Self Stabilization

CS553 Distributed Algorithms
Prof. Ajay Kshemkalyani

by
Islam Ismailov & Mohamed M. Ali

Introduction

- There is a possibility for a distributed system to go into an **illegitimate state**, for example, if a message is lost.
- **Self-stabilization**: regardless of initial state, system is guaranteed to converge to a legitimate state in a bounded amount of time without any outside intervention.
- **Problem**: nodes do not have a global memory that they can access instantaneoulsy.

System Model

- An abstract computer model: state machine.
- A distributed system model comprises of a set of n state machines called processors that communicate with each other, which can be represented as a graph.
- Message passing communication model: queue(s) $Q_{ij,}$ for messages from P_i to P_j
- System configuration is set of states, and message queues.
- In any case it is assumed that the topology remains connected, i.e., there exists a path between any two nodes.

Definition

- States satisfying P are called legitimate states and those not satisfying P are called illegitimate states.
- A system S is self-stabilizing with respect to predicate P if it satisfies the properties of closure and convergence

Dijkstra's self-stabilizing token ring system

- When a machine has a privilege, it is able to change its current state, which is referred to as a *move*.
- A legitimate state must satisfy the following constraints:
  - There must be at least one privilege in the system (liveness or no deadlock).
  - Every move from a legal state must again put the system into a legal state (closure).
  - During an infinite execution, each machine should enjoy a privilege an infinite number of times (no starvation).
  - Given any two legal states, there is a series of moves that change one legal state to the other (reachability).

Dijkstra considered a legitimate (or legal) state as one in which exactly one machine enjoys the privilege.

Dijkstra's system (contd)

- For any machine, we use symbols S, L, and R to denote its own state, state of the left neighbor and state of the right neighbor on the ring.
- The exceptional machine:

```
If L = S then
    S = (S+1) mod K;
```

- All other machines:

```
If L = S then
    S = L;
```

Dijkstra's system (contd)

- Note that a privilege of a machine is ability to change its current state on a Boolean predicate that consists of its current state and the states of its neighbors. When a machine has a privilege, it is able to change its current state, which is referred to as a move.
- Second solution (K = 3)
  - **The bottom machine, machine 0:**

    If (S+1) mod 3 = R then S = (S−1) mod 3;
  - **The top machine, machine** n−1**:**

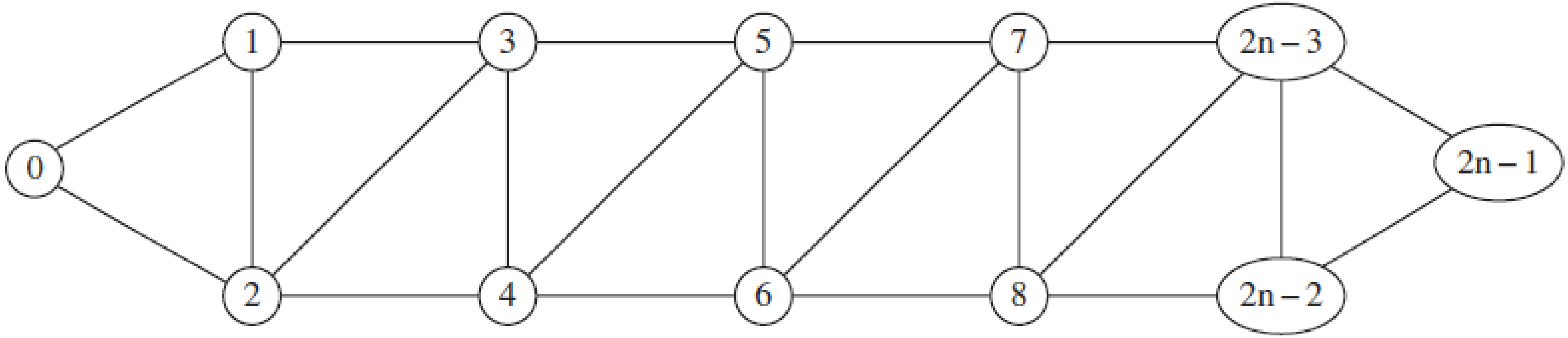
    If L = R and (L+1) mod 3 = S then S = (L+1) mod 3;
  - **The other machines:**

    If (S+1) mod 3 = L then S = L;

