# Review

<table>
<thead>
<tr>
<th>Binary Theory</th>
<th>Number Theory</th>
<th>Character Theory</th>
</tr>
</thead>
<tbody>
<tr>
<td>laws</td>
<td>Character Theory</td>
<td></td>
</tr>
<tr>
<td>proof</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bunches</th>
<th>Sets</th>
<th>Strings</th>
<th>Lists</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functions</th>
<th>Quantifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Specification</th>
<th>Refinement</th>
<th>Program Development</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time Calculation</th>
<th>real time</th>
<th>recursive time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space Calculation</td>
<td>maximum space</td>
<td>average space</td>
</tr>
<tr>
<td>assertions</td>
<td>exact precondition</td>
<td>exact postcondition</td>
</tr>
<tr>
<td>Scope</td>
<td>variable declaration</td>
<td>frame</td>
</tr>
<tr>
<td>Data Structures</td>
<td>array element assignment</td>
<td></td>
</tr>
<tr>
<td>Control Structures</td>
<td><strong>while-loop</strong></td>
<td>loop with <strong>exit</strong></td>
</tr>
</tbody>
</table>
## Review

<table>
<thead>
<tr>
<th><strong>Binary Theory</strong></th>
<th>laws</th>
<th>proof</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number Theory</strong></td>
<td>Character Theory</td>
<td></td>
</tr>
<tr>
<td><strong>Bunches</strong></td>
<td>Sets</td>
<td>Strings</td>
</tr>
<tr>
<td><strong>Functions</strong></td>
<td>Quantifiers</td>
<td></td>
</tr>
<tr>
<td>Specification</td>
<td>Refinement</td>
<td>Program Development</td>
</tr>
<tr>
<td>Time Calculation</td>
<td>real time</td>
<td>recursive time</td>
</tr>
<tr>
<td>Space Calculation</td>
<td>maximum space</td>
<td>average space</td>
</tr>
<tr>
<td>assertions</td>
<td>exact precondition</td>
<td>exact postcondition</td>
</tr>
<tr>
<td>Scope</td>
<td>variable declaration</td>
<td>frame</td>
</tr>
<tr>
<td>Data Structures</td>
<td>array element assignment</td>
<td></td>
</tr>
<tr>
<td>Control Structures</td>
<td><strong>while-loop</strong></td>
<td>loop with <strong>exit</strong></td>
</tr>
</tbody>
</table>
## Review

<table>
<thead>
<tr>
<th>Binary Theory</th>
<th>laws</th>
<th>proof</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number Theory</td>
<td>Character Theory</td>
<td></td>
</tr>
<tr>
<td>Bunches</td>
<td>Sets</td>
<td>Strings, Lists</td>
</tr>
<tr>
<td>Functions</td>
<td>Quantifiers</td>
<td></td>
</tr>
<tr>
<td><strong>Specification</strong></td>
<td><strong>Refinement</strong></td>
<td><strong>Program Development</strong></td>
</tr>
<tr>
<td><strong>Time Calculation</strong></td>
<td><strong>real time</strong></td>
<td><strong>recursive time</strong></td>
</tr>
<tr>
<td><strong>Space Calculation</strong></td>
<td><strong>maximum space</strong></td>
<td><strong>average space</strong></td>
</tr>
<tr>
<td>assertions</td>
<td>exact precondition</td>
<td>exact postcondition, invariant</td>
</tr>
<tr>
<td>Scope</td>
<td>variable declaration</td>
<td>frame</td>
</tr>
<tr>
<td>Data Structures</td>
<td>array element assignment</td>
<td></td>
</tr>
<tr>
<td>Control Structures</td>
<td><strong>while-loop</strong></td>
<td>loop with <strong>exit</strong>, <strong>for-loop</strong></td>
</tr>
</tbody>
</table>
### Review

