Distributed Algorithms

LECTURER: V.A. ZAKHAROV

Lecture 10.

Termination detection problem

The Dijkstra–Scholten Algorithm

The Shavit–Francez Algorithm

The Safra's Algorithm

The Credit-Recovery Algorithm

The Rana's Algorithm

Termination detection problem

A computation of a distributed algorithm terminates when the algorithm reaches a **terminal configuration**, i.e. a configuration in which no further steps of the algorithm are applicable.

Termination detection problem

A computation of a distributed algorithm terminates when the algorithm reaches a **terminal configuration**, i.e. a configuration in which no further steps of the algorithm are applicable.

A computation terminates explicitly if all processes of a system are in their terminal states . Then we say that the computation ends with processes termination .

Termination detection problem

A computation of a distributed algorithm terminates when the algorithm reaches a **terminal configuration**, i.e. a configuration in which no further steps of the algorithm are applicable.

A computation terminates explicitly if all processes of a system are in their terminal states . Then we say that the computation ends with processes termination .

A computation terminates implicitly if all channels are empty but some processes are not in their terminal states, i.e. these processes are in states that allow the receipt of messages. Then we say that the computation ends with message exchange termination .

Termination detection problem

A computation of a distributed algorithm terminates when the algorithm reaches a **terminal configuration**, i.e. a configuration in which no further steps of the algorithm are applicable.

A computation terminates explicitly if all processes of a system are in their terminal states . Then we say that the computation ends with processes termination .

A computation terminates implicitly if all channels are empty but some processes are not in their terminal states, i.e. these processes are in states that allow the receipt of messages. Then we say that the computation ends with message exchange termination .

Very often one needs to be able to convert algorithms in which some computations end with message exchange termination into algorithms in which all computations end with processes termination.

Termination detection problem

To this end we consider a uniform approach which supply an arbitrary distributed algorithm with two additional algorithms that interact with the given message-terminating algorithm and with each other.

One of these algorithms observes the computation and will detect somehow that the computation has reached a terminal configuration of the underlying algorithm. It then calls the second algorithm, which floods a termination message to all processes, causing them to enter a terminal state.

The most difficult part of the transformation turns out to be the algorithm that detects the termination. The flooding procedure is rather trivial.

It will be shown that termination detection is possible for all classes of networks for which a wave algorithm can be given.

Termination detection problem

The set of states Z_p of a process p is divided into two subfamilies of active and passive states.

A state c_p of a process p is active if an internal or send event of p can be performed in c_p ; otherwise this state is passive.

In a passive state c_p either only message receipt is possible, or no actions at all are applicable; in the latter case c_p is a terminal state of a process p .

We say that a process p is active if it is in one of its active states; otherwise p is called passive.

A message can be sent only by an active process, and a passive process can become active only when a message is received. An active process can become passive when it enters a passive state.

Termination detection problem

Some assumptions are made in order to simplify the description of the algorithms.

1. **An active process becomes passive only in an internal event.**

Termination detection problem

Some assumptions are made in order to simplify the description of the algorithms.

1. **An active process becomes passive only in an internal event.**
2. **A process always becomes active when a message is received.**

Termination detection problem

Some assumptions are made in order to simplify the description of the algorithms.

1. **An active process becomes passive only in an internal event.**
2. **A process always becomes active when a message is received.**
3. **The internal events in which p becomes passive are the only internal events of p .**

   Internal events in which p moves from one active state to another active state are ignored, because the termination-detection algorithm must be oblivious , i.e. it must not distinguish between the details of the computations.

Termination detection problem

Basic algorithm.

var $state_p$: ($active$, $passive$) ;

S$_p$: { $state_p = active$ }
begin send ⟨**mes**⟩ **end**

R$_p$: { Message ⟨**mes**⟩ is delivered to p }
begin receive ⟨**mes**⟩ ; $state_p := active$ **end**

I$_p$: { $state_p = active$ }
begin $state_p := passive$ **end**

This is what every process for a termination detection algorithm looks like.

Termination detection problem

The predicate **term**(γ) is defined to be true in every configuration γ , where no event of the basic computation is applicable.

