Computation Orchestration

Jayadev Misra

Department of Computer Science
University of Texas at Austin

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

Univ. of Texas
TECS Week, 2005

Orc

Compose basic computing elements called Sites. A site is a

- function: Compress MPEG file
- method of an object: LogOn procedure at a bank
- monitor procedure: read from a buffer
- web service: get a stock quote
- transaction: check account balance
- distributed transaction: move money from one bank to another

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.

Example: workflow

- An office assistant contacts a potential visitor.
- The visitor responds, sends the date of her visit.
- The assistant books an airline ticket and contacts two hotels for reservation.
- After hearing from the airline and any of the hotels: he tells the visitor about the airline and the hotel.
- The visitor sends a confirmation which the assistant notes.

Example: workflow, contd.

After receiving the confirmation, the assistant

- confirms hotel and airline reservations.
- reserves a room for the lecture.
- announces the lecture by posting it at a web-site.
- requests a technician to check the equipment in the room.

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

contact the visitor for a second time only after hearing from the airline and one of the hotels.

We look at only the third problem.

A new kind of assignment

$x :\in f$

where x is a variable and f is an Orc expression.

Evaluation of f may

start threads

yield zero or more values.

Assign the first value to x.

Simple Orc Expression

- CNN is a news service, d a date. Download the news page for d.

  $x :\in CNN(d)$

- Side-effect: Book ticket at airline A for a flight described by c.

  $x :\in A(c)$

The returned value is the price and the confirmation number.

Sites

- A site may not respond.

  Its response at different times (for the same input) may be different.

- A site call may change states (of external servers) tentatively or permanently.

  Tentative state changes are made permanent by explicit commitment.

Notation

- No arithmetic or logic capability in Orc.
- Can't write $u + v$ or $x \vee y$.
  Write $add(u, v)$ and $or(x, y)$, where add and or are sites.
- Convention: Write $u + v$ and $x \vee y$.
  Assume that a compiler converts these to $add(u, v)$ and $or(x, y)$.

Some Fundamental Sites

$let(x, y, \cdots)$: returns a tuple of the argument values.

$Rtimer(t)$: integer t, $t \geq 0$, returns a signal t time units later.

$Signal$ returns a signal immediately. Same as $Rtimer(0)$.

$if(b)$: boolean b,
returns a signal if b is true;
remains silent if b is false.

Composition Operators

- $CNN \;\; >x> \; Email(address, x)$ Sequencing
- $CNN \;\; | \; BBC$ Symmetric composition
- $(Email(address, x)$ where $x{:}{\in} \;\; (CNN \;\; | \; BBC))$
  Asymmetric composition

Syntax

$E \in$ Expression Name
$M \in$ Site
$x \in$ Variable
$p \in$ Parameter

$$
\begin{array}{llll}
f, g \in Expression & ::= & & \\
 & & \mathbf{0} & \text{Zero expression} \\
 & | & M(p*) & \text{Site call} \\
 & | & E(p*) & \text{Expression call} \\
 & | & f \;>x>\; g & \text{Sequential Composition} \\
 & | & f \mid g & \text{Symmetric Parallel Composition} \\
 & | & f \text{ where } x :\in\; g & \text{Asymmetric Parallel Composition}
\end{array}
$$

Examples

Convention: $>x>$ without x is $\gg$.

Precedence of binding powers: where , $:\in$, $|$, $\gg$

$N(x)$
$M \gg N(x)$
$M \;\; >u> \;\; N(u)$
$F(x,y) \;\gg\; N(u)$
$(M \gg \mathbf{0} \;|\; \{N(x)$ where $x{:\in}\; R \;|\; N(y)\})$

Site and Expression Call

- Site call $M(x)$: Call M if x is defined. Expression value is the response from M.
- Expression call: Similar to function call; may return many values.

Sequencing

- $M \gg N$: Call M; after hearing from M call N.
  Expression value is the response from N.

