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, Ian Wehrman
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 Orchestration language Orc

- A Program execution
  - calls sites, to invoke services.
  - publishes values.

- Orc is simple
  - Language has only 3 combinators.
  - Can handle time-outs, priorities, failures, synchronizations, · · ·
  - 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:
  
  \[
  \begin{align*}
  & \text{do } f \text{ and } g \text{ in parallel} & f \mid g & \text{Symmetric composition} \\
  & \text{for all } x \text{ from } f \text{ do } g & f \gg g & \text{Piping} \\
  & \text{for some } x \text{ from } g \text{ do } f & f \ll g & \text{Asymmetric composition}
  \end{align*}
  \]
Symmetric composition: \( f | g \)

\( CNN \mid BBC \): 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 \( Email \).
Figure 1: Schematic of $f > x > g$
Notation

$f \gg g$ for $f > x > g$, if $x$ unused in $g$.

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

$(f > x > g) \mid (h > y > u)$
Asymmetric parallel composition: \((f \prec x \prec g)\)

For some value published by \(g\) do \(f\).

- Evaluate \(f\) and \(g\) in parallel.
  Site calls that need \(x\) are suspended; other site calls proceed.
  \((M \mid N(x)) \prec x \prec g\)

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

- Values published by \(f\) are the values of \((f \prec x \prec g)\).

  \(Email(address, x) \prec x \prec (CNN \mid BBC)\)

Binds \(x\) to the first value from \(CNN \midBBC\). Sends at most one email.
Some Fundamental Sites

0: never responds.

\( \text{let}(x, y, \cdots) \): 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

\[ \begin{align*}
\text{MailOnce}(a) & \triangleq \\
\text{Email}(a, m) & \prec m \prec (\text{CNN} \mid \text{BBC}) \\
\text{MailLoop}(a, d) & \triangleq \\
\text{MailOnce}(a) & \succ \text{Rtimer}(d) \succ \text{MailLoop}(a, d)
\end{align*} \]

- 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.
Another Strategy

- Start $MailOnce(a)$ and $Rtimer(t)$ simultaneously.

$$MailLoop(a, d) \triangleq MailOnce(a) \mid Rtimer(d) \gg MailLoop(a, d)$$

Compare with

$$MailLoop(a, d) \triangleq MailOnce(a) \gg Rtimer(d) \gg MailLoop(a, d)$$

What if $MailOnce(a)$ does not respond for 3 days?
Metronome

Publish a signal at every time unit.

\[
\text{Metronome} \triangleleft \text{Signal} \mid (\text{Rtimer}(1) \gg \text{Metronome})
\]

Publish \( n \) signals.

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

- Site $\text{Query}$ returns a value (different ones at different times).

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

- Produce all acceptable values by calling $\text{Query}$ at unit intervals forever.

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

$$let(z)\quad <z< M \mid (Rtimer(t) \gg let(0))$$
Fork-join parallelism

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

Return their values as a tuple after both respond.

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

Notational Convention:

\[
\text{let}(u, v) \\
<u < M \\
<v < N
\]
Recursive definition with time-out

Call a list of sites.

Count the number of responses received within 10 time units.

\[
\begin{align*}
tally([]) & \triangleq let(0) \\
tally(M : MS) & \triangleq u + v \\
\langle u \rangle (M \triangleright let(1)) & | (Rtimer(10) \triangleright let(0)) \\
\langle v \rangle & tally(MS)
\end{align*}
\]
Barrier Synchronization in $M \gg f \mid N \gg g$

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

Rendezvous of CSP or CCS; $M$ and $N$ are complementary actions.

$$\begin{align*}
(\text{let}(u, v) \\
< u < M \\
< v < N)
\end{align*} \gg (f \mid g)$$
Priority

- Publish \( N \)'s response asap, but no earlier than 1 unit from now. Apply fork-join between \( R_{timer}(1) \) and \( N \).

\[
\text{Delay} \triangleq (R_{timer}(1) \gg \text{let}(u)) < u < N
\]

- Call \( M, N \) together.

