L\ j \rangle$$

$$\text{swap} = \langle i, j: 0, \dots, \#L \rightarrow L\ i := L\ j \parallel L\ j := L\ i \rangle$$

$$\text{sort}' (\#L) \Leftarrow \text{sort } 0 \Rightarrow \text{sort}' (\#L)$$

$$\text{sort } 0 \Rightarrow \text{sort}' (\#L) \Leftarrow \mathbf{for } n := 0; \dots, \#L \mathbf{do } \text{sort } n \Rightarrow \text{sort}' (n+1)$$

$$\text{sort } n \Rightarrow \text{sort}' (n+1) \Leftarrow$$

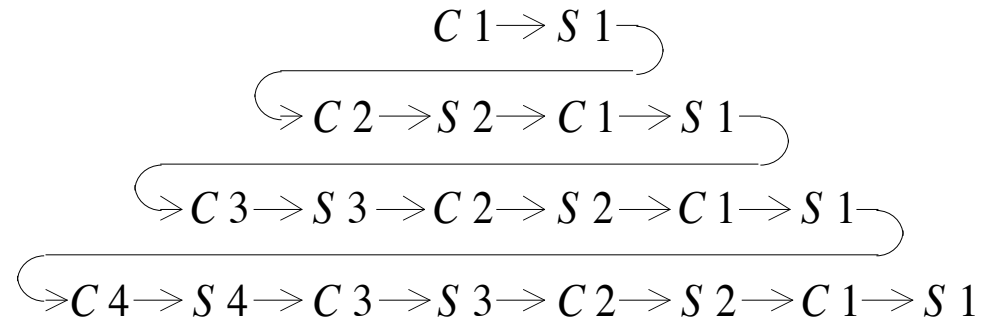
if $n=0$ **then** *ok*

else if $L (n-1) \leq L\ n$ **then** *ok*

else ($\text{swap } (n-1)\ n$. $\text{sort } (n-1) \Rightarrow \text{sort}'\ n$)

$$\begin{array}{cccccc} [L\ 0 & ; & L\ 1 & ; & L\ 2 & ; & L\ 3 & ; & L\ 4 &] \\ 0 & & 1 & & 2 & & 3 & & 4 & & 5 \end{array}$$

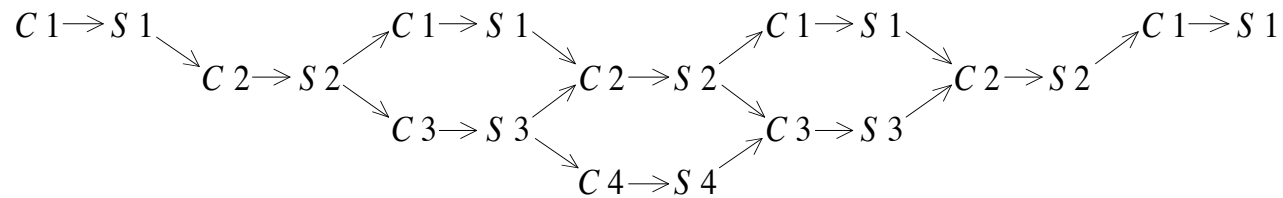
Insertion Sort



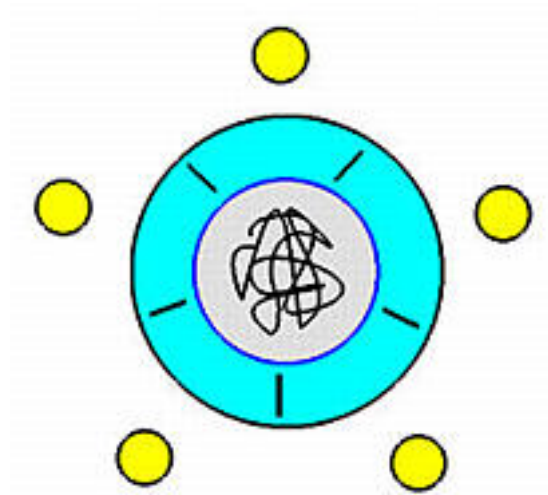
If $abs(i-j) > 1$ then S_i and S_j in parallel

If $abs(i-j) > 1$ then S_i and C_j in parallel

C_i and C_j in parallel



Dining Philosophers



Dining Philosophers

$life = (P_0 \vee P_1 \vee P_2 \vee P_3 \vee P_4). life$

$P_i = up\ i. up(i+1). eat\ i. down\ i. down(i+1)$

$up\ i = chopstick\ i := \top$

$down\ i = chopstick\ i := \perp$

$eat\ i = \dots\dots chopstick\ i \dots\dots chopstick(i+1) \dots\dots$

If $i \neq j$, $(up\ i. up\ j)$ becomes $(up\ i \parallel up\ j)$.

If $i \neq j$, $(up\ i. down\ j)$ becomes $(up\ i \parallel down\ j)$.

If $i \neq j$, $(down\ i. up\ j)$ becomes $(down\ i \parallel up\ j)$.

If $i \neq j$, $(down\ i. down\ j)$ becomes $(down\ i \parallel down\ j)$.

If $i \neq j \wedge i+1 \neq j$, $(eat\ i. up\ j)$ becomes $(eat\ i \parallel up\ j)$.

If $i \neq j \wedge i \neq j+1$, $(up\ i. eat\ j)$ becomes $(up\ i \parallel eat\ j)$.

If $i \neq j \wedge i+1 \neq j$, $(eat\ i. down\ j)$ becomes $(eat\ i \parallel down\ j)$.

If $i \neq j \wedge i \neq j+1$, $(down\ i. eat\ j)$ becomes $(down\ i \parallel eat\ j)$.

If $i \neq j \wedge i+1 \neq j \wedge i \neq j+1$, $(eat\ i. eat\ j)$ becomes $(eat\ i \parallel eat\ j)$.

Dining Philosophers

$life = (P_0 \vee P_1 \vee P_2 \vee P_3 \vee P_4). life$

$P_i = up\ i. up(i+1). eat\ i. down\ i. down(i+1)$

$up\ i = chopstick\ i := \top$

$down\ i = chopstick\ i := \perp$

$eat\ i = \dots\dots chopstick\ i \dots\dots chopstick(i+1) \dots\dots$

$life = P_0 \parallel P_1 \parallel P_2 \parallel P_3 \parallel P_4$

$P_i = (up\ i \parallel up(i+1)). eat\ i. (down\ i \parallel down(i+1)). P_i$