Structured Concurrent Programming

Jayadev Misra

Department of Computer Science
University of Texas at Austin

Email: misra@cs.utexas.edu
web: http://www.cs.utexas.edu/users/psp

Collaborators: William Cook, David Kitchin
Example: Airline

- Contact two airlines simultaneously for price quotes.
- Buy ticket from either airline if its quote is at most $300.
- Buy the cheapest ticket if both quotes are above $300.
- Buy any ticket if the other airline does not provide a timely quote.
- Notify client if neither airline provides a timely quote.
Wide-area Computing

Acquire data from remote services.
Calculate with these data.
Invoke yet other remote services with the results.

Additionally
Invoke alternate services for failure tolerance.
Repeatedly poll a service.
Ask a service to notify the user when it acquires the appropriate data.
Download an application and invoke it locally.
Have a service call another service on behalf of the user.
The Nature of Distributed Applications

Three major components in distributed applications:

Persistent storage management

databases by the airline and the hotels.

Specification of sequential computational logic

does ticket price exceed $300?

Methods for orchestrating the computations

We look at only the third problem.
Overview of Orc

- Orchestration language.
  - Invoke services by calling sites
  - Manage time-outs, priorities, and failures

- A Program execution
  - calls sites,
  - publishes values.

- Simple
  - Language has only 3 combinators.
  - Semantics described by labeled transition system and traces.
  - Combinators are (monotonic and) continuous.
Structure of Orc Expression

- **Simple**: just a site call, \( CNN(d) \)
  
  Publishes the value returned by the site.

- **composition** of two Orc expressions:

  do \( f \) and \( g \) in parallel \( f \mid g \)  
  for all \( x \) from \( f \) do \( g \) \( f \bowtie x \bowtie g \)  
  for some \( x \) from \( g \) do \( f \) \( f \text{ where } x \in g \)  
  
  Symmetric composition  
  Piping  
  Asymmetric composition
Symmetric composition: $f \mid g$

$CNN \mid BBC$: calls both $CNN$ and $BBC$ simultaneously.
Publishes values returned by both sites. (0, 1 or 2 values)

- Evaluate $f$ and $g$ independently.
- Publish all values from both.
- No direct communication or interaction between $f$ and $g$. They may communicate only through sites.
Pipe: \( f > x > g \)

For all values published by \( f \) do \( g \). Publish only the values from \( g \).

- \( CNN > x > Email(address, x) \)
  
  Call \( CNN \). Bind result (if any) to \( x \). Call \( Email(address, x) \).
  
  Publish the value, if any, returned by \( Email \).

- \( (CNN | BBC') > x > Email(address, x) \)

  May call \( Email \) twice. Publishes up to two values from \( Email \).
Write $f \gg g$ for $f > x > g$ if $x$ unused in $g$.

Precedence: $f > x > g \mid h > y > u$

$(f > x > g) \mid (h > y > u)$
Figure 1: Schematic of piping
Asymmetric parallel composition: \((f \text{ where } x : \in g)\)

For some value published by \(g\) do \(f\). Publish only the values from \(f\).

\[ Email(address, x) \text{ where } x : \in (CNN \mid BBC) \]

Binds \(x\) to the first value from \(CNN \mid BBC\).

- Evaluate \(f\) and \(g\) in parallel.
  Site calls that need \(x\) are suspended; other site calls proceed.
  \[(M \mid N(x)) \text{ where } x : \in g\]

- When \(g\) returns a value, assign it to \(x\) and terminate \(g\).
  Resume suspended calls.

- Values published by \(f\) are the values of \((f \text{ where } x : \in g)\).
Some Fundamental Sites

0: never responds.

\( \text{let}(x, y, \ldots) \): returns a tuple of its argument values.

\( \text{if}(b) \): boolean \( b \),
returns a signal if \( b \) is true; remains silent if \( b \) is false.

\( \text{Signal} \) returns a signal immediately. Same as \( \text{if}(\text{true}) \).

\( \text{Rtimer}(t) \): integer \( t \), \( t \geq 0 \), returns a signal \( t \) time units later.
Centralized Execution Model

- An expression is evaluated on a single machine (client).
- Client communicates with sites by messages.
- All fundamental sites are local to the client. All except *Rtimer* respond immediately.
- Concurrent and distributed executions are derived from an expression.
Expression Definition

\[ \text{MailOnce}(a) \triangleq \text{Email}(a, m) \text{ where } m : \in (\text{CNN} \mid \text{BBC}) \]

\[ \text{MailLoop}(a, t) \triangleq \text{MailOnce}(a) \gg \text{Rtimer}(t) \gg \text{MailLoop}(a, t) \]

- Expression is called like a procedure. May publish many values. \textit{MailLoop} does not publish a value.