  $Rtimer(1) \gg Email(address, message)$
  $Rtimer(1) \gg Rtimer(1)$
  $Email(address1, message) \gg Email(address2, message) \gg Notify$

- $M \; >x> \; R(x)$: Pass the value from M in x

  $CNN \; >x> \; Email(address, x)$
  $Discover(c) \; >m> \; Apply(m, y)$

$\gg$ is associative. $>x>$ is right associative.

Symmetric composition using $|$

- $M \ | \ N$: parallel threads call M and N. Two possible values.
- $f \ | \ g$: parallel threads call f and g. Stream of values from each, merged in time-order.

Example:

$$CNN \ | \ BBC$$
$$M \ | \ M$$
$$(M \gg N) \ | \ (M \gg R)$$
$$if(b) \gg M \ | \ if(\neg b) \gg N$$

$$(M \mid N) \gg R$$

Create two threads to evaluate M and N. Call R for each result.

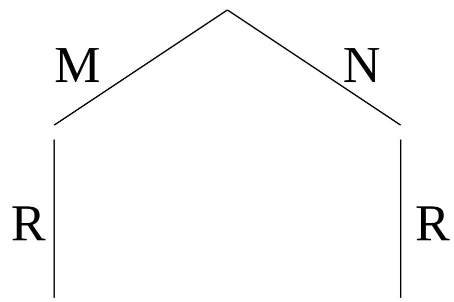

Figure 1: $(M \mid N) \gg R$

$$\{Email(address1, message) \mid Email(address2, message)\} \gg Notify$$

Double notification.

$f \;\; {>}x{>} \;\; g$

Example: $f \;\; {>}x{>} \;\; g$, where f produces values 0, 1 and 2.

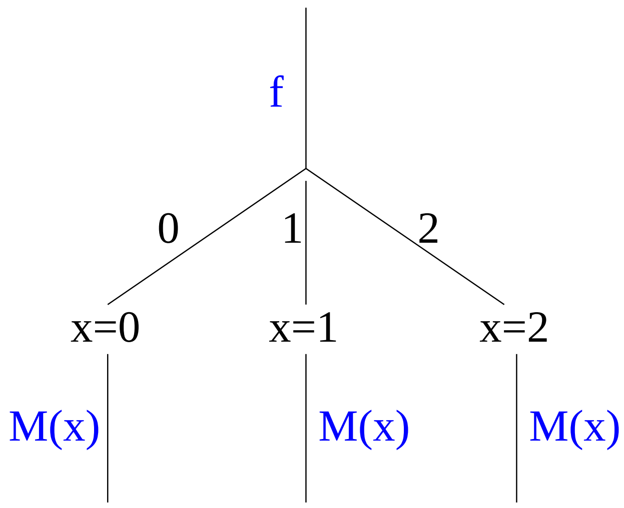

Figure 2: Computation of $f \;\; {>}x{>} \;\; M(x)$

Asymmetric parallel composition: $\{f$ where $x{:}\in\ g\}$

- Evaluate f and g in parallel.
- When g returns a result, assign the value to x and terminate g.
- Any site call in f which does not name x can proceed.
- A site calls which names x waits until x gets a value.
- Values produced by f are the values of $\{f$ where $x{:}\in\ g\}$.

Pruning the computation

$(CNN \mid BBC) \ {>}x{>} \ Email(address, x)$

May send two emails.

To send just one email:

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

Notify after both respond

$$\{Email(address1, message) \mid Email(address2, message)\} \gg Notify$$

Use

$\{\{let(u, v) \gg Notify$
where
$u :\in Email(address1, message)\}$
where
$v :\in Email(address2, message)\}$

Adopt the notation:

$\{let(u, v) \gg Notify$
where
$u :\in Email(address1, message)$
$v :\in Email(address2, message)\}$

Constant $\mathbf{0}$

$\mathbf{0}$ is a site which never responds.

