Part 1: Self stabilization

Part 3.1: Clock synchronization (of Gouda and Herman)

Clock synchronization

Shared memory, synchronous

A new approach to fault tolerance
A unified approach to transient failures by formally incorporating them into the design model

If \( t(p) = t(x) \) then \( t(p) := t(p) + 1 \):

Is it stabilizing? No! (may even deadlock)

Program at each processor \( p \):

Choose a neighbor \( x \) of \( p \);

If \( t(p) = t(x) \) then \( t(p) := t(p) + 1 \);

Stabilizing if all start with the same time, however...
if \( t(p) = t(s) \) then \( t(p) = t(p) + 1 \);
if \( t(p) < t(s) \) then \( t(p) = t(s) \).

Is it stabilizing? yes!

Program at each processor \( p \):
choose a neighbor \( s \) of \( p \) fairly:
if \( t(p) = t(s) \) then \( t(p) = t(p) + 1 \);
if \( t(p) < t(s) \) then \( t(p) = t(s) \).

Is it stabilizing? yes!

proof

\[
f(s) = n \times \text{range}(s) + \text{top}(s)
\]

\[
\begin{align*}
S: & \quad 6 \quad 7 \\
4 & \quad 7
\end{align*}
\]

\[
n=5, \text{range}(S)=7-4=3, \text{top}(S)=2
\]

\[
f(s) = 5 \times 3 + 2 = 17
\]
Part 3.2: Mutual exclusion (Dijkstra’s 1st algorithm)

- distributed control
- ring of (finite-state machine) processes
- for each machine define privileges (when a transition is enabled)
- nodes are influenced only by neighbors
- legitimate states
- in each legitimate state there is at least one privileged machine
- in each legitimate state each possible move will bring the system again to a legitimate state

A system is **self-stabilizing** if, regardless of the initial state and regardless of the privilege selected each time for the next move, at least one privilege will always be present and the system is guaranteed to find itself in a legitimate state after a finite number of moves.

Use for:
- correctness (especially termination) + complexity
In Dijkstra's model:

- Token ring.
- N machines, denoted 0 through N-1.
- A machine is privileged ("has token") or not privileged.
- daemon/scheduler determines which of the privileged machines will make the next move
- Legitimate States: all states with exactly one privileged machine.

Dijkstra's 1st algorithm

- N processors 0,1,...,N-1
- Solution with k-state Machines
- machine \( P_0 \)

  \[
  \text{if } x[0] = x[N-1] \text{ then } x[0] := x[0]+1 \mod k
  \]

- machine \( P_i \) (i = 1,2,...,N-1)

  \[
  \text{if } x[i] ≠ x[i-1] \text{ then } x[i] := x[i-1]
  \]

Notes to consider:

- scheduler (daemon):
  - points at each time to one of the privileged machines, or
  - point at each time to one of the machines
  - centralized scheduler, distributed scheduler
  - fairness:
    - Do we need fairness, and of what kind?
    - Unconditional Fairness: Each machine gets pointed to infinitely often.
    - Type of fairness:
      - each machine is pointed at infinitely often,
      - round robin, or
      - other.

N=4, K=3

State: 0,1,2
**Example 1**

Machine $P_0$: if $x[0] = x[N-1]$ then $x[0] = x[0] \bot x[0]$ mod $k$

Machine $P_i$: if $x[i] \neq x[i-1]$ then $x[i] = x[i-1]$

**Example 2**

Machine $P_0$: if $x[0] = x[N-1]$ then $x[0] = x[0] \bot x[0]$ mod $k$

Machine $P_i$: if $x[i] \neq x[i-1]$ then $x[i] = x[i-1]$

**Example 3**

Machine $P_0$: if $x[0] = x[N-1]$ then $x[0] = x[0] \bot x[0]$ mod $k$

Machine $P_i$: if $x[i] \neq x[i-1]$ then $x[i] = x[i-1]$

Note:

<table>
<thead>
<tr>
<th>Machine Number</th>
<th>Privileged</th>
<th>Non-privileged</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, ..., N-1</td>
<td>$X[0] = X[N-1]$</td>
<td>$X[0] \neq X[N-1]$</td>
</tr>
<tr>
<td>1, ..., N-1</td>
<td>$X[i] = X[i-1]$</td>
<td>$X[i] \neq X[i-1]$</td>
</tr>
</tbody>
</table>

Informally: "If machine is privileged, then make it non-privileged."
Intuition:

N=5, K = 5

Goal: Show that from an arbitrary state, the system stabilizes to a legitimate state in a finite number of steps.

Intuition:

N=5, K = 3

machine \( p_0 \): if \( x[0] = x[N-1] \) then \( x[0] := x[0] \mod k \)

machine \( p_i \): if \( x[i] \neq x[i-1] \) then \( x[i] := x[i-1] \)

Lemma 1: If all states are equal then the system is stabilized.

Lemma 2: (liveness) At every configuration, at least one machine is privileged.

Lemma 3: Starting from any configuration, \( p_0 \) will eventually make a move (hw)

Corollary: Starting at any configuration, \( p_0 \) will make an infinite number of steps.

(The still does not guarantee stabilization.)

Theorem:

Assuming a centralized scheduler: starting with any initial configuration, Dijkstra’s 1st algorithm stabilizes if \( K > N-2 \).
Theorem:
Assuming a centralized scheduler: starting with any initial configuration, Dijkstra’s 1st algorithm stabilizes iff $k > N-2$.

Proof:
Case 1: $k < N-1$  
Case 2: $k = N-1$  
Case 3: $k = N$  
Case 4: $k > N$  

Either one state is missing, ... Lemma 5  
or one state occurs twice.  

Machine $P_0$: if $x[0] = x[N-1]$ then $x[0] = x[N-1] \mod k$  
machine $P_1$: if $x[i] \neq x[i-1]$ then $x[i] = x[i-1]$

Definition: $P_0$ is called special in a configuration if $j \neq i \Rightarrow x[j] \neq x[i]$

Lemma 4: If $P_0$ is special in a configuration, then the system will eventually stabilize.

Lemma 5: If in a configuration one of the states is missing, then the system will eventually stabilize.
Theorem:

Assuming a distributed scheduler: starting with any initial configuration, Dijkstra's 1st algorithm stabilizes iff \[ K > N - 1. \]

\[ \text{hw: prove} \]

So, one state occurs twice.

Either \( P_0 \) is special, ... Lemma 4

or \( x[0] = x[1] \)

\( i < N - 1 \) \( P_0 \) is not enabled, and after any other move there is a missing state, and ... Lemma 5.

\( i = N - 1 \) If \( P_0 \), \( i = 0 \) makes a move, there is a missing state, and done by Lemma 5. If \( P_0 \) makes a move, we get back to the previous case.

Use of a potential function

\( S \) a configuration (a system state)

\( r(S) \) number of maximal runs of equal states

Example: \( S = 22201000, f(S) = 4 \)

\[ f(S) = \begin{cases} 
   r(S) + 1 & \text{if first}(S) = \text{last}(S) \\
   r(S) & \text{otherwise.}
\end{cases} \]

Note: \( S \) is legitimate iff \( f(S) = 2 \).

Self stabilization

- is a special kind of fault tolerance
- guarantees automatic recovery from a transient failure
- as a design goal is a too strong property and thus either too difficult to achieve or achieved at the expense of other goals

Some general comments on self stabilization

Applications

- Distributed data structures
- Digital and analog circuits
- Genetic algorithms
- Network protocols
- Sensor networks
Part 3.3: Mutual exclusion (Dijkstra’s 3rd algorithm)