Example

State of machine 0	State of machine 1	State of machine 2	State of machine 3	Privileged machines	Machine to make move
0	1	0	2	0, 2, 3	0
2	1	0	2	1, 2	1
2	2	0	2	1	1
2	0	0	2	0	0
1	0	0	2	1	1
1	1	0	2	2	2
1	1	1	2	2	2
1	1	2	2	1	1
1	2	2	2	0	0
0	2	2	2	1	1
0	0	2	2	2	2
0	0	0	2	3	3
0	0	0	1	2	2

Systems with less than three states per node



Ghosh system (contd)

For machine 0:

If $(s[0], s[1]) = (\tilde{b}, b)$ then $s[0] := b$

For machine $2n-1$:

If $(s[2n-1], s[2n-2]) = (b, b)$ then $s[2n-1] := \tilde{b}$

For even numbered machines:

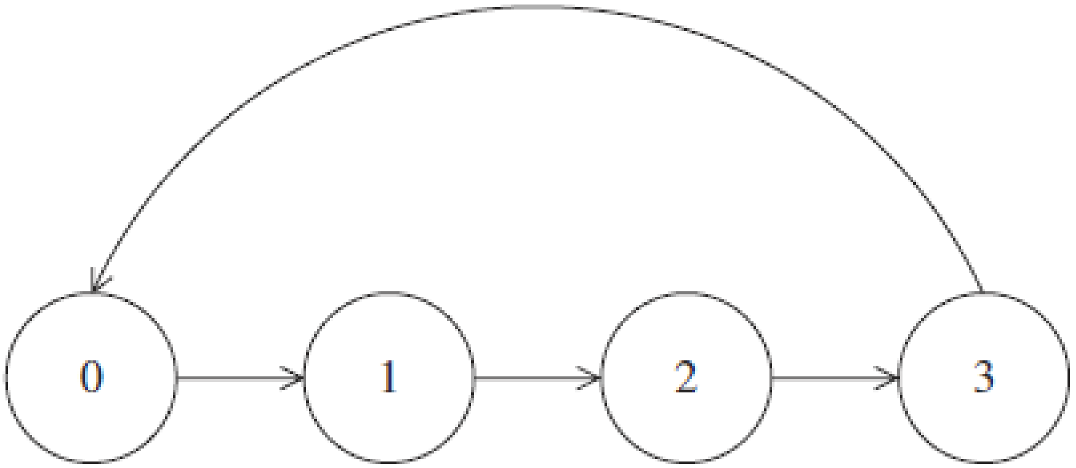
If $(s[2i-2], s[2i-1], s[2i], s[2i+1]) = (b, b, \tilde{b}, b)$ then $s[2i] := b$

For odd numbered machines:

If $(s[2i-2], s[2i-1], s[2i], s[2i+1]) = (b, b, b, \tilde{b})$ then $s[2i-1] := \tilde{b}$

Uniform vs Non-uniform

- From the examples of the preceding section, we notice that at least one of the machines (exceptional machine) had a privilege and executed steps that were different from other machines.
- The individual processes can be anonymous, meaning they are indistinguishable and all run the same algorithm.



Central and distributed daemons

- Generally, the presence of a central demon is assumed in self-stabilizing algorithms.
- Distributed demon is more desirable in distributed systems.
- The presence of a central demon considerably simplifies the verification of a weak correctness criterion of a self-stabilizing algorithm.

Reducing the number of states in token ring

- In a self-stabilizing token ring with a central demon and deterministic execution, Ghosh showed that a minimum of three states per machine is required.
- There exists a non-trivial self-stabilizing system with two states per machine. It requires a high degree of atomicity in each action.

Shared memory models

- Two processors, P_i and P_j, are neighbors, then there are two registers, i and j, between the two nodes. To communicate, P_i writes to i and reads from j and P_j writes to j and reads from i.
- Dolev et al. present a dynamic self-stabilizing algorithm for mutual exclusion. Node failures may cause an illegal global state, but the system again converges to a legal state.

Mutual Exclusion

- In a mutual exclusion algorithm, each process has a critical section of code. Only one process enters its critical section at any time, and every process that wants to enter its critical section, must be able to enter its critical section in finite time.
- A self-stabilizing mutual exclusion system can be described in terms of a token system, which has the processes circulating tokens. Initially, there may be more than one token in the system, but after a finite time, only one token exists in the system which is circulated among the processes

Costs of self-stabilization

- A study and assessment of cost factors is very important in any practical implementation.
- **Convergence span** The maximum number of transitions that can be executed in a system, starting from an arbitrary state, before it reaches a safe state.
- **Response span** The maximum number of transitions that can be executed in a system to reach a specified target state, starting from some initial state. The choice of initial state and target state depends upon the application.