Example: send an email but do not wait for its response:

$$\{Email(address1, message) \gg \mathbf{0} \quad | \quad Notify\}$$

Expression Definition

$$
\begin{array}{lll}
Asynch(M,N) & \underline{\Delta} & M \gg \mathbf{0} \mid N \\
Metronome & \underline{\Delta} & Signal \mid Rtimer(1) \gg Metronome
\end{array}
$$

$$
\begin{array}{lll}
BM(0) & \underline{\Delta} & \mathbf{0} \\
BM(n+1) & \underline{\Delta} & S \mid R \gg BM(n)
\end{array}
$$

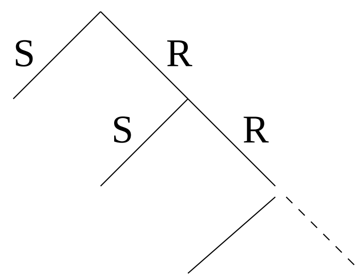

Figure 3: *Metronome*

Example of Expression call

- $Query$ returns a value (different ones at different times).
- $Accept(x)$ returns x if x is acceptable.
- Produce all acceptable values by calling $Query$ at unit intervals forever.

$$RepeatQuery \triangleq Metronome \gg Query >x> Accept(x)$$

Some Fundamental Sites

$let(x, y, \cdots)$: returns a tuple of the argument values.

$Rtimer(t)$: integer t, $t \geq 0$, returns a signal t time units later.

$Signal$ returns a signal immediately. Same as $Rtimer(0)$.

$if(b)$: boolean b,
returns a signal if b is true;
remains silent if b is false.

Small Examples

- Call site M four times, at unit time intervals.

$$M \quad | \quad Rtimer(1) \gg M \quad | \quad Rtimer(2) \gg M \quad | \quad Rtimer(3) \gg M$$

- Time-out: set z to M's response before t, 0 after t.

$$z :\in \quad M \quad | \quad Rtimer(t) \gg let(0)$$

- Receive N's response asap, but no earlier than 1 unit from now.

$$DelayedN \ \underline{\Delta} \quad \{Rtimer(1) \gg let(u) \text{ where } u :\in \ N\}$$

- Call M, N together. If M responds within one unit, take its response. Else, pick the first response.

$$x :\in \ M \quad | \ DelayedN$$

Recursive definition with time-out

Call a list of sites.

Count the number of responses received within 10 time units.

$$
\begin{array}{ll}
tally([\,]) & \underline{\Delta} \; let(0) \\
tally(x : xs) & \underline{\Delta} \\
& \{u + v \\
& \quad \text{where} \\
& \qquad u{:}\in \; x \gg let(1) \mid Rtimer(10) \gg let(0) \\
& \qquad v{:}\in \; tally(xs)\}
\end{array}
$$

Sequential Computing

- $(S;\ T)$ is $(S \gg T)$
- **if** b **then** S **else** T

  is

  $$if(b) \gg S \mid if(\neg b) \gg T$$

- **while** b **do** $x := S(x)$

  $$loop(x) \;\underline{\Delta}\; if(b) \gg S(x) >y> loop(y) \mid if(\neg b) \gg let(x)$$

Kleene Star

- For a given x, to produce the sequence of values

$$x,\ M(x),\ M(x)\ >y>\ M(y),\ M(x)\ >y>\ M(y)\ >z>\ M(z),\ \ldots$$

$$Mstar(x)\ \underline{\Delta}\ let(x)\ |\ M(x)\ >y>\ Mstar(y)$$

- To produce the same sequence of values without x, i.e.,

$$M(x),\ M(x)\ >y>\ M(y),\ M(x)\ >y>\ M(y)\ >z>\ M(z),\ \ldots$$

$$Mplus(x)\ \underline{\Delta}\ M(x)\ >y>\ (let(y)\ |\ Mplus(y))$$

Arbitration

In CCS: $\alpha.P + \beta.Q$

In Orc:

$$if(b) \gg P \mid if(\neg b) \gg Q$$

where

$$b :\in Alpha \gg let(\textsf{true}) \mid Beta \gg let(\textsf{false})$$

Time-out

Return $(x, \textit{true})$ if M returns x before t, and $(-, \textit{false})$ otherwise.