<table>
<thead>
<tr>
<th><strong>Binary Theory</strong></th>
<th><strong>Number Theory</strong></th>
<th><strong>Character Theory</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bunches</strong></td>
<td><strong>Sets</strong></td>
<td><strong>Strings</strong></td>
</tr>
<tr>
<td><strong>Functions</strong></td>
<td><strong>Quantifiers</strong></td>
<td><strong>Lists</strong></td>
</tr>
<tr>
<td><strong>Specification</strong></td>
<td><strong>Refinement</strong></td>
<td><strong>Program Development</strong></td>
</tr>
<tr>
<td><strong>Time Calculation</strong></td>
<td><strong>real time</strong></td>
<td><strong>recursive time</strong></td>
</tr>
<tr>
<td><strong>Space Calculation</strong></td>
<td><strong>maximum space</strong></td>
<td><strong>average space</strong></td>
</tr>
<tr>
<td><strong>assertions</strong></td>
<td><strong>exact precondition</strong></td>
<td><strong>exact postcondition</strong></td>
</tr>
<tr>
<td></td>
<td><strong>invariant</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Scope</strong></td>
<td><strong>variable declaration</strong></td>
<td><strong>frame</strong></td>
</tr>
<tr>
<td><strong>Data Structures</strong></td>
<td><strong>array element assignment</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Control Structures</strong></td>
<td><strong>while-loop</strong></td>
<td><strong>loop with exit</strong></td>
</tr>
<tr>
<td></td>
<td><strong>for-loop</strong></td>
<td></td>
</tr>
</tbody>
</table>
Review

Time Dependence

Assertions

Subprograms

Probabilistic Programming

Functional Programming

Recursive Data Definition

Recursive Program Definition

Theory Design and Implementation

Data Transformation

Concurrent Composition

Interactive Variables

wait

backtracking

function

procedure

random number generator

refinement

timing

construction

induction

construction

induction

data theory

program theory

sequential to concurrent transformation

Communication Channels
Review

**Time Dependence**
- wait

**Assertions**
- backtracking

**Subprograms**
- function
- procedure

**Probabilistic Programming**
- random number generator

**Functional Programming**
- refinement
- timing

- construction
- induction

**Recursive Data Definition**
- construction
- induction

**Recursive Program Definition**
- construction
- induction

**Theory Design and Implementation**
- data theory
- program theory

**Data Transformation**

**Concurrent Composition**
- sequential to concurrent transformation

**Interactive Variables**
- Communication Channels
# Review

<table>
<thead>
<tr>
<th>Time Dependence</th>
<th>wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assertions</td>
<td>backtracking</td>
</tr>
<tr>
<td>Subprograms</td>
<td>function procedure</td>
</tr>
<tr>
<td>Probabilistic Programming</td>
<td>random number generator</td>
</tr>
<tr>
<td>Functional Programming</td>
<td>refinement timing</td>
</tr>
<tr>
<td><strong>Recursive Data Definition</strong></td>
<td>construction induction</td>
</tr>
<tr>
<td><strong>Recursive Program Definition</strong></td>
<td>construction induction</td>
</tr>
<tr>
<td>Theory Design and Implementation</td>
<td>data theory program theory</td>
</tr>
<tr>
<td>Data Transformation</td>
<td></td>
</tr>
<tr>
<td>Concurrent Composition</td>
<td>sequential to concurrent transformation</td>
</tr>
<tr>
<td>Interactive Variables</td>
<td>Communication Channels</td>
</tr>
</tbody>
</table>
**Review**

Time Dependence
- wait

Assertions
- backtracking

Subprograms
- function
- procedure

Probabilistic Programming
- random number generator

Functional Programming
- refinement
- timing

Recursive Data Definition
- construction
- induction

Recursive Program Definition
- construction
- induction

**Theory Design and Implementation**

- data theory
- program theory

Data Transformation

Concurrent Composition
- sequential to concurrent transformation

Interactive Variables
- Communication Channels
Review

Time Dependence
Assertions
Subprograms
Probabilistic Programming
Functional Programming
Recursive Data Definition
Recursive Program Definition
Theory Design and Implementation

Data Transformation
Concurrent Composition
Interactive Variables

wait
backtracking
function procedure
random number generator
refinement timing
construction induction
construction induction
data theory program theory
sequential to concurrent transformation
Communication Channels
Review

Time Dependence
Assertions
Subprograms
Probabilistic Programming
Functional Programming
Recursive Data Definition
Recursive Program Definition
Theory Design and Implementation
Data Transformation

