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 Orchestration language \textbf{Orc}

- A Program execution
  - calls \textit{sites}, to invoke services.
  - publishes \textit{values}.

- Orc is simple
  - Language has only 3 combinators.
  - Can handle time-outs, priorities, failures, synchronizations, \ldots
  - 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 $f | g$ Symmetric composition
  for all $x$ from $f$ do $g$ $f >x> g$ Piping
  for some $x$ from $g$ do $f$ $f$ where $x:\in g$ Asymmetric composition
Symmetric composition: $f \mid g$

\[ CNN \mid BBC: \text{ calls both } CNN \text{ and } BBC \text{ simultaneously.} \]

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: \( f \rightarrow x \rightarrow g \)

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

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

- \( (CNN \mid BBC) \rightarrow x \rightarrow Email(address, x) \)
  
  May call \( Email \) twice. Publishes up to two values from \( Email \).
Figure 1: Schematic of piping

\[ f \succ x \succ g \]
\( f \gg g \) for \( f \gg x \gg g \),
if \( x \) unused in \( g \).

Precedence: \( f \gg x \gg g \mid h \gg y \gg u \) for
\[
(f \gg x \gg g) \mid (h \gg y \gg u)
\]
Asymmetric parallel composition: \((f \text{ where } x \in g)\)

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.
  \((M \mid N(x)) \text{ where } x \in g\)

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

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

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

Binds \(x\) to the first value from \(CNN \mid BBC\). 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 $Rt\text{ime}$ respond immediately.

- Concurrent and distributed executions are derived from an expression.
Expression Definition

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

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

- 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.

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

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.

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

\[
\text{let}(z) \text{ where } \\
z \in M \\
\quad | \ Rtimer(t) \gg \text{let}(0)
\]
Fork-join parallelism

Call $M$ and $N$ in parallel.

Return their values as a tuple after both respond.

$$(\text{let}(u, v)$$

where $u \in M$$$

where $v \in N$$$

Notational Convention:

$$\text{let}(u, v)$$

where $u \in M$$$

$v \in N$
Recursive definition with time-out

Call a list of sites.

Count the number of responses received within 10 time units.

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

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

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

In Orc:

\[ if(b) \rightarrow P \mid if(\neg b) \rightarrow Q \]

where

\[ b \in (\text{Alpha } \rightarrow \text{let(true)}) \mid (\text{Beta } \rightarrow \text{let(false)}) \]

Orc does not permit non-deterministic internal choice.
• Publish $N$’s response asap, but no earlier than 1 unit from now. Apply fork-join between $Rtimer(1)$ and $N$.

$$\text{Delay} \triangleq (Rtimer(1) \triangleright let(u)) \text{ where } u : \in N$$

• Call $M$, $N$ together.

  If $M$ responds within one unit, take its response.
  Else, pick the first response.

$$let(x) \text{ where } x : \in (M \mid Delay)$$
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

$$\text{let}(z) \text{ where } z \in (f \mid \text{Interrupt.get})$$
Parallel or

Sites $M$ and $N$ return booleans. Compute their parallel or.

$\text{ift}(b) \triangleq \text{if}(b) \Rightarrow \text{let(true)}$: returns $true$ if $b$ is $true$; silent otherwise.

$$\text{ift}(x) \mid \text{ift}(y) \mid \text{or}(x, y)$$

where

$x: \in M$, $y: \in N$

To return just one value:

$$\text{let}(z)$$

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.

$threshold(x)$ returns $x$ if $x < c$; silent otherwise.

$Min(x, y)$ returns the minimum of $x$ and $y$.

$$let(z)$$
where
$$z \in threshold(x) \mid threshold(y) \mid Min(x, y)$$

$x \in A$

$y \in B$
Sequential Computing

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

- if \(b\) then \(S\) else \(T\)

is

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

- while \(B(x)\) do \(x := S(x)\)

\[
\text{loop}(x) \triangleq \!
\begin{array}{c}
B(x) > b > (\text{if}(b) \gg S(x) > y > \text{loop}(y) \mid \text{if}(\neg b) \gg \text{let}(x))
\end{array}
\]
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.

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

- \(z\): 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 \( c\text{.put} \) and \( c\text{.get} \) as site calls.
- In our examples, \( c\text{.get} \) is blocking and \( c\text{.put} \) 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 \( e \):

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

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

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

User inputs an integer on channel $p$

Process outputs $true$ on channel $q$ iff the number is prime.

