Uniform Self-Stabilization
by J. Burns and J. Pachl

Self-Stabilizing System: $S = (G, R, \Gamma, \Delta)$

- $G = (V, E), \; |V| = n$
- $R$ - orientation of $G$
  specifies a total ordering on the neighbors for each $v$ in $V$
- $\Gamma = \Sigma_i \times \Sigma_i \times \ldots \times \Sigma_i$ - set of configurations where $\Sigma_i$ is a finite set of states of processor $P_i$
- $\Delta = \delta_1, \delta_2, \ldots, \delta_m$ - sequence of transition relations
  $\delta_i$ is the transition relation of $P_i$
  from $\Sigma_i \times \Sigma_i \times \ldots \times \Sigma_i$ to $\Sigma_i$
  $(R(i) = (v_0, \ldots, v_n))$

In words: $\delta_i(\gamma)$ is the set of configurations of the whole system if $P_i$ takes a step in $\gamma$.

Self-Stabilizing System: $S = (G, R, \Gamma, \Delta)$

For configuration $\gamma = (s_0, s_1, \ldots, s_n) \in \Gamma$ where
- $s_i$ is a state of $P_i$ (for all $i$).
- $x$ - possible next state of $P_i$ by $\delta_i$,
  i.e. if $R(i) = (v_0, \ldots, v_n)$ then $x \in \delta_i(s_0, s_1, \ldots, s_n)$.

Define $\delta_i(\gamma)$ as the set of configurations $\gamma'$ (of all processors)
  such that $\gamma' = (s_0, s_1, \ldots, s_i', \ldots, s_n)$ (for all possible values of $x$)

In words: $\delta_i(\gamma)$ is the set of configurations of the whole system if $P_i$ takes a step in $\gamma$.

Self-Stabilizing System: $S = (G, R, \Gamma, \Delta)$

- $P_i$ is enabled (privileged) at $\gamma$ if $\delta_i(\gamma)$ is not empty
- $\gamma \Rightarrow \gamma'$ is a step of $P_i$ if $\gamma' \in \delta_i(\gamma)$
- $\gamma \Rightarrow \gamma'$ is a step of $P_i$ for some $i$
- $\gamma_0 \Rightarrow \gamma_1 \Rightarrow \ldots$ is a computation of the system where
  $\gamma_j \Rightarrow \gamma_{j+1}$ for all $j$

Only one processor takes a step at a time - central daemon

An n-processor uniform ring

A system $S = (G, R, \Gamma, \Delta)$ where
- $G$ is a cycle
- processors are denoted by $0, 1, \ldots, n-1$ clockwise
- $R(i) = \{(i-1) \mod n, (i+1) \mod n\}$
- $\Sigma_0 = \Sigma_1 = \ldots = \Sigma_n$
- $\delta_0 = \delta_1 = \ldots = \delta_n$

Definition:
A system $S = (G, R, \Gamma, \Delta)$ is self-stabilizing if
  there is a set $\Lambda \subseteq \Gamma$, called the legitimate configurations of $S$,
satisfying:

1. No Deadlock
   For every $\gamma \in \Gamma$ there is a $\gamma' \in \Gamma$ such that $\gamma \Rightarrow \gamma'$
2. Closure
   For every $\lambda \in \Lambda$, every $\lambda'$ such that $\lambda \Rightarrow \lambda'$ is in $\Lambda$
3. No Livelock
   Every infinite computation of $S$ contains a configuration in $\Lambda$
4. Mutual Exclusion
   For every $\lambda \in \Lambda$, exactly one processor is enabled
5. Fairness
   For every processor $P_i$ every infinite computation consisting of
     configurations in $\Lambda$ contains an infinite number of steps by $P_i$

2, 4 and 5 deal only with legitimate states. The problem is 1 and 3.
**Theorem:**
There is no uniform n-processor self-stabilizing ring if n is composite.