Methodologies for design

- After malicious adversary disrupts the normal operation of the system. If enough components are left for the system to operate, then a self-stabilizing system will slowly resume
- Normal operation after the attack. If not, system is destroyed.

Layering and modularization

- Self-stabilization is amenable to layering because the self-stabilization relation is transitive.
- Time in shared memory systems must meet these properties:
- **Safety** All clocks have the same value. (Differ at most 1)
- **Progress** At each step, each clock is incremented by the same amount. (i+1 when neighbors are i or i+1)
- Topology based primitives: leader election.

Communication protocols

- Communication protocol might be affected due to:
  - Initialization to an illegal state.
  - A change in the mode of operation. Not all processes get the request for the change at the same time, so an illegal global state may occur.
  - Transmission errors because of message loss or corruption.
  - Process failure and recovery.
  - A local memory crash which changes the local state of a process.

Communication protocols

- Communication protocol must satisfy the following three properties to be self-stabilizing:
  - It must be non-terminating.
  - There are an infinite number of safe states.
  - There are timeout actions in a non-empty subset of processes.

Dolev, Israeli, and Moran Algorithm

- Self stabilizing BFS spanning-tree construction.
- Properties:
  - Semi-uniform systems
  - Central daemon
  - Assume read/write atomicity
- Every node maintains:
  - A pointer to one of its incoming edges.
  - An integer measuring number of hops from root of tree.

Dolev, Israeli, and Moran Algorithm (cont.)

- Nodes periodically exchange their distance value with each other, (root node always sends a value of 0).
- Each node chooses the neighbor with minimum distance as its new parent, and updates its distance accordingly.
- After reading all neighbors values for k times, distance value of a process is at least $k+1$.

Variables:

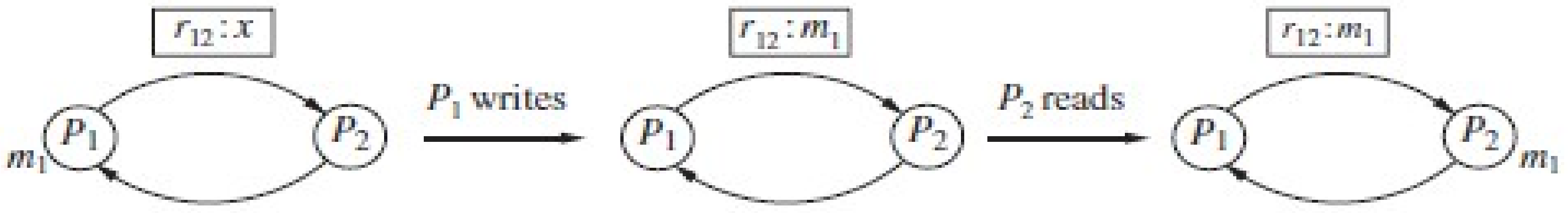
$no\_neighbors$ = Number of processor's neighbors
i = the writing processor
m = for whom the data is written
lr_{ji} (local register r_{ji}) the last value of r_{ji} read by P_i

```
Root Node:
{do forever}
while TRUE do
     for $m := 1$ to $no\_neighbors$ do
          write $lr_{im} := < 0, 0 >$
     end
end
```