Theorem 1.

$$\textbf{term}(\gamma) \iff (\forall p \in \mathbb{P} : state_p = passive)$$
$$\wedge \quad (\forall pq \in E : M_{pq} \text{ does not contain a message } \langle \textbf{mes} \rangle).$$

Termination detection problem

The predicate **term**(γ) is defined to be true in every configuration γ , where no event of the basic computation is applicable.

Theorem 1.

$$\textbf{term}(\gamma) \iff (\forall p \in \mathbb{P} : state_p = passive)$$
$$\wedge \quad (\forall pq \in E : M_{pq} \text{ does not contain a message } \langle \textbf{mes} \rangle).$$

Proof. If all processes are passive, no internal or send event is applicable. If, moreover, no channel contains a message, no receive event is applicable, hence, no basic event is applicable at all.

If some process is active, a send or internal event is possible in that process, and if some channel contains a message the receipt of this message is applicable. □

Termination detection problem

The problem under consideration: add a control algorithm to the system which brings the processes into a terminal state after the basic computation has reached a terminal configuration.

The control algorithm exchanges (control) messages, and these can be sent by passive processes and do not make a passive process active when they are received.

The control algorithm consists of termination detection sub-algorithm and termination-announcement procedure which brings all processes to the terminal states.

Termination detection problem

The detection algorithm must satisfy the following three requirements.

Termination detection problem

The detection algorithm must satisfy the following three requirements.

1. **Non-interference.** It must not influence the computation of the basic algorithm.

Termination detection problem

The detection algorithm must satisfy the following three requirements.

1. **Non-interference.** It must not influence the computation of the basic algorithm.
2. **Liveness.** If **term** holds then termination-announcement procedure *Announce* must be called within a finite number of steps.

Termination detection problem

The detection algorithm must satisfy the following three requirements.

1. **Non-interference.** It must not influence the computation of the basic algorithm.
2. **Liveness.** If **term** holds then termination-announcement procedure *Announce* must be called within a finite number of steps.
3. **Safety.** If a procedure *Announce* is called then the configuration must satisfy **term**.

Termination detection problem

var $SentStop_p$: **bool** **init** *false* ;
$RecStop_p$: **integer** **init** 0 ;

Procedure *Announce*:
begin if not $SentStop_p$ **then**
begin $SentStop_p := true$;
forall $q \in Out_p$ **do** send $\langle$**stop**$\rangle$ to q
end
end

{Message $\langle$**stop**$\rangle$ was delivered to p}
begin receive $\langle$**stop**$\rangle$; $RecStop_p := RecStop_p + 1$;
Announce ;
if $RecStop_p = \#In_p$ **then halt**
end

Termination detection problem

Theorem 2.
For every termination-detection algorithm there exists a basic computation that exchanges M basic messages and for which the detection algorithm exchanges at least M control messages.

This theorem follows from the non-interference requirement: a detection algorithm must not look «inside» computations it supervises.

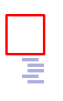
Termination detection problem

Theorem 3.

Detecting the termination of a decentralized basic computation requires the exchange of at least W control messages in the worst case, where W is a message complexity of optimal wave algorithm.

Proof.

Consider the basic computation that does not exchange any message and where each active process becomes passive as its first event. Then the detection algorithm to be a wave algorithm if the detection (the call to *Announce*) is regarded as the decision.
Indeed, a call to *Announce* must occur within a finite number of steps; hence, the detection algorithm itself terminates and decides.
If the decision is not preceded by an event in some process q , a different basic computation is considered where q remains active. The decision does not depend causally on any event in q , so the detection algorithm may erroneously call *Announce*.
As the detection algorithm is a wave, it exchanges at least W messages.



Dijkstra–Scholten Algorithm

Dijkstra–Scholten Algorithm is applicable when

- basic algorithm is centralized;
- network topology is arbitrary;
- channels are bi-directed;
- control algorithm is centralized and it is initiated in the same process as the basic algorithm.

Dijkstra–Scholten Algorithm

Termination detecting algorithm builds and maintains computation tree $T = (V_T, E_T)$. This is a virtual tree which has the following properties.

1. Either T is empty, or it is a directed tree with root p_0 .
2. The set V_T includes all active processes and all basic messages in transit.

The initiator p_0 calls the procedure *Announce*, if $p_0 \notin V_T$; by the first property T in this case is empty, and by the second property **term** holds.