**Proof:**
- \( (i + |P_i| \mod p) \)
- Symmetry: all the \( P_i \)s in \( [i] \) are in the same state in \( \gamma \)
- If Sym(\( \gamma \)) holds, we can construct an infinite computation \( \gamma_0 \rightarrow \gamma_1 \rightarrow \gamma_2 \rightarrow \ldots \) for which Sym(\( \gamma \)) holds for every \( i \)

\[ \gamma_i: \]

\[
\begin{array}{c c c c}
S_1 & P_1 & P_2 & S_2 \\
S_2 & P_3 & P_4 & S_1 \\
S_1 & P_1 & P_2 & S_2 \\
S_2 & P_3 & P_4 & S_1 \\
\end{array}
\]

**Main Result**
Algorithm for a ring of a prime size:
- uniform (no distinguished processor)
- deterministic algorithm
- central scheduler
- \( O(n^2) \) states
- unidirectional

**The algorithm**
- deterministic
- unidirectional
- (the value of \( \delta(s_i, s_{i+1}, s_{i+2}) \) depends only on \( s_i \) and \( s_{i+1} \))

**Notation:**
\[ s_{i+1} \rightarrow x \] means that \( x \in \delta(\{s_i, s_{i+2}, s_{i+3}\}) \)

**A First Attempt**
- \( \Sigma_0 = \Sigma_1 = \ldots = \Sigma_n = \{0, 1, \ldots, n-2\} \)
- arithmetic modulo n

The rule: if \( s_i \neq s_{i+1} \) then \( s_{i+2}, s_{i+3} \rightarrow s_{i+4} \)

**The rule: if \( s_i \neq s_{i+1} \) then \( s_{i+2}, s_{i+3} \rightarrow s_{i+4} \)**

The cycle of the intended legitimate configurations

<table>
<thead>
<tr>
<th>( P_0 )</th>
<th>( P_1 )</th>
<th>( P_2 )</th>
<th>( P_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

A configuration sequence which does not converge to the legitimate configurations

<table>
<thead>
<tr>
<th>( P_0 )</th>
<th>( P_1 )</th>
<th>( P_2 )</th>
<th>( P_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
A Correct Protocol

The Algorithm:

- \( n \geq 3 \)
- states are \((L_1, T_1)\)
- labels \( \in \{0, 1, \ldots, n-2\} \)
- tags \( \in \{0\} \cup \{2, 3, \ldots, n-2\} \)

Rule A:
- If \( L_1 \neq L_1+1 \) and 
  - \( L_0 \neq 0 \) or \( T_1 = 0 \) or \( T_1 \neq L_1 \) or \( T_1 = T_1 \)
  then \( L_1 \neq T_1 \) \( \xrightarrow{L_1} \) \( L_1 \neq T_1 \)
- If \( L_1 = L_1+1 \) and \( L_1 \neq 0 \) and \( T_1 = T_1 \)
  then \( L_1 \neq T_1 \) \( \xrightarrow{L_1} \) \( L_1 \neq T_1 \)

Rule B:
- If \( L_1 \neq L_1+1 \) and \( L_1 \neq 0 \) or \( T_1 = T_1 \)
  then \( L_1 \neq T_1 \) \( \xrightarrow{L_1} \) \( L_1 \neq T_1 \)

Definitions:
Let \( \gamma \in \Gamma \) be a configuration
- if \( L_1 \neq L_1+1 \) (mod \( n-1 \)), then \( P_1 \) and \( P_{n+1} \) form a gap of \( \gamma \)
- the gap size of \( \gamma \) is \( g(\gamma) = \frac{L_1 - L_1}{\text{mod } n-1} \)
- a segment of \( \gamma \) is a maximal cyclically contiguous sequence \( s = (P_1, P_2, \ldots, P_{n+1}) \) which contains no gaps
  - \( P_1 \) is the left end of \( s \), \( P_{n+1} \) is the right end of \( s \)
  - the gap size of \( s \) \( g(s, \gamma) \) is the gap size of its right end, \( g(\gamma) \)
  - a segment is zero based if its left end is \( 0 \) (\( L_1 = 0 \))

Example:

\[
\begin{array}{cccccc}
\text{P}_0 & \text{P}_1 & \text{P}_2 & \text{P}_3 & \text{P}_4 & \text{P}_5 \\
1&3&2&3&1&.52&0.2&1.1&0.4 & label.tag \\
\end{array}
\]
- \( n=7 \)
- labels \( \in \{0, 1, \ldots, 5\} \), arithmetic mod 6
- segment \( s_1 = (p_0, p_2, p_4, p_6) \)
- \( p_0, p_2, p_4, p_6 \) is zero based
  - \( g(s_1, \gamma) = 2 \)
  - \( g(s_2, \gamma) = 5 \)

