Lecture 19
Reasoning with CSP

David Garlan
Carnegie Mellon University

Models of Software Systems
15-671
Fall 1995

CSP Concurrency: Four Lectures

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

Last Time

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

$a \rightarrow P \mid b \rightarrow Q$: 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

$P \parallel Q$: P parallel Q

run P and Q in parallel, synchronizing on shared events

Special processes: STOP, RUN

Skip

- Another useful process is SKIP, representing successful termination.
- Formally, SKIP = $\surd \rightarrow$ STOP,
- where $\surd$ is the “success event”
- sometimes called “tick”

Input/Output
Channels and Messages

- "A communication is an event that is described by a pair c.v where c is the name of the channel ... and v is the value of the message that passes."
- For an event c.v we define

  channel(c.v) = c

  message(c.v) = v
- <u>Example</u>:

  $P = c.3 \rightarrow STOP$

  $Q = c.3 \rightarrow out.(3+3) \rightarrow STOP$

  P || Q --- Q reads 3 from P (on the "c" channel) and outputs 6 (on the "out" channel).

Problems with this

- Doesn’t distinguish between the input-er and the output-er.
- In general, we want Q to accept any value that P may choose to provide -- not just 3.
- What Q does after reading a value on its channel may depend on the value it reads.
- To fix these problems we introduce some syntactic conventions.

General Input/Output

- **c!v denotes output of message v on channel c.**

  So, (c!v $\rightarrow$ P) = (c.v $\rightarrow$ P)

- **c?x denotes input on channel c, with the result assigned to the variable x.**

  So, (c?x $\rightarrow$ P(x)) = (y:Any-c-Msg $\rightarrow$ P(message(y)))

  where Any-c-Msg = {event | channel(event) = c}

Communication

- Two such processes communicate by acting in parallel.

  $(c!v \rightarrow P) \mid\mid (c?x \rightarrow Q(x))$
- Formally this is equivallent to

  $c.v \rightarrow (P \mid\mid Q(v))$
- We are assuming that P and Q share alphabets of the form {c.x | x ranges over space of values}.
- Note Q(v) is a function from values to processes.

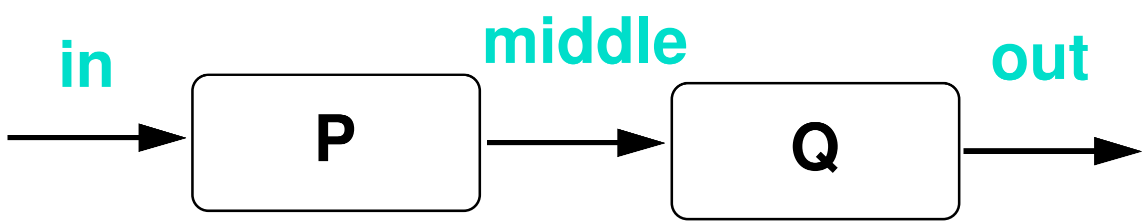
  eg., $Q(x) = out.(x + x) \rightarrow STOP$

Examples

- <u>Producer</u>: A process that continuously writes "yes" on channel c.
  YES = c!yes $\rightarrow$ YES
- <u>Consumer</u>: A process that eats messages from channel c (and doesn't do anything with them).
  P = c?x $\rightarrow$ P
- <u>Buffer (1)</u>: A process that copies messages from its input to its output.
  P = in?x $\rightarrow$ (out!x $\rightarrow$ P)
- <u>Buffer (2)</u>:
  P<> = in?x $\rightarrow$ P<x>
  P<x> = (in?y $\rightarrow$ P<x,y>) | (out!x $\rightarrow$ P<>)
  P<x,y> = out!x $\rightarrow$ P<y>

Another Example

<u>Let</u> $P = in?x \rightarrow middle!x \rightarrow P$
<u>and</u> $Q = middle?x \rightarrow out!x \rightarrow Q$
Then $P \,||\, Q$ is a two-place buffer



Qn: is this equivalent to the previous 2-slot buffer?
(Hint: another process can interact with the “middle” channel)

More Examples

- **Buffer (infinite)**: P<> , where

  P<> = left?x → P<x>

  P<x>^s = left?y → P<x>^s^<y>

  | right!x→ Ps

- **Stack**: P<> , where

  P<> = (empty → P<>) | (left?x → P<x>)