Dijkstra–Scholten Algorithm

When a process p sends a basic message $\langle \mathbf{mes} \rangle$, it is inserted (virtually) into the tree and the father of $\langle \mathbf{mes} \rangle$ is p .

When a process p , not in the tree, becomes active by the receipt of a message from q , the node q becomes the father of p .

To represent the sender of a message explicitly, a basic message $\langle \mathbf{mes} \rangle$ sent by q will be denoted as $\mathbf{cmes}_q$.

Dijkstra–Scholten Algorithm

The removal of nodes from T is also necessary, for two reasons.

1) A basic message is deleted when it is received.

2) To ensure progress of the detection algorithm the tree must collapse within a finite number of steps after termination. Messages are leaves of the tree T ; processes maintain a variable that counts the number of their sons in T . The deletion of a son of process p occurs in a different process q when

a) either this son is a basic message delivered to q ,

b) or this son is the process q itself.

To prevent corruption of p 's son account, a signal message $\mathbf{sig}_p$ is sent to p when its son is deleted. This message replaces the deleted son of p ; its deletion occurs in p , and p decrements its son count when it receives a signal.

Dijkstra–Scholten Algorithm

var $state_p$: $(active, passive)$ **init if** $p = p_0$ **then** $active$ **else** $passive$;
sc_p : **integer** **init** 0 ;
$father_p$: $\mathbb{P}$ **init if** $p = p_0$ **then** p **else** $udef$;

S$_p$: { $state_p = active$ }
begin send $\langle$**mes**, $p\rangle$; $sc_p := sc_p + 1$ **end**

R$_p$: {A message $\langle$**mes**, $q\rangle$ has arrived at p }
begin receive $\langle$**mes**, $q\rangle$; $state_p := active$;
if $father_p = udef$ **then** $father_p := q$ **else** send $\langle$**sig**, $q\rangle$ to q
end

I$_p$: { $state_p = active$ }
begin $state_p := passive$;
if $sc_p = 0$ **then**(* Delete p from T *)
begin if $father_p = p$
then *Announce*
else send $\langle$**sig**, $father_p\rangle$ to $father_p$;
$father_p := udef$
end
end

Dijkstra–Scholten Algorithm

$\mathbf{A}_p$: { Signal $\langle \mathbf{sig}, p \rangle$ arrives at p }
begin receive $\langle \mathbf{sig}, p \rangle$; $sc_p := sc_p - 1$;
if $sc_p = 0$ **and** $state_p = passive$ **then**
begin if $father_p = p$
then *Announce*
else send $\langle \mathbf{sig}, father_p \rangle$ to $father_p$;
$father_p := udef$
end
end

Dijkstra–Scholten Algorithm

The correctness proof rigorously establishes that the graph T , as defined, is a tree and that it becomes empty only after termination of the basic computation.

For every configuration γ define two sets of nodes

$$\begin{aligned} V_T &= \{p : father_p \neq udef\} \cup \{\langle \mathbf{mes}, p\rangle \text{ in transit}\} \\ &\cup \{\langle \mathbf{sig}, p\rangle \text{ in transit}\} \end{aligned}$$

and

$$\begin{aligned} E_T &= \{(p, father_p) : father_p \neq udef \wedge father_p \neq p\} \\ &\cup \{(\langle \mathbf{mes}, p\rangle, p) : \langle \mathbf{mes}, p\rangle \text{ in transit}\} \\ &\cup \{(\langle \mathbf{sig}, p\rangle, p) : \langle \mathbf{sig}, p\rangle \text{ in transit}\}. \end{aligned}$$

Dijkstra–Scholten Algorithm

The safety of the algorithm will follow from the assertion P :

$$
\begin{aligned}
P \equiv \quad & state_p = active \implies p \in V_T \\
\wedge \quad & (u, v) \in E_T \implies u \in V_T \wedge v \in V_T \cap \mathbb{P} \\
\wedge \quad & sc_p = \#\{v : (v, p) \in E_T\} \\
\wedge \quad & V_T \neq \varnothing \implies T \text{ is a tree with root } p_0 \\
\wedge \quad & (state_p = passive \wedge sc_p = 0) \implies p \notin V_T.
\end{aligned}
$$

