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`

Lectures for NATO Summer School, Marktoberdorf
August, 2004

Computation Orchestration

Given are basic computing elements. How to compose them?

- Computing elements are logic gates: $\wedge$, $\vee$, $\neg$

  Composition is a circuit.

- Computing elements are functions.

  Composition is through higher-order functions.

- Computing elements are processes.

  Composition is through CCS or CSP operators.

Orc

Computing elements are Sites, such as

- 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

Structure of the Lectures

- Programming Notation: the composition operators, their usage
- Programming Methodology: Parallelism, Synchronization, Interrupt
- Semantics, Implementation
- Site Specification, Commitment, Revocation

Other Possible Topics:

- Program Structuring
- Concurrency

Lecture Material

Computation Orchestration: A Basis for Wide-Area Computing

http://www.cs.utexas.edu/users/psp/Wide-area.pdf

Exercises in your Handouts

I will give additional exercises during the lecture.

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.

Orc

A new kind of assignment

$x :\in f$

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

Evaluation of f yields zero or more values.

Assign the first value to x.

An Orc expression is

- Simple: Site (Function call, method, web service, transaction)
- Compound: $f \mid g$, $f \gg g$, $f * g$, $\{ f \ \mathbf{where}\ x :\in g \}$

Simple Orc Expression

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

  $x :\in M(d)$

- 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.

Properties of 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.

Structure of response

- The response from a site has:
  value, which the programmer can manipulate, and
  pledge, which the programmer cannot manipulate.
- Pledge is used to commit this site call.
  Pledge is valid for some time period.
  Value is meaningful during then.
- By committing a valid pledge (during the given period), the programmer establishes some fact.

Nesting

- (Data Piping) Retrieve a news page for date d from M and email it to address a. Here, $Email$ is a site.

$$Email(a, M(d))$$

- (Higher-order site) Call discovery service D with parameter x to locate a site; call that site with parameter y.

$$Apply(D(x), y)$$

Simple Orc Expression: Sequencing

M, N, R are sites for 3 professors.

s is a set of possible meeting times.

$M(s)$ is a subset of s, the times when M can meet.

$M(N(R(s)))$ is the possible meeting times of all three professors.

Parallel, Strict evaluation

Arguments of a site call are evaluated in parallel.

A site is called only after all its arguments have been evaluated.

Fork-join parallelism

$A(c)$ and $B(c)$ return ticket prices from airlines A and B.

Min returns the minimum of its arguments.

$Min(A(c), B(c))$:

Compute $A(c)$ and $B(c)$ in parallel.

Call Min when both quotes are available.

Predefined sites

- $Fail$ never responds.

- $let(x, y, \cdots)$ returns a tuple of argument values as soon they are available. $let(\theta)$ is $skip$.

- $random$ returns a random number (in a specified range), instantaneously.

- fst returns the value of the first argument as soon all argument values are available.

- $timer(t)$, where t is a non-negative integer, returns a signal exactly after t time units.

- $timer(t, x)$ is $fst(x, timer(t))$; returns x after t time units.

Composing Expressions

- (Alternation) $f \mid g$: evaluate f and g in parallel; values of $f \mid g$ are those from f and from g.

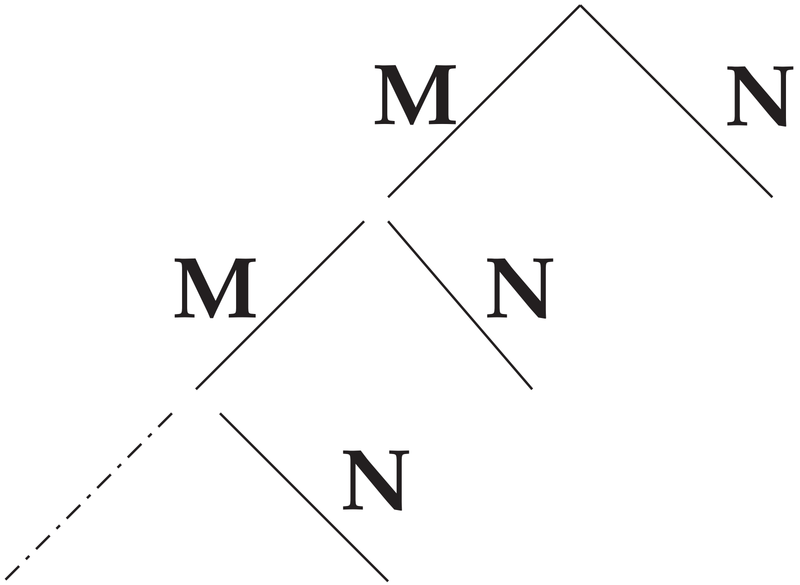
- (Piping) $f \gg g$: Evaluate g for all values of f; values of $f \gg g$ are those from g.

- (Iteration) $f * g$: values from g after zero or more piping steps of f.

$$
\begin{array}{ll}
 & f * g \\
= & g \mid (f \gg (f * g)) \\
= & g \mid (f \gg (g \mid f \gg (g \mid f \gg \cdots)))
\end{array}
$$

- (Definition) $\{\ f \ \mathbf{where}\ x{:}{\in}\ g \}$

Pictorial Depiction of $M * N$



Values of $M * N$ are the ones returned by N.

Binding power

$|$ has the lowest binding power.

$\gg$ and $*$ have equal binding powers.

$$f * g \mid h \gg g \;\equiv\; (f * g) \mid (h \gg g)$$

Example of Orc expression:

$$G(q) \gg (\langle M(q) \mid R(\theta, q) \gg G(\theta) \rangle * S(\theta))$$

Default Parameter

- $M \gg N(x, \theta)$
- $(M \mid S) \gg (N(x, \theta) \mid R(\theta))$
- Start computation of f with value v for θ:
  $let(v) \gg f$.
- Start an iteration where $x_0 = v$ and $x_{i+1} = M(x_i)$.
  Values returned are $N(x_i)$, for $i \geq 0$.

$$let(v) \gg (M(\theta) * N(\theta))$$

Properties of the timer

$$
\begin{array}{lcl}
x :\in \; timer(t) \;\mid\; timer(u) & \equiv & x :\in \; timer(t)\text{, given } t \leq u\text{,} \\
timer(t) \;\gg\; timer(u) & \equiv & timer(t+u)
\end{array}
$$

Alternation, Piping

- Assign the first value from $M(c)$ or $N(d)$ to z.

  $$z :\in M(c) \mid N(d)$$

- assign to z the value from M if it arrives before t, 0 otherwise.

  $$z :\in M \mid timer(t,0)$$

- Interruption

  $$f \mid Interrupt.get$$

- Make four requests to site M, in intervals of one time unit each.

  $$M \mid timer(1) \gg M \mid timer(2) \gg M \mid timer(3) \gg M$$

Priority

Request M and N for values. Give priority to M.

- Allocate one extra time unit for M to respond.

