Lecture 20
CSP - Non-determinism

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, interleaving, hiding
  - > the need for a richer semantic model
- **Applications**
  - > pipes
  - > specification of software architectures

Today's Topics

- Quick review of concepts
- Tidy up from last lecture:
  > A word about recursion
  > Proof by induction over traces
- Nondeterminism in sequencial systems
- Non-determinism in concurrent systems
- Interleaving
- Hiding

Lecture 1

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

$P \mid\mid Q$: P parallel Q

run P and Q in parallel, synchronizing on shared events

Special processes: STOP, RUN

Lecture 2

- **Skip to represent successful termination**
  SKIP = √ → STOP
- **Input/output**
  c?in, c!out
- **Semantics of CSP**
  meaning of a process is its set of event traces
  specifications and "satisfies"
  induction over traces sets
- **Axioms**
  permit reasoning
  allow transformation of process expressions
  can eliminate parallel construct

Concurrency review

- Recall: two processes can only continue if they can agree on a shared event to do jointly, or one process executes a private event
- Formally

$$(a \rightarrow P) \,||\, (a \rightarrow Q) \quad = \quad a \rightarrow (P \,||\, Q)$$
$$(a \rightarrow P) \,||\, (b \rightarrow Q) \quad = \quad STOP$$

- Consequence 1:
  any trace of a parallel composition, when restricted to the events of one of the processes will be a valid trace for that process.
- Consequence 2:
  if P satisfies S(t); Q satisfies T(t), then P || Q satisfies $S(t \uparrow \alpha P) \wedge T(t \uparrow \alpha Q)$

When is Recursion Uniquely Defined?

- Suppose process P is defined by a recursive equation: P = F(P)
- There is a theorem that guarnatees this equation will always have a unique solution, provided F is “guarded”
- That is, F must introduce a prefix event.
- Note that P = P (where F is the identity) does not produce a unique solution.

Induction for CSP Processes

- Suppose P is defined by a recursive equation P = F(P), where F is guarded.
- To show that P satisfies specification S by induction we must demonstrate two things:
  1. Base case: STOP satisfies S
  2. Inductive step: whenever a process X satisfies S, then F(x) also satisfies S.

Example

- VMS = coin $\rightarrow$ choc $\rightarrow$ VMS

  (note F(X) = coin $\rightarrow$ choc $\rightarrow$ X, and F is guarded)
- Show that VMS satisfies

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

Proof

1. Base case: <u>STOP satisfies NOLOSS</u>

   True because tr(STOP) = {<>}, and

   (<>$\downarrow$ choc) = 0 $\leq$ 0 = (<> $\downarrow$ coin)

   so forall t: tr(STOP), t satsifies NOLOSS.

2. Inductive step: suppose a process X satisfies NOLOSS. Consider the process Y = F(X), which is coin $\rightarrow$ choc $\rightarrow$ X.

   What are the traces of Y?

$tr(Y) = \{<>, <coin>, <coin,choc>\} \cup$

$\{\ t: tr(X) \bullet <coin, choc>\ \hat{}\ t\ \}$

1. $<>$ satisfies NOLOSS, as argued before.

2. $<coin>$ satisfies NOLOSS, since

$(<coin> \downarrow choc) = 0 \leq 1 = (<coin> \downarrow coin)$.

3. $<coin,choc>$ satisfies NOLOSS, since

$(<coin,choc> \downarrow choc) = 1 \leq 1 = (<coin,choc> \downarrow coin)$.

4. $\langle coin,choc \rangle \mathbin{\hat{}} t$ satisfies NOLOSS, since

$(\langle coin,choc \rangle \mathbin{\hat{}} t) \downarrow choc$

$=$ property of $\downarrow$

$(\langle coin,choc \rangle \downarrow choc) + (t \downarrow choc)$

$=$ def of $\downarrow$

$1 + (t \downarrow choc)$

$\leq$ induction hypothesis

$1 + (t \downarrow coin)$

$=$ def of $\downarrow$

$\langle coin,choc \rangle \downarrow coin) + (t \downarrow coin)$

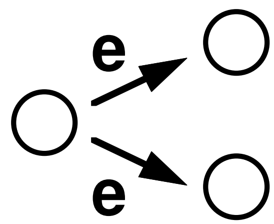
$=$ property of $\downarrow$

$(\langle coin,choc \rangle \mathbin{\hat{}} t) \downarrow coin$

Nondeterminism - Sequential Systems

Have already seen several instances of non-determinism:

1. <u>State machines</u>



2. <u>Relations</u>

choose: Set $\rightarrow$ Element

{(Fred,blue) (Fred,green), (Charlie,pink)}

3. <u>Schemas</u>

S

x, y: $\mathbb{N}$

Enlarge

Δ S

x' > x

Nondeterminism - Concurrent Systems

- Non-determinism has also arisen in the context of concurrency.
- Example:

  Suppose P and Q are processes
  with $\alpha P = \alpha Q$, $a \notin \alpha Q$, $b \notin \alpha P$. Then
  $(a \rightarrow P) \,||\, (b \rightarrow Q) =$

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

  $b \rightarrow a \rightarrow (P \,||\, Q)$
- As with many concurrent models, non-determinism is used to represent parallel computation.

Nondeterminism - Continued

- Non-determinism arises in CSP in other ways, too.
- Recall the choice operator (|) allows us to write:

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

  where $\{a,b\} \subseteq \alpha P$, $\alpha P = \alpha Q$, <u>provided that</u> $a \neq b$.
- But suppose we wrote

  $(a \rightarrow P) \mid (a \rightarrow Q)$
- What might this mean?