  P<x>^s = left?y → P<y>^<x>^s

  | right!x → Ps

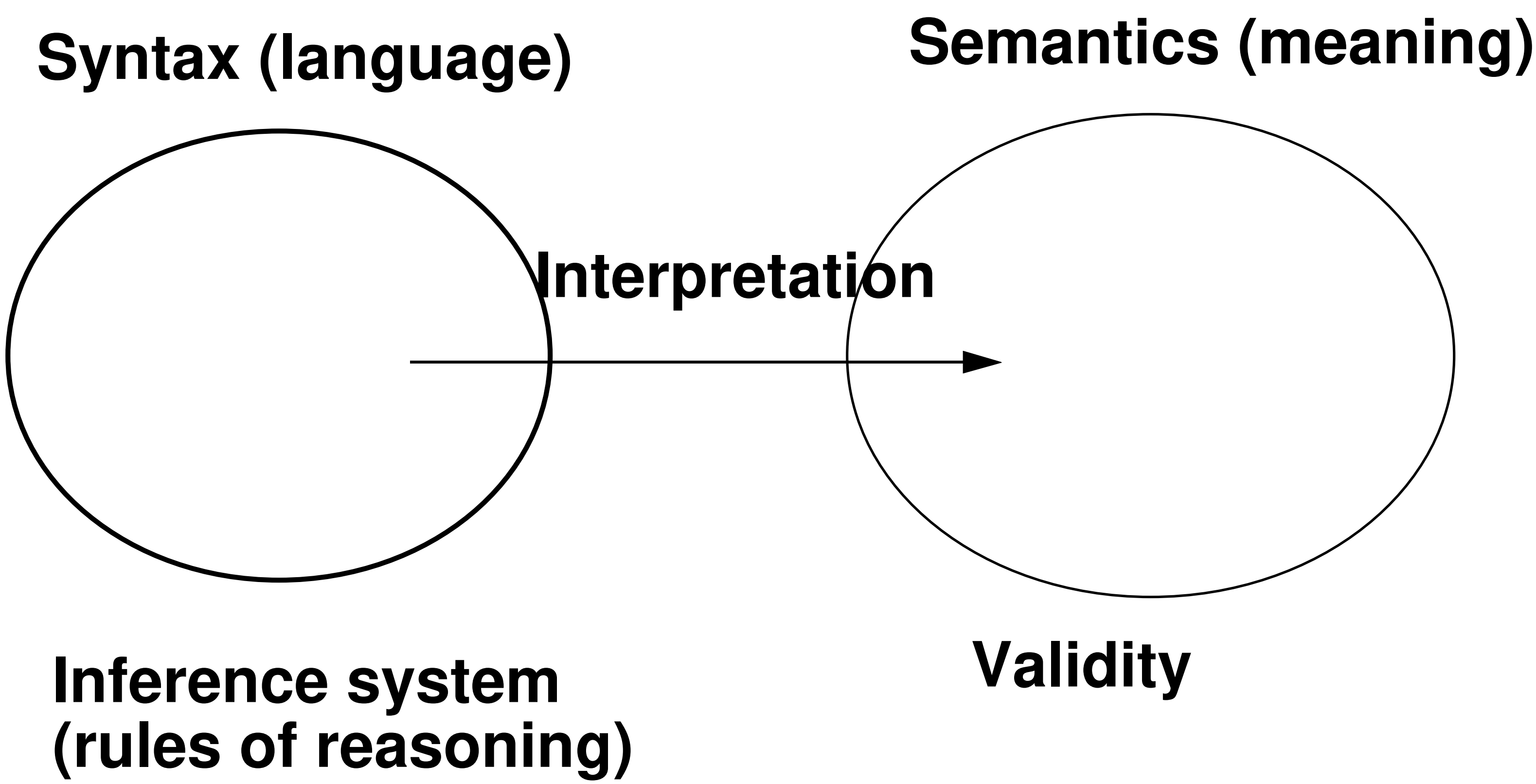
- **Shared Variable**: Var, where

  Var = write?x → Var-x

  Var-x = write?y → Var-y

  | read!x → Var-x

Basics of a Formal System



Semantics (for now)

- The meaning of a CSP process P is the set of traces, tr(P), that it produce.
- That is, we only care about the visible event behavior.
- Example:

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

  tr(SIMPLE_VEND} =

  {<>, <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 we can perform operations on them:**

  concatenation, size, head, tail, etc.
- **One useful operator counts the number of occurrences of an event e in trace tr: $tr \downarrow e$**
- **Example:**

  <coin,choc,coin,choc,coin> $\downarrow$ coin = 3
- **Another useful operator hides events from a trace: $tr \upharpoonright S$**
- **Example:**