Dijkstra–Scholten Algorithm

$$
\begin{aligned}
P \equiv \quad & state_p = active \implies p \in V_T && (1)\\
\wedge \quad & (u, v) \in E_T \implies u \in V_T \wedge v \in V_T \cap \mathbb{P} && (2)\\
\wedge \quad & sc_p = \#\{v : (v, p) \in E_T\} && (3)\\
\wedge \quad & V_T \neq \varnothing \implies T \text{ is a tree with root } p_0 && (4)\\
\wedge \quad & (state_p = passive \wedge sc_p = 0) \implies p \notin V_T. && (5)
\end{aligned}
$$

By definition, the node set of T includes all messages (basic as well as control messages), and by (1) it also includes all active processes.

Dijkstra–Scholten Algorithm

$$
\begin{aligned}
P \equiv \quad & state_p = active \implies p \in V_T && (1)\\
\wedge \quad & (u, v) \in E_T \implies u \in V_T \wedge v \in V_T \cap \mathbb{P} && (2)\\
\wedge \quad & sc_p = \#\{v : (v, p) \in E_T\} && (3)\\
\wedge \quad & V_T \neq \varnothing \implies T \text{ is a tree with root } p_0 && (4)\\
\wedge \quad & (state_p = passive \wedge sc_p = 0) \implies p \notin V_T. && (5)
\end{aligned}
$$

Clause (2) is rather technical; it states that T is indeed a graph and all edges are directed towards processes.

Dijkstra–Scholten Algorithm

$$
\begin{aligned}
P \equiv \quad & state_p = active \implies p \in V_T && (1)\\
\wedge \quad & (u, v) \in E_T \implies u \in V_T \wedge v \in V_T \cap \mathbb{P} && (2)\\
\wedge \quad & sc_p = \#\{v : (v, p) \in E_T\} && (3)\\
\wedge \quad & V_T \neq \varnothing \implies T \text{ is a tree with root } p_0 && (4)\\
\wedge \quad & (state_p = passive \wedge sc_p = 0) \implies p \notin V_T. && (5)
\end{aligned}
$$

Clause (3) expresses the correctness of the son count of each process.

Dijkstra–Scholten Algorithm

$$
\begin{aligned}
P \equiv \quad & state_p = active \implies p \in V_T && (1)\\
\wedge \quad & (u, v) \in E_T \implies u \in V_T \wedge v \in V_T \cap \mathbb{P} && (2)\\
\wedge \quad & sc_p = \#\{v : (v, p) \in E_T\} && (3)\\
\wedge \quad & V_T \neq \varnothing \implies T \text{ is a tree with root } p_0 && (4)\\
\wedge \quad & (state_p = passive \wedge sc_p = 0) \implies p \notin V_T. && (5)
\end{aligned}
$$

The clause (4) states that T is a tree and p_0 is the root.

Dijkstra–Scholten Algorithm

$$
\begin{aligned}
P \equiv \quad & state_p = active \implies p \in V_T && (1)\\
\wedge \quad & (u, v) \in E_T \implies u \in V_T \wedge v \in V_T \cap \mathbb{P} && (2)\\
\wedge \quad & sc_p = \#\{v : (v, p) \in E_T\} && (3)\\
\wedge \quad & V_T \neq \varnothing \implies T \text{ is a tree with root } p_0 && (4)\\
\wedge \quad & (state_p = passive \wedge sc_p = 0) \implies p \notin V_T. && (5)
\end{aligned}
$$

Clause (5) is used to show that the tree indeed collapses if the basic computation terminates.

Dijkstra–Scholten Algorithm

Lemma 1.
The assertion P is an invariant of Dijkstra–Scholten Algorithm.

Proof.
This is Your HOMETASK 1.

Dijkstra–Scholten Algorithm

Theorem 4.
The Dijkstra–Scholten algorithm is a correct termination-detection algorithm which uses M control message exchanges for any basic computation with M basic message exchange.

Proof.
Follows from Lemma 1.

Dijkstra–Scholten Algorithm

The Dijkstra–Scholten algorithm achieves an attractive balance between the control communication and the basic communication; for each basic message sent from p to q the algorithm sends exactly one control message from q to p .

