

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

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

Example: in variable x

$$x := x+1 = x' = x+1$$

in variables y and z

$$y := y+2 = y' = y+2 \wedge z' = z$$

in variables x , y , and z

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

Partitioning:

If either x' or $x:=$ appears in a process specification, then x belongs to that process, so 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.

$$x:=y \parallel y:=x \quad = \quad x'=y \wedge y'=x \wedge z'=z$$

x belongs to the left process

y belongs to the right process

z belongs to either process

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

In boolean variable b and integer variable x ,

$$\begin{aligned} & b:= x=x \parallel x:= x+1 && \text{replace } x=x \text{ by } \top \\ = & b:= \top \parallel x:= x+1 \end{aligned}$$

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

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

versus

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

With time, independent composition is defined as

$$\begin{aligned} P \parallel Q &= \exists tP, tQ. \quad (\text{substitute } tP \text{ for } t' \text{ in } P) \\ &\quad \wedge (\text{substitute } tQ \text{ for } t' \text{ in } Q) \\ &\quad \wedge t' = \max tP \ tQ \end{aligned}$$

Laws of Independent Composition

$(x := e \parallel y := f). P =$ (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

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

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

List Concurrency

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

partition within lists

Example: maximum item in a nonempty list

findmax 0 (#L) where

$$\textit{findmax} = \lambda i, j. i < j \Rightarrow L' i = \textit{MAX} L [i;..j]$$

findmax i j \Leftarrow **if** j-i = 1 **then** ok

else ((*findmax* i (div (i+j) 2)

|| findmax (div (i+j) 2) j).

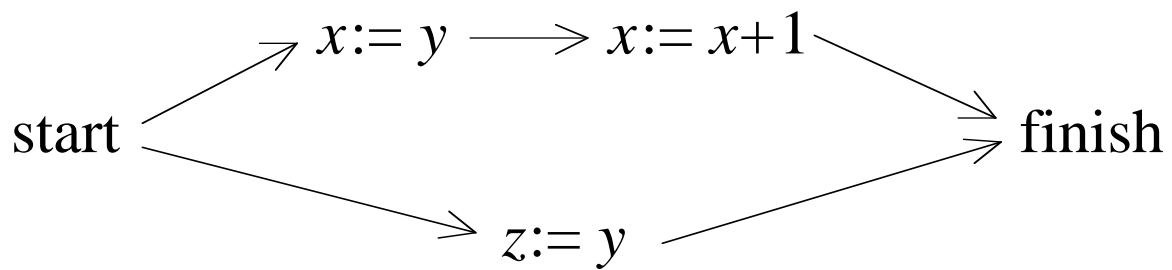
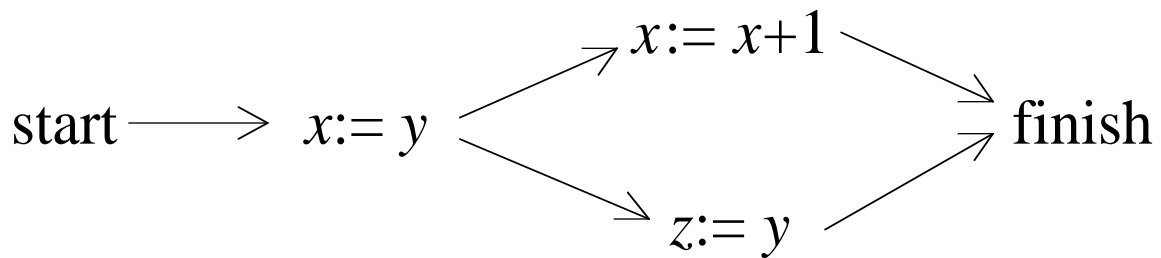
L i := max (L i) (L (div (i+j) 2)))

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

Sequential to Parallel Transformation

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

start \longrightarrow $x:=y$ \longrightarrow $x:=x+1$ \longrightarrow $z:=y$ \longrightarrow finish



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.

