Lecture 18
CSP - Processes

David Garlan

Carnegie Mellon University

Models of Software Systems

15-671

Fall 1995

CSP: Four Lectures

- **CSP Processes**
  - > basic notions of processes
  - > prefix, recursion, choice, parallel combination
  - > deadlock
- **Reasoning with CSP**
  - > skip, input/output
  - > semantics: traces, specif
  - > laws of inference
- **Nondeterminism**
  - > external choice
  - > the need for a richer semantic model
- **Applications**
  - > pipes
  - > specification of software architectures

Next time:
Maybe move input/output to this lecture
do skip also

CSP: Overview

- Communicating Sequential Processes
  - > C.A.R. Hoare
- Based on events
  - > processes interact by coordinating events
  - > semantics defined by traces of events
- One of many similiar kinds of notations
  - > CCS
  - > Lotos
  - > Occam
- Has an associated algebraic theory
- Has compositional semantics

The Challenge of Concurrency

I added this and the next slide into this lecture. Needs integrating

- Consider state machines:

  S1, with m states and j transitions.

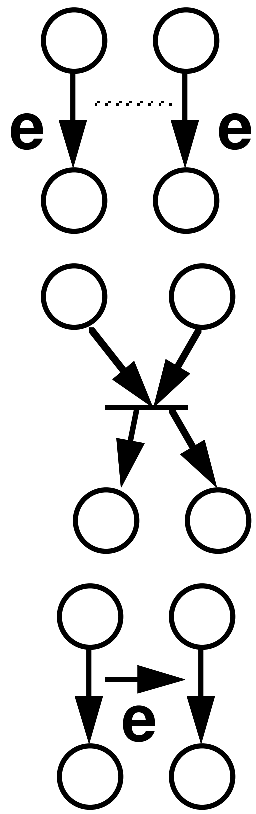
  S2, with n states and k transitions.

- If we "run these in parallel" we get a new state machine with up to n*m states and j*k transitions.
- This quickly leads to combinatorial explosion!
- Hence would like to find formalisms and models that

  (a) allow us to describe machines separately

  (b) reason compositionally about their combined (concurrent) behavior.

- **This is easy if the machines don't interact, since we can simply consider each machine independently.**
- **But in general machines influence each other.**
- **Some possible solutions:**
  - **<u>CSP</u>: consider the intersection of their alphabets as a set of *synchronizing events*: whenever one makes a transition so does the other.**
  - **<u>Petri Nets</u>: define *synchronizing barriers* that both machines must pass through.**
  - **<u>CCS</u>: define *input/output event pairs* ; a special operator (hiding) designates which events must be synchronized.**



A look back

- **Up to now, we have seen primarily state machines**
  - > State Machines
  - > Z
  - > Algebraic models
  - > Shared Variable Concurrency

Elaborate with actual topics covered

Transitions as State Relations

- Each of these describes conditions for changing from one *state* to another.
  - > Z = operations relate schemas
  - > State machine = precondition/postcondition
  - > Shared memory communication = local/global variables
- CSP describes <u>processes</u>, and focuses on the events that take place

$$\text{Process1} = \text{event} \rightarrow \text{Process2}$$

This slide needs to be fixed.
Not clear what the main point is

Events

- Events are simply names
- Examples
  - > coin
  - > chocolate
  - > output!2 (read output-bang-2)
  - > input?x (read input-query-x)
  - > handin-homework
- Each process has associated with it a set of events that it “understands”
  - > this is called it’s *alphabet*
  - > if P is a process αP is its alphabet

STOP: The Stuck Process

$STOP_A$

- **The process with alphabet A that never does anything when any event of A occurs.**
- **I.e., the broken process.**
- **Different from (succesfully) terminating process -- but that will come later.**

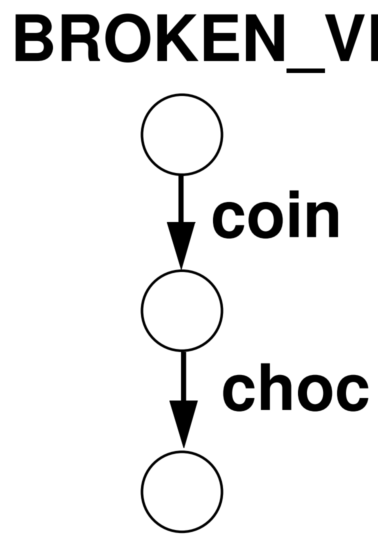
Prefix

$\rightarrow$

- **If P is a process and e is an event in its alphabet, then**
- **Q = e $\rightarrow$ P represents the process that does (or “engages in) e and then behaves like process P**
- **Verbally, we say “e *then* P”**
- **The operator $\rightarrow$ associates to the right, so a $\rightarrow$ b $\rightarrow$ STOP = a $\rightarrow$ (b $\rightarrow$ STOP)**