  $z :\in M \mid timer(1) \gg N$ or $z :\in M \mid N \gg timer(1, \theta)$

- Accept the response from M if it arrives within one time unit, else accept the first response.

  $z :\in M \mid fst(N, timer(1))$

Iteration

- Call M forever at unit time intervals until it returns a value.

  $$z{:}\in timer(1) * M$$

  which is

  $$z{:}\in M \mid timer(1) \gg (M \mid timer(1) \gg (M \mid \cdots))$$

- Same as above, but stop calling after 10 time units.

  $$z{:}\in timer(1) * M \;\mid\; timer(10)$$

Iteration; Contd.

- Site M returns stock price of company abc

  Site $C(x)$: returns x if $x < 20$; silent otherwise.

  $$M * C(\theta)$$

  either never returns a value (if abc never falls below 20)

  or returns a value lower than 20. Initially, $\theta \geq 20$.

- Variation: Poll M once every hour for 6-hours:

  $$timer(1) * \langle M \gg C(\theta) \rangle \mid timer(6)$$

Definition within Orc expression

- A machine is assembled from two parts, u and v.
- Two vendors for each part: $u1$ and $u2$ for u, and $v1$ and $v2$ for v.
- Solicit quotes from all vendors.
- Accept the first quote for each part.
- Compute the machine cost to be 20% above the sum of the part costs.

$$
\begin{array}{l}
cost{:}\in\ \{\ (u+v)\times 1.2 \\
\qquad \textbf{where} \\
\qquad\qquad u{:}\in u1 \mid u2 \\
\qquad\qquad v{:}\in v1 \mid v2 \\
\quad \}
\end{array}
$$

General Orc Statements

```
$z :\in \{ \; f(\cdots x \cdots y \cdots)$
      where
          $x :\in g$
          $y :\in h$
      $\}$
```

Example: M , N , R , S are sites.

```
$z :\in \{ \; (M(x) \mid N(y)) \gg M(y)$          $\gg \; \{ \; M(y)$
        where                                              where
            $x :\in R(y) \mid N(y)$                            $y :\in S$
            $y :\in \{ \; R \mid N(t)$                         $\}$
                   where   $t :\in S$
                $\}$
      $\}$
```

Syntax

$$
\begin{array}{lll}
\text{statement} & ::= & \text{defn} \\
 & & \\
\text{defn} & ::= & \text{variable} \;\; :\in \;\; \text{expr} \\
 & & \\
\text{expr} & ::= & \quad \text{term} \\
 & & \mid \;\; \text{expr} \;\; | \;\; \text{expr} \\
 & & \mid \;\; \text{expr} \;\; \gg \;\; \text{expr} \\
 & & \mid \;\; \text{expr} \;\; * \;\; \text{expr} \\
 & & \mid \;\; \{ \; \text{expr} \;\; \textbf{where} \;\; \text{defn} \; \} \\
 & & \\
\text{term} & ::= & \text{site}([\text{parameter}]) \\
 & & \\
\text{parameter} & ::= & \text{variable} \;\; \mid \;\; \theta
\end{array}
$$

[parameter] is a list of parameters, possibly empty.

Free and Bound Variables

- Variable assigned in a statement is the goal variable.

- Variables named in an expression are global or local.

- Free variables:

$$\begin{aligned}
\mathit{free}(M(L)) &= \{x \mid x \in L,\ x \neq \theta\} \\
\mathit{free}(f\ \mathit{op}\ g) &= \mathit{free}(f) \cup \mathit{free}(g), \text{ where } \mathit{op} \in \{\ |\ ,\ \gg\ ,\ *\ \} \\
\mathit{free}(\{f \ \ \textbf{where} \ \ x{:}{\in}\ g\}) &= (\mathit{free}(f) - \{x\}) \cup \mathit{free}(g)
\end{aligned}$$

- In $\{\ f$ **where** $x{:}{\in}\ g\}$, any free occurrence of x in f is bound to the variable shown.

- $z{:}{\in}\ f$ is well-formed if all the free variables in f are global variables.

Flat Expression

Flat expression: without a **where** clause.
Non-flat expression:

Flat expression is a regular expression of language theory.

Terms are symbols.

Syntactic Conventions: Omit Braces; group where

```
{{ $f$
       where
           $x$ :∈ $g$
   }
   where
       $y$ :∈ $h$
}
```

is

```
  $f$
  where
     $x$ :∈ $g$
     $y$ :∈ $h$
```

or, $\{$ f **where** x:∈ g, y:∈ $h\}$

Syntactic Conventions: Nested Site Calls

- $Email(a, M(d))$ is not an expression.

  It means:

  $\{\ Email(a, u)$ **where** $u{:}\in M(d)\,\}$

- We allow $R(f, g)$ where f and g are expressions.
  It means : $\{\ R(x, y)$ **where** $x{:}\in f,\ y{:}\in g\,\}$

- $timer(f, g)$ is (after f_0 time units return g_0)

  $\{\ fst(x, y)$
  where $x{:}\in g$
  $y{:}\in \{timer(u)$ **where** $u{:}\in f\,\}$
  $\}$

Argument Evaluation in Nested Site Calls

Consider $Q(N(x), N(x), N(x))$.

For the first two arguments:
evaluate $N(x)$ once and use the value for both.

For the last argument:
reevaluate $N(x)$.

```
{ $Q(u, u, v)$
    where
        $u :\in N(x)$
        $v :\in N(x)$
}
```

Operational Semantics

```
$z :\in \{A(x) \mid B(y)$                    $\gg$        $\{\ C(p, \theta)$
      where                                                  where
          $x :\in M \mid R$                                   $p :\in N$
          $y :\in N$
   $\}$                                                  $\}$
```

- Execute the defns of x, y and evaluate $A(x) \mid B(y)$, all in parallel.
- Suspend evaluation of $A(x) \mid B(y)$ until x or y gets a value.
- When x gets a value, resume evaluation of $A(x)$.
- When y gets a value, resume evaluation of $B(y)$.
- Suppose $A(x)$ returns v. Evaluate $C(p, v)$. Start with $p :\in N$.

Execution Rules

- State: Variable, value pair. Value for θ in every state.

  $p.x$ is the value of x in state p.

- In the initial state only globals and θ have values.

- Rules describe the bag of values computed for expression f, by structural induction on f.

$$\begin{array}{lcl}
\text{expr} & ::= & \text{term} \\
 & \mid & \text{expr} \mid \text{expr} \\
 & \mid & \text{expr} \gg \text{expr} \\
 & \mid & \text{expr} \ast \text{expr} \\
 & \mid & \{\ \text{expr}\ \textbf{where}\ \text{defn}\ \}
\end{array}$$

Execution Rules; Starting state p

- term $M(x, y)$: call M with parameters $p.x$ and $p.y$.
  - M never responds: computation never terminates.
  - M responds with value v: Only one result state q, $q.\theta = v$ and $q.x = p.x$ for all other x.
- $f \mid g$: evaluate f and g in state p, in parallel.