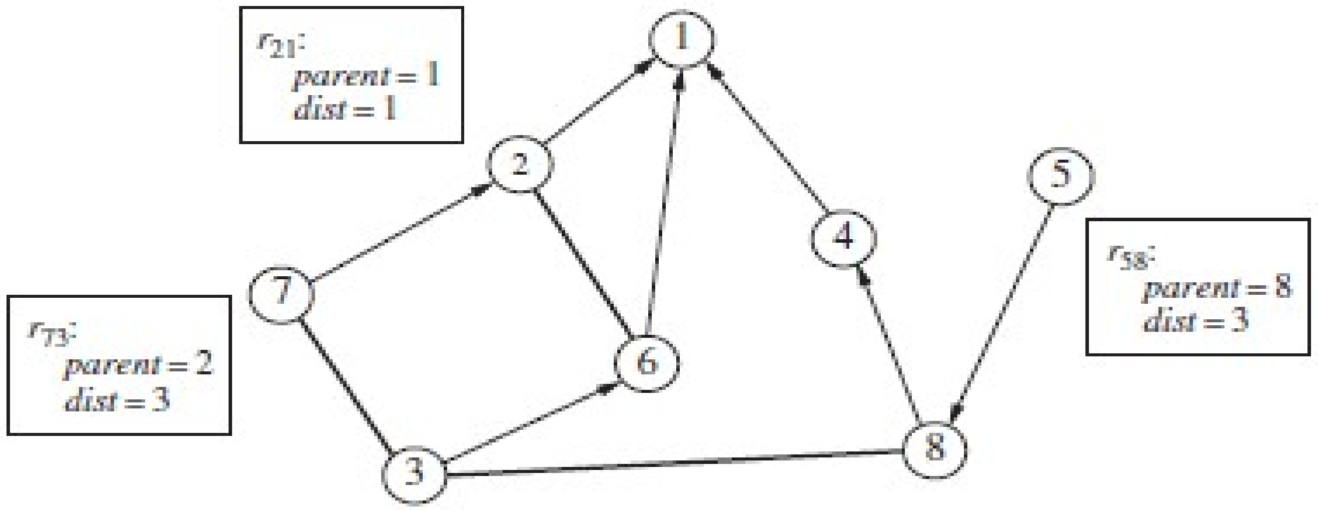
```
Other Nodes:
{do forever}
while $TRUE$ do
     for $m := 1$ to $no\_neighbors$ do
          $lr_{mi}$ := read($lr_{mi}$)
          $FirstFound := false$
           $dist := 1 + \min(lr_{mi}.dist)\ \forall\ m$: $1 \leq m \leq no\_neighbors$
          for $m := 1$ $to$ $no\_neighbors$ do
               if not $FirstFound$ and $lr_{\ mi}.dis = dist - 1$ then
               write $r_{im} := <1, dist>$
               $FirstFound := true$
               else write $r_{im} := <0, dist>$
          end
     end
end
```

Example

(a) The distributed system – computation step



(b) Spanning-tree, system and code



r_{21}: *parent* = 1 *dist* = 1
r_{73}: *parent* = 2 *dist* = 3
r_{58}: *parent* = 8 *dist* = 3
1
2
3
4
5
6
7
8

Afek, Kutten, and Yung Algorithm

- BFS spanning-tree, in read/write atomicity model.
- No distinguished process assumption.
- All nodes have globally unique identifiers that can be totally ordered.
- The largest identifier will be the root of tree.
- Similar to Dolev *et al.*, but also exchange the current root which a node think it is present.
- If a larger root appears, send a join request to the other sub-tree, and wait for grant message.

Arora and Gouda Algorithm

- BFS spanning-tree, in composite atomicity model, with central daemon assumption.
- All nodes have globally unique identifiers that can be totally ordered.
- The largest identifier will be the root of tree.
- Needs a bound N on the number n of nodes in network to work correctly.
- Cycles are detected when distance bound grows to exceed N.
- O(N^2) Vs. O(n^2) for Afek *et al.*

Afek and Bremler Algorithm

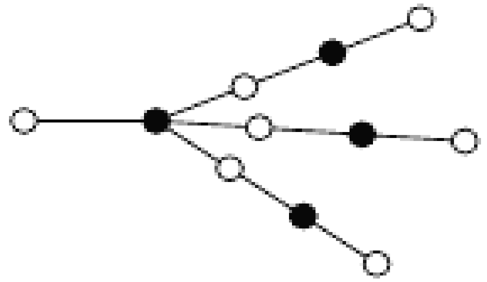
- For synchronous, and asynchronous networks.
- Node with smallest identifier is considered the root.
- Based on “Power Supply” idea.
- Power is a continuous flow of messages, one per round.
- When a node receives power from a neighbor with a smaller identifier, it attaches itself to the tree.

Afek and Bremler Algorithm

- Weak messages are exchanged between nodes to synchronize their states, while strong messages carry power.
- Stabilizes in O(n) rounds without process to have the knowledge of n.

1-Maximal Independent Set

- A maximal independent set is a set of nodes such that every node not in the set is adjacent to a node in the set.
- A 1-maximal independent set is maximal independent set provided one cannot increase the cardinality of the independent set by removing one node and adding more nodes.



