CIS 505 Software Systems
Lecture Note on CSP

Instructor: Insup Lee
Department of Computer and Information Science
University of Pennsylvania

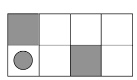
[The slides are originally prepared by U. Sammapun, based on the CSP book by C.A.R. Hoare]

Communicating Sequential Processes (CSP)

- Capture real world behaviors
  - Chocolate vending machine
- **Event** (instantaneous actions)
  - *coin*
  - *choc*
- **Process** (behavior patterns of objects)
  - *VMC – chocolate vending machine*
- **Alphabet** (set of possible events, denoted as αP)
  - $\alpha VMC = \{coin, choc\}$
- $STOP_A$ (behavior or process of a broken object with alphabet A)
  - $STOP_{\alpha VMC}$

Prefix

- **Prefix**: $(x \rightarrow P)$
  - Process with same alphabet as P
  - $\alpha(x \rightarrow P) = \alpha P$ where $x \in \alpha P$
- Broken chocolate vending machine
  - $coin \rightarrow STOP_{\alpha VMC}$
  - $(coin \rightarrow (choc \rightarrow (coin \rightarrow (choc \rightarrow STOP_{\alpha VMC}))))$
- Counter board



  - $\alpha CRT = \{up, right\}$
  - $CRT = (right \rightarrow up \rightarrow right \rightarrow \text{right} \rightarrow STOP_{\alpha CRT})$
- **Incorrect**
  - $P \rightarrow Q$
  - $x \rightarrow y$ (Correct syntax: $x \rightarrow (y \rightarrow STOP)$)

Recursion

- Describe repetitive behaviors
- Clock
  - $\alpha CLOCK = \{tick\}$
  - $CLOCK = tick \rightarrow CLOCK$
  - *CLOCK*
    - $= (tick \rightarrow CLOCK)$ [original equation]
    - $= (tick \rightarrow (tick \rightarrow CLOCK))$ [by substitution]
    - $= (tick \rightarrow (tick \rightarrow (tick \rightarrow CLOCK)))$ [similarly]
    - $= \ldots\ldots\ldots$
    - $= tick \rightarrow tick \rightarrow tick \rightarrow tick \rightarrow tick \rightarrow \ldots\ldots$ [unfolding CLOCK many times]
- Useless equation?
  - $X = X$

Recursion

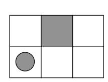
- **Guarded**
  - A process description that begins with a prefix
  - *CLOCK* = (*tick* → *CLOCK*)
- If *F(X)* is a guarded expression containing process *X* and alphabet *A*
  - Then *X* = *F(X)* has a unique solution with alphabet *A*
  - Solution: *μ X*: *A* • *F(X)*
- *X* is a local name and can be changed
  - *μ X*: *A* • *F(X)* = *μ* Y: *A* • *F(Y)*
  - Because a solution for *X* = *F(X)* is also a solution for *Y* = *F(Y)*

Recursion Examples

- Clock
  - *CLOCK* = *μ X*: *{tick}* • *(tick* → *X)*
- Working chocolate vending machine!
  - *VMC* = (*coin* → (*choc* → *VMC*))
  - Or formally, *VMC* = *μX: {coin,choc}* • *(coin* → *(choc* → *X))*
- Machine gives change for 5p repeatedly
  - *αCH5A = {in5p, out2p, out1p}*
  - *CH5A* = (*in5p* → *out2p* → *out1p* → *out2p* → *CH5A*)
- Different change-giving machine with same alphabet
  - *CH5B* = (*in5p* → *out1p* → *out1p* → *out1p* → *out2p* → *CH5B*)

Choice

- **Interaction with environment**
  - Machine with 1p coin slot and 2p coin slot
  - It's a customer's choice
- **Syntax**
  - *(x* → *P* | *y* → *Q)*
    where *α(x* → *P* | *y* → *Q)* = *αP*
    provided {x,y} ⊆ *αP* and *αP* = *αQ*
- **Examples**
  - Possible counter moves
    - *(up* → *STOP* | *right* → *right* → *up* → *STOP)*



  - Machine with two combination of changes
    - *CH5C = in5p* → *(out1p* → *out1p* → *out1p* → *out2p* → *CH5C*
      | *out2p* → *out1p* → *out2p* → *CH5C)*