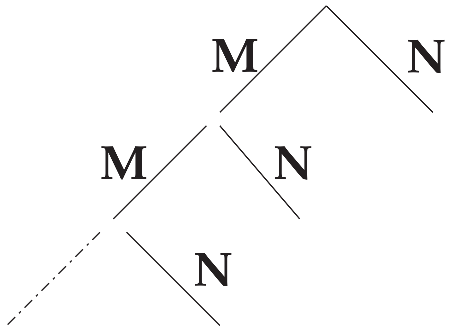
  The (bag of) result states are the ones returned by both f and g.
- $f \gg g$: evaluate f in p. For each result state q, evaluate g in q.

  Value computed for f is in $q.\theta$. Value of $f \gg g$ are the states returned by g.

Execution Rule for $f * g$ in state p

Evaluate both f and g in p and in any q returned by f.

The result states are the ones returned by g.



Values are returned by N.

Execution Rule for $\{\ f$ **where** $x{:}{\in}\ g\}$ in state p

- Evaluate f and g in state p, in parallel.
- When g returns state q, augment p with $(x, q.\theta)$.
- In evaluating f, if we need the value of x:

  wait until the value is available (in an augmented state).
- Result states of $\{\ f$ **where** $x{:}{\in}\ g\}$ are from f.

  Remove the tuple for x from the state because x is not defined outside this scope.

Execution Rule for $z{:}\in f$ in state p

- Evaluate f in p.
- Any result state, q, has the same set of variables as p, and $p.x = q.x$, for all x except θ; the result of evaluation is $q.\theta$.
- Let r be the first result state. Augment p by $(z, r.\theta)$, and return this as the result state of $z{:}\in f$.
- If f never responds, state p is never augmented.

Fork-Join parallelism

$$\begin{array}{lcl} z :\in fst(\textit{true}, x) & \quad | \quad & fst(\textit{false}, x) \\ \quad \textbf{where} & & \textbf{where} \\ \qquad x :\in timer(1) & & \qquad x :\in timer(2) \end{array}$$

z is assigned *true* after 1 time unit.

Angelic Nondeterminism

In $(M \mid N) \gg R$, R may be called twice. We have

$$(M \mid N) \gg R \;=\; M \gg R \mid N \gg R,$$

More generally, Right Distributivity of $\gg$ over $\mid$:

$$(f \mid g) \gg h \;=\; (f \gg h \mid g \gg h)$$

Demonic Nondeterminism; where clause

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

is *not* equivalent to

$$let(x) \gg M$$

where

$$x {:}{\in}\ N \mid R$$

Idempotence and Left Distributivity do not hold

Following laws do not hold.

(Idempotence of $\mid$)	$f \mid f \;=\; f$
(Left Distributivity of $\gg$ over $\mid$)	$f \gg (g \mid h) \;=\; (f \gg g) \mid (f \gg h)$

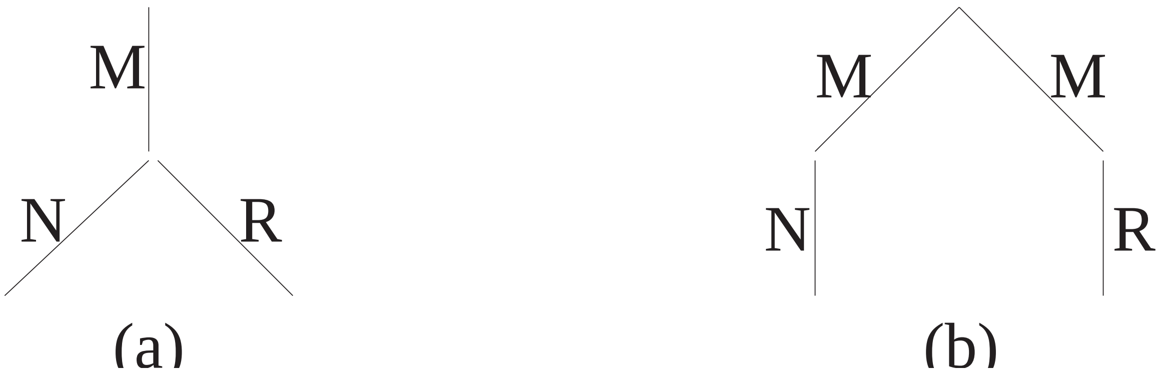


Figure 1: Schematic for $M \gg (N \mid R)$ and $M \gg N \;\mid\; M \gg R$

Parallel or

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

$$z :\in ift(x) \mid ift(y) \mid or(x, y)$$

where

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

Similarly, evaluate any function f of the form

$$f(x, y) = \begin{cases} p(x) & \text{if } c(x) \\ q(y) & \text{if } d(y) \\ r(x, y) & \text{otherwise} \end{cases}$$

Eight queens

- 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.
- $check(x{:}xs)$: Given xs valid, return

  $x : xs$, if it is valid

  remain silent, otherwise.

Eight queens; Contd.

$$
\begin{array}{l}
let([]) \\
\gg\ \langle check(0:\theta) \mid check(1:\theta) \mid check(2:\theta) \cdots \mid check(7:\theta)\rangle \\
\gg\ \langle check(0:\theta) \mid check(1:\theta) \mid check(2:\theta) \cdots \mid check(7:\theta)\rangle \\
\gg\ \langle check(0:\theta) \mid check(1:\theta) \mid check(2:\theta) \cdots \mid check(7:\theta)\rangle \\
\gg\ \langle check(0:\theta) \mid check(1:\theta) \mid check(2:\theta) \cdots \mid check(7:\theta)\rangle \\
\gg\ \langle check(0:\theta) \mid check(1:\theta) \mid check(2:\theta) \cdots \mid check(7:\theta)\rangle \\
\gg\ \langle check(0:\theta) \mid check(1:\theta) \mid check(2:\theta) \cdots \mid check(7:\theta)\rangle \\
\gg\ \langle check(0:\theta) \mid check(1:\theta) \mid check(2:\theta) \cdots \mid check(7:\theta)\rangle \\
\gg\ \langle check(0:\theta) \mid check(1:\theta) \mid check(2:\theta) \cdots \mid check(7:\theta)\rangle
\end{array}
$$

$$
\begin{array}{l}
let([]) \gg \langle \gg i : 0 \le i \le 7 : \\
\qquad\qquad \langle \mid j : 0 \le j \le 7 :\ check(j:\theta)\rangle \\
\qquad \rangle
\end{array}
$$