  If \( M \) responds within one unit, take its response.

  Else, pick the first response.

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

Introduce two sites:

- $\text{Interrupt.set}$: to interrupt $f$
- $\text{Interrupt.get}$: responds after $\text{Interrupt.set}$ has been called.

Instead of $f$, evaluate

$$\text{let}(z) <z< (f \mid \text{Interrupt.get})$$
Sites $M$ and $N$ return booleans. Compute their \textit{parallel or}.

\[
\text{ift}(b) \triangleq \text{if}(b) \gg \text{let}(\text{true}):
\]

returns \textit{true} if $b$ is \textit{true}; silent otherwise.

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

To return just one value:

\[
\text{let}(z)
\]
\[
<z < \text{ift}(x) \mid \text{ift}(y) \mid \text{or}(x, y)
\]
\[
<x < M
\]
\[
<y < N
\]
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 $y$.

\[
\text{let}(z)
\begin{align*}
&<z< \text{threshold}(x) \mid \text{threshold}(y) \mid \text{Min}(x, y) \\
&<x< A \\
&<y< B
\end{align*}
\]
Sequential Computing

- \((S; T)\) is \((S \gg T)\)

- \(\text{if } b \text{ then } S \text{ else } T\)
  
  is

  \(\text{if}(b) \gg S \mid \text{if}(\neg b) \gg T\)

- \(\text{while } B(x) \text{ do } x := S(x)\)

\(\text{loop}(x) \triangleq \frac{B(x) > b > (\text{if}(b) \gg S(x) > y > \text{loop}(y) \mid \text{if}(\neg b) \gg \text{let}(x))}{\text{while } B(x) \text{ do } x := S(x)}\)
Angelic 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.

\[
\begin{align*}
\text{extend}(z, 1) & \triangleq \text{valid}(0:z) \mid \text{valid}(1:z) \mid \cdots \mid \text{valid}(7:z) \\
\text{extend}(z, n) & \triangleq \text{extend}(z, 1) > y > \text{extend}(y, n - 1)
\end{align*}
\]

- \(z\): partial placement of queens (list of values from 0..7)

- \(\text{extend}(z, n)\) publishes all valid extensions of \(z\) with \(n\) additional queens.

- \(\text{valid}(z)\) returns \(z\) if \(z\) is valid; silent otherwise.

- Solve the original problem by calling \(\text{extend}([], 8)\).
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 \( e \):

\[
P(c, e) \triangleq c.get \; \triangleright x \triangleright \text{Compute}(x) \triangleright y \triangleright e.put(y) \implies P(c, e)
\]

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

\[
Net(c, d, e) \triangleq P(c, e) \mid P(d, e)
\]
Interaction: Run a dialog

User inputs an integer on channel \( p \)

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

Site \( \text{Prime}\?(x) \) returns \( true \) iff \( x \) is prime.

\[
\begin{align*}
\text{Dialog}(p, q) & \triangleq \\
p.\text{get} & > x > \\
\text{Prime}\?(x) & > b > \\
q.\text{put}(b) & \Rightarrow \\
\text{Dialog}(p, q)
\end{align*}
\]
Laws of Kleene Algebra

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

\[
\begin{align*}
  f | 0 &= f \\
  f | g &= g | f \\
  (f | g) | h &= f | (g | h) \\
  f | f &= f \\
  (f \gg g) \gg h &= f \gg (g \gg h) \\
  0 \gg f &= 0 \\
  f \gg 0 &= 0 \\
  Signal \gg f &= f \\
  f \gg x \gg let(x) &= f \\
  f \gg (g | h) &= (f \gg g) \gg (f \gg h) \\
  (f | g) \gg h &= (f \gg h | g \gg h)
\end{align*}
\]
(Idempotence of $|\ )$ \hspace{1cm} f \mid f = f
(Right zero of $\gg$) \hspace{1cm} f \gg 0 = 0
(Left Distributivity of $\gg$ over $|$) \hspace{1cm} f \gg (g \mid h) = (f \gg g) \mid (f \gg h)
Additional Laws

(Distributivity over $\gg$) if $g$ is $x$-free
$((f \gg g) < x < h) = (f < x < h) \gg g$

(Distributivity over $|$) if $g$ is $x$-free
$((f | g) < x < h) = (f < x < h) | g$

(Distributivity over where) if $g$ is $y$-free
$((f < x < g) < y < h) = ((f < y < h) < x < g)$

(Elimination of where) if $f$ is $x$-free, for site $M$
$(f < x < M) = f | (M \gg 0)$
Why Bother with a Formal Semantics?