Choice Example

- Chocolate / toffee machine
  - VMCT = *μ X* • *coin* → *(choc* → *X* | *toffee* → *X)*
- Machine with choices of coins / goods / change
  - *VM = (in2p* →*(large* → *VM*
    | *small* → *out1p* → *VM)*
    | *in1p* → *(small* → *VM*
    | *in1p* → *(large* → *VM*
    | *in1p* → *STOP)))*
- Machine that trusts customers
  - *VMCRED* = *μ X* • *(coin* → *choc* → *X* | *choc* → *coin* → *X)*
- To prevent loss, an initial payment is required
  - *VM2 = (coin* → *VMCRED)*
- Copying Process
  - COPYBIT = *μ X* • *(in.0* → *out.0* → *X* | *in.1* → *out.1* → *X)*

Choice

- **More than two alternatives**
  - $(x \rightarrow P \mid y \rightarrow Q \mid \ldots \mid z \rightarrow R)$
  - where *x, y, z* are distinct events
- **Incorrect syntax**
  - $(x \rightarrow P \mid x \rightarrow Q)$
  - $(x \rightarrow P \mid (y \rightarrow Q \mid z \rightarrow R))$
- **In general**, choice is written as
  - $(x : B \rightarrow P(x))$
  - Offers a choice of any event *x* in *B*, and behaves like *P(x)*
  - A set *B* is called the *"initial menu"* of the process
- **Example**: Process which can engage in any event in alphabet A
  - $\alpha RUN_A = A$
  - $RUN_A = (x : A \rightarrow RUN_A)$
- *There are other kinds of choice: e.g., non-deterministic choice*

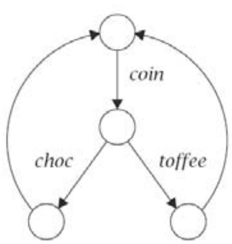
Mutual Recursion

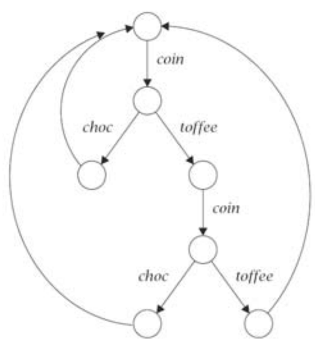
- **Orange-Lemon drink dispenser (*DD)***
  - Pressing two buttons: *setorange*, *setlemon*
  - Dispensing drinks: *orange*, *lemon*
  - $\alpha DD = \alpha O = \alpha L = \{setorange, setlemon, orange, lemon\}$

$$DD = (setorange \rightarrow O \mid setlemon \rightarrow L)$$
$$O = (orange \rightarrow O \mid setlemon \rightarrow L \mid setorange \rightarrow O)$$
$$L = (lemon \rightarrow L \mid setorange \rightarrow O \mid setlemon \rightarrow L)$$

- **Object movement**
  - On the ground: move up or around
    - $CT_0 = (up \rightarrow CT_1 \mid around \rightarrow CT_0)$
  - In the air: move up or down
    - $CT_{n+1} = (up \rightarrow CT_{n+2} \mid down \rightarrow CT_n)$

Pictures





Laws

- **Law1:**
  - $(x \rightarrow P \mid y \rightarrow Q) = (y \rightarrow Q \mid x \rightarrow P)$
  - $(x \rightarrow P) \neq STOP$
  - $(x \rightarrow P) \neq (y \rightarrow Q)$ if $x \neq y$
  - $(x \rightarrow P) = (x \rightarrow Q)$ implies $P = Q$
  - Example:
    - $(coin \rightarrow choc \rightarrow coin \rightarrow choc \rightarrow STOP) \neq (coin \rightarrow STOP)$
    - $\mu X \bullet (coin \rightarrow (choc \rightarrow X \mid toffee \rightarrow X))$
      $= \mu X \bullet (coin \rightarrow (toffee \rightarrow X \mid choc \rightarrow X))$

Laws

- **Law2:** let *F(X)* be a guarded expression
  - $(Y = F(Y)) \equiv (Y = \mu X \bullet F(X))$
  - $\mu X \bullet F(X) = F(\mu X \bullet F(X))$

  - Example:
    - *Let VM1 = (coin → VM2)*
      VM2 = (choc → VM1)

    - Then VM1 = (coin → VM2)
      = (coin → (choc → VM1))
      = VMC

Traces

- **Trace of a behavior of a process**
  - Finite sequence of symbols recording events up to some moment in time