$let(z, b)$
where
$(z, b) :\in\ M\ \ >x>\ let(x, \textit{true})\ \ |\ \ Rtimer(t)\ \ >x>\ let(x, \textit{false})$

Fork-join parallelism

Call M and N in parallel.

Return their values as a tuple after they both respond.

$$\{ \ let(u, v)$$
$$\quad \text{where} \ u :\in M$$
$$\qquad\qquad v :\in N$$
$$\}$$

Screen Refresh

Get: screen image, keyboard input, mouse position every time unit.

Call $Draw$ with this triple.

$$
\begin{array}{l}
Metronome \\
\gg \{ \; let(i, k, m) \\
\quad \text{where} \quad i \;:\in\; Image \\
\qquad\qquad\; k \;:\in\; Keyboard \\
\qquad\qquad\; m :\in\; Mouse \\
\} \\
>x> \; Draw(x)
\end{array}
$$

Barrier Synchronization

Synchronize $M \gg f$ and $N \gg g$:

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

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

$$
\begin{array}{l}
\{\ let(u,v) \\
\quad \text{where } u{:}\in M \\
\quad\quad\quad\quad\ \ v{:}\in N\} \\
\gg (f \mid g)
\end{array}
$$

To pass values from M and N to f and g, modify last line:

$$>(u,v)>\ (f \mid g)$$

Interrupt handling

- Orc statement can not be directly interrupted.
- $Interrupt$ site: a monitor.
- $Interrupt.set$: to interrupt the Orc statement
- $Interrupt.get$: responds after $Interrupt.set$ has been called.

$$z :\in f$$

is changed to

$$z :\in f \quad | \quad Interrupt.get$$

Interrupt; contd.

Determine if there has been an interrupt:

$$
\begin{array}{l}
callM \ \underline{\Delta} \\
\quad (let(z, b) \\
\qquad \text{where} \\
\qquad\quad (z, b) :\in \ M \ \ >x> \ let(x, \textsf{true}) \ \ | \ \ Interrupt.get \ \ >x> \ let(x, \textsf{false}) \\
\quad)
\end{array}
$$

Process Interrupt:

$$
\begin{array}{l}
\qquad\qquad callM \\
>(z, b)> \\
\qquad\qquad \{ \ \ if\,(b) \ \ \gg \ \text{“Normal processing with value } z\text{”} \\
\qquad\qquad \ \ | \ \ if\,(\neg b) \ \gg \ \text{“Interrupt Processing”} \ \ \}
\end{array}
$$

Parallel or

Let sites M and N return booleans. Compute their parallel or.

$$\{if(x) \;\mid\; if(y) \;\mid\; or(x,y)$$

where

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

Return just one value.

$$\{let(z)$$

where

$$z :\in if(x) \;\mid\; if(y) \;\mid\; or(x,y)$$
$$x :\in M$$
$$y :\in N\}$$

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.

$$\{threshold(x) \mid threshold(y) \mid Min(x, y)$$

where

$$x :\in A$$
$$y :\in B\}$$

Processes

Run a dialog with the client.

Forever: client gives an integer; Process determines if it is prime.

Use channel tty: $tty.get$ and $tty.put$ are sites.

$$
\begin{array}{ll}
Dialog \triangleq & \\
\quad tty.get & >x> \\
\quad Prime?(x) & >b> \\
\quad tty.put(b) & \gg \\
\quad Dialog &
\end{array}
$$

Refinement of Dialog

The client specifies the communication channels.

$$
\begin{array}{ll}
Dialog(p,q) \ \underline{\Delta} & \\
\quad p.get & >x> \\
\quad Prime?(x) & >b> \\
\quad q.put(b) & \gg \\
\quad Dialog(p,q) &
\end{array}
$$

Typical Iterative Process

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

$$P(c,e) \;\underline{\Delta}\; c.get \;\; >x> \;\; Compute(x) \;\; >y> \;\; e.put(y) \;\gg\; P(c,e)$$