- Precise definition of execution
- Prove certain identities as reality check
- Guidance for implementation
- Avoid oversight of issues
- Possibility of enhancement of the programming model
Asynchronous Orc: No notion of time

- no \textit{Rtimer}

- no requirement that anything be done:
  In \( f | g \), \( f \) may start whereas \( g \) may never start.
Events, Transitions

\[ f \xrightarrow{a} f' \]

denotes evaluation of expression \( f \) may cause event \( a \) and then transit to \( f' \). Example: \( \text{let}(v) \xrightarrow{!v} 0 \)

Event:

- Site Call: \( M_k(v) \), call site \( M \) with parameter values \( v \); call identified by \( k \).
- Site Response: \( k?m \), site responds with value \( m \) for call \( k \).
- Publication: \( !v \), publish \( v \).
- Silent (internal): \( \tau \)
Rules for Site Call

\[ M(v) \xrightarrow{M_k(v)} ?k, \ k \text{ fresh} \]  
\text{(SITECALL)}

\[ ?k \xrightarrow{k?v} \text{let}(v) \]  
\text{(SITERET)}

\[ \text{let}(v) \xrightarrow{!v} 0 \]  
\text{(LET)}
Symmetric Composition

\[ f \xrightarrow{a} f' \]
\[ f \mid g \xrightarrow{a} f' \mid g \]  \hspace{1cm} \text{(Sym1)}

\[ g \xrightarrow{a} g' \]
\[ f \mid g \xrightarrow{a} f \mid g' \]  \hspace{1cm} \text{(Sym2)}
### Sequencing

\[
\begin{align*}
  f & \xrightarrow{a} f' & a \neq !v \\
  f > x > g & \xrightarrow{a} f' > x > g \\
  f & \xrightarrow{!v} f' \\
  f > x > g & \xrightarrow{\tau} (f' > x > g) \mid [v/x].g
\end{align*}
\]

\((\text{SEQ1N})\) \hspace{10cm} \((\text{SEQ1V})\)
Asymmetric Composition

\[
\begin{align*}
f & \xrightarrow{a} f' \\
f < x < g & \xrightarrow{a} f' < x < g
\end{align*}
\]

\[
\begin{align*}
g & \xrightarrow{a} g' \\
& \quad a \neq !v \\
\end{align*}
\]

\[
\begin{align*}
f < x < g & \xrightarrow{a} f < x < g'
\end{align*}
\]

\[
\begin{align*}
g & \xrightarrow{!v} g' \\
 f < x < g & \xrightarrow{\tau} [v/x].f
\end{align*}
\]
Expression Call

\[
\frac{[[ E(x) \triangleq f ]] \in D}{E(p) \xrightarrow{\tau} [p/x].f}
\]  

(Def)
$M(v) \xrightarrow{M_k(v)} ?k, \ k \text{ fresh}$

$\begin{align*}
?k & \xrightarrow{k?v} \text{let}(v) \\
\text{let}(v) & \xrightarrow{!v} 0 \\
f & \xrightarrow{a} f' \\
\begin{array}{c}
\frac{f | g}{f | g'} \\
g' \\
\end{array} \\
E(p) & \xrightarrow{\tau} [p/x].f
\end{align*}$