The control communication equals the lower bound given in Theorem 2, so the algorithm is a worst-case optimal algorithm for termination detection of centralized computations.

Shavit–Francez Algorithm

The Dijkstra–Scholten algorithm was generalized to decentralized basic computations by Shavit and Francez.

In their algorithm, the computation graph is a computation forest of which each tree is rooted at an initiator of the basic computation.

The tree rooted at p is denoted T_p .

Shavit–Francez Algorithm

The Shavit–Francez Algorithm builds and maintains such a graph $F = (V_F, E_F)$, that

(1) either F is empty, or F is a forest of which each tree is rooted in an initiator;

(2) the set V_F includes all active processes and all basic messages.

Termination is detected when the graph becomes empty.

But in the case of a forest it is not easy to see whether the graph is empty.
When the computation tree is rooted at p_0 the emptiness of the tree is observed by p_0 , which calls **Announce** when the tree is empty.
In the case of a forest, each initiator only observes the emptiness of its own tree, but this does not imply the emptiness of the forest.

Shavit–Francez Algorithm

The verification that all trees have collapsed is done by a single wave.

A forest F is built so that if any tree T_p has become empty, it remains empty thereafter. This does not prevent p from becoming active; but if p becomes active after the collapse of its tree, it is inserted in the tree of another initiator.

Each process participates in the wave only if its tree has collapsed; when the wave decides, *Announce* is called.

Shavit–Francez Algorithm

Every process p has a variable $father_p$.

Its value is udef, if $p \notin V_T$.

If p is a root of a tree, then the value of $father_p$ is the ID of p .

And, finally, if p is a non-root node of a tree T , then the value of $father_p$ is the ID of the father of p .

Variable sc_p gives the number of sons of p in T .

The boolean variable $empty_p$ is true iff p 's tree is empty.

Shavit–Francez Algorithm

var $state_p$: ($active$, $passive$) **init if** p is initiator **then** $active$ **else** $passiv$
sc_p : **integer** **init** 0 ;
$father_p$: $\mathbb{P}$ **init if** p is initiator **then** p **else** $udef$;
$empty_p$: **bool** **init if** p is initiator **then** $false$ **else** $true$;
S$_p$: { $state_p = active$ }
begin send $\langle$**mes**, $p\rangle$; $sc_p := sc_p + 1$ **end**
R$_p$: {Message $\langle$**mes**, $q\rangle$ has been delivered to p }
begin receive $\langle$**mes**, $q\rangle$; $state_p := active$;
if $father_p = udef$ **then** $father_p := q$ **else** send $\langle$**sig**, $q\rangle$ to q
end
I$_p$: { $state_p = active$ }
begin $state_p := passive$;
if $sc_p = 0$ **then** (* Delete p from F *)
begin
if $father_p = p$ **then** $empty_p := true$ **else** send $\langle$**sig**, $father_p\rangle$
$father_p := udef$
end
end

Shavit–Francez Algorithm

$\mathbf{A}_p$: { Signal $\langle \mathbf{sig}, p \rangle$ arrives at p }
begin receive $\langle \mathbf{sig}, p \rangle$; $sc_p := sc_p - 1$;
if $sc_p = 0$ **and** $state_p = passive$ **then**
begin
if $father_p = p$ **then** $empty_p := true$ **else** send $\langle \mathbf{sig}, father_p \rangle$
$father_p := udef$
end
end

Shavit–Francez Algorithm

Comments.

1. In the above description, the wave algorithm is not explicitly selected.
2. The wave algorithm is run only by the initiators of basic computation.
3. All processes carry out parallel execution of the wave algorithm in such a way that sending messages or making decisions is allowed only to those processes p for which variable $empty_p$ has value true .
4. When the event **decide** occurs the procedure *Announce* is called.