Site $Prime? (x)$ returns $true$ iff $x$ is prime.

$$Dialog(p, q) \triangleq$$
$$p.\text{get} \quad > x >$$
$$Prime?(x) \quad > b >$$
$$q.\text{put}(b) \quad \Rightarrow$$
$$Dialog(p, q)$$
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 \( \mid \) )

\[
\begin{align*}
(f \mid 0) & = f \\
(f \mid g) & = g \mid f \\
(f \mid g) \mid h & = f \mid (g \mid h) \\
(f \mid f) & = f \\
(f \gg g) \gg h & = f \gg (g \gg h) \\
0 & \gg f = 0 \\
f & \gg 0 = 0 \\
Signal \gg f & = f \\
f \gg x \gg \text{let}(x) & = f \\
(f \mid g) \mid h & = (f \gg g) \mid (f \gg h) \\
(f \mid g) \gg h & = (f \gg h) \mid (g \gg h)
\end{align*}
\]
Laws which do not hold

(Idempotence of \( | \) ) \( f | f = f \)

(Right zero of \( \gg \) ) \( f \gg 0 = 0 \)

(Left Distributivity of \( \gg \) over \( | \) ) \( f \gg (g | h) = (f \gg g) | (f \gg h) \)
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

\[ \frac{k \text{ fresh}}{M(v) \xrightarrow{M_k(v)} ?k} \]  \hspace{1cm} (SITE\text{CALL})

\[ ?k \xrightarrow{k?v} \text{let}(v) \]  \hspace{1cm} (SITE\text{RET})

\[ \text{let}(v) \xrightarrow{!v} 0 \]  \hspace{1cm} (LET)
Symmetric Composition

\[
\begin{align*}
f & \xrightarrow{\alpha} f' \\
f \mid g & \xrightarrow{\alpha} f' \mid g
\end{align*}
\]  
\text{(SYM1)}

\[
\begin{align*}
g & \xrightarrow{\alpha} g' \\
f \mid g & \xrightarrow{\alpha} f \mid g'
\end{align*}
\]  
\text{(SYM2)}
Sequencing

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

\text{(SEQ1N)}

\text{(SEQ1V)}
Asymmetric Composition

\[
\frac{\frac{f \xrightarrow{a} f'}{f \text{ where } x \in g \xrightarrow{a} f' \text{ where } x \in g}}{(ASYM1N)}
\]

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

\[
\frac{\frac{g \xrightarrow{a} g'}{a \neq !v}}{(ASYM2)}
\]
Expression Call

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

(DEF)
\[
\begin{align*}
\begin{array}{c}
\text{k fresh} \\
\hline
M(v) \quad M_k(v) \\
?k \\
?k \quad \text{let}(v) \\
\text{let}(v) \quad !v \quad 0 \\
f \quad \frac{a}{f'} \\
\hline
f \mid g \quad \frac{a}{f' \mid g} \\
g \quad \frac{a}{g'} \\
\hline
f \mid g \quad \frac{a}{f \mid g'} \\
\hline
[[E(x) \ \Delta \ f]] \in D \\
\hline
E(p) \quad \frac{\tau}{[p/x].f} \\
\end{array}
\end{align*}
\]

**Rules**

\[
\begin{align*}
f \quad \frac{a}{f'} & \quad a \neq !v \\
f \quad \frac{\tau}{f' >x> g} & \quad a \neq !v \\
\hline
f \quad \frac{a}{f'} & \quad a \neq !v \\
\hline
f \quad \frac{\tau}{(f' >x> g) \mid [v/x].g} \\
\hline
f \quad \frac{a}{f'} \\
\hline
f \text{ where } x: \in g \quad \frac{a}{f' \text{ where } x: \in g} \\
\hline
g \quad \frac{\tau}{g'} \\
\hline
f \text{ where } x: \in g \quad \frac{\tau}{[v/x]\cdot f} \\
\hline
g \quad \frac{a}{g'} & \quad a \neq !v \\
\hline
f \text{ where } x: \in g \quad \frac{a}{f \text{ where } x: \in g'}
\end{align*}
\]
Example

\[((M(x) \mid let(x)) \rightarrow y > R(y)) \text{ where } x : \in (N \mid S)\]

\[
\begin{array}{c}
\frac{S_k}{\text{Call } S : S \xrightarrow{S_k} k ; N \mid S \xrightarrow{S_k} N \mid k} \\
\end{array}
\]

\[((M(x) \mid let(x)) \rightarrow y > R(y)) \text{ where } x : \in (N \mid k)\]

\[
\begin{array}{c}
\frac{N_l}{\text{Call } N} \\
\end{array}
\]

\[((M(x) \mid let(x)) \rightarrow y > R(y)) \text{ where } x : \in (?l \mid k)\]

\[
\begin{array}{c}
\frac{l \xrightarrow{?5}}{\{ ?l \xrightarrow{?5} let(5) ; ?l \mid ?k \xrightarrow{?5} let(5) \mid k \}} \\
\end{array}
\]

\[((M(x) \mid let(x)) \rightarrow y > R(y)) \text{ where } x : \in (let(5) \mid k)\]
Example; contd.