A tree with the smallest 1-maximal independent set

Shi *et al.* Algorithm

- A connected, undirected graph with node set V and edge set E.
- N(i) denotes a set of neighbors of node *i*.
- Algorithm is presented as a set of rules, each with a boolean predicate and action.
- A node will be *privileged* if predicate is true.
- If a node is privileged, it may execute the corresponding action, called move.
- A central daemon is assumed to exist.
- Nodes in state '0' will be in the desired set.

Shi *et al.* Algorithm (cont.)

- Rules:
  1. If not involved in a transition process, then set state to the number of neighbors in state 0 or state 0'.
  2. If in state 0 and adjacent to at least two 1s, change to state 0.
  3. If in state 1 and adjacent to a 0', change state to 1'.
  4. If in state 0' and adjacent to at least two 1's, change state to 2'.
  5. If in state 1' and adjacent to no 0', change state to 0.
  6. If in state 2' and adjacent to no 1', change state to 2.

```
{* All moves are tried in the listed order *}
V1: if flag is set and node is incident on bad edge
    and after clearing flag node would not be incident on any bad edge
      then clear flag
V2: if flag is clear and $x' = f(0/0')$ and $(f(0/0') \geq 1$ or $f(1'/2') = 0)$
      then set $x = f(0/0')$
C1: if $x = 0$ and $f(1) = 2$ and $f(0/0'/2') = 0$
      then set $x = 0'$
C2: if $x = 0'$ and $(f(1/1') \leq 1$ or $f(0/0') \geq 1)$
      then set $x = f(0/0')$
C3: if $x = 0'$ and $f(1') = 2$ and $f(0'/2') = 0$
      then set $x = 2'$
C4: if $x = 2'$ and $f(1') = 0$
      then set $x = f(0/0')$
C5: if $x = 1$ and $f(0') = 1$ and $f(0/1'/2') = 0$
      then set $x = 1'$
C6: if $x = 1'$ and $(f(0') \neq 1$ or $f(0/1'/2') \geq 1)$
      then set $x = f(0/0')$
```

Shi *et al.* Algorithm (cont.)