Shavit–Francez Algorithm

For every configuration γ define the following set of nodes:

$$\begin{aligned} V_T &= \{p : father_p \neq udef\} \cup \{\langle \mathbf{mes}, p\rangle \text{ in transit}\} \\ &\cup \{\langle \mathbf{sig}, p\rangle \text{ in transit}\} \end{aligned}$$

и дуг

$$\begin{aligned} E_F &= \{(p, father_p) : father_p \neq udef \ \wedge \ father_p \neq p\} \\ &\cup \{(\langle \mathbf{mes}, p\rangle, p) : \langle \mathbf{mes}, p\rangle \text{ in transit}\} \\ &\cup \{(\langle \mathbf{sig}, p\rangle, p) : \langle \mathbf{sig}, p\rangle \text{ in transit}\}. \end{aligned}$$

Shavit–Francez Algorithm

The safety of the algorithm will follow from the assertion Q :

$$
\begin{aligned}
Q \iff & \quad state_p = active \implies p \in V_F \\
& \wedge \ (u, v) \in E_F \implies u \in V_F \wedge v \in V_F \cap \mathbb{P} \\
& \wedge \ sc_p = \#\{v : (v, p) \in E_F\} \\
& \wedge \ V_F \neq \varnothing \implies F \text{ is a forest} \\
& \wedge \ (state_p = passive \wedge sc_p = 0) \implies p \notin V_F \\
& \wedge \ empty_p \iff T_p \text{ is empty}
\end{aligned}
$$

Shavit–Francez Algorithm

Lemma 2.
The assertion Q is an invariant of the Shavit–Francez Algorithm.

Proof.
This is Your HOMEWORK 2.

Shavit–Francez Algorithm

Theorem 5.
The Shavit–Francez Algorithm is a correct termination detection algorithm which uses $M + W$ control message exchanges (where W is a message exchange complexity of the wave algorithm).

Proof.
Follows from Lemma 2.

Safra's Algorithm

This termination detection algorithm can be applied when

- A basic algorithm is decentralized.

Safra's Algorithm

This termination detection algorithm can be applied when

- A basic algorithm is decentralized.
- Communication channels are one-way.

Safra's Algorithm

This termination detection algorithm can be applied when

- A basic algorithm is decentralized.
- Communication channels are one-way.
- There is a Hamiltonian cycle in the network along which control messages propagate; basic messages can be transmitted on any channel.

Safra's Algorithm

This termination detection algorithm can be applied when

- A basic algorithm is decentralized.
- Communication channels are one-way.
- There is a Hamiltonian cycle in the network along which control messages propagate; basic messages can be transmitted on any channel.
- Control algorithm is a centralized wave algorithm in one-way rings.

Safra's Algorithm

Basic principles.

- Every process may have some color *white* or *black* ; the current color of a process is stored in the variable *color* .

Safra's Algorithm

Basic principles.

- Every process may have some color *white* or *black* ; the current color of a process is stored in the variable *color* .
- Control messages (tokens) may have also some color *white* or *black* .

Safra's Algorithm

Basic principles.

- Every process may have some color *white* or *black* ; the current color of a process is stored in the variable *color* .
- Control messages (tokens) may have also some color *white* or *black* .
- Token is transmitted only by passive processes.

Safra's Algorithm

Basic principles.

- Every process may have some color *white* or *black* ; the current color of a process is stored in the variable *color* .
- Control messages (tokens) may have also some color *white* or *black* .
- Token is transmitted only by passive processes.
- Initiator of the control algorithm issues and transmits a *white* token.

Safra's Algorithm

Basic principles.

- Every process may have some color *white* or *black* ; the current color of a process is stored in the variable *color* .
- Control messages (tokens) may have also some color *white* or *black* .
- Token is transmitted only by passive processes.
- Initiator of the control algorithm issues and transmits a *white* token.
- After receiving a basic message a process becomes *black* .

Safra's Algorithm

Basic principles.

- Every process may have some color *white* or *black* ; the current color of a process is stored in the variable *color* .
- Control messages (tokens) may have also some color *white* or *black* .
- Token is transmitted only by passive processes.
- Initiator of the control algorithm issues and transmits a *white* token.
- After receiving a basic message a process becomes *black* .
- A *black* process colors a token *black* .

Safra's Algorithm

Basic principles.

- Every process may have some color *white* or *black* ; the current color of a process is stored in the variable *color* .
- Control messages (tokens) may have also some color *white* or *black* .
- Token is transmitted only by passive processes.
- Initiator of the control algorithm issues and transmits a *white* token.
- After receiving a basic message a process becomes *black* .
- A *black* process colors a token *black* .
- Each process turns *white* immediately after sending the token.