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

$$Net(c,d,e) \;\underline{\Delta}\; P(c,e) \;|\; P(d,e)$$

Mutual Exclusion

- Process i writes a site name on channel c_i. $Multiplexor_i$ collects inputs from c_i, writes on e.

$$Multiplexor_i \quad \underline{\Delta} \quad c_i.get \;\; >x> \;\; e.put(x) \;\; \gg \;\; Multiplexor_i$$

- $Multiplexor$ collects inputs from all channels, writes them on e.

$$Multiplexor \quad \underline{\Delta} \quad (\; | \; i :: Multiplexor_i)$$

- $Arbiter$ picks an item g from e; grants resource by calling $g.Grant$. $g.Grant$ responds after the process completes the resource usage.

$$Arbiter \quad \underline{\Delta} \quad e.get \;\; >g> \;\; g.Grant \;\; \gg \;\; Arbiter$$

- $Mutex$ coordinates all the activities.

$$Mutex \quad \underline{\Delta} \quad Multiplexor \; | \; Arbiter$$

Dining Philosophers

A philosopher's life is depicted by

$$
\begin{array}{l}
P_i \;\underline{\Delta} \\
\qquad \{let(x,y) \gg Eat \gg Fork_i.put \gg Fork_{i'}.put \\
\qquad\quad \text{where} \quad x :\in Fork_i.get \\
\qquad\qquad\qquad\; y :\in Fork_{i'}.get \\
\qquad \} \\
\qquad \gg P_i
\end{array}
$$

where $Fork_i$ and $Fork_i'$ are sites.

Represent the ensemble of N philosophers by

$$DP \;\underline{\Delta}\; (\;|\; i{:}\; 0 \leq i < N{:}\; P_i)$$

Synchronized Communication: Byzantine Protocol

- Process i sends values to j over channel c_{ij}.

$$Send_i(v) \;\underline{\Delta}\; \{Signal \mid (\mid j :: c_{ij}.put(v) \gg \mathbf{0})\}$$
$$Read_i \;\underline{\Delta}\; \{let(X) \text{ where } (\forall j :: X_j :\in c_{ji}.get)\}$$

- $Round_i(v, n)$: For process i to run n rounds with initial value v.

$$Round_i(v, 0) \;\underline{\Delta}\; let(v)$$
$$Round_i(v, n) \;\underline{\Delta}\; Send_i(v) \gg Read_i \;>X>\; Compute_i(X) \;>u>\; Round_i(u, n-1)$$

- $Byz(V, n)$: All processes run n rounds; initial value vector is V.