- **Example**
  - Chocolate vending machine after serving 2 customers
    - ⟨ *coin, choc, coin, choc* ⟩
  - Before anyone puts coins in
    - ⟨ ⟩ (called empty trace, the shortest possible trace)
  - Change-giving machine – customer is waiting for the last 2p
    - ⟨ *in5p, out2p, in1p* ⟩

Operations on Traces

- **Concatenation**: $s \,\hat{}\, t$
  - ⟨ *coin, choc* ⟩ ^ ⟨ *coin, toffee* ⟩ = ⟨ *coin, choc, coin, toffee* ⟩

- **Head**: s_0 **Tail**: s'
  - $\langle x, y, x \rangle_0 = x$
  - $\langle x, y, x \rangle' = \langle y, x \rangle$

- **Ordering**: $s \le t = (\exists\, u \bullet s \,\hat{}\, u = t)$
  - *"s* is a prefix of *t"*
  - $\langle x, y \rangle \le \langle x, y, x, w \rangle$

- **Star**:
  - $A^* = \{ t \mid t = \langle\rangle$ or
    $(t_0 \in A$ and $t' \in A^*) \}$

Operation on Traces

- **After**: *P / s* where $s \in traces(P)$
  - *"P after s"*
  - *(VMC / ⟨ coin ⟩) = (choc → VMC)*
  - *(VMC / ⟨ coin, choc ⟩) = VMC*

- **Restriction**: $(t \uparrow A)$ "trace *t* when restricted to symbols in *A*"
  - ⟨ *around, up, down, around* ⟩ ↑ { *up, down* } = ⟨ *up, down* ⟩

- **Length**: # t
  - # ⟨ *x, y, x* ⟩ = 3

Traces of Process

- **Complete set of all possible traces of *P***
  - *traces(P)*
- **Example**
  - $traces(STOP) = \{ \langle\rangle \}$
  - $traces(coin \rightarrow STOP) = \{ \langle\rangle, \langle coin\rangle \}$
  - $traces(\mu X \bullet tick \rightarrow X) = \{ \langle\rangle, \langle tick\rangle, \langle tick,tick\rangle, \ldots \}$
    $= \{ tick \}^*$
  - Chocolate vending machine
    - $trace(\mu X \bullet coin \rightarrow choc \rightarrow X) = \{ s \mid \exists n \bullet s \le \langle coin,choc\rangle^n \}$

Specifications

- **Describe intended behaviors of products**
  - Assume *tr* is a variable for an arbitrary trace
  - Let $(tr \downarrow choc)$ denote $\#(tr \uparrow \{ choc \})$
    - Number *choc* events in trace *tr*
- **Example**
  - VM owner: # of chocolate must never exceed # of coins inserted
    - $NOLOSS = (tr \downarrow choc) \le (tr \downarrow coin)$
  - VM customers: VM won't take more coins until it dispenses paid chocolate
    - $FAIR = (tr \downarrow coin) \le (tr \downarrow choc) + 1$
  - Hence, VM manufacturer must meet spec. from both VM owner and customers
    - $VMSPEC = NOLOSS \land FAIR$
      $= 0 \le (tr \downarrow coin) - (tr \downarrow choc) \le 1$

Satisfaction

- ***P* sat *S***
  - If a product *P* meets a specification *S*, then *P* satisfies *S*
  - Formally, *P* sat *S* iff $\forall\, tr \bullet tr \in traces(P) \Rightarrow S$
- **Example**: *VMC* sat *VMSPEC*
  - Recall:
    - $VMC = (coin \rightarrow (choc \rightarrow VMC))$
    - $VMSPEC = NOLOSS \land FAIR$
      $= 0 \le (tr \downarrow coin) - (tr \downarrow choc) \le 1$

Concurrency

- **2 or more processes operating together**
  - Syntax: $P \parallel Q$
  - Example: Both customers and vending machines can be viewed as processes interacting with one another
- **Interaction**
  - Interact via shared events between processes
- **Concurrency**
  - Specifies how shared and private events in processes are joined

Interaction Example

- A greedy customer tries to get chocolate or toffee without paying
- If doesn't work, he reluctantly pays for chocolate
  - GRCUST = (toffee → GRCUST
    | choc → GRCUST
    | coin → choc → GRCUST)
- When using VMCT machine,
  - VMCT = μ *X* • *coin* → *(choc* → *X* | *toffee* → *X)*
- he can't get goods without paying, hence, he only gets chocolate
  - *(GRCUST || VMCT) =* μ *X* • *(coin* → *choc* → *X)*

