Goals of the lecture: Conjunctive Predicates

- Direct dependency algorithm
- Token based decentralized algorithm
- Channel Predicates

Reference: Chapter 5.
Algorithm for application process $P_i$

- Assume fully connected network
- Mattern’s vector clock
- Notation:
  - $(i, k)$: the $k$th state on process $P_i$ (or simply $k$)
Monitor Processes for WCP

- Monitor processes responsible for searching for a WCP cut.
- The token stores a candidate cut.
- The token also stores information to determine whether the candidate cut is consistent.
Informal Description

- A token is sent to a process $P_i$ only when the current cut is not consistent. Specifically, when current state from $P_i$ happened before some other state in the candidate cut.

- Once the monitor process for $P_i$ has eliminated the current state,
  - receive a new state from the application process
  - check for consistency conditions again.

- This process is repeated until
  - all states are eliminated from some process $P_i$ or
  - the WCP is detected.
Token

- A monitor process is active only if it has the token.
- token consists of two vectors $G$ and color.
  - $G$ is a global state vector represents the candidate global cut
    - $G[i] = k$: indicates that state $(i, k)$ is part of the current cut.
    - We maintain the invariant that $G[i] = k$ implies that any global cut $C$ with $(i, s) \in C$ and $s < k$ cannot satisfy the WCP.
  - color, indicates which states have been eliminated.
    - If $color[i] = red$ then state $(i, G[i])$ has been eliminated and can never satisfy the global predicate.
    - If $color[i] = green$, then there is no state in $G$ such that $(i, G[i])$ happened before that state.
Monitor Process Algorithm

```plaintext
var
candidate:array[1..n] of integer;

on receiving the token (G,color)
  while (color[i] = red) do
    receive candidate from application process $P_i$
    if (candidate.vclock[i] > G[i]) then
      G[i] := candidate.vclock[i]; color[i] := green;
  endwhile
for j ≠ i:
  if (candidate.vclock[j] > G[j]) then
    G[j] := candidate.vclock[j];
    color[j] := red;
  endif
endfor
if (∃ j: color[j] = red) then send token to $P_j$
else detect := true;
```

Figure 1: Monitor Process Algorithm
Correctness of WCP Detection Algorithm

The algorithm correctly detects the first cut that satisfies a WCP.

Lemma 1 For any $i$,

1. $G[i] \neq 0 \land \text{color}[i] = \text{red} \Rightarrow \exists j: j \neq i : (i, G[i]) \rightarrow (j, G[j])$

2. $\text{color}[i] = \text{green} \Rightarrow \forall k : (i, G[i]) \not\rightarrow (k, G[k])$

3. $(\text{color}[i] = \text{green}) \land (\text{color}[j] = \text{green}) \Rightarrow (i, G[i]) \parallel (j, G[j])$

4. If $(\text{color}[i] = \text{red})$, then there is no global cut satisfying the WCP which includes $(i, G[i])$. 
Analysis of Single-Token WCP Algorithm

- **time complexity**: the total computation time for all processes is $O(n^2m)$
  - Every time a state is eliminated, $O(n)$ work is performed
  - There are at most $mn$ states.

- **Message complexity**: the total number of messages $O(mn)$.
  - The token is sent at most $mn$ times.
  - Each monitor receives at most $m$ messages from its application process.

- **Communication bit complexity**: $O(n^2m)$.
  - Size of both the token and the candidate messages is $O(n)$.

- **space complexity**: $O(mn)$ space is required by the algorithm for every process.
  - The buffer for holding messages
Channel Predicates

- A channel predicate: any boolean function of the accumulation of send and receive events on that channel.
- Only uni-directional channels
  
  \( s, t \): states at different processes.

  \( s.\text{send}[t.p] \): string of all messages sent at or before state \( s \) from \( s.p \) to \( t.p \).

  \( t.\text{received}[s.p] \): string of all messages received at or before state \( t \) from \( t.p \) to \( s.p \).

The channel predicate can then be written as:

\[
c_j(s.\text{send}[t.p], t.\text{received}[s.p])
\]

or in short notation as:

\[
c_j(S, R) \equiv c_j(s.\text{send}[t.p], t.\text{receive}[s.p])
\]

- Requirements for monotonicity
Examples

Example 1 *Empty channels*: \( \text{len}(S) = \text{len}(R) \): This says that if a channel predicate is false, then it cannot be made true by sending more messages without receiving more messages.

Example 2 *Nonempty channels*: \((\text{ns} > \text{nr})\): \((\text{ns} - \text{nr} > nk)\) /* at least k messages in the channel */
GCP-cuts

$C$: global cuts that satisfy a GCP with monotone channel predicates

- $C \leq D$ iff $\forall i : C[i] \leq D[i]$. We show that the concept of first cut that satisfies a GCP is well-defined.

**Theorem 2** If $C, D \in C$, then their greatest lower bound is also in $C$.

**Proof:**
Example: no first cut in general

predicate: There are an odd number of messages in the channel. true only at points $C[1]$ and $D[1]$ for $P_1$, and $C[2]$ and $D[2]$ for $P_2$.
the GCP is true in the cut $C$ and $D$ but not in their greatest lower bound.