- Site calls are strict; expression calls non-strict.
**Metronome**

Publish a signal at every time unit.

\[ \text{Metronome} \triangleright Signal \mid (R\text{timer}(1) \gg \text{Metronome}) \]

Publish \( n \) signals.

\[
\begin{align*}
BM(0) \triangleright 0 \\
BM(n) \triangleright Signal \mid (R\text{timer}(1) \gg BM(n - 1))
\end{align*}
\]
Example of Expression call

- Site $Query$ returns a value (different ones at different times).

- Site $Accept(x)$ returns $x$ if $x$ is acceptable; it is silent otherwise.

- Produce all acceptable values by calling $Query$ at unit intervals forever.

\[ \text{Metronome} \Rightarrow Query \succ x \succ Accept(x) \]
Publish $M$’s response if it arrives before $t$, and 0 otherwise.

\[
\begin{align*}
let(z) \\
&\text{where} \\
&z : \in \\
&M \\
&| \text{Rtimer}(t) \Rightarrow let(0)
\end{align*}
\]
Recursive definition with time-out

Call a list of sites.

Count the number of responses received within 10 time units.

\[
tally([], \triangleq \text{let}(0))
\]
\[
tally(M : MS), \triangleq u + v
\]

where
\[
u : \in (M \Rightarrow \text{let}(1)) \mid (\text{Rtimer}(10) \Rightarrow \text{let}(0))
\]
\[
v : \in tally(MS)
\]
Fork-join parallelism

Call \( M \) and \( N \) in parallel.

Return their values as a tuple after both respond.

\[
\text{let}(u, v) \\
\text{where } u \in M \\
v \in N
\]

This stands for:

\[
(\text{let}(u, v) \\
\text{where } u \in M) \\
\text{where } v \in N
\]
Barrier Synchronization in $M \gg f \mid N \gg g$

$f$ and $g$ start only after both $M$ and $N$ complete.

\[
( \text{let}(u, v) \\
\text{where} \ u \in M \\
v \in N )
\gg (f \mid g)
\]
In CCS/ Pi-Calculus: \( \alpha.P + \beta.Q \)

In Orc:

\[
if(b) \Rightarrow P \mid if(\neg b) \Rightarrow Q \\
\text{where} \\
\quad b \in (\text{Alpha} \Rightarrow \text{let(true)}) \mid (\text{Beta} \Rightarrow \text{let(false)})
\]

Orc does not permit non-deterministic internal choice.
Publish $N$'s response asap, but no earlier than 1 unit from now.

\[ \text{Delay} \triangleq (\text{Rtimer}(1) \gg \text{let}(u)) \text{ where } u \in N \]

Call $M$, $N$ together.

If $M$ responds within one unit, take its response.

Else, pick the first response.

\[ \text{let}(x) \text{ where } x \in (M \mid \text{Delay}) \]
Evaluation of \( f \) can not be directly interrupted.

Introduce two sites:

- \texttt{Interrupt.set}: to interrupt \( f \)
- \texttt{Interrupt.get}: responds after \texttt{Interrupt.set} has been called.

Instead of \( f \), evaluate

\[
\textit{let}(z) \text{ where } z : \varepsilon (f \mid \texttt{Interrupt.get})
\]
Parallel or

Sites \( M \) and \( N \) return booleans. Compute their parallel or.

\[ \text{ift}(b) \oplus \text{if}(b) \Rightarrow \text{let}(\text{true}) : \text{returns true if } b \text{ is true; silent otherwise.} \]

\[
\text{ift}(x) \mid \text{ift}(y) \mid \text{or}(x, y) \\
\text{where} \\
x \in M, \ y \in N
\]

To return just one value:

\[
\text{let}(z) \\
\text{where} \\
z \in \text{ift}(x) \mid \text{ift}(y) \mid \text{or}(x, y) \\
x \in M \\
y \in N
\]
Airline quotes: Application of Parallel or

Contact airlines $A$ and $B$.

Return any quote if it is below $c$ as soon as it is available, otherwise return the minimum quote.