Generalized Choice

- CSP provides an operator that allows such expressions:

  [] -- generalized choice

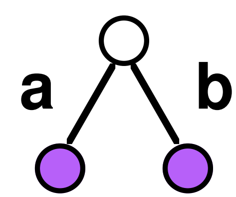
  where,

  $(a \rightarrow P)\ []\ (b \rightarrow Q) = (a \rightarrow P) \mid (b \rightarrow Q)$ **when** $a \neq b$

  otherwise one of the two branches is picked (non-deterministically) by the machine itself!

  traces($(a \rightarrow b \rightarrow STOP)\ []\ (a \rightarrow c \rightarrow STOP)$ **) =**

  {<>, <a>, <a,b>, <a,c>}



Non-deterministic OR

- A somewhat different need for non-determinism arises when we want to explicitly distinguish "<u>internal</u>" from "<u>external</u>" choice.
- CSP uses the operator $\sqcap$ to indicate internal choice.
- Example 1:

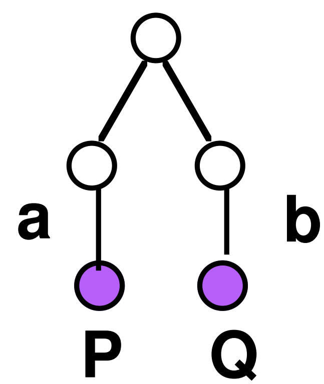
  $(a \rightarrow P) \sqcap (b \rightarrow Q)$

  Either a or b will occur,

  but the "machine" decides.

  Acts like an unlabeled transition.

  Sometimes represented as τ in other systems



Example 2:

Chg = qtr $\rightarrow$ (DimeFirst $\sqcap$ NicklesFirst)

DimeFirst = dime $\rightarrow$ nickle $\rightarrow$ nickle $\rightarrow$ Chg

NicklesFirst = nickle $\rightarrow$ nickle $\rightarrow$ dime $\rightarrow$ Chg

Example 3:

CokeLover =

qtr $\rightarrow$ selCoke $\rightarrow$ getCoke $\rightarrow$ CokeLover

PepsiLover =

dime $\rightarrow$ selPepsi $\rightarrow$ getPepsi $\rightarrow$ PepsiLover

VeryThirsty = CokeLover $\sqcap$ PepsiLover

Laws of Nondeterminism

$P \sqcap P = P$

$x \rightarrow (P \quad Q) = (x \rightarrow P) \sqcap (x \rightarrow Q)$

$(P \sqcap Q) \parallel R = (P \parallel R) \sqcap (Q \parallel R)$

other laws are similar

Semantics of Nondeterminism

- **What are the (trace) semantics of $\sqcap$?**

  traces $(P \sqcap Q)$ = traces$(P) \cup$ traces(Q)

- **Is this a problem?**

  $(a \rightarrow STOP) \mid (b \rightarrow STOP)$ has traces

  $\{<>, <a>, <b>\}$

- **But, according to the above**

  $(a \rightarrow STOP) \sqcap (b \rightarrow STOP)$ also has traces

  traces$(a \rightarrow STOP) \cup$ traces$(b \rightarrow STOP)$ =

  $\{<>, <a>\} \cup \{<>, <b>\} = \{<>, <a>, <b>\}$

- **So apparently these are the same processes. *Is this true?***

Semantics (cont)

Let

$P = (a \rightarrow STOP)$ and $Q = (b \rightarrow STOP)$

where $\alpha P = \alpha Q = \{a, b\}$

Now

$(P \mid Q) \parallel P = (a \rightarrow STOP)$

but

$(P \sqcap Q) \parallel P = (P \parallel P) \sqcap (Q \parallel P)$

$= P \sqcap STOP$

In other words, the first must accept a, while the second can choose to do b and then get stuck immediately

Thus the two processes are not "behaviorally equivalent".

Fixing the Semantics

- When any formal system has the situation that two terms have the same semantics, but are not behaviorially equivalent, this is a PROBLEM.
- Technically, we say the semantics isn't "fully abstract".
- To fix the problem our semantics must be refined.
- Instead of only considering the traces of a process, we must also consider under what situations the process can deadlock.

Failures

- **We define failures(P) to be a relation**

  traces(P) $\leftrightarrow \mathbb{P}$ (α P)

  where (tr, events) $\in$ failures(P)

  iff P can execute *tr* and then deadlock on any of the events in *events.*

- **Example:**

  for P = (a $\rightarrow$ STOP) and Q = (b $\rightarrow$ STOP)

  failures (P | Q) =

  { (<>,{}), (<a>, {a,b}), (<b>, {a,b}) }

  failures (P $\sqcap$ Q) =

  { (<>,{a}), (<>,{b}), (<a>, {a,b}), (<b>, {a,b}) }

Interleaving

- Sometimes we want to make sure two machines don't interact at all.
- CSP provides an interleaving operator.
  - P ||| Q
- The traces of two interleaved processes are the arbitrary interleaving of the traces.
- Example:
  - (a $\rightarrow$ b $\rightarrow$ STOP) ||| (c $\rightarrow$ STOP)
- has traces
  - {< >, <a>, <c>, <a,b>, <a,c>, <c,a>, <c,a,b>, < a,c,b> <a,b,c>}

Hiding

- Sometimes it is useful to "hide" events of a machine from its environment.
- In that way those events can't be influenced by external processes.
- The general form is P \ Event-set.
- Example:

  If $P = (a \rightarrow b \rightarrow STOP) \mid (b \rightarrow c \rightarrow STOP)$

  then $P \setminus \{b\} = (a \rightarrow STOP) \mid (c \rightarrow STOP)$