- N processors 0,1,...,N-1
- Solution with 3-state Machines
- \( P_0 \): if \( x[0]+1=x[1] \) then \( x[0]:=x[0]-1 \)
- \( P_{N-1} \):
  - if \( (x[N-2]=x[0]) \wedge (x[N-1] \neq x[0]+1) \) then \( x[N-1]:=x[0]+1 \)
  - \( P_i \) (\( i = 1,2,...,N-2 \)):
    - if \( (x[i]+1=x[i-1]) \vee (x[i]-1=x[i+1]) \) then \( x[i]:=x[i]+1 \)

Back to Dijkstra’s algorithm

- Machine \( P_0 \): if \( x[0]=x[N-1] \) then \( x[0]:=x[0]+1 \mod k \)
- Machine \( P_i \): if \( x[i] \neq x[i-1] \) then \( x[i]:=x[i-1] \)

Uniform protocol: same program at each processor

Theorem: There is no uniform protocol that solves the mutual exclusion problem (under either a centralized or a distributed scheduler)
\( P_i \ (i = 1, 2, \ldots, N-2) : \)
\[
\text{if } (x[i]+1 = x[i-1]) \lor (x[i]+1 = x[i+1]) \text{ then } x[i] := x[i]+1;
\]

\[
\text{if } (x[0]+1 = x[1]) \lor (x[0]+1 = x[n-1]) \text{ then } x[0] := x[0]+1;
\]

\[
\begin{align*}
&x[0] \quad x[1] \quad x[2] \quad \ldots \quad x[n-1] \\
&0 \quad 2 \quad 1
\end{align*}
\]
Lemma 1: If there is a unique arrow in a configuration, then it moves back and forth infinitely.

Corollary: It suffices to show that from any configuration we eventually reach a configuration with a unique arrow.

Lemma 2: (liveness) Starting from any initial configuration, at least one process is privileged.

Lemma 4: Between every two successive moves of $P_{N-1}$ there is at least one move of $P_0$.

Proof: recall the program for $P_{N-1}$:
- if $(x[N-2]=x[0]) \land (x[N-1] \neq x[0]+1)$
  then $x[N-1] := x[0]+1$.
After $P_{N-1}$ makes a step $x[N-1]=x[0]+1$.
And this condition makes $P_{N-1}$ non-privileged.
This condition can become true only after $P_0$ makes a step $P_0$, if $x[0]+1=x[1]$ then $x[0]:=x[0]-1$.

Lemma 3: Starting from any configuration, eventually $f \geq 1$.

Proof: if during an execution $f \geq 1$ - OK.
else $f=0$, but then there is no arrow, and in the next step there is a unique arrow, and then apply Lemma 1.

For two neighbors

\[ x[i] = x[i+1] \]
\[ x[i] \rightarrow x[i+1] \]
\[ x[i] \leftarrow x[i+1] \]
Lemma 5: A sequence of moves in which $P_0$ does not move is finite.
(therefore - impossible by Lemma 2)

Proof (cont.):
from some point only steps 1 and 2.
Follows from topology.

Proof (cont.)
$P_0 \ldots P_0 \ldots P_0 \ldots P_0 \ldots \ldots$
phases
"there exists a rightmost arrow, and it points to the right"
Falsified in each phase:
$P_0 \ldots P_0 \ldots P_0 \ldots P_0 \ldots \ldots$

$F \ F \ F \ F \ F \ F \ F$

Complexity?

$\text{hw: } \frac{5}{6}n^2 \leq \text{stabilization time} \leq \frac{13}{18}n^2$

(following Chernoy, Shalom and Z.)

Theorem: Starting from any initial configuration, the system eventually stabilizes.

Proof: by Lemma 5 $P_0$ makes infinite no. of moves.
$P_0 \ldots P_0 \ldots P_0 \ldots P_0 \ldots \ldots$

This is done only in steps 3, 4, 6
6 - ok
so, assume only 3 and 4.
in each phase:
they have $\Delta f=-3$.
0 and 7 $\Delta f=2$.
others $\Delta f=0$.
so, each phase $f$ is decreased by at least 1.
References

- M. G. Gouda and T. Herman, Stabilizing Unison, Department of Computer Science, 1990.