Whenever two programs occur in sequence, and neither assigns to a variable appearing in the other, they can be placed in parallel without any copying of initial values.

Buffer

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

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

control = *produce. consume. control*

P → *C* → *P* → *C* → *P* → *C* → *P* → *C* →



control = *produce. newcontrol*

newcontrol = *consume. produce. newcontrol*

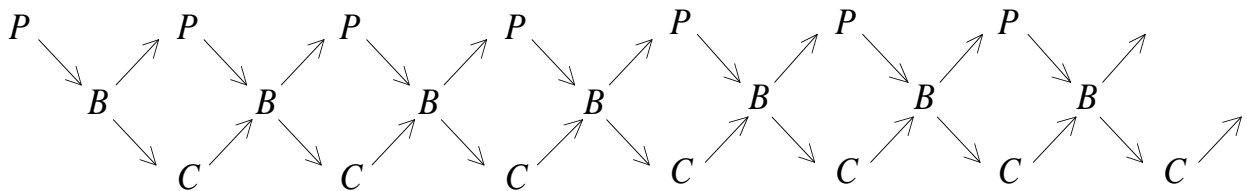
newcontrol = (*consume || produce*). *newcontrol*

produce =*p*:= *e*.....

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

control = *produce*. *newcontrol*

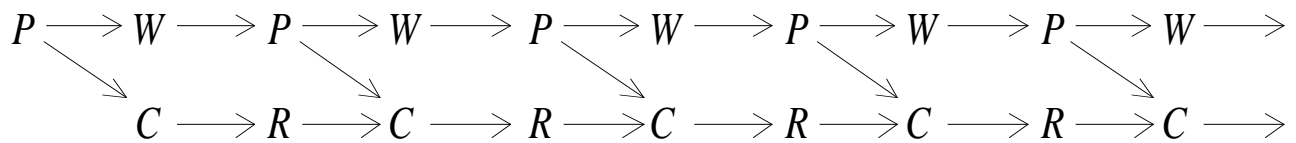
newcontrol = *c*:= *p*. (*consume* || *produce*). *newcontrol*



produce =*bw*:= *e*.....

consume =*x*:= *br*.....

control = *produce*. *w*:= *w*+1. *consume*. *r*:= *r*+1. *control*



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

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

control

Insertion Sort

define

$$\text{sort} = \lambda n. \forall i, j: 0, \dots, n. i \leq j \Rightarrow L_i \leq L_j$$

$$\text{swap } i \ j = L_i := L_j \parallel L_j := L_i$$

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

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

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

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

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

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

[L_0 ; L_1 ; L_2 ; L_3 ; L_4]

0 1 2 3 4 5

Dining Philosophers

$$life = (P\ 0 \vee P\ 1 \vee P\ 2 \vee P\ 3 \vee P\ 4). \text{ life}$$

$$P\ i = \text{up } i. \text{ up}(i+1). \text{ eat } i. \text{ dn } i. \text{ dn}(i+1)$$

$$\text{up } i = \text{chopstick } i := \top$$

$$\text{dn } i = \text{chopstick } i := \perp$$

$$\text{eat } i = \dots\dots\text{chopstick } i\dots\dots\text{chopstick}(i+1)\dots\dots$$

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

If $i \neq j$, $(\text{up } i. \text{ dn } j)$ becomes $(\text{up } i \parallel \text{dn } j)$.

If $i \neq j$, $(\text{dn } i. \text{ up } j)$ becomes $(\text{dn } i \parallel \text{up } j)$.

If $i \neq j$, $(\text{dn } i. \text{ dn } j)$ becomes $(\text{dn } i \parallel \text{dn } j)$.

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

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

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

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

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

$$life = P\ 0 \parallel P\ 1 \parallel P\ 2 \parallel P\ 3 \parallel P\ 4$$

$$P\ i = (\text{up } i \parallel \text{up}(i+1)). \text{ eat } i. (\text{dn } i \parallel \text{dn}(i+1)). P\ i$$