**Concurrent Composition**
Sequential to concurrent transformation

Interactive Variables
Communication Channels
Review

Time Dependence
Assertions
Subprograms
Probabilistic Programming
Functional Programming
Recursive Data Definition
Recursive Program Definition
Theory Design and Implementation
Data Transformation
Concurrent Composition
Interactive Variables

wait
backtracking
function procedure
random number generator
refinement timing
construction induction
construction induction
data theory program theory
sequential to concurrent transformation

Communication Channels
Disjoint Composition

Concurrent composition $P \parallel Q$ requires that $P$ and $Q$ have no variables in common, although each can make use of the initial values of the other's variables by making a private copy. An alternative, let's say disjoint composition, is to allow both $P$ and $Q$ to use all the variables with no restrictions, and then to choose disjoint sets of variables $v$ and $w$ and define

$$P \mid v \mid w \mid Q = (P. v' = v) \land (Q. w' = w)$$
Disjoint Composition

Concurrent composition $P\parallel Q$ requires that $P$ and $Q$ have no variables in common, although each can make use of the initial values of the other's variables by making a private copy. An alternative, let's say disjoint composition, is to allow both $P$ and $Q$ to use all the variables with no restrictions, and then to choose disjoint sets of variables $v$ and $w$ and define

$$P \upharpoonright v \upharpoonright w \parallel Q = (P. v' = v) \land (Q. w' = w)$$
Disjoint Composition

Concurrent composition $P||Q$ requires that $P$ and $Q$ have no variables in common, although each can make use of the initial values of the other's variables by making a private copy. An alternative, let's say disjoint composition, is to allow both $P$ and $Q$ to use all the variables with no restrictions, and then to choose disjoint sets of variables $v$ and $w$ and define

$$P |v|w| Q = (P. \ v' = v) \land (Q. \ w' = w)$$
Concurrent composition \( P \parallel Q \) requires that \( P \) and \( Q \) have no variables in common, although each can make use of the initial values of the other's variables by making a private copy. An alternative, let's say disjoint composition, is to allow both \( P \) and \( Q \) to use all the variables with no restrictions, and then to choose disjoint sets of variables \( v \) and \( w \) and define

\[
P \mid_v \mid_w Q = (P. \ v' = v) \land (Q. \ w' = w)
\]

(a) Prove that if \( P \) and \( Q \) are implementable specifications, then \( P \mid_v \mid_w Q \) is implementable.
Disjoint Composition

Concurrent composition $P||Q$ requires that $P$ and $Q$ have no variables in common, although each can make use of the initial values of the other's variables by making a private copy. An alternative, let's say disjoint composition, is to allow both $P$ and $Q$ to use all the variables with no restrictions, and then to choose disjoint sets of variables $v$ and $w$ and define

$$P|v|w|Q = (P. v'=v) \land (Q. w'=w)$$

(a) Prove that if $P$ and $Q$ are implementable specifications, then $P|v|w|Q$ is implementable.

Application Law $\langle v \cdot b \rangle a = (\text{substitute} \ a \ \text{for} \ v \ \text{in} \ b)$
Disjoint Composition

Concurrent composition $P \parallel Q$ requires that $P$ and $Q$ have no variables in common, although each can make use of the initial values of the other's variables by making a private copy. An alternative, let's say disjoint composition, is to allow both $P$ and $Q$ to use all the variables with no restrictions, and then to choose disjoint sets of variables $v$ and $w$ and define

$$P \mid v \mid w \mid Q = (P. \ v' = v) \land (Q. \ w' = w)$$

(a) Prove that if $P$ and $Q$ are implementable specifications, then $P \mid v \mid w \mid Q$ is implementable.

Application Law $\langle v \cdot b \rangle a = (\text{substitute } a \text{ for } v \text{ in } b)$