Local object

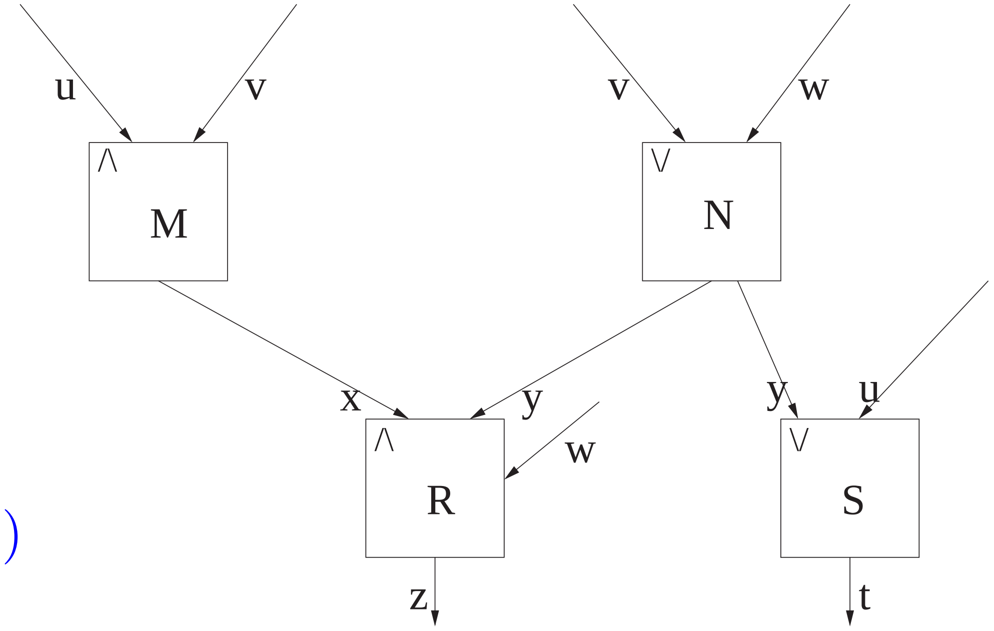
- Call sites M, N and R.
- Terminate after receiving two response.

Object $count$ with integer state. Initially, 0.

- $count.incr$ increments state;
- returns a signal if state ≥ 2, otherwise, remains silent.

$$
\begin{array}{l}
c :\in \\
\qquad M \gg count.incr \\
\quad | \; N \gg count.incr \\
\quad | \; R \gg count.incr
\end{array}
$$

And-Or graph



r:∈ $let(z, t)$

where

z:∈ $R(x, y, w)$

t:∈ $S(y) \mid S(u)$

where

x:∈ $M(u, v)$

y:∈ $N(v) \mid N(w)$

Airline

- Return any quote, from A or B, provided it is below 300.
- If neither quote is below 300, then return the cheapest quote or any quote available by time t.
- If no quote is available by t, return ∞.

Min returns the minimum of its argument values.

$threshold(x)$ returns x if x is below 300; silent otherwise.

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

where

$$\begin{array}{l} x :\in A \mid timer(t, \infty) \\ y :\in B \mid timer(t, \infty) \end{array}$$

Workflow: Visit Coordination

- $Email(p, s)$: contact p with dates s; response is date d from s.
- $Hotel(d)$: booking from hotel.
- $Airline(d)$: booking from airline.
- $Ack(p, t)$: similar to $Email$; response is an acknowledgment.
- $Confirm(t)$: confirm reservation t (for hotel or airline).
- $Room(d)$: reserve room for d. Response q: room number, time.
- $Announce(p, q)$: announce the lecture.
- $AV(q)$: contact technician with room and time information in q.

Workflow; Contd.

$z :\in let(b)$

where

$b :\in Ack(p, h, f)$

$\gg let(c, e)$

where

$c :\in Confirm(h)$

$e :\in Confirm(f)$

$\gg let(u, v)$

where

$u :\in Announce(p, q)$

$v :\in AV(q)$

where

$q :\in Room(d)$

where

$h :\in Hotel(d)$

$f :\in Airline(d)$

where

$d :\in Email(p, s)$

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

$$z :\in f \mid Interrupt.get$$

Processing Interrupt

$$z :\in \{ \ f(x,y) \quad \textbf{where} \quad x :\in g, \ \ y :\in h \ \}$$

If f is interrupted, call M and N with parameters x and y, respectively, to cancel the effects of g and h.

$$z :\in \ \mathit{Normal}(t) \ | \ \mathit{Interr}(t) \gg \mathit{let}(X,Y)$$

where

$$X :\in M(x)$$
$$Y :\in N(y)$$

where

$$t \ :\in f(x,y) \ | \ \mathit{Interrupt.get}$$

where

$$x \ :\in g$$
$$y \ :\in h$$

Phase Synchronization

Process starts its $(k+1)^{th}$ phase only after all processes have completed their k^{th} phases.

Consider $M \gg f$ and $N \gg g$.

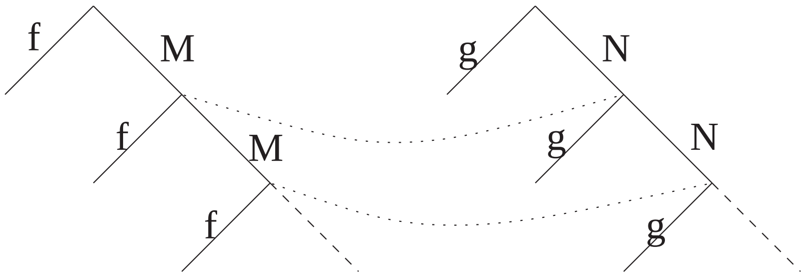
$$\{let(x, y)$$

where

$$x {:\in}\ M$$
$$y {:\in}\ N\}$$
$$\gg$$
$$fst(\theta) \gg f \ | \ snd(\theta) \gg g$$

Phase Synchronization; Contd.

Synchronize $M * f$ and $N * g$.



$$\{let(x, y)$$

where

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

$$*$$

$$fst(\theta) \gg f \mid snd(\theta) \gg g$$

Heat transfer computation over a grid

- Value x_{ij} at point (i,j) in a phase is the average of its neighbors' values in the previous phase.
- Site $average$ returns the average of its arguments.
- Site $converge$ returns its argument value if the values have converged sufficiently, otherwise, it remains silent.

```
$z :\in \{let(x)$
        where
            $\langle \forall i,j :: x_{ij} :\in average(\theta_{i-1,j}, \theta_{i+1,j}, \theta_{i,j-1}, \theta_{i,j+1}) \rangle$
      $\}$
  $* \; converge(\theta)$
```

Environment

An environment is a set of tuples. Each tuple has:

- a name (of a variable or θ),
- a val component (its value) and
- a $clock$ component, the time at which this value was computed.

Example: p: $\langle(x, \textit{false}, 27), (y, \textit{true}, 12), (\theta, 13, 20)\rangle$

$p.x.val = \textit{false}$ and $p.x.clock = 27$.

An environment is a statement about a computation:
the value of x, computed at time 27, is *false*, and the value of y,
computed at time 12, $\cdots$.

Relation over Bags of Environments

Each expression and defn is a binary relation over bags of environments.

Notation: P, Q are bags of environments. $\cup$ is bag union.

Write $P\ f\ Q$ and $P\ (z{:}{\in}\ f)\ Q$.

Coercion rule: $$\frac{\langle \forall p : p \in P : \{p\}\ f\ Q_p \rangle}{P\ \ f\ \ \langle \cup p : p \in P : Q_p \rangle}$$

Consequently, we need only consider $\{p\}\ f\ Q$.