Prefix Example

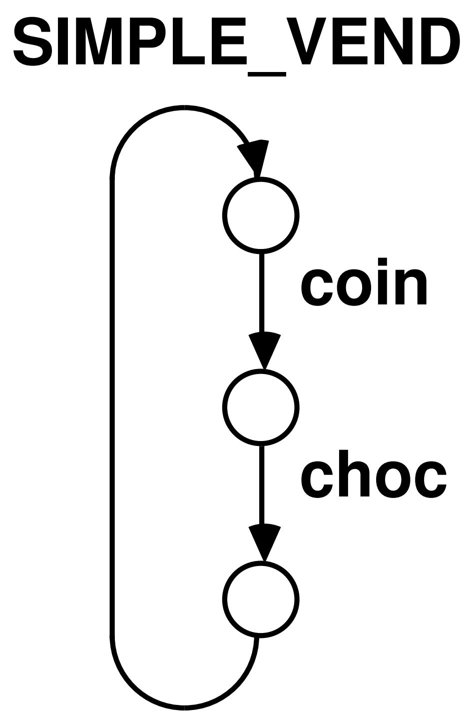
BROKEN_VEND = coin $\rightarrow$ (choc $\rightarrow$ STOP)



α BROKEN_VEND = {coin, choc}

Example (Better)

SIMPLE_VEND = coin $\rightarrow$ (choc $\rightarrow$ SIMPLE_VEND)



Note: definitions of the form P = process-expression may be recursive

More Simple Examples

1. Clock = tick $\rightarrow$ Clock

α Clock = {tick}

2. Clock = tick $\rightarrow$ tock $\rightarrow$ Clock

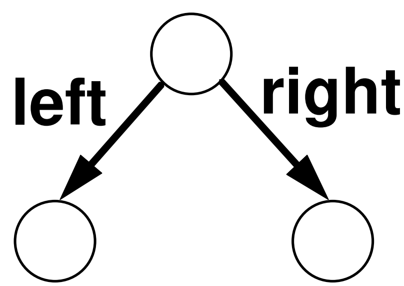
α Clock = {tick, tock}

Need more examples

Choice

| - the choice operator

DECIDE = (left →STOP) | (right →STOP)



Choice

The choice operator (|) combines two proceses, prefixed by a events.

If $a \rightarrow P$ and $b \rightarrow Q$ are processes

with $a \neq b$, then

$a \rightarrow P \mid b \rightarrow Q$ is a process that will do either an "a" event or a "b" event, and then behave like P or Q (resp.)

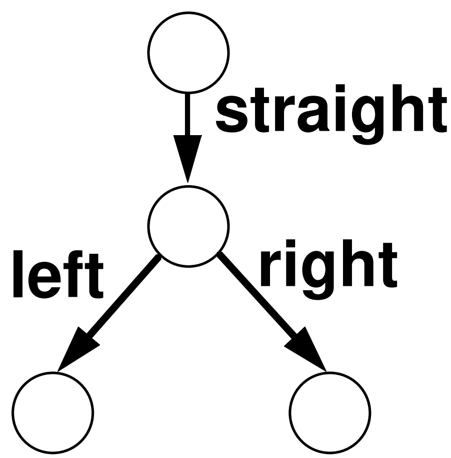
Verbally, we say "P *choose* Q"

May be used as an n-ary operator:

$a \rightarrow P \mid b \rightarrow Q \mid c \rightarrow R$

Combining operations

DECIDE2 = straight $\rightarrow$ (left $\rightarrow$ STOP | right $\rightarrow$ STOP)



Add a recursive version

More Interesting Example

VEND =

(in5 → 5IN) | (in10 → 10IN)