Let the remaining variables (if any) be $x$. 
Disjoint Composition

$P. \; v' = v$
P. \( v' = v \)

expand sequential composition

\[
= \exists v'', w'', x'' \cdot \langle v', w', x' \cdot P \rangle v'' w'' x'' \land v' = v''
\]
Disjoint Composition

\[ P. \ v' = v \]

\[ = \exists v'', w'', x'' \cdot \langle v', w', x' \cdot P \rangle v'' w'' x'' \land v' = v'' \]

expand sequential composition

\[ = \exists w'', x'' \cdot \langle v', w', x' \cdot P \rangle v' w'' x'' \]

one-point \ v''
Disjoint Composition

\[ P. \ v' = v \]

\[ = \ \exists v'', w'', x''. \langle v', w', x' \cdot P \rangle v'' w'' x'' \quad \land \quad v' = v'' \quad \text{expand sequential composition} \]

\[ = \ \exists w'', x''. \langle v', w', x' \cdot P \rangle v' w'' x'' \quad \text{one-point } v'' \]

\[ = \ \exists w'', x''. \langle v', w', x' \cdot P \rangle v' w'' x'' \quad \text{rename } w'', x'' \text{ to } w', x' \]
Disjoint Composition

\[ P \cdot v' = v \]

\[ = \exists v'', w'', x'' \cdot \langle v', w', x' \cdot P \rangle v'' w'' x'' \land v' = v'' \]

expand sequential composition

one-point \( v'' \)

\[ = \exists w'', x'' \cdot \langle v', w', x' \cdot P \rangle v' w'' x'' \]

rename \( w'', x'' \) to \( w', x' \)
Disjoint Composition

\[ P. \quad v' = v \]

\[ = \exists v'', w'' , x'' . \langle v', w', x' . P \rangle v'' w'' x'' \wedge v' = v'' \quad \text{expand sequential composition} \]

\[ = \exists w'', x'' . \langle v', w', x' . P \rangle v' w'' x'' \quad \text{one-point } v'' \]

\[ = \exists w' , x' . \langle v', w', x' . P \rangle v' w' x' \quad \text{rename } w'', x'' \text{ to } w', x' \]
Disjoint Composition

\[ P. \ v' = v \]

\[ = \ \exists v'', w'', x'' \cdot \langle v', w', x' \cdot P \rangle v'' w'' x'' \land v' = v'' \]

expand sequential composition

one-point \ v''

\[ = \ \exists w'', x'' \cdot \langle v', w', x' \cdot P \rangle v' w'' x'' \]

rename \ w'', x'' to \ w', x'

\[ = \ \exists w', x' \cdot \langle v', w', x' \cdot P \rangle v' w' x' \]

apply

\[ = \ \exists w', x' \cdot P \]
Disjoint Composition

\[ P. \ v' = v \]

\[ = \exists v'', w'', x''. \langle v', w', x' \cdot P \rangle v'' w'' x'' \land v' = v'' \]

expand sequential composition

\[ = \exists w'', x''. \langle v', w', x' \cdot P \rangle v' w'' x'' \]

one-point \( v'' \)

\[ = \exists w', x'. \langle v', w', x' \cdot P \rangle v' w' x' \]

rename \( w'', x'' \) to \( w', x' \)

\[ = \exists w', x'. P \]

apply

\[ Q. \ w' = w \]

\[ = \exists v', x'. Q \]
Disjoint Composition

\[ P. \quad v' = v \]
\[ = \exists v'', w'', x'' \cdot \langle v', w', x' \cdot P \rangle v'' w'' x'' \wedge v' = v'' \quad \text{expand sequential composition} \]
\[ = \exists w'', x'' \cdot \langle v', w', x' \cdot P \rangle v' w'' x'' \quad \text{one-point } v'' \]
\[ = \exists w', x' \cdot \langle v', w', x' \cdot P \rangle v' w' x' \quad \text{rename } w'', x'' \text{ to } w', x' \]
\[ = \exists w', x' \cdot P \quad \text{apply} \]

\[ Q. \quad w' = w \]
\[ = \exists v', x' \cdot Q \]

\[ P \mid v \mid w \mid Q \]
Disjoint Composition

\[ P. \ v' = v \]

\[ = \ \exists v'', w'', x''. \langle v', w', x' \cdot P \rangle v'' w'' x'' \land v' = v'' \]

expand sequential composition  

one-point \ v''

\[ = \ \exists w'', x''. \langle v', w', x' \cdot P \rangle v' w'' x'' \]

rename \ w'', x'' to \ w', x'

\[ = \ \exists w', x'. \langle v', w', x' \cdot P \rangle v' w' x' \]

apply

\[ = \ \exists w', x'. P \]

\[ Q. \ w' = w \]

\[ = \ \exists v', x'. Q \]

