Termination detection

Think about this

An initiator sends query to each process: has your local computation terminated? Assume that everyone said YES.
Will this signal global termination?

Not necessarily true!

Each process periodically becomes active or passive (local termination). A passive process may wake up and become active by receiving a message.

Conditions for global termination

1. All processes are passive
2. No message is in transit
A computation subgraph is a network of active processes.

A computation subgraph with 5 processes 1-5. Process 0 is the *initiator*. The subgraph grows and shrinks, and finally becomes empty. We need to introduce a signaling mechanism to detect when the subgraph completely shrinks.
The signaling mechanism

(Keeping the flavor of the original paper)

The computation graph is a directed graph. An edge from j to k means node j engages node k.

There is one initiator with in-degree = 0

(Game plan) Initiator initiates the detection by sending out a signal over each outgoing edge. When it receives an ack via each outgoing edge, it detects termination.

The signaling mechanism determines when and to whom a node will send signals and acks.
Over any edge, *deficit* = # signals - # ack. Let

\[ C = \text{deficit over incoming edges} \]
\[ D = \text{deficit over outgoing edges} \]

Initially for all node \( C=0 \) and \( D=0 \). The initiator changes it to \( C = 0 \) and \( D = k \) (out-degree)

**Invariant 1.** \( (C \geq 0) \land (D \geq 0) \)

Dijkstra and Scholten *proposed* that every non-initiator satisfies the following invariant:

**Invariant 2.** \( (C > 0) \implies (D = 0) \)

*Sending signals and ack must preserve both invariants.*

Sending an ack decrements \( C \). So,

\[
(C-1 \geq 0) \land (C-1 > 0) \land (D = 0) = (C \geq 1) \land (C > 1) \land (D = 0) = (C > 1) \land (C = 1 \land D = 0)
\]
**Termination detection program**

```
program detect {for an internal node}
define C, D : integer
    m : (signal, ack)
    state : (active, passive)
initially C=0, D=0, parent = i

do m = signal (C=0) C:=1;
    state:= active*;
    parent := sender
    m = ack D:= D-1
    (C=1D=0) state = passive send ack to parent%
    C:= 0; parent := i
    m = signal (C=1) send ack to the sender;
od

* This node is now engaged by its parent. It can stay active for some time, or turn passive, which depends on the computation

% This node now turns passive.
```
Observations

1. The algorithm only detects termination, and does not detect non-termination.

2. The edges with positive deficits form a spanning tree. The spanning tree can be different in different runs.

3. The total number of messages required for termination detection is twice the number of messages exchanged by the underlying algorithm.
A token passing algorithm for a unidirectional ring

(Dijkstra, Feijen, Van Gasteren)

(Main idea) Initiator sends a white token. If all processes are passive then they pass the token, and the white token returns to the initiator. An active process does not accept the white token until it turns white.

What are the problems? How do you fix them?
Revised rules

Assumption 1. When a process sends a message, it turns black, and
Assumption 2. When a black process sends a token, the token turns black.
Assumption 3. When a black process sends a token to its successor, it turns white.

program term {for process i > 0}
define color : (black, white)
            state : (active, passive)
do   token ⊸ (state = passive) ⊸
       send token;
       if color = black ⊸ color = white;
          token turns black
       fi
       i sends a message ⊸ color(i) := black
od
\{for process 0\}

send a white token;

\textbf{do} token = black \[  
  \quad \text{send a white token}  
\]

\textbf{od};

termination detected;