Note: $\{\}\ f\ \{\}$

Relation over Bags of Environments; Contd.

$\{p\}\ f\ Q$: evaluation of f started in p at time $p.\theta.clock$ yields all environments in Q in some computation.

Q may be empty: non-terminating computation.

Q may have duplicates: as in evaluating $M \mid M$.

Example

p: $\langle(x, \textsf{\textit{false}}, 27), (y, \textsf{\textit{true}}, 12), (\theta, 13, 20)\rangle$.

$\{p\}$ $let(x, y)$ $\{\langle(x, \textsf{\textit{false}}, 27), (y, \textsf{\textit{true}}, 12), (\theta, (\textsf{\textit{false}}, \textsf{\textit{true}}), 27)\rangle\}$

$\{p\}$ $timer(2)$ $\{\langle(x, \textsf{\textit{false}}, 27), (y, \textsf{\textit{true}}, 12), (\theta, SIGNAL, 22)\rangle\}$

$\{p\}$ $u{:}{\in}\ let(x, y)$
$\{\langle(x, \textsf{\textit{false}}, 27), (y, \textsf{\textit{true}}, 12), (u, (\textsf{\textit{false}}, \textsf{\textit{true}}), 27), (\theta, 13, 20)\rangle\}$

$\{p\}$ $let(z)$ $\{\}$, because z is not defined in p.

$\{p\}$ $u{:}{\in}\ let(z)$ $\{p\}$

Semantics of Term

Evaluate $M(L)$ in environment p. Result is at most one environment.

- $x \in L$ and $x \notin p$: no result environment.
- Otherwise: call M with values $p.x.val$ for all x in L, at maximum of $p.\theta.clock$ and clock values of all parameters in L.
- If M responds with value v at time t, the result environment is q, where

  $q.x = p.x$, for all x in p, $x \neq \theta$
  $q.\theta.val = v$
  $q.\theta.clock = t$

Axioms about terms

Notation: $p \backslash x$: remove the tuple for x from q. $x \notin p \Rightarrow p \backslash x = p$.

- $\{p\}\ M(L)\ \{\}$, if $x \in L$ and $x \notin p$.
- Given $\{p\}\ M(L)\ \{q\}$:
  - $q.x = p.x$, for all x in p, $x \neq \theta$, and $q.\theta.clock \geq p.\theta.clock$
  - if $x \notin L$, then $\{p \backslash x\}\ M(L)\ \{q \backslash x\}$
  - if $x \in L$, let $p' = p \backslash x$ except
    $p'.\theta.clock = \max(p.\theta.clock, p.x.clock)$.
    Then, $\{p'\}\ M(L[x := p.x.val])\ \{q \backslash x\}$
- (parameters may be renamed) For $y \notin p$ and $y \notin L$,

$$\{p\}\ M(L)\ \{q\}\ \equiv\ \{p[x := y]\}\ M(L[x := y])\ \{q[x := y]\}$$

Semantics of some sites

- $\{p\}\ \mathit{Fail}\ \{\}$
- $\{p\}\ \mathit{let}(x, y)\ \{q\}$:

$$q.\theta.val = (p.x.val, p.y.val)$$
$$q.\theta.clock = \max(p.\theta.clock, p.x.clock, p.y.clock)$$

In particular, $\{p\}\ \ \mathit{let}(\theta)\ \ \{p\}$.

- $\{p\}\ \mathit{random}\ \{q\}$:

$q.\theta.val =$ a number from the specified range

$q.\theta.clock = p.\theta.clock$

Semantics of some sites; Contd.

- $\{p\}\ fst(x, y)\ \{q\}$:

$$\begin{aligned} q.\theta.val &= p.x.val \\ q.\theta.clock &= \max(p.\theta.clock, p.x.clock, p.y.clock) \end{aligned}$$

- $\{p\}\ timer(t)\ \{q\}$:

$$\begin{aligned} q.\theta.val &= SIGNAL \\ q.\theta.clock &= p.\theta.clock + t \end{aligned}$$

Exercise: Properties of the timer

$$x{:}\in\ timer(t)\ \mid\ timer(u) \quad\equiv\quad x{:}\in\ timer(t)\text{, given }\ t \le u\text{,}$$
$$timer(t)\ \gg\ timer(u) \quad\equiv\quad timer(t+u)$$

Semantics of Defn

- $$\frac{\{p\}\ f\ \{\}}{\{p\}\ \ (z{:}{\in}\ f)\ \ \{p\}}$$

- $$\frac{\{p\}\ f\ Q,\ q\ ismin\ Q}{\{p\}\ \ (z{:}{\in}\ f)\ \ \{p + (z, q.\theta)\}}$$

$q\ ismin\ Q$: $q \in Q$ and $q.\theta.clock \leq r.\theta.clock$ for every r in Q.

$+$ denotes expansion of an environment by a tuple.

Semantics of Expression

- $$\frac{\{p\}\ (x{:}{\in}\ g)\ \{q\},\ \ \{q\}\ f\ Q}{\{p\}\ \{f\ \ \textbf{where}\ \ x{:}{\in}\ g\}\ (Q\backslash x)}$$

- $$\frac{\{p\}\ f\ Q,\ \ \{p\}\ g\ R}{\{p\}\ \ (f\ |\ g)\ \ (Q \cup R)}$$

- $$\frac{\{p\}\ f\ Q,\ \ Q\ g\ R}{\{p\}\ \ (f \gg g)\ \ R}$$

- $$\frac{\{p\}\ f\ Q,\ \ Q\ f^*\ R}{\{p\}\ \ f^*\ \ (\{p\} \cup R)}$$

$f * g \equiv f^* \gg g$

$f^* \equiv f\ *\ \ \mathbf{1}$, where $\mathbf{1} = let(\theta)$.

Notes

- $\gg$ is relational composition.
- $|$ is not relational union,

  $P\ f\ Q$ does not imply $P\ (f \mid g)\ Q$.

  Under relational union M and $M \mid M$ would be identical. We treat them differently.