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

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

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

\[ \tau \{ let(5) \xrightarrow{15} 0; \ M(5) \mid let(5) \xrightarrow{15} M(5) \mid 0; \ f \xrightarrow{!v} f' \text{ implies } f \succ y \succ g \xrightarrow{\tau} (f' \succ y \succ g) \mid [v/y] \cdot g \} \]

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

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

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

\[(M(5) \mid 0) \triangleright y \triangleright R(y) \mid ?n\]
\[\{?n \quad n?7 \quad \text{let}(7)\}\]

\[(M(5) \mid 0) \triangleright y \triangleright R(y) \mid \text{let}(7)\]

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

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

The sequence of events:
\[S_k \quad N_l \quad l?5 \quad \tau \quad \tau \quad R_n(5) \quad n?7 \quad \text{!7}\]

The sequence minus \(\tau\) events:
\[S_k \quad N_l \quad l?5 \quad R_n(5) \quad n?7 \quad \text{!7}\]
Executions and Traces

Define

\[ f \xrightarrow{\epsilon} f \]

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

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

- 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

- $f \mid 0 \sim f$
- $f \mid g \sim g \mid f$
- $f \mid (g \mid h) \sim (f \mid g) \mid h$
- $f \succ x \succ (g \succ y \succ h) \sim (f \succ x \succ g) \succ y \succ h$, if $h$ is $x$-free.
- $0 \succ x \succ f \sim 0$
- $(f \mid g) \succ x \succ h \sim f \succ x \succ h \mid g \succ x \succ h$
- $(f \mid g)$ where $x: \in h \sim (f$ where $x: \in h) \mid g$, if $g$ is $x$-free.
- $(f \succ y \succ g)$ where $x: \in h \sim (f$ where $x: \in h) \succ y \succ g$, 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 \text{ is an equality}

Given \: f \sim g, \text{ show}

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

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

3. \: \begin{align*}
   f \text{ where } x: \in h &\sim g \text{ where } x: \in h \\
   h \text{ where } x: \in f &\sim h \text{ where } x: \in 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 $let(1) > x > 0 \neq let(1) > x > let(x)$

- Then 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, \( \text{let}(x)^{[1/x]} \rightarrow \text{let}(1) \).
  
  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:
  
  \[ f \mid g \xrightarrow{[v/x]} [v/x] \cdot f \mid g \]
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: $f \equiv g$ if $\langle f \rangle = \langle g \rangle$.

**Theorem** (Combinators preserve $\equiv$)

Given $f \equiv g$ and any combinator $* : f * h \equiv g * h$, $h * f \equiv h * g$

Specifically, given $f \equiv g$

1. $\begin{align*}
f &\equiv g \\
h &\equiv h \\
f &\equiv g
\end{align*}$

2. $\begin{align*}
f &\equiv g \\
h &\equiv h \\
f &\equiv g
\end{align*}$

3. $\begin{align*}
f &\equiv g \\
h &\equiv h \\
f &\equiv g
\end{align*}$
Monotonicity, Continuity

- Define: \( f \sqsubseteq g \) if \( \langle f \rangle \sqsubseteq \langle g \rangle \).
  
  **Theorem** (Monotonicity) Given \( f \sqsubseteq g \) and any combinator \( * \)
  
  \[ f * h \sqsubseteq g * h, \ h * f \sqsubseteq h * g \]

- Chain \( f : f_0 \sqsubseteq f_1, \cdots f_i \sqsubseteq f_{i+1}, \cdots \)

  **Theorem:** \( \bigsqcup (f_i * h) \cong (\bigsqcup f) * h \).
  
  **Theorem:** \( \bigsqcup (h * f_i) \cong h * (\bigsqcup f) \).
Least Fixed Point

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

\[ M_0 \cong 0 \]
\[ M_{i+1} \cong S \mid R \triangleright M_i, \ i \geq 0 \]

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

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