\[ P \mid v \mid w \mid Q = (P. \ v' = v) \land (Q. \ w' = w) \]
**Disjoint Composition**

\[ P. \ v' = v \quad \text{expand sequential composition} \]

\[ = \exists v'', w'', x'' \cdot \langle v', w', x' \cdot P \rangle v'' w'' x'' \land v' = v'' \quad \text{one-point } v'' \]

\[ = \exists w'', x'' \cdot \langle v', w', x' \cdot P \rangle v' w'' x'' \quad \text{rename } w'', x'' \text{ to } w', x' \]

\[ = \exists w', x' \cdot \langle v', w', x' \cdot P \rangle v' w' x' \quad \text{apply} \]

\[ = \exists w', x' \cdot P \]

**Q.** \( w' = w \)

\[ = \exists v', x' \cdot Q \]

\[ P \mid v \mid w \mid Q = (P. \ v' = v) \land (Q. \ w' = w) = (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q) \]
Disjoint Composition

\[ P \mid v \mid w \mid Q \text{ is implementable} \]
Disjoint Composition

\[( P |v|w| Q \text{ is implementable}) \]

\[= \forall v, w, x \cdot \exists v', w', x' \cdot P |v|w| Q \]

definition of implementable
Disjoint Composition

\[
\begin{align*}
(P \mid v\mid w \mid Q \text{ is implementable}) & \quad \text{definition of implementable} \\
= & \quad \forall v, w, x \cdot \exists v', w', x' \cdot P \mid v\mid w \mid Q \quad \text{use previous result} \\
= & \quad \forall v, w, x \cdot \exists v', w', x' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q)
\end{align*}
\]
Disjoint Composition

\[(P \mid v \mid w \mid Q \text{ is implementable})\]  

\[= \forall v, w, x \cdot \exists v', w', x' \cdot P \mid v \mid w \mid Q\]  

\[= \forall v, w, x \cdot \exists v', w', x' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q)\]  

\[\text{definition of implementable}\]  

\[\text{use previous result}\]
Disjoint Composition

\[(P \mid v \mid w \mid Q \text{ is implementable}) \quad \text{definition of implementable}\]

\[= \forall v, w, x \cdot \exists v', w', x' \cdot P \mid v \mid w \mid Q \quad \text{use previous result}\]

\[= \forall v, w, x \cdot \exists v', w', x' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q)\]
Disjoint Composition

\[
(P | v | w | Q \text{ is implementable})
\]

\[= \forall v, w, x \cdot \exists v', w', x' \cdot P | v | w | Q \quad \text{definition of implementable}
\]

\[= \forall v, w, x \cdot \exists v', w', x' \cdot P | v | w | Q \quad \text{use previous result}
\]

\[= \forall v, w, x \cdot \exists v', w', x' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q) \quad \text{identity for } x'
\]

\[= \forall v, w, x \cdot \exists v', w' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q)
\]
Disjoint Composition

\[(P | v|w| Q \text{ is implementable})\]

\[= \forall v, w, x \cdot \exists v', w', x' \cdot P | v|w| Q\]

\[= \forall v, w, x \cdot \exists v', w', x' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q)\]

\[= \forall v, w, x \cdot \exists v', w' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q)\]

\[\uparrow \uparrow\]
Disjoint Composition

\( (P \mid v \mid w \mid Q \text{ is implementable}) \)  

\[ = \forall v, w, x \cdot \exists v', w', x' \cdot P \mid v \mid w \mid Q \]  

definition of implementable  

use previous result  

identity for \( x' \)

\[ = \forall v, w, x \cdot \exists v', w', (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q) \]  

\[ = \forall v, w, x \cdot \exists v', (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q) \]  

\[ = \forall v, w, x \cdot \exists v' \cdot \exists w' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q) \]
Disjoint Composition