Interaction Laws and Traces

- Laws:
  - *P || Q = Q || P*
  - *P || (Q || R) = (P || Q) || R*
  - *P ||* $STOP_{\alpha P}$ = $STOP_{\alpha P}$
  - *P ||* $RUN_{\alpha P}$ = *P*
  - *(c → P) || (c → Q) = (c → (P || Q)*
  - *(c → P) || (d → Q) = STOP* if $c \neq d$
- Traces:
  - *traces(P || Q) = traces(P)* ∩ *traces(Q)*
  - *(P || Q) / s = (P / s) || (Q / s)*

Concurrency

- In general, it's possible that
  - $\alpha P \neq \alpha Q$
  - for $x \in (\alpha P - \alpha Q)$,
    - *P* may engage alone with no concern to *Q*
    - and vice versa
- Hence,
  - $\alpha(P \,||\, Q) = \alpha P \cup \alpha Q$

Concurrency Example

- Noisy vending machine
  - α*NOISYVM* = {*coin, choc, clink, clunk, toffee*}
  - *NOISYVM = (coin → clink → choc → clunk → NOISYVM)*
- Customer who curses when gets chocolate instead of toffee
  - α*CUST* = {*coin, choc, curse, toffee*}
  - *CUST = (coin → (toffee → CUST*
    | curse → choc → CUST)
- Cursing customer using noisy machine
  - *(NOISY || CUST) =*
    μ *X* • *(coin → (clink → curse → choc → clunk → X*
    | curse → clink → choc → clunk → X))

Concurrency Law

- || is symmetric and associative
- $P \parallel STOP_{\alpha P} = STOP_{\alpha P}$
- $P \parallel RUN_{\alpha P} = P$

- Let
  - $a \in (\alpha P - \alpha Q)$
  - $b \in (\alpha Q - \alpha P)$
  - $\{c,d\} \subseteq (\alpha P \cap \alpha Q)$

- Then
  - $(c \to P) \parallel (c \to Q) = c \to (P \parallel Q)$
  - $(c \to P) \parallel (d \to Q) = STOP$ if $c \neq d$
  - $(a \to P) \parallel (c \to Q) = a \to (P \parallel (c \to Q))$
  - $(c \to P) \parallel (b \to Q) = b \to ((c \to P) \parallel Q)$
  - $(a \to P) \parallel (b \to Q) = (a \to (P \parallel (b \to Q)) \mid (b \to ((a \to P) \parallel Q))$

Concurrency Traces

- traces$(P \parallel Q)$ is all possible traces of process $(P \parallel Q)$

- $traces(P \parallel Q) =$
  $\{ t \mid (t \uparrow \alpha P) \in traces(P) \wedge$
  $(t \uparrow \alpha Q) \in traces(Q) \wedge$
  $t \in (\alpha P \cup \alpha Q)^* \}$

  traces of all events where *P* participates

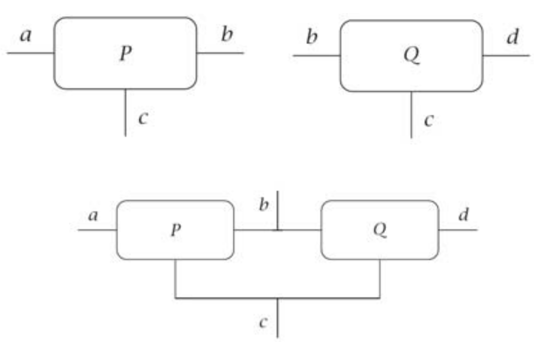
  traces of all events where *Q* participates

  every event in *t* must be in either αP or αQ

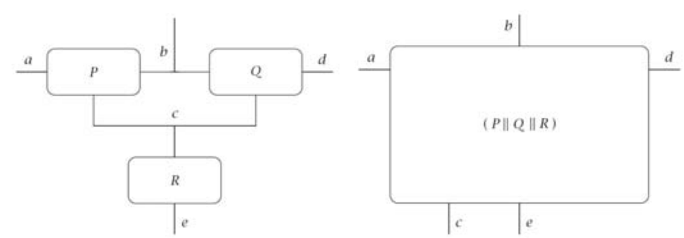
- t1 = $\langle$ *coin, clink, curse* $\rangle \in traces(NOISYVM \parallel CUST)$
  - $t1 \uparrow \alpha NOISYVM = \langle coin, clink \rangle \in traces(NOISYVM)$
  - $t1 \uparrow \alpha CUST = \langle coin, curse \rangle \in traces(CUST)$