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 $\mid$)	$(f \mid g) \gg h = (f \gg h \mid g \gg h)$
(Recursive Expansion of Kleene star)	$f^* = \mathbf{1} \mid f \gg f^*$

Corollaries

(Left Distributivity of $*$ over $|$) $f * (g \mid h) \;=\; (f * g \mid f * h)$

(Regrouping $*$ over $\gg$) $(f * g) \gg h \;=\; f * (g \gg h)$

Additional Properties of Non-flat Expressions

- (Narrowing the scope) Given that x is not free in g:

$$\begin{array}{lcl}
\{g \ \textbf{where} \ x{:}\in h\} & = & g \\
\{f \mid g \ \textbf{where} \ x{:}\in h\} & = & \{f \ \textbf{where} \ x{:}\in h\} \mid g \\
\{f \gg g \ \textbf{where} \ x{:}\in h\} & = & \{f \ \textbf{where} \ x{:}\in h\} \gg g
\end{array}$$

- (Bound variable renaming) In the following, y is not free in f or g.

$$\{f \ \textbf{where} \ x{:}\in g\} = \{f[x := y] \ \textbf{where} \ y{:}\in g\}$$

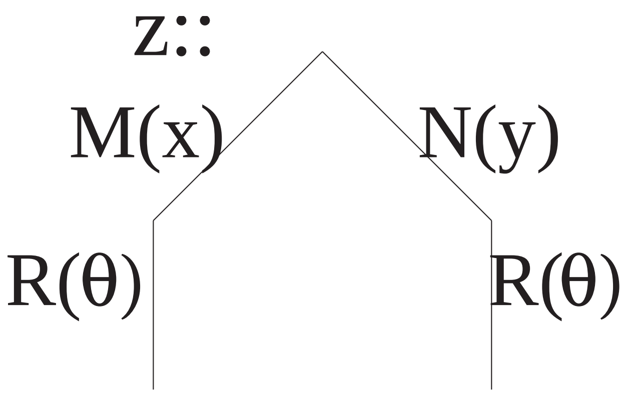
- (Independent defn) y is not free in g and x is not free in h.

$$\begin{array}{ll}
 & \{\{f \ \textbf{where} \ x{:}\in g\} \ \textbf{where} \ y{:}\in h\} \\
= & \{\{f \ \textbf{where} \ y{:}\in h\} \ \textbf{where} \ x{:}\in g\}
\end{array}$$

Implementation

- Compile the statement into a (set of) finite state automata
- explore the automata to compute the goal variable.

For $z{:}\in (M(x) \mid N(y)) \gg R(\theta)$

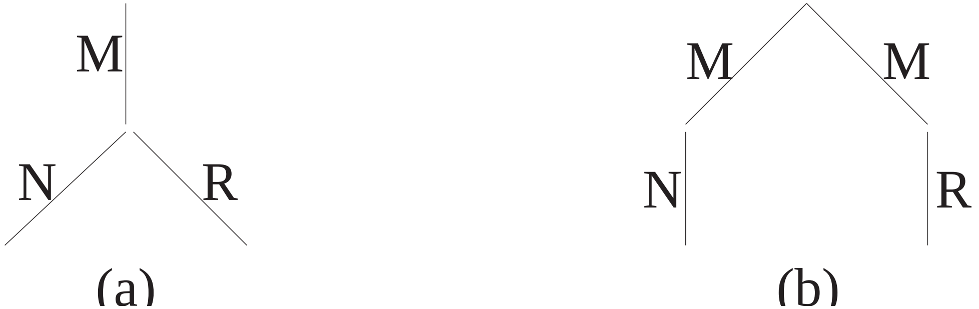


Difficulties In Automata construction

- How to compile a non-flat expression:

  $z{:}\in \{M(x) \mid N \;\; \textbf{where} \;\; x{:}\in R\}$

- We cannot use standard procedures. In automata theory, $M \gg (N \mid R) \equiv M \gg N \;\mid\; M \gg R$



- Non-determinism: M is called twice in $M \gg N \;\mid\; M \gg R$.

Flat program

Transform $\{f$ **where** $x{:}\in g\}$ to

$open(x) \gg f \gg close(x)$ and add the defn $x{:}\in g$.

```
z:∈ {M(x)                                  |        { R(u)
          where                                       where
             x:∈ {A(y)                                   u:∈ B}
                      where   y:∈ B}
     }
```

The corresponding flat program is

$$z{:}\in open(x) \gg M(x) \gg close(x) \mid open(u) \gg R(u) \gg close(u)$$
$$x{:}\in open(y) \gg A(y) \gg close(y)$$
$$y{:}\in B$$
$$u{:}\in B$$

$open$ and $close$

- $open$ and $close$ are treated differently from usual sites.
- Calling $open(x)$ starts a new computation of x on a clone.
- A state includes the values for the global variables and θ, but reference to clone for local variable.
- $close(x)$ removes reference to x from the state.

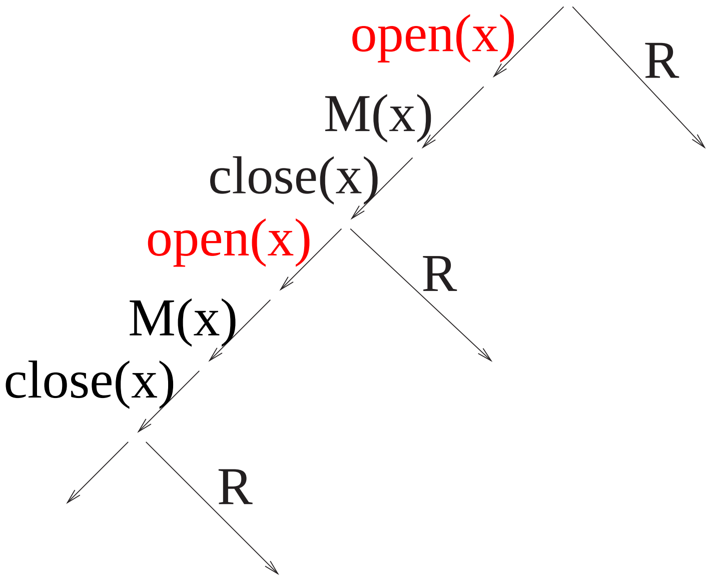
clone

Several clones of an fsa may be simultaneously in existence.

$z{:}\in\ \{\ M(x)$ **where** $x{:}\in N\}\ *\ R$ has the flat program

$z{:}\in \langle open(x) \gg M(x) \gg close(x)\rangle * R$

$x{:}\in N$



Finite State Automata (fsa) from Flat Program

- An Orc fsa is a finite directed graph.
- Its edges are labeled with terms (including $open$ and $close$).
- A pair of nodes in the fsa may have multiple edges between them, possibly with the same label.
- Two distinguished nodes: begin and end.
- No incoming edge to begin node; no outgoing edge of end.
- For every edge, there is a path from begin to end that includes the edge.