\( (P \mid v \mid w \mid Q \text{ is implementable}) \quad \text{definition of implementable} \)

\[
\begin{align*}
    &= \forall v, w, x \cdot \exists v', w', x' \cdot P \mid v \mid w \mid Q \\
    &= \forall v, w, x \cdot \exists v', w', x' \cdot (\exists w', x' \cdot P) \wedge (\exists v', x' \cdot Q) \\
    &= \forall v, w, x \cdot \exists v', w' \cdot (\exists w', x' \cdot P) \wedge (\exists v', x' \cdot Q) \\
    &= \forall v, w, x \cdot \exists v' \cdot \exists w' \cdot (\exists w', x' \cdot P) \wedge (\exists v', x' \cdot Q)
\end{align*}
\]
Disjoint Composition

\[
(P \mid v \mid w \mid Q \text{ is implementable}) = \forall v, w, x \cdot \exists v', w', x' \cdot P \mid v \mid w \mid Q
\]

use previous result

\[
= \forall v, w, x \cdot \exists v', w', x' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q)
\]

identity for \( x' \)

\[
= \forall v, w, x \cdot \exists v', w' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q)
\]

distribution (factoring)

\[
= \forall v, w, x \cdot \exists v' \cdot (\exists w', x' \cdot P) \land (\exists w' \cdot \exists v', x' \cdot Q)
\]
Disjoint Composition

\( (P \mid v \mid w \mid Q \text{ is implementable}) \)

= \( \forall v, w, x \cdot \exists v', w', x' \cdot P \mid v \mid w \mid Q \) \hspace{1cm} \text{definition of implementable}

= \( \forall v, w, x \cdot \exists v', w', x' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q) \) \hspace{1cm} \text{use previous result}

= \( \forall v, w, x \cdot \exists v', w' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q) \) \hspace{1cm} \text{identity for } x'

= \( \forall v, w, x \cdot \exists v' \cdot \exists w' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q) \) \hspace{1cm} \text{distribution (factoring)}

= \( \forall v, w, x \cdot \exists v' \cdot (\exists w', x' \cdot P) \land (\exists w' \cdot \exists v', x' \cdot Q) \)
Disjoint Composition

\( (P \mid v \mid w \mid Q \text{ is implementable}) \)
definition of implementable

\[ = \forall v, w, x \cdot \exists v', w', x' \cdot P \mid v \mid w \mid Q \]
use previous result

\[ = \forall v, w, x \cdot \exists v', w', x' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q) \]
identity for \( x' \)

\[ = \forall v, w, x \cdot \exists v', w' \cdot (\exists v', x' \cdot P) \land (\exists v', x' \cdot Q) \]
distribution (factoring)

\[ = \forall v, w, x \cdot \exists v' \cdot (\exists w', x' \cdot P) \land (\exists w' \cdot \exists v', x' \cdot Q) \]
distribution (factoring)

\[ = \forall v, w, x \cdot (\exists v' \cdot \exists w', x' \cdot P) \land (\exists w' \cdot \exists v', x' \cdot Q) \]
Disjoint Composition

\[(P |v|w|Q \text{ is implementable})\]

\[= \forall v, w, x \cdot \exists v', w', x' \cdot P |v|w|Q\]

\[= \forall v, w, x \cdot \exists v', w', x' \cdot (\exists w', x' \cdot P) \wedge (\exists v', x' \cdot Q)\]

\[= \forall v, w, x \cdot \exists v'. w' \cdot (\exists w', x' \cdot P) \wedge (\exists v', x' \cdot Q)\]

\[= \forall v, w, x \cdot \exists v'. (\exists w', x' \cdot P) \wedge (\exists v', x' \cdot Q)\]

\[= \forall v, w, x \cdot (\exists v' \cdot \exists w', x' \cdot P) \wedge (\exists v' \cdot w', x' \cdot Q)\]

\[= \forall v, w, x \cdot (\exists v', w', x' \cdot P) \wedge (\exists v', w', x' \cdot Q)\]

\[= \forall v, w, x \cdot (\exists v', w', x' \cdot P) \wedge (\exists v', w', x' \cdot Q)\]

\[\text{definition of implementable}\]

\[\text{use previous result}\]

\[\text{identity for } x'\]

\[\text{distribution (factoring)}\]

\[\text{distribution (factoring)}\]

\[\text{distribution (factoring)}\]
Disjoint Composition

\[(P \mid v \mid w \mid Q \text{ is implementable})\]

\[= \forall v, w, x \cdot \exists v', w', x' \cdot P \mid v \mid w \mid Q\]  
\[\text{definition of implementable}\]

\[= \forall v, w, x \cdot \exists v', w', x' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q)\]  
\[\text{use previous result}\]