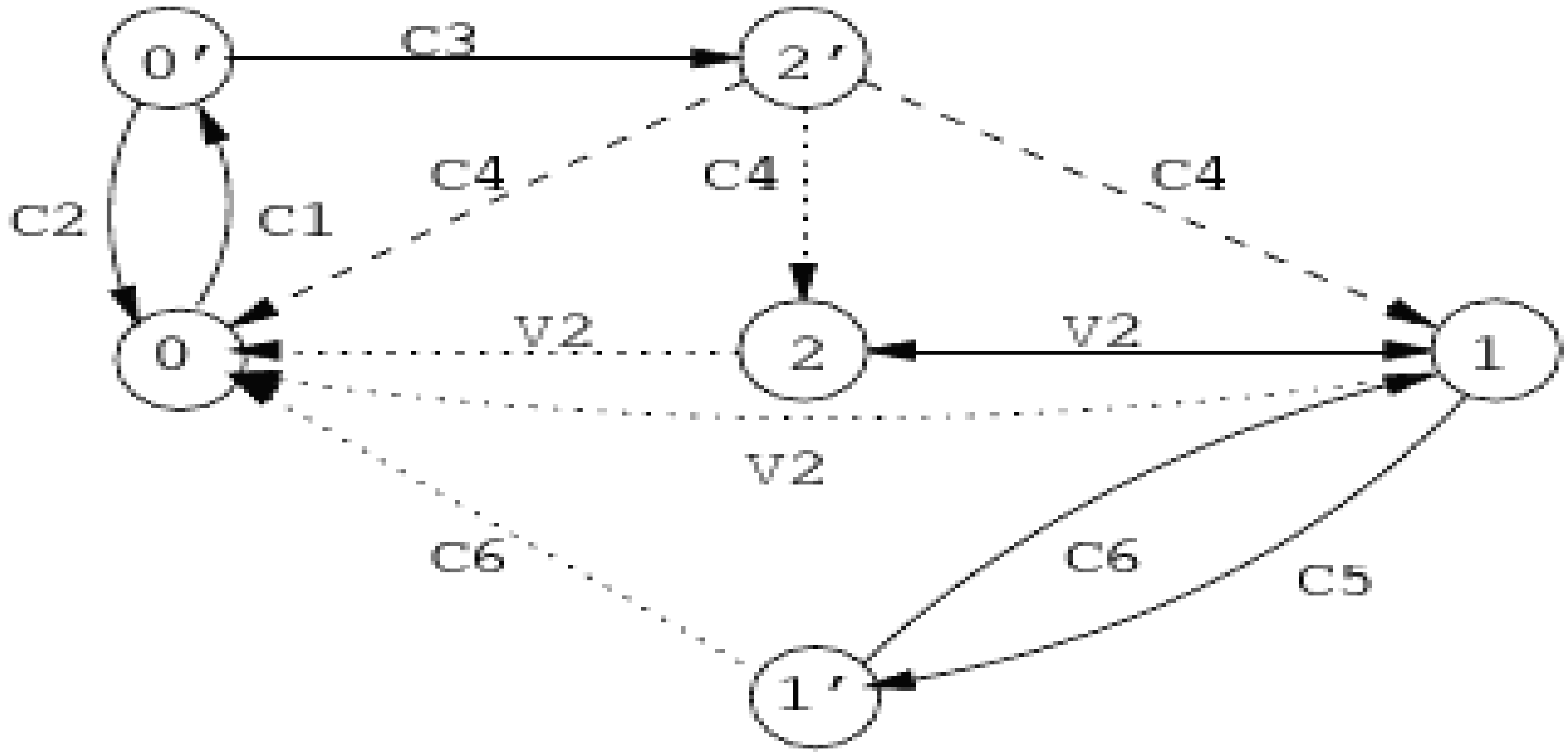
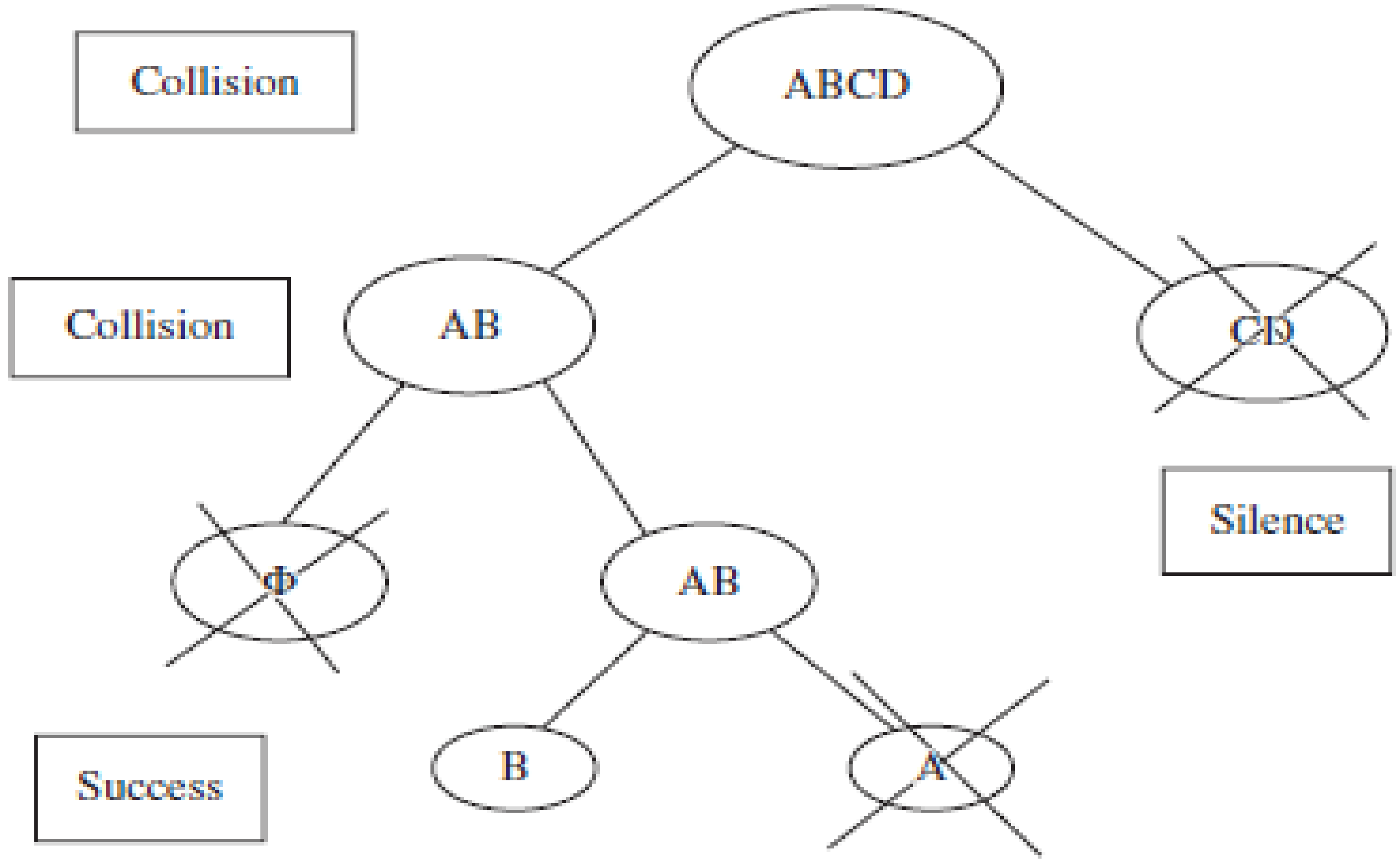