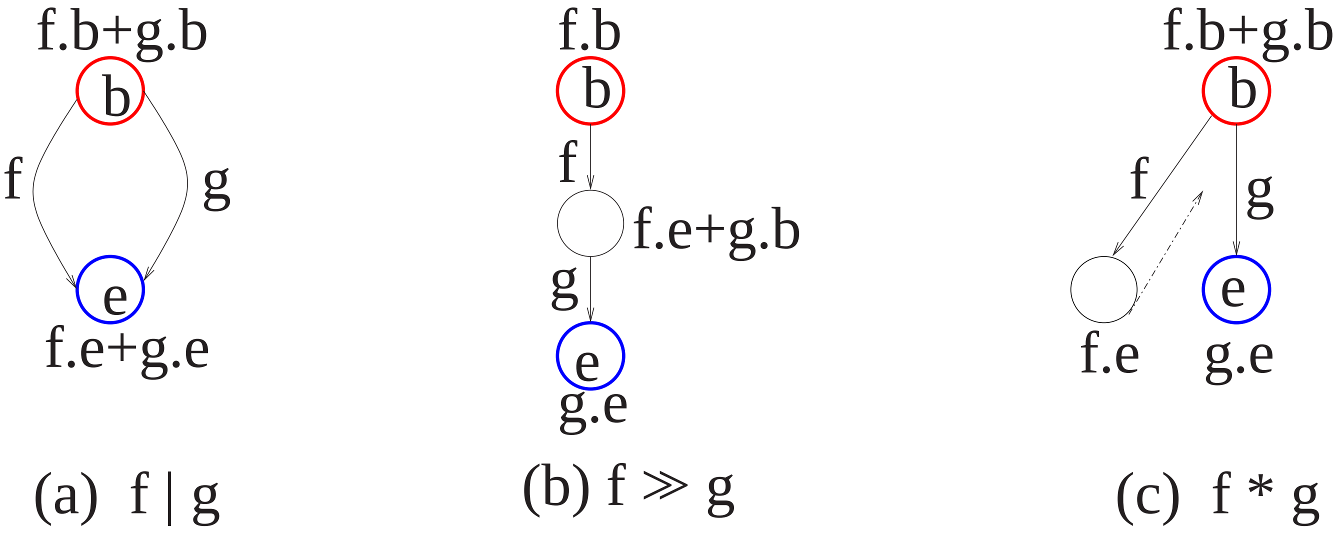
Recursive fsa Construction

merge of x, y is $x + y$: incoming, outgoing edges of x, y.

$f.b$ and $f.e$ for begin and end nodes of fsa for f.

term $M(L)$: The fsa has one edge from begin to end, labeled $M(L)$.

For edge from $(f * g).b$ to x with label r: make edge $(f.e, x)$ with r.

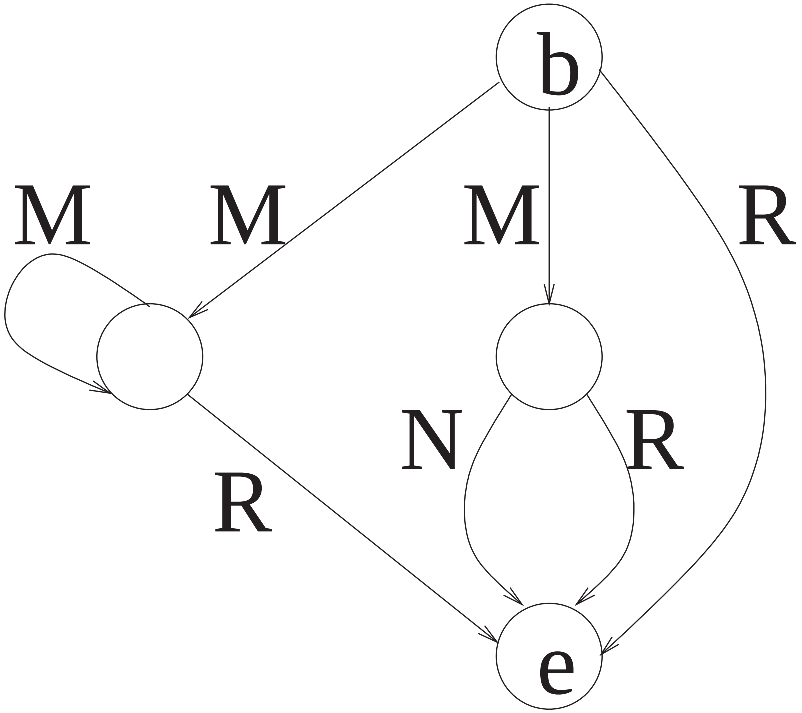


(a) f | g

(b) f ≫ g

(c) f * g

fsa for $M \gg (N \mid R) \mid M * R$



Notes on fsa

- Traditional deterministic fsa construction is P-space complete.

  Orc fsa construction is linear (see exercise).

- Each fsa can be unrolled to a (possibly infinite) tree.

fsa exploration; token

- token associated with an edge of a clone of an fsa.
- Corresponds to a single step of computation.
- token has a state and a parent. parent is a token in the same clone or NIL.
- token processes label $M(x)$ on edge e:

  wait until x has a value, then call site M. On receiving v from M:

  – creates children tokens on all successor edges of e,
  – bequeaths to them the state with θ value v,
  – if e has no successor edge, reports v as the value of the computation.

fsa exploration; token; contd.

Asssume: single outgoing edge from each begin node (begin edge).

Ensure by adding an edge with label $\mathbf{1}$.

- The token processes label $open(x)$ on edge e:
  - initiate computation on a clone of the fsa for x
  - return reference to the clone as part of the state
- $close(x)$: remove the reference to the clone for x from the state.
- To initiate computation on a clone in state s:

  place a token on its begin edge with state s and NIL parent.

Site

- Any level of granularity in site,

  from simple message transmission

  to business-business transaction involving many servers

- May spawn new processes, start servers and change database contents

- May interact with peripheral devices, including displays and keyboards

Site Specification

The specification of site M is a predicate p over a triple (x, y, t):

x is the value of actual parameters,

y is the result returned by M,

t is an (absolute) time instant.

Two stage Site Operation

A site operates in two stages.

- response: Client calls. Site returns y or remains silent.

  also returns a pledge which is invisible to the Orc statement.

  pledge carries a deadline by which it should be committed.

- commit: If the caller commits the pledge at time t before the deadline,

  the site executes its commit stage,

  which establishes predicate $p(x, y, t)$.

Examples of Site Specification

- Function f:

  returns y where $y = f(x)$. The deadline is irrelevant.

  predicate: $y = f(x)$. No commitment needed.

- Site $postOffice$:

  called with description of a parcel and returns y, the cost of delivery.

  The deadline is the instant t, the time of response.

  predicate: the cost of delivery of x at time t is y.

  needs no commitment to establish this predicate.

Examples of Site Specification; Contd.

- Object $count$: has an integer value $count.v$ and two methods, $incr$ and $read$.

- Initially, $count.v = 0$.

- $incr$: $count.v := count.v + 1$;
  $read$: returns v = $count.v$.
  The moment of response, t, is the deadline.

- predicate: $count.v \geq v$ beyond t. No commitment required.

- Another spec: $count.v = v$ at the moment of commitment s, $s \leq t$.

  Implement specification by: lock $count.v$ until the moment of commitment or t, whichever comes first.

Examples of Site Specification; Contd.

- Transaction sells 80 shares of stock pqr if price is above $25 and buys 100 shares of stock abc if it is below $20 a share.
- Both price conditions are met before the transaction returns a signal.
- The deadline is very short.
- If the client commits within the deadline, establishes the predicate:

  client has bought and sold the requisite number of shares for the given prices at the moment of commitment.

Examples of Site Specification; Contd.