$\text{threshold}(x)$ returns $x$ if $x < c$; silent otherwise.
$\text{Min}(x, y)$ returns the minimum of $x$ and $y$.

\[
\text{let}(z) \\
\text{where} \\
z \in \text{threshold}(x) \mid \text{threshold}(y) \mid \text{Min}(x, y) \\
x \in A \\
y \in B
\]
Sequential Computing

- $(S; T) \text{ is } (S \Rightarrow T)$
- if $b$ then $S$ else $T$
  
  is
  
  $\text{if}(b) \Rightarrow S \mid \text{if}(\neg b) \Rightarrow T$
- while $B(x)$ do $x := S(x)$

\[
\text{loop}(x) \triangleq 
B(x) \triangleright b \triangleright \text{if}(b) \Rightarrow S(x) \triangleright y \triangleright \text{loop}(y) \mid \text{if}(\neg b) \Rightarrow \text{let}(x))
\]
Angelica vs. Demonic non-determinism

- for all \( x \) from \( f \) do \( g \): implements angelic non-determinism.
  All paths of computation are explored.

- for some \( x \) from \( f \) do \( g \): implements demonic non-determinism.
  Some selected path of computation is explored.
Backtracking: Eight queens

Figure 2: Backtrack Search for Eight queens
Eight queens; contd.

- **Configuration**: placement of queens in the last $i$ rows. Represented by a list of $i$ values from 0..7.

- **Valid configuration**: no queen captures another. 
  
  $valid(z)$ returns $z$ if configuration $z$ is valid; silent otherwise.

- **Produce all** valid extensions of $z$ by placing $n$ additional queens:

  $$extend(z, 1) \triangleq valid(0; z) \mid valid(1; z) \mid \cdots \mid valid(7; z)$$

  $$extend(z, n) \triangleq extend(z, 1) > y > extend(y, n - 1)$$

- **Solve the original problem by calling** $extend([], 8)$. 
Processes

- Processes typically communicate via channels.
- For channel \( c \), treat \( c.put \) and \( c.get \) as site calls.
- In our examples, \( c.get \) is blocking and \( c.put \) is non-blocking.
- Other kinds of channels can be programmed as sites.
Typical Iterative Process

Forever: Read $x$ from channel $c$, compute with $x$, output result on $e$:

$$P(c, e) \triangleq c\text{.get} \triangleright x \triangleright Compute(x) \triangleright y \triangleright e\text{.put}(y) \triangleright P(c, e)$$

Process (network) to read from both $c$ and $d$ and write on $e$:

$$Net(c, d, e) \triangleq P(c, e) \mid P(d, e)$$
Run a dialog with a child.

**Forever:** child inputs an integer on channel \( p \)

Process outputs \( true \) on channel \( q \) iff the number is prime.

Sites: \( c.get \) and \( c.put \), for channel \( c \).

\( Prime?(x) \) returns \( true \) iff \( x \) is prime.

\[
\text{Dialog}(p, q) \triangleq \\
p.get > x > \\
Primes? (x) > b > \\
q.put (b) \Rightarrow \\
\text{Dialog}(p, q)
\]
Laws of Kleene Algebra

(Zero and \( \mid \) )
(Commutativity of \( \mid \) )
(Associativity of \( \mid \) )
(Idempotence of \( \mid \) )
(Associativity of \( \gg \) )
(Left zero of \( \gg \) )
(Right zero of \( \gg \) )
(Left unit of \( \gg \) )
(Right unit of \( \gg \) )
(Left Distributivity of \( \gg \) over \( \mid \) )
(Right Distributivity of \( \gg \) over \( \mid \) )

\[
\begin{align*}
\text{(Zero and } \mid \text{)} & \quad f \mid 0 = f \\
\text{(Commutativity of } \mid \text{)} & \quad f \mid g = g \mid f \\
\text{(Associativity of } \mid \text{)} & \quad (f \mid g) \mid h = f \mid (g \mid h) \\
\text{(Idempotence of } \mid \text{)} & \quad f \mid f = f \\
\text{(Associativity of } \gg \text{)} & \quad (f \gg g) \gg h = f \gg (g \gg h) \\
\text{(Left zero of } \gg \text{)} & \quad 0 \gg f = 0 \\
\text{(Right zero of } \gg \text{)} & \quad f \gg 0 = 0 \\
\text{(Left unit of } \gg \text{)} & \quad \text{Signal } \gg f = f \\
\text{(Right unit of } \gg \text{)} & \quad f \gg \text{let}(x) = f \\
\text{(Left Distributivity of } \gg \text{ over } \mid \text{)} & \quad f \gg (g \mid h) = (f \gg g) \mid (f \gg h) \\
\text{(Right Distributivity of } \gg \text{ over } \mid \text{)} & \quad (f \mid g) \gg h = (f \gg h \mid g \gg h)
\end{align*}
\]
Laws which do not hold