The Protocol - Revised:

Rule A:
- If \( L_1 \neq L_1+1 \) and 
  - \( L_0 \neq 0 \) or \( T_1 = 0 \) or \( T_1 \neq L_1 \) or \( T_1 = T_1 \)
  then \( L_1 \neq T_1 \) \( \xrightarrow{L_1} \) \( L_1 \neq T_1 \)

Rule B:
- If \( L_1 = L_1+1 \) and \( L_1 \neq 0 \) and \( T_1 = T_1 \)
  then \( L_1 \neq T_1 \) \( \xrightarrow{L_1} \) \( L_1 \neq T_1 \)

- There are less distinct labels than processes, so a segment always exists
Recall:
A system $S = (G, R, \Gamma, \Delta)$ is self-stabilizing iff there is a set $\mathcal{A} \subseteq \Gamma$, called the legitimate configurations of $S$, satisfying:

1. **No Deadlock**
   For every $\gamma \in \Gamma$ there is a $\gamma' \in \Gamma$ such that $\gamma \rightarrow \gamma'$

2. **Closure**
   For every $\lambda \in \mathcal{A}$, every $\lambda'$ such that $\lambda \rightarrow \lambda'$ is in $\mathcal{A}$

3. **No Livelock**
   Every infinite computation of $S$ contains a configuration in $\mathcal{A}$

4. **Mutual Exclusion**
   For every $\lambda \in \mathcal{A}$, exactly one processor is enabled

5. **Fairness**
   For every processor $P_i$, every infinite computation consisting of configurations in $\mathcal{A}$ contains an infinite number of steps by $P_i$

So, we have to prove:
A system $S = (G, R, \Gamma, \Delta)$ is self-stabilizing iff there is a set $\mathcal{A} \subseteq \Gamma$, called the legitimate configurations of $S$, satisfying:

1. **No Deadlock**
   For every $\gamma \in \Gamma$ there is a $\gamma' \in \Gamma$ such that $\gamma \rightarrow \gamma'$

2. **Closure**
   For every $\lambda \in \mathcal{A}$, every $\lambda'$ such that $\lambda \rightarrow \lambda'$ is in $\mathcal{A}$

3. **No Livelock**
   Every infinite computation of $S$ contains a configuration in $\mathcal{A}$

4. **Mutual Exclusion**
   For every $\lambda \in \mathcal{A}$, exactly one processor is enabled

5. **Fairness**
   For every processor $P_i$, every infinite computation consisting of configurations in $\mathcal{A}$ contains an infinite number of steps by $P_i$
Lemma: for every $\gamma \in \Gamma$ there is a $\gamma' \in \Gamma$ such that $\gamma \rightarrow \gamma'$

Rule A: if $L_1, L_2, L_3$ and $(\emptyset, \emptyset)$ or $T_i, L_1 \rightarrow T_i, L_1$

Rule B: if $L_1, L_2, L_3$ and $L_4 \neq \emptyset$ and $L_4 \neq \emptyset$, then $L_1, L_2, L_3 \rightarrow L_1, L_2, L_3$

• Case 1: one zero-based segment of length $n$
  $\ldots \; O, T_i \; \ldots$
  if $T_i = 0$ rule A applies
  else $T_i \neq 0$, so $T_i = L_i - 1$, and again rule A applies

• Rule A doesn’t apply, so all the segments are zero-based

• Rule B doesn’t apply: all tags within each segment are equal

For every infinite computation of $S$ contains a configuration in $L$

• Rule A doesn’t apply, so all the segments are zero-based

• Rule B doesn’t apply: all tags within each segment are equal

For every processor $P_i$, every infinite computation consisting of configurations in $L$ contains an infinite number of steps by $P_i$
Definition:

Let $C = \gamma_1 \gamma_2 \ldots$ be a quiet computation

- a dynamic gap is a function $g$ from indices of computation into indices of processors, satisfying the following:
  1. processor $z(j)$ is the right end of the segment in $\gamma_i$
  2. if $\gamma_j > \gamma_i$ is a step of $P$, by Rule A and $z(j) = i-1$ then $z(j) = i$, else $z(j) = j-1$

Then $g(z(j)) = g(z(j+1), \gamma_j)$ for all $j$.