Figure 2: The clean moves

Probabilistic Self-Stabilizing Leader Election Algorithm

- All stations try to send messages via the channel. Collision!
- For a station S, flip a coin for retransmission. Accordingly, either retransmit or keep silent.
- Keep applying till no collision occurs, and accordingly leader is elected.
- Is there a probability of all being silent!?

Example



Self-Stabilizing Compilers

- Sequential Programs:
  - Rule based program (Brown et al.)
  - In initialization, a *rule* is a multiple assignment statement with an enabling condition called *guard*.
  - *guard* is a predicate over the variables of the program, which is updated at each state.
  - A computation is a sequence of rule firings, where at each step an enabled rule is non-deterministically selected for execution.
  - A program terminates when reaching a *fixed point* state where values of variables no longer change.

Self-Stabilizing Compilers (cont.)

- To force self-stabilization while preserving termination, a program must be:
  - Of acyclic data dependence graph.
  - Each rule in the program assigns only one variable.
  - For any pair of enabled rules with same target variable, both rules will assign the same value to the variable.
- Message Passing Systems:
  - Three component algorithm:
    1. A self-stabilizing version of Chandy-Lamport's global snapshot algorithm.
    2. A self-stabilizing reset algorithm that is superposed on it.
    3. A non-self-stabilizing program on which the former two are imposed to obtain self-stabilizing program.

Self-Stabilizing Compilers (cont.)

- Distinguished initiator repeatedly takes global snapshots.
- After taking a snapshot, initiator evaluates a predicate (assumed decidable), on the collected state.
- If an illegitimate global state is detected, reset algorithm is initiated.

Fault Tolerance

- The following transient faults can be handled by a self-stabilizing system:
  - Inconsistent initialization: Different processes initialized to local states that are inconsistent with one another.
  - Mode of change: There can be different modes of execution of a system. In changing the mode of operation, it is impossible for all processes to effect the change in same time.
  - Transmission errors: Loss, corruption, or reordering of messages.
  - Memory crash

Factors Preventing Self-Stabilization

- Symmetry: Processes should not be identical/symmetric because solution generally relies on a distinguished process.
- Termination: If any unsafe global state is a final state, system will not be able to stabilize. Exception case of finite state sequential programs.
- Isolation: Inadequate communication among processes can lead to local states consistent with some safe global state, however, the resulting global state is not safe!

Factors Preventing Self-Stabilization (cont.)

- Look-alike configurations: Such configurations result when the same computation is enabled in two different states with no way to differentiate between them. Then system cannot guarantee convergence from the unsafe state.

Limitations of Self-Stabilizing

- Need for an exceptional machine
- Convergence-response tradeoffs
  - Convergence span denotes the maximum number of critical transitions made before the system reaches a legal state.
  - Response span denotes the maximum number of transitions to get from the starting state to some goal state.
  - Critical transitions. Ex.: A process moves into a critical section, while another is already in!

Limitations of Self-Stabilizing (cont.)

- Pseudo-stabilization: Weaker, but less expensive w.r.t self-stabilization. Every computation only needs to have some state such that the suffix of the computation beginning at this state is in the set of legal computations.
- Verification of self-stabilizing system
  - Verification may be difficult.
  - Stair method developed; Proving the algorithm stabilizes in each step verifies correctness of the entire algorithm, where interleaving assumptions are relaxed.