(Idempotence of $|)$ $\quad f | f = f$
(Right zero of $\gg$) $\quad f \gg 0 = 0$
(Left Distributivity of $\gg$ over $|$) $\quad f \gg (g | h) = (f \gg g) | (f \gg h)$
Additional Laws

(Distributivity over $\gg$ ) if $g$ is $x$-free
\[(f \gg g \text{ where } x: \in h) = (f \text{ where } x: \in h) \gg g\]

(Distributivity over $|$ ) if $g$ is $x$-free
\[(f | g \text{ where } x: \in h) = (f \text{ where } x: \in h) | g\]

(Distributivity over where ) if $g$ is $y$-free
\[
((f \text{ where } x: \in g) \text{ where } y: \in h) =
((f \text{ where } y: \in h) \text{ where } x: \in g)
\]

(Elimination of where) if $f$ is $x$-free, for site $M$
\[(f \text{ where } x: \in M) = f | M \gg 0\]
Rules for Site Call

\[
\frac{u \text{ fresh}}{M(c) \xrightarrow{\text{SITECALL}} M\langle c, u \rangle \xrightarrow{?u}} \quad (\text{SITECALL})
\]

\[
?u \xrightarrow{u?c} \text{let}(c) \quad (\text{SITERET})
\]

\[
\text{let}(c) \xrightarrow{i_c} 0 \quad (\text{LET})
\]
Symmetric Composition

\[
\begin{align*}
  f \xrightarrow{l} f' \\
  f \mid g \xrightarrow{l} f' \mid g
\end{align*}
\]  

(SYM1)

\[
\begin{align*}
  g \xrightarrow{l} g' \\
  f \mid g \xrightarrow{l} f \mid g'
\end{align*}
\]  

(SYM2)
### Sequencing

\[
\begin{align*}
  f & \xrightarrow{l} f' & l \neq {!c} \\
  f \seq x g & \xrightarrow{l} f' \seq x g
\end{align*}
\]

\((\text{SEQ1N})\)

\[
\begin{align*}
  f & \xrightarrow{!c} f' \\
  f \seq x g & \xrightarrow{\tau} (f' \seq x g) \mid [c/x]g
\end{align*}
\]

\((\text{SEQ1V})\)
Asymmetric Composition

\[ \frac{f \xrightarrow{l} f'}{l \neq \neg c} \]
\[ g \text{ where } x : \in f \xrightarrow{l} g \text{ where } x : \in f' \quad (*) \text{(ASYM1N)} \]

\[ \frac{f \xrightarrow{!c} f'}{g \text{ where } x : \in f \xrightarrow{\tau} [c/x]g} \]
\[ (*) \text{(ASYM1V)} \]

\[ \frac{g \xrightarrow{l} g'}{g \text{ where } x : \in f \xrightarrow{l} g' \text{ where } x : \in f} \]
\[ \text{(ASYM2)} \]
Expression Call

\[
\frac{[[ E(q) \mathrel{\Delta} f ]] \in D}{E(p) \xrightarrow{\tau} \left[ p/q \right] f}
\]
\[
\begin{align*}
\text{Rules} & \\
\text{**k** fresh} & \\
M(v) & \xrightarrow{M_k(v)} ?k \\
?k & \xrightarrow{k?v} \text{let}(v) \\
\text{let}(v) & \xrightarrow{!v} 0 \\
f & \xrightarrow{a} f' \\
f & \xrightarrow{\alpha} f' \\
\frac{f | g \xrightarrow{\alpha} f' | g}{g \xrightarrow{\alpha} g'} \\
\frac{[[ E(x) \triangle f ]] \in D}{E(p) \xrightarrow{\tau} [p/x].f} \\
\text{where} \ x : \in g & \xrightarrow{\alpha} f' \text{ where } x : \in g \\
g & \xrightarrow{!v} g' \\
\frac{f \xrightarrow{\alpha} f' \text{ where } x : \in g}{\frac{\text{where } x : \in g \xrightarrow{\tau} [v/x].f}{g \xrightarrow{\alpha} g'} \\
g & \xrightarrow{a} g' \\
\frac{\text{where } x : \in g}{f' \text{ where } x : \in g'} \\
f & \xrightarrow{a \neq !v} f' \text{ where } x : \in g'
\end{align*}
\]
Example