5IN =

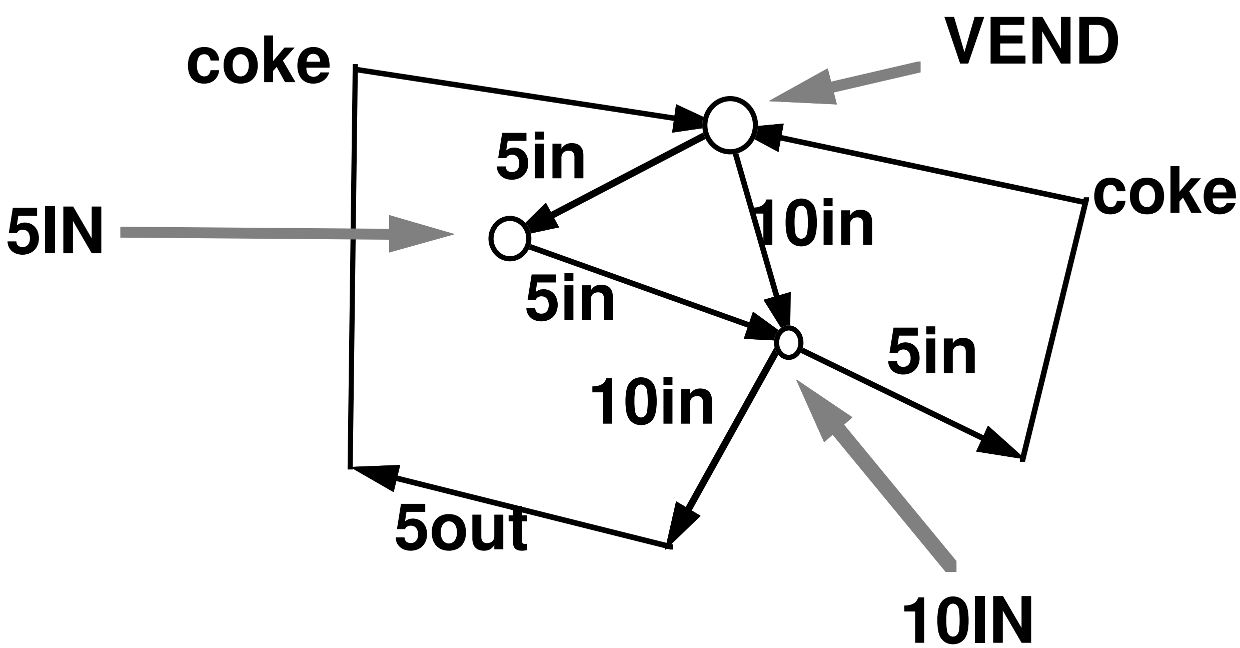
(in5 → 10IN) | (in10 → coke → VEND)

10IN =

(in5 → coke → VEND)

| (in10 → out5 → coke → VEND)

Diagram of VEND



Note on Representing State

- In CSP states are represented through process names.
- A process P with state S will be represented as P_{State}
- Not all transitions need be associated with a pair of named states, as illustrated in the previous example.
- Could introduce names for all of these (eg. “Dispense”) but this would clutter the description.

Another example

$$\text{Stack}_{\langle\ \rangle} = \text{push.x} \rightarrow \text{Stack}_{\langle x \rangle}$$

$$\text{Stack}_{\langle y \rangle \wedge s} = \text{push.x} \rightarrow \text{Stack}_{\langle x \rangle \wedge \langle y \rangle \wedge s}$$

$$\mid \text{pop} \rightarrow \text{Stack}_{s}$$

Generalized Choice

- **When there are too many events to enumerate all of the choices we can write**

$$(x{:}B \rightarrow P(x))$$

x is an Event

P is a function

P: B $\rightarrow$ Process

- **This is the process that can do any event, x, in B, and then becomes P(x)**

- **Verbally, "x from B then P of x"**

Examples

1. $Square = x: N \rightarrow (x^2 \rightarrow Square)$

   The process that squares a number

   P is the function $P(x) = x^2 \rightarrow Square$

2. $RUN_A = (x:A \rightarrow RUN_A)$

   The process that will accept any event of A

   P is the function $P(x) = RUN$

Interacting Processes

$SIMPLE\_VEND =$

$\quad coin \rightarrow choc \rightarrow SIMPLE\_VEND$

$GREEDY\_CUST =$

$\quad (choc \rightarrow GREEDY\_CUST)$

$\quad | \ (coin \rightarrow choc \rightarrow GREEDY\_CUST)$

How do we combine these?

The Parallel operator (||)

SIMPLE_VEND || GREEDY_CUST

"SIMPLE_VEND in parallel with GREEDY_CUST"

Meaning:

for every event, SIMPLE_VEND and GREEDY_CUST must *both* participate in the event.

SIMPLE_VEND || GREEDY_CUST = P

where P = coin $\rightarrow$ choc $\rightarrow$ P

Problem

SIMPLE_VEND =

$\quad$ coin $\rightarrow$ choc $\rightarrow$ SIMPLE_VEND

STUPID_CUST =

$\quad$ coin $\rightarrow$ STUPID_CUST

SIMPLE_VEND || STUPID_CUST =

$\quad$ coin $\rightarrow$ STOP

This is called <u>deadlock</u>.