$$Byz(V, n) \;\underline{\Delta}\; \{ \mid i :: Round_i(V_i, n)\}$$

Backtracking: Eight queens

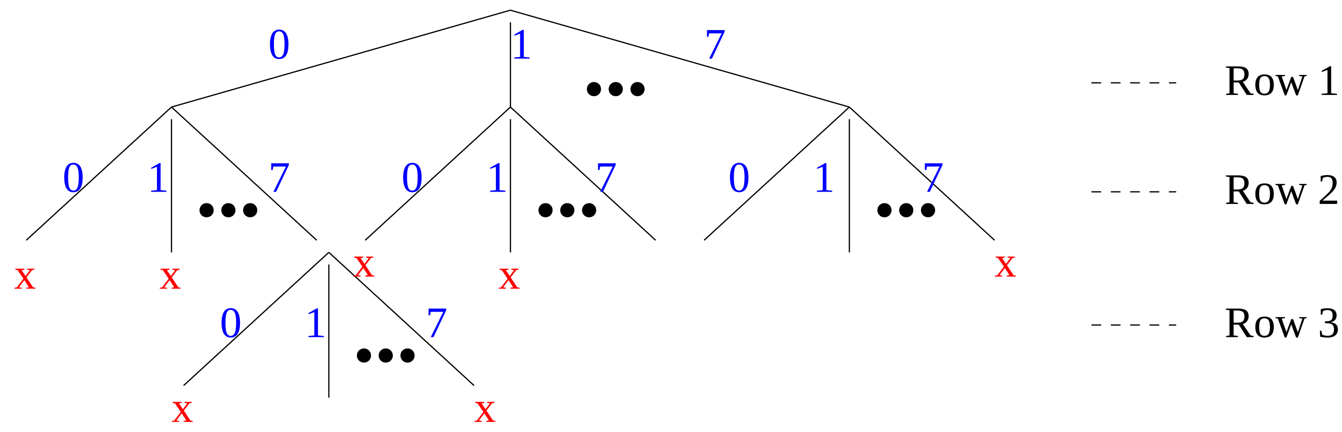

Figure 4: Backtrack Search for Eight queens

Eight queens; contd.

- configuration: placement of queens in the last i rows.
- Represent a configuration by a list of integers j, $0 \leq j \leq 7$.
- Valid configuration: no queen captures another.
- Site $check(x{:}xs)$: Given xs is valid, return

  $x{:}xs$, if it is valid

  remain silent, otherwise.

Eight queens; contd.

$extend(x, n)$: where x is a valid configuration, $1 \le n$ and $|x| + n \le 8$,
Produce all valid extensions of x by placing n additional queens.

Solve the original problem by calling $extend([\,], 8)$.

$$extend(x, 1) \quad \underline{\Delta} \quad \{\ \mid i{:}\ 0 \le i < 8{:}\ check(i{:}x)\}$$
$$extend(x, n) \quad \underline{\Delta} \quad extend(x, 1)\ \ {>}y{>}\ extend(y, n-1)$$

Laws of Kleene Algebra

(Zero and $\mid$)	$f \mid \mathbf{0} = f$
(Commutativity of $\mid$)	$f \mid g = g \mid f$
(Associativity of $\mid$)	$(f \mid g) \mid h = f \mid (g \mid h)$
(Idempotence of $\mid$)	$f \mid f = f$
(Associativity of $\gg$)	$(f \gg g) \gg h = f \gg (g \gg h)$
(Left zero of $\gg$)	$\mathbf{0} \gg f = \mathbf{0}$
(Right zero of $\gg$)	$f \gg \mathbf{0} = \mathbf{0}$
(Left unit of $\gg$)	$\mathbf{1} \gg f = f$
(Right unit of $\gg$)	$f \gg \mathbf{1} = f$
(Left Distributivity of $\gg$ over $\mid$)	$f \gg (g \mid h) = (f \gg g) \mid (f \gg h)$
(Right Distributivity of $\gg$ over $\mid$)	$(f \mid g) \gg h = (f \gg h \mid g \gg h)$

Laws which do not hold

(Idempotence of $\mid$)	$f \mid f = f$
(Right zero of $\gg$)	$f \gg \mathbf{0} = \mathbf{0}$
(Left Distributivity of $\gg$ over $\mid$)	$f \gg (g \mid h) = (f \gg g) \mid (f \gg h)$

Figure 5: Schematic for $M \gg (N \mid R)$ and $M \gg N \mid M \gg R$

Additional Laws

Provided both sides are well-formed:

(Distributivity over $\gg$)

$$\{f \gg g \text{ where } x{:}\in h\} = \{f \text{ where } x{:}\in h\} \gg g$$

(Distributivity over $|$)

$$\{f \mid g \text{ where } x{:}\in h\} = \{f \text{ where } x{:}\in h\} \mid g$$

(Distributivity over where)

$$\{\{f \text{ where } x{:}\in g\} \text{ where } y{:}\in h\}$$
$$= \{\{f \text{ where } y{:}\in h\} \text{ where } x{:}\in g\}$$

Program Structuring: Running an Auction

- Advertize the item and a minimum bid price v: call $Adv(v)$
- Get bids: $Bids(v)$ returns a stream of increasing bids, all above v.
- Post successive bids at a web site: call $PostNext$

$$Auction_1(v) \;\underline{\Delta}$$
$$Adv(v) \gg Bids(v) \; >u> \; PostNext(u) \gg \mathbf{0}$$