$\begin{align*}
f & \xrightarrow{a} f' \quad a \neq !v \\
\begin{array}{c}
f >x> g \xrightarrow{a} f' >x> g \\
f & \xrightarrow{!v} f' \\
f >x> g & \xrightarrow{\tau} (f' >x> g) | [v/x].g \\
f & \xrightarrow{a} f' \\
f <x< g & \xrightarrow{a} f' <x< g \\
g & \xrightarrow{a} g' \quad a \neq !v \\
f & \xrightarrow{a} f' <x< g' \\
g & \xrightarrow{!v} g' \\
f & \xrightarrow{\tau} [v/x].f
\end{array}$
Example

\[ ((M(x) \mid \text{let}(x)) > y > R(y)) < x < (N \mid S) \]

\[ S_k \rightarrow \{ \text{Call } S : S \xrightarrow{S_k} ?k ; N \mid S \xrightarrow{S_k} N \mid ?k \} \]

\[ ((M(x) \mid \text{let}(x)) > y > R(y)) < x < (N \mid ?k) \]

\[ N_l \rightarrow \{ \text{Call } N \} \]

\[ ((M(x) \mid \text{let}(x)) > y > R(y)) < x < (?l \mid ?k) \]

\[ l?5 \rightarrow \{ ?l \xrightarrow{l?5} \text{let}(5) ; ?l \mid ?k \xrightarrow{l?5} \text{let}(5) \mid ?k \} \]

\[ ((M(x) \mid \text{let}(x)) > y > R(y)) < x < (\text{let}(5) \mid ?k) \]
Example; contd.

\[(M(x) \mid \text{let}(x)) > y > R(y) \triangleleft x < (\text{let}(5) \mid ?k)\]

\[
\tau \rightarrow \{ \text{let}(5) \stackrel{15}{\rightarrow} 0; \ \text{let}(5) \mid ?k \stackrel{15}{\rightarrow} 0 \mid ?k \}\]

\[(M(5) \mid \text{let}(5)) > y > R(y)\]

\[
\tau \rightarrow \{ \text{let}(5) \stackrel{15}{\rightarrow} 0; \ M(5) \mid \text{let}(5) \stackrel{15}{\rightarrow} M(5) \mid 0; \ f \stackrel{v}{\rightarrow} f' \implies f > y > g \stackrel{\tau}{\rightarrow} (f' > y > g) \mid [v/y].g \}\]

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

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

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

\[
((M(5) | 0) > y > R(y)) | ?n
\]

\[
\frac{n?7}{?n \quad \frac{n?7}{let(7)}}
\]

\[
((M(5) | 0) > y > R(y)) | let(7)
\]

\[
\frac{!7}{\{ f \mid let(7) \quad \frac{!7}{f \mid 0} \}}
\]

\[
((M(5) | 0) > y > R(y)) | 0
\]

The sequence of events:

\[S_k \quad N_l \quad l?5 \quad \tau \quad \tau \quad R_n(5) \quad n?7 \quad !7\]

The sequence minus \(\tau\) events:

\[S_k \quad N_l \quad l?5 \quad R_n(5) \quad n?7 \quad !7\]
Executions and Traces

Define

\[ f \xrightarrow{\epsilon} f \]
\[ f \xrightarrow{a} f'', f'' \xrightarrow{s} f' \]
\[ f \xrightarrow{a \cdot s} f' \]

- 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

Define $f \sim g$: Executions of $f$ and $g$ are equal.

- $f | 0 \sim f$
- $f | g \sim g | f$
- $f | (g | h) \sim (f | g) | h$
- $f >x> (g >y> h) \sim (f >x> g) >y> h$, if $h$ is $x$-free.
- $0 >x> f \sim 0$
- $(f | g) >x> h \sim f >x> h | g >x> h$
- $(f | g) <x< h \sim (f <x< h) | g$, if $g$ is $x$-free.
- $(f >y> g) <x< h \sim (f <x< h) >y> g$, if $g$ is $x$-free.
- $(f <x< g) <y< h \sim (f <y< h) <x< g$, if $g$ is $y$-free
  - $h$ is $x$-free.
Relation \sim is an equality

Given \ f \sim g, show

1. \quad \begin{align*}
&f | h \sim g | h \\
&h | f \sim h | g
\end{align*}

2. \quad \begin{align*}
&f >x> h \sim g >x> h \\
&h >x> f \sim h >x> g
\end{align*}

3. \quad \begin{align*}
&f <x< h \sim g <x< h \\
&h <x< f \sim h <x< g
\end{align*}
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: $let(x) \sim 0$
  But $let(1) > x > 0 \neq let(1) > x > let(x)$

- If we work with only open expressions, we can’t show
  $$let(x) \mid let(y) \sim let(y) \mid let(x)$$
Substitution Event

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

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

- Earlier rules apply to base events only.