  <5in,5in,10in,choc,5out> $\upharpoonright$ {5in,10in,5out} =

  <5in,5in,10in,5out>

More Formally:Trace Axioms

- (Stop) $tr(STOP) = \{\langle\rangle\}$
- (Skip) $tr(SKIP) = \{\langle\rangle, \langle\surd\rangle\}$
- (Gen) $tr(e \rightarrow P) = \{t: tr(P) \bullet \langle e\rangle \hat{} t\} \cup \{\langle\rangle\}$
- (Choice) $tr(P \mid Q) = tr(P) \cup tr(Q)$
- ($||_{= alph}$) $tr(P \parallel Q) = tr(P) \cap tr(Q)$
- ($||_{\neq alph}$) $tr(P \parallel Q) = \{t: seq\ (\alpha P \cup \alpha Q) \mid (t \upharpoonright \alpha P) \in tr(P) \wedge (t \upharpoonright \alpha Q) \in tr(Q)\}$

Others

- (Non-empty) $\langle\rangle \in tr(P)$
- (Prefix) $t1 \leq t2 \wedge t2 \in tr(P) \Rightarrow t1 \in tr(P)$

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?

CSP: Rules of Inference

- CSP rules of inference allow you to change the form of a process description through syntactic manipulations.
- This an alternative to reasoning about traces.
- Allows one to add or reduce the parallelism in an expression.
- Every parallel process can be written as an equivalent process without the parallel operator.
- Rules fall into categories

Laws about || Processes

- (Idempotence) $P \parallel P = P$
- (Commutativity) $P \parallel Q = Q \parallel P$
- (Associativity) $(P \parallel Q) \parallel R = P \parallel (Q \parallel R)$

Assume:

$$e_1 \in (\alpha P - \alpha Q)$$
$$e_2 \in (\alpha Q - \alpha P)$$
$$e_3, e_4 \in (\alpha P \cap \alpha Q)$$

Different alphabets

- **Deadlock** $(e_3 \rightarrow P) \parallel (e_4 \rightarrow Q) = STOP$
- **Synch** $(e_3 \rightarrow P) \parallel (e_3 \rightarrow Q) =$

  $e_3 \rightarrow (P \parallel Q)$
- **Stop** $P \parallel STOP_{\alpha P} = STOP_{\alpha P}$
- **Independent events come first**

  $(e_1 \rightarrow P) \parallel (e_3 \rightarrow Q) = (e_1 \rightarrow (P \parallel e_3 \rightarrow Q)$
- **Interleaving**

  $(e_1 \rightarrow P) \parallel (e_2 \rightarrow Q) =$

  $e_1 \rightarrow (P \parallel e_2 \rightarrow Q) \mid e_2 \rightarrow (e_1 \rightarrow P) \parallel Q$
- **Alphabets**

  $\alpha(P \parallel Q) = \alpha P \cup \alpha Q$

More Laws

- **Distribution**

$$P \,||\, (Q \mid R) = (P \,||\, Q) \mid (P \,||\, R)$$

$$(P \mid Q) \,||\, R) = (P \,||\, R) \mid (Q \,||\, R)$$

- **Run identity**

$$P \,||\, RUN_{\alpha P} = P$$

Example

Let

$$P = a \rightarrow ((b \rightarrow P) \mid (c \rightarrow P))$$
$$Q = (a \rightarrow b \rightarrow Q)$$

Simplify

$$P \mid\mid Q$$

Ans:

$\alpha P = \{a, b, c\}$ and $\alpha Q = \{a, b\}$

$P \mid\mid Q$

= <expand definitions>

$a \rightarrow ((b \rightarrow P) \mid (c \rightarrow P)) \mid\mid (a \rightarrow b \rightarrow Q)$

= <synch>

$a \rightarrow ((b \rightarrow P) \mid (c \rightarrow P) \mid\mid (b \rightarrow Q))$

= <distribution>

$a \rightarrow ((b \rightarrow P \,||\, b \rightarrow Q) \;|\; (c \rightarrow P \,||\, b \rightarrow Q))$

= <synch>

$a \rightarrow ((b \rightarrow P \,||\, Q) \;|\; (c \rightarrow P \,||\, b \rightarrow Q))$

= <independent events first>

$a \rightarrow ((b \rightarrow P \,||\, Q) \;|\; (c \rightarrow (P \,||\, b \rightarrow Q))$

= <expand def of P>

$a \rightarrow ((b \rightarrow P \,||\, Q) \;|\; (c \rightarrow (a \rightarrow \ldots \,||\, b \rightarrow Q))$

= <deadlock>

$a \rightarrow ((b \rightarrow P \,||\, Q) \;|\; (c \rightarrow STOP))$

So P|| Q = R, where

$R = a \rightarrow ((b \rightarrow R) \;|\; (c \rightarrow STOP))$