Partial Synchronization

Suppose

$CLOCK = tick \rightarrow tock \rightarrow CLOCK$

$GAME = tick \rightarrow tack \rightarrow toe \rightarrow GAME$

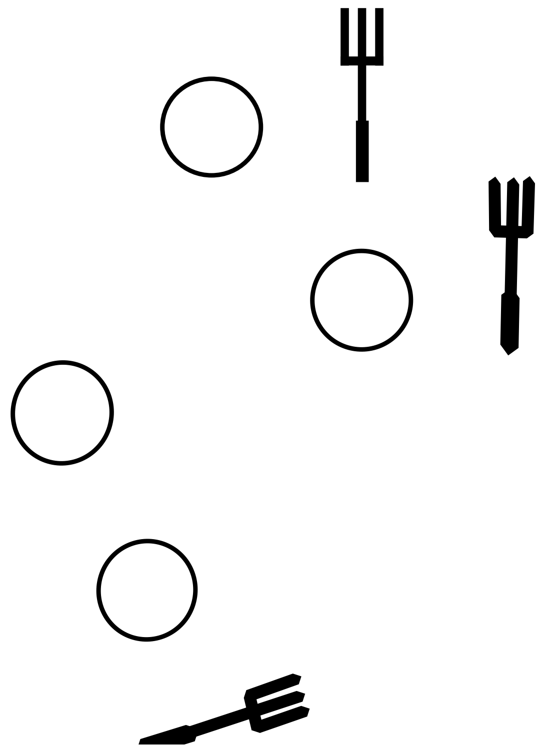
CLOCK || GAME = ???

Alphabets and ||

αCLOCK = {tick,tock}
αGAME = {tick,tack,toe}
αCLOCK $\cap$ αGAME = {tick}
CLOCK and GAME must synchronize on {tick}.
P || Q means that P and Q must synchronize on every event in the intersection of their alphabet.
So CLOCK || GAME = P, where

$$
\begin{aligned}
P = \ & tick \rightarrow \\
& (tock \rightarrow tack \rightarrow toe \rightarrow P \mid \\
& \ tack \rightarrow tock \rightarrow toe \rightarrow P \mid \\
& \ tack \rightarrow toe \rightarrow tock \rightarrow P\ )
\end{aligned}
$$

Dining Philosophers



Dining Philosophers

$PHIL_i = think_i \rightarrow upfork_{(i,i)} \rightarrow upfork_{(i,i+1 \bmod n)} \rightarrow eat_i \rightarrow downfork_{(i,i)} \rightarrow downfork_{(i,i+1 \bmod n)} \rightarrow PHIL_i$

$FORK_i =$

$(upfork_{(i,i)} \rightarrow downfork_{(i,i)} \rightarrow FORK_i)$

$|$

$(upfork_{(i-1 \bmod n,i)} \rightarrow downfork_{(i-1 \bmod n,i)} \rightarrow FORK_i)$

Dining Philosophers (cont.)

PHILOSOPHERS =
 PHIL0 || PHIL1 || PHIL2 || PHIL3 || PHIL4

FORKS =
 FORK0 || FORK1 || FORK2 || FORK3 || FORK4

ROOM = PHILOSOPHERS || FORKS

Summary

α **P: alphabet of P**

the events that P can engage in

e $\rightarrow$ P : e *then* P

the process that does event e and then acts like P

a $\rightarrow$ P | b $\rightarrow$ Q : a then P *choose* b then Q

do a then act like P or do b then act like Q

(x:B $\rightarrow$ P(x)) : x *from* B then P of x

do one of the events from B

P || Q : P parallel Q

run P and Q in parallel, synchronizing on shared events

Special processes: STOP, RUN

Last Slide

Semantics

- The meaning of a CSP process is the set of traces that it can produce.
- That is, we only care about the visible event behavior.
- Example:
  - SIMPLE_VEND = coin $\rightarrow$ choc $\rightarrow$ SIMPLE_VEND
  - has meaning
  - {<>, <coin>, <coin,choc>, <coin,choc,coin>, ...}
- Note:
  - Each trace is finite.
  - There can be an infinite number of them.
  - The set of traces is "prefix closed"

Operations on Traces

- Since traces are just sequences then can perform the usual operations on them

  concatenation, size, head, tail, etc.

- One useful operator is the one that counts the number of occurrences of a given event

  tr ↓ e

- Example:

<coin,choc,coin,choc,coin>↓ coin = 3

Specifications

- A specification is a predicate that all traces of a desired process should satisfy.
- Example

  Let VMS = coin $\rightarrow$ choc $\rightarrow$ VMS

  and

  NOLOSS = (tr $\downarrow$ choc) $\leq$ (tr $\downarrow$ coin)

  FAIR = (tr $\downarrow$ coin) $\leq$ (tr $\downarrow$ choc + 1)

  Does the process satisfy the specification?

  How would you prove it?

Alternate Description of Recursion

SIMPLE_VEND = μX. coin $\rightarrow$ choc $\rightarrow$ X

SIMPLE_VEND is the solution to the equation

X = coin $\rightarrow$ choc $\rightarrow$ X

More Alphabets

SIMPLE_VEND = coin $\rightarrow$ choc $\rightarrow$ SIMPLE_VEND

αSIMPLE_VEND = {coin,choc}

DECIDE = left $\rightarrow$STOP | right $\rightarrow$STOP

αDECIDE = {left,right}