From \( f \xrightarrow{[v/x]} [v/x] \cdot f \), we can not conclude:

\[ f \mid g \xrightarrow{[v/x]} [v/x] \cdot f \mid g \]
Define Orc combinators over trace sets, \( S \) and \( T \). Define:

\[
S \mid T, \quad S >x> T, \quad S <x< T.
\]

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

Theorem

\[
\langle f \mid g \rangle = \langle f \rangle \mid \langle g \rangle \\
\langle f >x> g \rangle = \langle f \rangle >x> \langle g \rangle \\
\langle f <x< g \rangle = \langle f \rangle <x< \langle g \rangle
\]
Trace equality is Equality

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

**Theorem** (Combinators preserve \( \cong \))

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

Specifically, given \( f \cong g \)

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

2. \( f \rightarrow x \rightarrow h \cong g \rightarrow x \rightarrow h \)
   \( h \rightarrow x \rightarrow f \cong h \rightarrow x \rightarrow g \)

3. \( f \leftarrow x \leftarrow h \cong g \leftarrow x \leftarrow h \)
   \( h \leftarrow x \leftarrow f \cong h \leftarrow x \leftarrow g \)
Assigning Meaning to Recursion

\[ M \triangle S | R \gg M \] — Metronome

To compute executions (traces) of \( M \), define

\[
\begin{align*}
M_0 & \triangle 0 \\
M_{i+1} & \triangle S | R \gg M_i, \quad i \geq 0
\end{align*}
\]

Take \( M \) to be the least upper bound of the chain \( M_0, M_1, \ldots \).

We need:

- (Monotonicity) A partial order \( \sqsubseteq \) such that \( M_0 \sqsubseteq M_1 \sqsubseteq \cdots \)

- (Continuity) Least upper bound of this chain exists and has desirable properties.
Monotonicity, Continuity

• Define: \( f \sqsubseteq g \) if \( \langle f \rangle \subseteq \langle g \rangle \).

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

\[
f \ast h \sqsubseteq g \ast h, \quad h \ast f \sqsubseteq h \ast g
\]

• Given chain \( f : f_0 \sqsubseteq f_1, \ldots, f_i \sqsubseteq f_{i+1}, \ldots \), its least upper bound exists. Write it as \( \sqcup f \).

**Theorem:** \( \sqcup (f_i \ast h) \cong (\sqcup f) \ast h \).

**Theorem:** \( \sqcup (h \ast f_i) \cong h \ast (\sqcup f) \).
Treatment of Time

\[ f \overset{t,a}{\rightarrow} f' \]

Evaluation of \( f \) may cause event \( a \) at \( t \) units after the evaluation starts. No event precedes \( a \). Transition to \( f' \).

\[
\begin{align*}
let(7) & \xrightarrow{0,!7} 0 \\
Rtimer(3) & \xrightarrow{3,!} 0 \\
Rtimer(3) \mid Rtimer(5) & \xrightarrow{3,!} 0 \mid Rtimer(2)
\end{align*}
\]
**Time-Shifted Expression**

\( f^t \): expression resulting from \( f \) after \( t \) units have elapsed without occurrence of an event.

\( f^t = \bot \) if \( t \) units cannot elapse without an event.

\[
\begin{align*}
R_{\text{timer}}(5)^0 &= R_{\text{timer}}(5) \\
R_{\text{timer}}(5)^3 &= R_{\text{timer}}(2) \\
R_{\text{timer}}(5)^6 &= \bot \\
let(v)^1 &= \bot
\end{align*}
\]

Expression containing \( \bot \) is \( \bot \).