$$\langle (M(x) \mid \text{let}(x)) \rangle \triangleright y \triangleright R(y) \rangle \text{ where } x: \in (N \mid S)$$

$$S_k \{ \text{Call } S; \ S \xrightarrow{S_k} ?k; \ N \mid S \xrightarrow{S_k} N \mid ?k \}$$

$$\langle (M(x) \mid \text{let}(x)) \rangle \triangleright y \triangleright R(y) \rangle \text{ where } x: \in (N \mid ?k)$$

$$N_l \{ \text{Call } N \}$$

$$\langle (M(x) \mid \text{let}(x)) \rangle \triangleright y \triangleright R(y) \rangle \text{ where } x: \in (?l \mid ?k)$$

$$l?5 \{ ?l \xrightarrow{l?5} \text{let}(5); \ ?l \mid ?k \xrightarrow{l?5} \text{let}(5) \mid ?k \}$$

$$\langle (M(x) \mid \text{let}(x)) \rangle \triangleright y \triangleright R(y) \rangle \text{ where } x: \in (\text{let}(5) \mid ?k)$$
Example; contd.

\[ ((M(x) \mid let(x)) \triangleright y \triangleright R(y)) \text{ where } x \in (let(5) \mid ?k) \]

\[ \xrightarrow{\tau} \{ \; let(5) \xrightarrow{!5} 0 ; \; let(5) \mid ?k \xrightarrow{!5} 0 \mid ?k \} \]

\[ (M(5) \mid let(5)) \triangleright y \triangleright R(y) \]

\[ \xrightarrow{\tau} \{ \; let(5) \xrightarrow{!5} 0 ; \; M(5) \mid let(5) \xrightarrow{!5} M(5) \mid 0 ; \; f \xrightarrow{!v} f' \text{ implies } f \triangleright y \triangleright g \xrightarrow{\tau} (f' \triangleright y \triangleright g) \mid [v/y].g \} \]

\[ ((M(5) \mid 0) \triangleright y \triangleright R(y)) \mid R(5) \]

\[ \xrightarrow{R_n(5)} \{ \text{call } R \text{ with argument } (5) \} \]

\[ ((M(5) \mid 0) \triangleright y \triangleright R(y)) \mid ?n \]
Example; contd.

\[
\begin{align*}
((M(5) \mid 0) > y > R(y)) \mid \ ?n \\
\rightarrow^n \{ \ ?n \rightarrow^n \ let(7) \} \\
((M(5) \mid 0) > y > R(y)) \mid let(7) \\
\rightarrow^7 \{ \ f \mid let(7) \rightarrow^7 f \mid 0 \} \\
((M(5) \mid 0) > y > R(y)) \mid 0
\end{align*}
\]

The sequence of events:
\[S_k \ N_i \ l?5 \ \tau \ \tau \ R_n(5) \ n?7 \ \uparrow^7\]

The sequence minus \(\tau\) events:
\[S_k \ N_i \ l?5 \ \ R_n(5) \ n?7 \ \uparrow^7\]
Executions and Traces

Define

\[ f \xrightarrow{\varepsilon} f \]

\[ f \xrightarrow{a} f'', \quad f'' \xrightarrow{s} f' \]

\[ f \xrightarrow{\alpha s} f' \]

- Given \( f \xrightarrow{s} f' \), \( s \) is an execution of \( f \).

- A trace is an execution minus \( \tau \) events.

- The set of executions of \( f \) (and traces) are prefix-closed.
Laws, using strong bisimulation

- $f \mid 0 \sim f$
- $f \mid g \sim g \mid f$
- $f \mid (g \mid h) \sim (f \mid g) \mid h$
- $f >x> (g >y> h) \sim (f >x> g) >y> h$, if $h$ is $x$-free.
- $0 >x> f \sim 0$
- $(f \mid g) >x> h \sim f >x> h \mid g >x> h$
- $(f \mid g)$ where $x:\in h \sim (f$ where $x:\in h) \mid g$, if $g$ is $x$-free.
- $(f >y> g)$ where $x:\in h \sim (f$ where $x:\in h) >y> g$, if $g$ is $x$-free.
- $(f$ where $x:\in g)$ where $y:\in h \sim (f$ where $y:\in h)$ where $x:\in g$, if $g$ is $y$-free, $h$ is $x$-free.
Relation $\sim$ is an equality