- Similar for t2 = $\langle$ *coin, curse, clink* $\rangle$

Concurrency Pictures

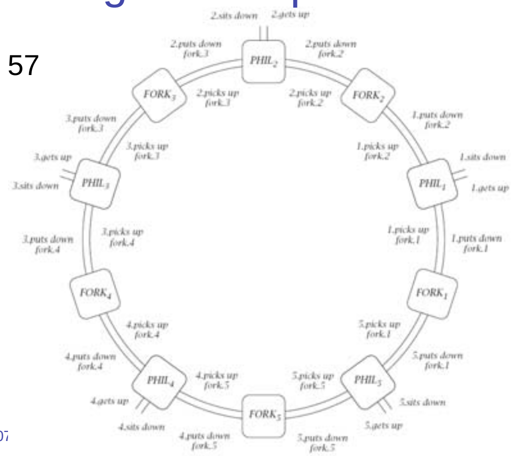


Concurrency Pictures



Dining Philosophers

- Book p. 57



Change of Symbol

- **Define groups of processes with similar behaviors**
  - One-on-one function (injection) maps alphabet of P onto a set of symbols A
  - $f : \alpha P \rightarrow A$
- **Example**:
  - Recall: Chocolate / toffee machine
    - VMCT = *coin* → *(choc* → *VMCT* | *toffee* → *VMCT)*
  - Want: Gum / pretzel machine
    - *f(coin) = coin*
    - *f(choc) = gum*
    - *f(toffee) = pretzel*
  - Now, *VMGP = f(VMCT)*

Process Labeling

- Label identical but independent processes
- Example:
  - Two machine standing side by side
    - (*left* : *VMC*) || (*right* : *VMC*)
  - Possible trace
    - ⟨ *left.coin, right.coin, right.choc* ⟩

Specifications and Satisfaction

- Satisfaction for concurrent processes

- **If** *P* sat *S(tr)*
  and *Q* sat *T(tr)*
  then *(P || Q)* sat $S(tr \uparrow \alpha P) \wedge T(tr \uparrow \alpha Q)$

P satisfies a spec S

Q satisfied a spec T

P || *Q* satisfies if in trace *tr*
- events of *P* satisfy *S*
- events of *Q* satisfy *T*

Spec, Sat Example

- **Let** $\alpha P = \{ a, c \}$ and $\alpha Q = \{ b, c \}$
  - $P = (a \rightarrow c \rightarrow P)$
  - $Q = (c \rightarrow b \rightarrow Q)$
- **Want to know** if $(P \parallel Q)$ sat $0 \leq (tr \downarrow a) - (tr \downarrow b) \leq 2$

num of *a* and *b* differ at most 2

- **Obviously**,
  - P sat $0 \leq (tr \downarrow a) - (tr \downarrow c) \leq 1$
  - Q sat $0 \leq (tr \downarrow c) - (tr \downarrow b) \leq 1$

In P, num of *a* and *c* differ at most 1

In Q, num of *c* and *b* differ at most 1

- **Hence**,
  - $(P \parallel Q)$ sat $(0 \leq (tr \uparrow \alpha P) \downarrow a - (tr \uparrow \alpha P) \downarrow c \leq 1 \wedge$
    $0 \leq (tr \uparrow \alpha Q) \downarrow c - (tr \uparrow \alpha Q) \downarrow b \leq 1)$
  - Since $(tr \uparrow A) \downarrow a = tr \downarrow a$ when $a \in A$

num of *a* when trace has only events in A and $a \in A$

  - $(P \parallel Q)$ sat $(0 \leq (tr \downarrow a) - (tr \downarrow c) \leq 1 \wedge$
    $0 \leq (tr \downarrow c) - (tr \downarrow b) \leq 1)$
  - $(P \parallel Q)$ sat $(0 \leq (tr \downarrow a) - (tr \downarrow b) \leq 2)$

Theory of Deterministic Processes

- CSP Book P. 72 – 79
- It shows that
  - CSP laws are in fact true
  - a recursively defined process is indeed a solution of the corresponding recursive equation (fixed point theory)
  - there exists a unique solution