\( f \xrightarrow{t,a} \bot \)

means \( f \) does not engage in \((t,a)\).
Definition of Time-shifted Expressions

\[(f \mid g)^t \triangleq f^t \mid g^t\]

\[(f \gg g)^t \triangleq f^t \gg g\]

\[(f \ll g)^t \triangleq f^t \ll g^t\]

\[M(x)^t \triangleq M(x)\]

\[M(m)^t \triangleq \begin{cases} M(m) & \text{if } t = 0 \\ \bot & \text{otherwise.} \end{cases}\]
Facts about Time-Shifted Expressions

\[ f^0 = f \]
\[ (f^s)^t = f^{s+t} \]
\[ f^s \xrightarrow{t,a} h \equiv f^s \xrightarrow{s+t,a} h \]
$M(v) \xrightarrow{0, M_k(v)} ?k, \ k \text{ fresh}$

$\begin{align*}
?k & \xrightarrow{t, ?v} \text{let}(v) \\
\text{let}(v) & \xrightarrow{0, !v} 0 \\
f & \xrightarrow{t, a} f' \\
f | g & \xrightarrow{t, a} f' | g^t \\
g & \xrightarrow{t, a} g' \\
f | g & \xrightarrow{t, a} f^t | g' \\
[[E(x) \triangle f]] & \in D \\
E(p) & \xrightarrow{0, \tau} [p/x].f
\end{align*}$

$\begin{align*}
\frac{f \xrightarrow{t, a} f' \quad a \neq !v}{f \xrightarrow{t, !v} f'} \\
\frac{f \xrightarrow{t, a} f' \quad a \neq !v}{f \xrightarrow{t, \tau} (f' \xrightarrow{t, \tau}) | [v/x].g} \\
\frac{f \xrightarrow{t, a} f' \quad a \neq !v}{f \xrightarrow{t, \tau} f'} \\
\frac{f \xrightarrow{t, a} f' \quad a \neq !v}{f \xrightarrow{t, \tau} f' \xrightarrow{t, \tau} f^t | g^t} \\
\frac{f \xrightarrow{t, a} f' \quad a \neq !v}{f \xrightarrow{t, \tau} f^t | g^t} \\
\frac{f \xrightarrow{t, a} f' \quad a \neq !v}{f \xrightarrow{t, \tau} f^t | g^t} \\
\frac{f \xrightarrow{t, a} f' \quad a \neq !v}{f \xrightarrow{t, \tau} f^t | g^t}
\end{align*}$
**Execution, Trace, Substitution**

**Execution**

\[
\begin{align*}
& \frac{f \xrightarrow{\epsilon} f}{f \xrightarrow{(t,a)} f'', f'' \xrightarrow{u} f'} \\
& \frac{f \xrightarrow{(t,a)u_t} f'}
\end{align*}
\]

**Substitution Event**

\[
\begin{align*}
& f \xrightarrow{t,[v/x]} [v/x].(f^t)
\end{align*}
\]
Example of Execution

\[(\text{Rtimer}(3) \gg M(x)) | N\]

\[0, N_k \rightarrow \{(\text{SYM2}), (\text{CALL})\}\]

\[(\text{Rtimer}(3) \gg M(x)) | ?k\]

\[2, k?7 \rightarrow \{(\text{SYM2}), (\text{RETURN}), \text{response for } ?k \text{ is } (2,7)\}\]

\[\text{let}(7)\]

\[0, \!7 \rightarrow \text{Rule for } \text{let}\]

\[\text{let}(7)\]

\[0 \rightarrow \{(\text{SYM1}), (\text{SEQ1V}), (\text{RETURN}), \text{Rtimer}(3)^2 = \text{Rtimer}(1)\}\]

\[((0 \gg M(x)) | M(x)) | 0\]

\[= \{\text{simplifying}\}

\[M(x)\]

\[u = (0, N_k) (2, k?7) (2, \!7) (3, \tau) \]

\[\text{trace}(u) = (2, \!7)\]