Figure 2: consistent cuts satisfying a GCP is not a lattice.
**Non-checker process algorithm**

initially $\forall j: j \neq i : \text{lcmvector}[j] = 0$

$\text{lcmvector}[i] = 1$

firstflag = true; incsend = increcv = Ø

**For** sending m **do**

send (lcmvector, m);

$\text{lcmvector}[i]++$

firstflag := true;

incsend := incsend $\oplus$ m;

**Upon** receive (msg_lcmvector, m) **do**

$\text{lcmvector} := \text{max}(\text{lcmvector}, \text{msg_lcmvector})$

firstflag := true;

increcv := increcv $\oplus$ m;

**Upon** (local_pred = true) $\land$ firstflag **do**

send (lcmvector, incsend, increcv) to checker;

firstflag := false; incsend := increcv := Ø
Data Structures of the Checker Process - per-process data

- **cut**: array[1..n] of struct v: vector of integer; color: red, green
  - The color of a state is either red or green.
    - green: the current state is concurrent with the current states from all other green processes.
    - red: the current state cannot be part of a GCP cut

- A FIFO queue of successive local snapshots from this process.

- **q**: array[1..n] of queues of struct
  - v: vector of integer;
  - incsend: array[1..n] of sequences of messages;
  - incrcv: array[1..n] of sets of messages;
Per-Channel Data

three data structures for each channel:

1. A pending-send list: messages sent but not yet received S[i,j]: sequence of message info;

2. A pending-receive list: ordered list of message sequence numbers. R[i,j]: sets of message info;

3. A CP-state flag. Value of channel predicates
   - T (true) only if the channel predicate for that channel is true for the current cut
   - F (false) only if the channel predicate for that channel is false for the current cut.

The CP-state flag can take the value X (unknown) at any time. cp[i,j]: X, F, T
Formal description

\[ S[1..n,1..n], R[1..n,1..n] : \text{sequence of message}; \]
\[ cp[1..n,1..n] : \{X, F, T\}; \]
\[ cut : \text{array}[1..n] \text{ of struct} \{ \]
\[ \quad \text{v : vector of integer}; \]
\[ \quad \text{color : \{red, green\}}; \]
\[ \quad \text{incsend, increcv : sequence of messages} \} \]

**initially**

\[ \text{cut}[i].v = 0; \text{cut}[i].\text{color} = \text{red}; S[i,j], R[i,j] = \emptyset; \]

**repeat**

\[ \text{while } (\exists i : (\text{cut}[i].\text{color} = \text{red}) \land (q[i] \neq \emptyset)) \]
\[ \quad \text{cut}[i] := \text{receive}(q[i]); \]
\[ \quad \text{paint-state}(i); \]
\[ \quad \text{update-channels}(i); \]

**endwhile**

\[ \text{if } (\exists i,j : \text{cp}[i,j] = X \land \text{cut}[i].\text{color} = \text{green} \land \text{cut}[j].\text{color} = \text{green}) \text{ then} \]
\[ \quad \text{cp}[i,j] := \text{chanp}(S[i,j]); \]
\[ \quad \text{if } (\text{cp}[i,j] = F) \text{ then} \]
\[ \quad \quad \text{if } (\text{send-mono}(i,j)) \text{ cut}[j].\text{color} := \text{red}; \]
\[ \quad \quad \text{else cut}[i].\text{color} := \text{red}; /* \text{receive-mono}(i,j) */ \]
\[ \text{until } (\forall i : \text{cut}[i].\text{color} = \text{green}) \land (\forall i,j : \text{cp}[i,j] = T) \]
\[ \text{detect} := \text{true}; \]
**Update Channels**

update-channels(i)

for \((j : \text{cut}[i].\text{incsend}[j] \neq \emptyset)\) do

\[S' := S[i,j];\]
\[R' := R[i,j];\]
\[S[i,j] := S' \oplus (\text{cut}[i].\text{incsend}[j] - R');\]
\[R[i,j] := R' - \text{cut}[i].\text{incsend}[j];\]

if \((\neg \text{send-mono}(i,j) \lor \text{cp}[i,j] = T)\) \(\text{cp}[i,j] := X;\)

for \((j : \text{cut}[i].\text{increcv}[j] \neq \emptyset)\) do

\[S' := S[j,i];\]
\[R' := R[j,i];\]
\[R[j,i] := R' \oplus (\text{cut}[i].\text{increcv}[j] - S');\]
\[S[j,i] := S' - \text{cut}[i].\text{increcv}[j];\]

if \((\neg \text{recv-mono}(j,i) \lor \text{cp}[j,i] = T)\) \(\text{cp}[j,i] := X;\)
Overhead analysis

- **Time complexity:**
  - any state is compared to at most $n$ other states.
  - There are $mn$ states in all. Therefore, $mn^2$ comparisons
  - at most two evaluations of the predicate per message.
  - at most $2mn$ message send and receive events.
  - each predicate evaluation takes at most $c$ time units, The total time spent is $2mnc$.

- **Space complexity**
  - $n$ queues each with at most $m$ elements. assume that component of each vector and every message: a constant number of bits.
  - Therefore, for each queue: $O(mn)$.
  - Summing up all incremental channel histories, we get $O(m)$.
  - Total space required by the checker process is $O(mn^2)$. 

©Vijay K. Garg

Distributed Systems Spring 96
• **Message Complexity** Every process sends at most $m$ messages to the checker process. Using same assumptions (space complexity): $O(mn)$ bits sent by each process.