Introduction to Self-Stabilization

Mikhail Nesterenko
Kent State University

The Definition of Self-Stabilization

Distributed system is *self-stabilizing* wrt the set of *legitimate states* if, regardless of the initial state, it is guaranteed to eventually arrive at a legitimate state and never leave the set of legitimate states after that.

SS distributed system:

- does not need to be initialized
- recovers from transient failures (local state corruption, message loss, etc.)
- adapts to changes in system's topology (if topology is considered part of the state)

On the other hand:

- SS system does not guarantee correct execution during recovery

Guarded Command Language (GCL)

```
*[
  guard1 ♦ command1
  []guard2 ♦ command2
  ⋮
]
```

- *[…] - execution repeats forever
- $guard_i$ - binary predicate on local vars, received messages, etc.;
- $command_i$ - list of assignment statements;

command is executed when corresponding *guard* is true;

guards are selected nondetermenistically,

Advantages:

- GCL allows to easily reason about algorithms and their executions: the program counter position is irrelevant or less important;
- we don't have to consider execution starting in the middle of guard or command (*serializability property*);

Dijkstra's K-State Token Circulation Algorithm

Objective: circulate a single token among processors

Processor ρ_0
*[
$s_0 = s_k$ ♦ $s_0 := (s_0 + 1) \bmod K$
]

Processor ρ_i $(0 < i \leq K)$
*[
$s_i \ ? \ s_{i-1}$ ♦ $s_i := s_{i-1}$
]

- the system consists of a ring of *K* processors (*id*s 0 through *K*-1)
- each processor maintains a state variable *s*; a processor can see the state of it's left (smaller id) neighbor
- guard evaluates to **true** - processor has a privilege (token)
- all processors evaluate their guards, only **one at a time** changes state (C-Daemon)
- after the state change all processors re-evaluate the guards

Alternating Bit Protocol

```
processor p
*[
   receive ack(i) ♦
      if i = ns then
         ns := ↓ns
         ms := get()
         send data(ms, ns)
   []timeout ♦
      send data(ms, ns)
]
```

```
processor q
*[
   receive data(m, i) ♦
      put(m)
      send ack(i)
]
```

The objective is to transmit data reliably from sender node to receiver node unreliable channel

processor *p* - sender
processor *q* - receiver
two types of messages: *data* and *ack*
ns - boolean sequence number (sn) of *data* last sent
ms - last message sent
get() - returns the next message to be sent
put() - delivers received message
timeout - enabled when both channels empty

problems:
•does not work if messages are present in the channel initially
•hard to estimate the length of the timeout

SS Bounded Alternating Bit Protocol

```
processor p
*[
   receive ack(i) ♦
      if i = ns then
         ns := ns + 1
         ms := get()
         send data(ms, ns)
   []timeout ♦
      send data(ms, ns)
]
```

```
processor q
*[
   receive data(m, i) ♦
      if i ? nr then
         put(m)
         nr := i
      send ack(i)
]
```

The objective is to transmit data reliably from sender node to receiver node over bounded (bound=g) unreliable channel

processor *p* - sender
processor *q* - receiver
two types of messages: *data* and *ack*
ns - sequence number(sn) of *data* last sent
nr - sn of the last correct *data*
ms - last message sent
get() - returns the next message to be sent
put() - delivers received message
timeout - always enabled

all variables are bounded,
bound B is set to be greater than $2g$.
addition is done modulo B