- A site call may cause state change during the response stage.
- An airline issues a price quote and changes its state during the response stage, even if no call commits.
- Its only obligation is to issue a ticket at the given price if the client commits within the deadline.

Type of pledge

- Instant pledge: site makes immediate commitment for the caller

  The pledge can only be revoked next.

  Calling site $email$ sends an email, without waiting for commitment.

  Common in concurrent computing.

- Deferred pledge: Commitment by deadline establishes the associated predicate.

Structure of pledge

A pledge has:

- An id,
- A deadline, an absolute time instant by which it has to be committed or revoked (if already committed),
- A set of pertinent arguments, the arguments of the site call which must be committed in order to commit this pledge.

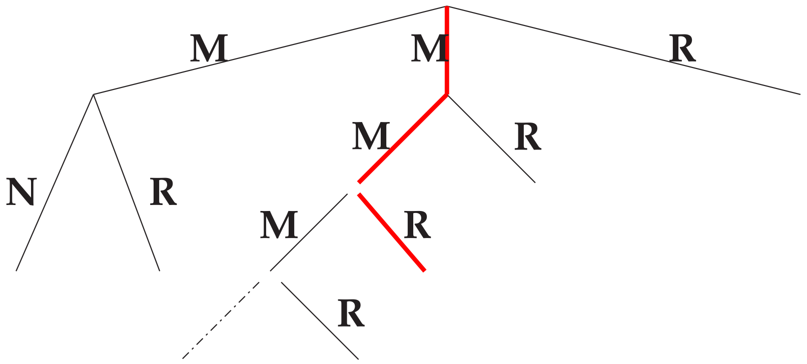
$z :\in Min(x, y)$
where
$x :\in A$
$y :\in B$

Pertinent argument from Min is the cheaper of x and y.

Critical Path

Response from a terminal edge, e, is assigned as the variable value.

Edge e defines the Critical Path.



Commit all pledges and their pertinent pledges along the critical path. Transitive closure needed.

Revoke all other instant pledges.

Definitions

For any clone c:

$child(c)$ = $\{d|$ d is a clone spawned by an $open$ in $c\}$

$pert(c)$ = $\{d|$ d is not a global variable or θ, and
d is a pertinent argument in a deferred pledge
along c's critical path$\}$

$instant(c)$ = instant pledge ids received in c

$deferred(c)$ = deferred pledge ids received in c

$instant^{+}(c)$ = instant pledge ids received along the critical path in c

$deferred^{+}(c)$ = deferred pledge ids received along the critical path in c

Transitive closure definitions

For any clone c:

$all\_instant(c)$ instant pledge ids received in c and its descendants.

$all\_deferred(c)$ deferred pledge ids received in c and its descendants.

$all\_instant^{+}(c)$ and $all\_deferred^{+}(c)$, limited to the critical paths.

$$\begin{array}{lcl}
all\_instant(c) & = & instant(c) \\
 & & \cup \ \langle \cup d : d \in child(c) : all\_instant(d) \rangle \\
all\_deferred(c) & = & deferred(c) \\
 & & \cup \ \langle \cup d : d \in child(c) : all\_deferred(d) \rangle \\
all\_instant^{+}(c) & = & instant^{+}(c) \\
 & & \cup \ \langle \cup d : d \in pert(c) \ : all\_instant^{+}(d) \rangle \\
all\_deferred^{+}(c) & = & deferred^{+}(c) \\
 & & \cup \ \langle \cup d : d \in pert(c) \ : all\_deferred^{+}(d) \rangle
\end{array}$$

Pledges to be Committed, Revoked

For any clone c:

$all^{+}(c)$: pledges necessary and sufficient to commit c (received by c and its descendants).

$$all^{+}(c) \quad = \quad all\_instant^{+}(c) \cup all\_deferred^{+}(c)$$

Instant pledges in $all^{+}(c)$ are already committed.

$pos(c)$: pledges which remain to be committed in $all^{+}(c)$; i.e.,

$$pos(c) \quad = \quad all\_deferred^{+}(c)$$

$neg(c)$: pledges which need to be revoked, i.e., already committed and not part of $all^{+}(c)$,

$$neg(c) \quad = \quad all\_instant(c) - all^{+}(c), \text{ i.e.,}$$
$$neg(c) \quad = \quad all\_instant(c) - all\_instant^{+}(c)$$

Commitment and Revocation Algorithm

- Client: sends to the appropriate sites,

  $commit_{init}$ for pledges in $pos(c)$,

  $revoke_{init}$ for pledges in $neg(c)$

- Site: responds with

  ack_{init} if it is ready to commit/revoke the pledge or

  $nack_{init}$ if it can not.

  Treat failure to respond (timely) as $nack_{init}$.

Commitment and Revocation Algorithm; Contd.

- Client:
  - If all responses are ack_{init}, sends $commit_{final}$ to sites corresponding to $pos(c)$ and $revoke_{final}$ to sites corresponding to $neg(c)$.
  - Otherwise, sends $abort_{final}$ to all sites.
- Site:

  commit after receiving $commit_{final}$

  revoke after receiving $revoke_{final}$,

  recovery computation after receiving $abort_{final}$.

Explicit Commit and Revoke

To handle transactions of differing deadlines.

Commit and Revoke sites.

A successful call to $Commit(x, y)$:

returns *true* and guarantees that x and y are committed with their pertinent variables. Similarly, $Revoke$.

Make explicit the commit in $z{:}\in f$

```
$z{:}\in$ { $Commit(y) \gg let(y)$
        **where**
            $y{:}\in f$
      }
```

Example

Reserve a hotel room and an airline ticket.

The hotel responds after a long delay but gives a long deadline.

The airline usually responds quickly but gives a short deadline.

Strategy

- Contact the hotel. After it responds, contact the airline.
- Airline responds before the hotel's deadline:

  if its quote is excessively high: cancel vacation plan and assign ∞ to the goal variable.

  Otherwise: commit to both and return the sum of the quotes as the goal variable value.

Example; Contd.

- Airline does not respond before the hotel's deadline: commit to the hotel; and wait 1 unit for the airline response.
  - Airline responds before the new deadline:
    if its quote is excessive: revoke the hotel commitment and cancel vacation plans.
    Otherwise: commit to the airline and return the sum of the quotes as the goal variable value.
  - Airline does not respond before the new deadline: revoke the hotel commitment and cancel vacation plans.

Example; Contd.

```
$z :\in$
    $\{ \quad exc(a) \gg let(\infty)$
    $\mid \neg exc(a) \gg Commit(a, h) \gg Plus(a, h)$
    $\mid let(t0) \gg Commit(h) \gg$
      $( \; (exc(a) \mid let(t1)) \gg Revoke(h) \gg let(\infty)$
       $\mid \neg exc(a) \gg Commit(a) \gg Plus(a, h)$
      $)$
    where
        $(h, d) :\in Hotel$
        $a :\in let(h) \gg Airline$
        $t0 :\in timer(d)$
        $t1 :\in let(t0) \gg timer(1)$
    $\}$
```