Claim: in a quiet computation gap sizes remain constant.

Informally:

\[
\begin{array}{c|c|c|c|c}
1 & 2 & 3 & 8 & 9 & 10 & 11 \\
\hline
1 & 2 & 3 & 4 & 9 & 10 & 11 \\
\end{array}
\]

Formally ...

For a dynamic gap $g$, $g(z)$ denotes this common value.

Formally $g(z) = g(z(j), \gamma_j)$.

Claim: For each dynamic segment in a quiet computation there is a configuration in which the right end has label 0, and there is a configuration in which the left end has label 0.

Informally ...

Lemma:

Let $C = \gamma_1 \gamma_2 \ldots$ be a quiet computation and $z$ a dynamic gap of $C$.

Then $g(z(j)), \gamma_j = g((z(j)), \gamma_j)$ for all $j$.

Proof:

- If $P_i$ takes a step and $z(j) = i$, the segment disappears – a contradiction
  \[
  g(z(j), \gamma_j) = g(z(j+1), \gamma_j)
  \]
- $P_i = z(j) = i+1$ takes a step - does not affect the gap
- $P_i = z(j) = i-1$ takes a step - does not affect the gap
- $P_i = z(j) = i$ remains unchanged

So: in a quiet computation gap sizes remain constant.

Formally $g(z) = g(z(j), \gamma_j)$.

Claim: For each dynamic segment in a quiet computation there is a configuration in which the right end has label 0, and there is a configuration in which the left end has label 0.
But it is applied to $\gamma$.

**Claim:** For each dynamic segment in a quiet computation there is a configuration in which the right end has label 0, and there is a configuration in which the left end has label 0.

Formally...

...the right end has label 0:

**Lemma:** Let $C = \gamma_i, \gamma_j$ be an infinite quiet computation and $z$ a dynamic gap of $C$.

Then there is $i$, such that the state of processor $z(j)$ in $\gamma_i$ is $0.g(z)$.

**Proof:** Consider $z$ at a time $j'$.

$\gamma_i$:  

\[ \begin{array}{c|c|c} \hline \ & \ P_i & \ P_j \\ \hline a^* & \hline \end{array} \]

$z(j)$

Rule A will be applied to $P_i$, and to $P_j$.

But it is applied to $P_i$ before it can applied to $P_j$, otherwise the number of segments decreases, which is impossible since the computation is quiet.

...the left end has label 0:

**Lemma:** Let $C = \gamma_0, \gamma_1$ be an infinite quiet computation and $z$ a dynamic gap of $C$.

Then there is $k \geq 0$, such that the state of processor $z(k) + 1$ in $\gamma_k$ is $0.u$ for some $u$.

**Proof:** we eventually get to a configuration

$\gamma_i$:  

\[ \begin{array}{c|c|c} \hline \ & \ P_i & \ P_j \\ \hline 0.g(z) & \hline \end{array} \]

$z(k) + 1 = P_{i+1}$

If $P_{k+1}$ is the first element of $z'(j)$, then we are done.

Else, consider the first time $P_i$ takes a step by rule $A$.

When Rule A is applied to $P_i$, we get from

$\gamma_i$:  

\[ \begin{array}{c|c|c} \hline \ & \ P_i & \ P_j \\ \hline a^* & \hline \end{array} \]

$z(j')$

to

$\gamma_i$:  

\[ \begin{array}{c|c|c} \hline \ & \ P_i & \ P_j \\ \hline a^* \cdot [z(j) + 1.g(z)] & \hline \end{array} \]

$z(j'')$

So eventually we get to a configuration

$\gamma_i$:  

\[ \begin{array}{c|c|c} \hline \ & \ P_i & \ P_j \\ \hline 0.g(z) & \hline \end{array} \]

$z(j)$

We consider the first time $P_i$ takes a step by rule $A$.

But therefore $P_{i+1}$ cannot take any step by Rule $A$, so its label must remain 0.