Program $Bids$

Get the next bid exceeding v.
Assume that bidders put their bids on channel c.

$$
\begin{array}{l}
nextBid(v) \;\underline{\Delta} \\
\qquad c.get \\
\quad >x> \\
\qquad \{ \quad if\,(x > v) \;\gg\; let(x) \\
\qquad\;\; \mid\; if\,(x \le v) \;\gg\; nextBid(v) \\
\qquad \}
\end{array}
$$

Output successively increasing bids, all above v.

$$Bids(v) \;\underline{\Delta}\; nextBid(v) \;>u>\; (let(u) \;\mid\; Bids(u))$$

A Terminating Auction

- Terminate if no higher bid arrives for an hour (h time units).
- Post the winning bid by calling $PostFinal$.
- Return the value of the winning bid.

$$
\begin{array}{lll}
Auction_2(v) \ \underline{\Delta} & & \\
 & & Adv(v) \\
 \gg & & Tbids(v) \\
 >(x,b)> & & \\
 & & \{ \quad if\,(b) \quad \gg PostNext(x) \gg \mathbf{0} \\
 & & \ \ \mid \ if\,(\neg b) \quad \gg PostFinal(x) \gg let(x) \\
 & & \}
\end{array}
$$

$Tbids$

$Tbids(v)$ returns a stream of pairs (x, b):
x is a bid, $x \geq v$, and b is boolean.

$b \quad \Rightarrow x$ exceeds the previous bid
$\neg b \quad \Rightarrow x$ equals the previous bid,
i.e., no higher bid has been received in an hour.

$Tbids(v) \ \underline{\Delta}$
$\{let(x, b) \ | \ if\,(b) \gg Tbids(x)$
where
$(x, b) :\in \ nextBid(v) \ >u> \ let(u, \textit{true})$
$| \ Rtimer(h) \ \gg \ let(v, \textit{false})$
$\}$

Batch Processing the Bids

- Post higher bids only once each hour.
- As before, terminate if no higher bid arrives for an hour.
- As before, post the winning bid by calling $PostFinal$.
- As before, return the value of the winning bid.

$$
\begin{array}{lll}
Auction_3(v) \ \underline{\Delta} & & \\
 & & Adv(v) \\
\gg & & Hbids(v) \\
>(x,b)> & & \\
 & & \{ \quad if\,(b) \quad \gg PostNext(x) \quad \gg \mathbf{0} \\
 & & \quad | \quad if\,(\neg b) \quad \gg PostFinal(x) \quad \gg let(x) \\
 & & \}
\end{array}
$$

$Hbids$

$Hbids(v)$ returns a stream of pairs (x, b), one per hour:
x is a bid, $x \geq v$, and b is boolean.

$b \quad \Rightarrow x$ is the best bid in the last hour and exceeds the last bid
$\neg b \quad \Rightarrow x$ equals the previous bid,
i.e., no higher bid has been received in an hour.

$$
\begin{array}{ll}
Hbids(v) \ \underline{\Delta} & \\
 & clock \\
>t> & bestBid(t+h, v) \\
>x> & \\
 & \{ \quad let(x, x = v) \\
 & \quad \mid \ if(x \neq v) \gg Hbids(x) \\
 & \}
\end{array}
$$

$bestBid$

- $bestBid(t, v)$ where t is an absolute time and v is a bid,
- Returns x, $x \geq v$, where x is the best bid received up to t.
- If $x = v$ then no better bid than v has been received up to t.

$$
\begin{array}{l}
bestBid(t, v) \;\underline{\Delta} \\
\quad \{ if(b) \gg bestBid(t, y) \;\;|\;\; if(\neg b) \gg let(v) \\
\qquad \text{where} \\
\qquad\quad (y, b) :\in \;\; nextBid(v) \;\; >x> \;\; let(x, \textsf{true}) \\
\qquad\qquad\qquad |\;\; Atimer(t) \;\; >x> \;\; let(x, \textsf{false}) \\
\quad \}
\end{array}
$$