Given $f \sim g$, show

1. $\frac{f}{h} \sim \frac{g}{h}$
   $\frac{h}{f} \sim \frac{h}{g}$

2. $\frac{f}{x} \sim \frac{g}{x}$
   $\frac{h}{x} \sim \frac{h}{g}$

3. $\frac{f}{h}$ where $x:\in h$ \sim \frac{g}{h}$ where $x:\in h$
   $\frac{h}{f}$ where $x:\in f$ \sim \frac{h}{g}$ where $x:\in g$
Treatment of Free Variables

Closed expression: No free variable.
Open expression: Has free variable.

- Law $f \sim g$ holds only if both $f$ and $g$ are closed.
  Otherwise: $\text{let}(x) \sim 0$
  But $\text{let}(1) > x > 0 \neq \text{let}(1) > x > \text{let}(x)$

- Then we can't show $\text{let}(x) | \text{let}(y) \sim \text{let}(y) | \text{let}(x)$
Substitution Event

\[ f \xrightarrow{[v/x]} [v/x].f \quad \text{(SUBST)} \]

- Now, \( \text{let}(x) \xrightarrow{[1/x]} \text{let}(1). \)
  
  So, \( \text{let}(x) \neq 0 \)

- Earlier rules apply to base events only.
  
  From \( f \xrightarrow{[v/x]} [v/x].f \), we can not conclude:
  
  \[ f \parallel g \xrightarrow{[v/x]} [v/x].f \parallel g \]
Traces as Denotations

Define Orc combinators over trace sets, \( S \) and \( T \). Define:

\[
S \mid T, \quad S \triangleright x \triangleright T, \quad S \text{ where } x \in T.
\]

Notation: \( \langle f \rangle \) is the set of traces of \( f \).

**Theorem**

\[
\begin{align*}
\langle f \mid g \rangle & = \langle f \rangle \mid \langle g \rangle \\
\langle f \triangleright x \triangleright g \rangle & = \langle f \rangle \triangleright x \triangleright \langle g \rangle \\
\langle f \text{ where } x \in g \rangle & = \langle f \rangle \text{ where } x \in \langle g \rangle
\end{align*}
\]
Expressions are equal if their trace sets are equal

Define: \( f \simeq g \) if \( \langle f \rangle = \langle g \rangle \).

**Theorem** (Combinators preserve \( \simeq \))

Given \( f \simeq g \) and any combinator \( *: f \ast h \simeq g \ast h, \ h \ast f \simeq h \ast g \)

Specifically, given \( f \simeq g \)

1. \( f \upharpoonright h \simeq g \upharpoonright h \)
   \( h \upharpoonright f \simeq h \upharpoonright g \)

2. \( f \upharpoonright x \upharpoonright h \simeq g \upharpoonright x \upharpoonright h \)
   \( h \upharpoonright x \upharpoonright f \simeq h \upharpoonright x \upharpoonright g \)

3. \( f \text{ where } x : \in h \simeq g \text{ where } x : \in h \)
   \( h \text{ where } x : \in f \simeq h \text{ where } x : \in g \)
Monotonicity, Continuity

- Define: $f \sqsubseteq g$ if $\langle f \rangle \sqsubseteq \langle g \rangle$.

Theorem (Monotonicity) Given $f \sqsubseteq g$ and any combinator $*$

$$f \ast h \sqsubseteq g \ast h, \quad h \ast f \sqsubseteq h \ast g$$

- Chain $f: f_0 \sqsubseteq f_1, \ldots f_i \sqsubseteq f_{i+1}, \ldots$

Theorem: $\sqcup (f_i \ast h) \simeq (\sqcup f) \ast h$.

Theorem: $\sqcup (h \ast f_i) \simeq h \ast (\sqcup f)$. 
Least Fixed Point

\[ M \triangleq S \mid R \triangleright M \]

\[ \begin{align*}
M_0 & \simeq 0 \\
M_{i+1} & \simeq S \mid R \triangleright M_i, \quad i \geq 0
\end{align*} \]

\( M \) is the least upper bound of the chain \( M_0 \subseteq M_1 \subseteq \cdots \)
Weak Bisimulation

\[
\begin{align*}
\text{signal} \gg f & \equiv f \\
 f \gg x \gg \text{let}(x) & \equiv f
\end{align*}
\]