\[= \forall v, w, x \cdot \exists v', w' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q)\]  
\[\text{identity for } x'\]

\[= \forall v, w, x \cdot \exists v' \cdot \exists w' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q)\]  
\[\text{distribution (factoring)}\]

\[= \forall v, w, x \cdot \exists v' \cdot (\exists w', x' \cdot P) \land (\exists w' \cdot \exists v', x' \cdot Q)\]  
\[\text{distribution (factoring)}\]

\[= \forall v, w, x \cdot (\exists v' \cdot \exists w', x' \cdot P) \land (\exists w' \cdot \exists v', x' \cdot Q)\]  

\[= \forall v, w, x \cdot (\exists v', w', x' \cdot P) \land (\exists v', w', x' \cdot Q)\]  
\[\text{splitting law}\]

\[= (\forall v, w, x \cdot \exists v', w', x' \cdot P) \land (\forall v, w, x \cdot \exists v', w', x' \cdot Q)\]
Disjoint Composition

\[( P \mid v \mid w \mid Q \text{ is implementable}) \]

\[= \forall v, w, x \cdot \exists v', w', x' \cdot P \mid v \mid w \mid Q \]

\[= \forall v, w, x \cdot \exists v', w', x' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q) \]

\[= \forall v, w, x \cdot \exists v', w' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q) \]

\[= \forall v, w, x \cdot \exists v' \cdot \exists w' \cdot (\exists w', x' \cdot P) \land (\exists v', x' \cdot Q) \]

\[= \forall v, w, x \cdot (\exists v' \cdot \exists w', x' \cdot P) \land (\exists v' \cdot \exists w', x' \cdot Q) \]

\[= \forall v, w, x \cdot (\exists v', w', x' \cdot P) \land (\exists v', w', x' \cdot Q) \]

\[= \forall v, w, x \cdot (\exists v', w', x' \cdot P) \land (\forall v, w, x \cdot \exists v', w', x' \cdot Q) \]

\[= (\forall v, w, x \cdot \exists v', w', x' \cdot P) \land (\forall v, w, x \cdot \exists v', w', x' \cdot Q) \]

\[= ( P \text{ is implementable}) \land ( Q \text{ is implementable}) \]
Disjoint Composition

Concurrent composition $P \parallel Q$ requires that $P$ and $Q$ have no variables in common, although each can make use of the initial values of the other's variables by making a private copy. An alternative, let's say disjoint composition, is to allow both $P$ and $Q$ to use all the variables with no restrictions, and then to choose disjoint sets of variables $v$ and $w$ and define

$$P \mid v \mid w \mid Q = (P. v' = v) \land (Q. w' = w)$$

(b) Describe how $P \mid v \mid w \mid Q$ can be executed.
Disjoint Composition

Concurrent composition $P||Q$ requires that $P$ and $Q$ have no variables in common, although each can make use of the initial values of the other's variables by making a private copy. An alternative, let's say disjoint composition, is to allow both $P$ and $Q$ to use all the variables with no restrictions, and then to choose disjoint sets of variables $v$ and $w$ and define

$$P \mid v \mid w \mid Q = (P. \ v' = v) \land (Q. \ w' = w)$$

(b) Describe how $P \mid v \mid w \mid Q$ can be executed.

Make a copy of all variables. Execute $P$ using the original set of variables and in parallel execute $Q$ using the copies. Then copy back from the copy $w$ to the original $w$. Then throw away the copies.
Disjoint Composition

Concurrent composition $P \parallel Q$ requires that $P$ and $Q$ have no variables in common, although each can make use of the initial values of the other's variables by making a private copy. An alternative, let's say disjoint composition, is to allow both $P$ and $Q$ to use all the variables with no restrictions, and then to choose disjoint sets of variables $v$ and $w$ and define

$$P |v|w|Q = (P. \; v' = v) \land (Q. \; w' = w)$$

(b) Describe how $P |v|w|Q$ can be executed.

$$P |v|w|Q \iff \text{var } cv := v \cdot \text{var } cw := w \cdot \text{var } cx := x \cdot (P \parallel \langle v, w, x, v', w', x' \cdot Q \rangle_{cv \; cw \; cx \; cv' \; cw' \; cx'}). \; w := cw$$