03_05_Termination Detection

1. Definition

1.1. General Definition

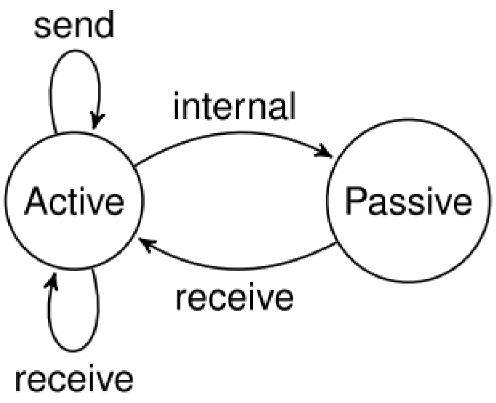
In a transition system a configuration is a **deadlocked or terminating** if there is **no outgoing transition** from that configuration

1. **terminating configuration** is reached when the algorithm **successfully completes** its task

1.2. LTS perspective

A node in isolation, can remain in one of the two states:

1. A process is active, if it has something to execute.
2. Otherwise, it is passive. A passive process will not necessarily always remain passive.



LTS_perspective

1.2.1. Condition of Termination

1. The system is deadlocked
2. the global state of the system satisfies the desired postcondition

2. Termination Detection

2.1. Why Termination Detection

- Many distributed algorithms run in **phases.**
- In order to run phase i + 1, the initiator has to ascertain that phase i has terminated for every process.
- Devise a DA that enables one (or all) of the processes to **determine that the computation has finished**

2.2. Preliminary

1. Termination detection is always **performed for a basic algorithm**, Termination detection **should not influence basic computations** (e.g., freezing).
2. A **control algorithm** consists of **termination detection** and **announcement**, we only focus on detection

2.3. Assumption

- Each of the processes is either **active** or **passive**
- Only **active process can send** message related to the computation
- a p**assive process can only become activ**e because of the reception of a message of the computation (containing some intermediate results)

2.4. Termination detection in an asynchronous unidirectional ring with FIFO communication

Assumption

- a **unidirectional** ring with **FIFO** links
- Process P_0 can only send to process P_{n-1}, and process P_i can only send to process P_{i-1},

Process

- introduce a color for the token and the processes
    - when the token starts, it is **white**

- when a process **spoils** the passiveness “behind the token’s back,” it turns **black**
- when the token **arrives at a black process**, it turns **black**, and the process **turns white** again to prepare for the next attempt
- when the token returns P_0:
  - **white:** conclude termination
  - **black**: try again

Understanding

A black token can turn into white when:

- reinitiate a round

So if a black process exist, there will definitely be next round

```
I. Spontaneous state change
when (state = active) do
   state ← passive

II. Receiving a message of the computation
upon receipt of (message) do
   state ← active

III. Sending a message to a higher-numbered process
send(message) to P_j
if (j > i) then
   color_p ← black

IV. (Re-)initiating the token in P_0
when (token_present) do
   token_present ← false
   send(token;white)
   color_p ← white

V. Receiving the token in P_i, i = 1, 2, ..., n − 1
upon receipt of (token;color_t) do
   token_present ← true
   if (color_t = black) or (color_p = black) then
      token_present ← false
      send(token;black)
      color_p ← white
   else if (state = passive) then
      token_present ← false
      send(token;white)

VI. Sending the token in P_i, i = 1, 2, ..., n − 1
when (token_present and ((state = passive) or (color_p = black)) do
   token_present ← false
   send(token;color_p)
   color_p ← white

VII. Receiving the token in P_0
upon receipt of (token;color_t) do
   token_present ← true
   if ((color_t = white) and (color_p = white) and (state = passive)) then
      decide termination
```

> Termination Detection Implementation.png

Correctness: (See slides)

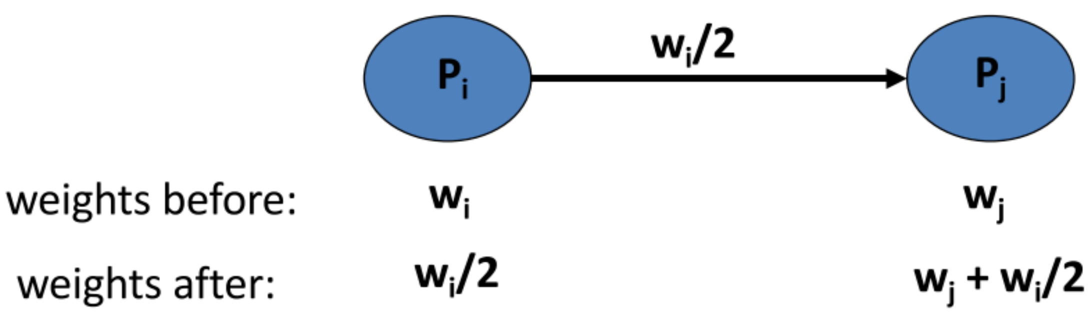
2.5. General Termination-detection

Preparation

- There is one **special process P** that does not participate in the application itself
- All processes and all messages have **non-negative weights**
- Initial weights of the processes:
  - process P: 1
  - all other n processes: 0
- Initialization:
  - P sets its own weight to **0**,
  - and gives each of the processes weight **1/n**

Process

- When a process **sends** a message:
  - it cuts its own weight **in half**
  - it **gives** the message a weight equal to the other half
- When a process **receives** a message:
  - it **adds** the weight of the message to its own weight



- When process P_i with weight w_i becomes **idle**:
  - it sends a message with weight w_i to P
  - and sets its own weight to 0
- When the weight of P **return to 1**, there is a **termination**