So, from the configuration

$\gamma_i$:  

\[ \begin{array}{c|c|c} \hline \ & \ P_i & \ P_j \\ \hline a & \hline \end{array} \]

$z(k) + 1 = P_{i+1}$

We eventually get to a configuration

$\gamma_i$:  

\[ \begin{array}{c|c|c} \hline \ & \ P_i & \ P_j \\ \hline a.g(z) & \hline \end{array} \]

$z(k) + 1 = P_{i+1}$

$[z(j) + 1.g(z)]$

$z(j)$

$z(k) + 1 = P_{i+1}$
Recall: we want to show that the number of segments is eventually reduced to one.

Definition:
A segment is well formed if all of its tags are equal to its gap size.

Claim: From some point all segments are well formed.

Lemma: Let \( C = \gamma_0, \gamma_1, \ldots \) be an infinite quiet computation. Then there is \( j_0 \), such that for all \( j \geq j_0 \), all segments in \( \gamma_j \) are well formed.

Proof: Consider a specific segment. We get to
\[
\gamma_i: \quad \ldots \quad 0.g(z) \quad T
\]
We know that Rule A will be applied to \( P_{u_i} \).
Rule A cannot be applied to \( P_i \) earlier, since then the number of segments will decrease.
Rule B cannot be applied to \( P_i \) earlier, since \( L_i = 0 \).

So, eventually we get to
\[
\gamma_i: \quad \ldots \quad 0.g(z) \quad 1.g(z) \quad \ast.g(z) \quad \ast.g(z)
\]
Where all tags in a segment are equal.
Rule B cannot be applied within this segment.
So this segment will remain well-formed from now on.
Apply this argument, from this configuration, to the \( 2^{th} \), \( 3^{rd} \), \ldots segment.
So, eventually all segments are well-formed.

Claim: for all gaps, \( g(z) = 0 \).
Proof: if there is only one segment, the gap must be 0. Otherwise, consider a maximal gap, \( g(z) > 0 \).
Let \( z' \neq z \). At some point \( z' \) is 0 based
\[
s_{z'} \quad s_z
\]
\[
\ast.g(z) \quad 0.g(z) \quad \ast.g(z)
\]
\( P_{u_i} \) can make a step only after \( P_i \) but such a step would destroy \( s_z \Rightarrow contradiction \)
Rule A: if $L_i \neq L_{i-1} + 1$ and ($L_i \neq 0$ or $T_{i-1} = 0$ or $T_i \neq L_i$ or $T_i < T_{i-1}$) then $L_i$.T$_{i-1}$ L$_i$.T$_i$ $\rightarrow$ ($L_i - 1 + 1$).($L_i - L_{i-1}$)

Rule B: if $L_i = L_{i-1} + 1$ and $L_0 \neq 0$ and $T_{i-1} \neq T_i$ then $L_i$.T$_{i-1}$ L$_i$.T$_i$ $\rightarrow$ L$_i$.T$_i$

So, all gaps have $g(z) = 0$.

There is either 1 segment of length $n$, or $n$ segments of length 1.

If we have $n$ segments of length 1, then any Rule A decreases the number of segments – a contradiction.

So, there is a single segment of length $n$, as desired.

Recall: the set of legitimate configurations, $\mathcal{A}$, is the set of all cyclic permutations of the form
0.0 1.0 ... a-1.0 a.0 a.0 a+1.0 ... n-2.0
for $a=0, 1, ..., n-2$

Eventually the system reaches a configuration $\gamma$:
\[ \gamma: \_ \_ a\_0a \_ \_ \_ \_ a0 a+1.0 \_ \_ \_ \_ \]

with one well-formed segment, $g(z) = 0$, and all tags = 0.
This is a legitimate configuration.
This completes the proof.

Conclusion
We saw an algorithm for a ring of a prime size:
• uniform (no distinguished processor)
• deterministic algorithm
• central scheduler
• O($n^2$) states
• unidirectional

Questions
• Can we find a simpler algorithm?
• Can we find an algorithm with $o(n^2)$ states?
• Estimate the convergence time of the algorithm.
• Algorithms for other graphs?
  For Cayley graphs?
• Give a lower bound for the number of states and the convergence time.