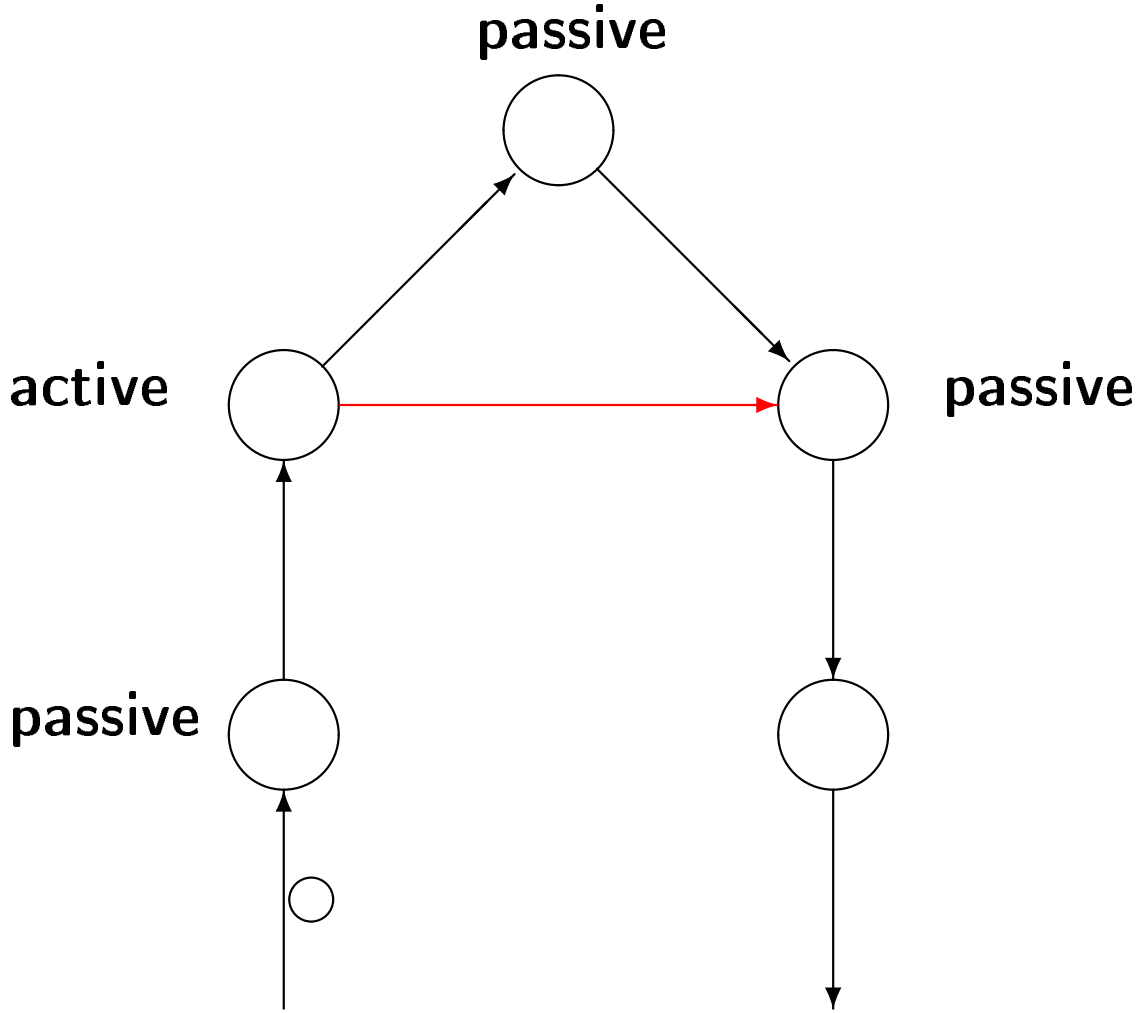
Safra's Algorithm

Basic principles.

- Every process may have some color *white* or *black* ; the current color of a process is stored in the variable *color* .
- Control messages (tokens) may have also some color *white* or *black* .
- Token is transmitted only by passive processes.
- Initiator of the control algorithm issues and transmits a *white* token.
- After receiving a basic message a process becomes *black* .
- A *black* process colors a token *black* .
- Each process turns *white* immediately after sending the token.
- When the initiator receives a *white* token it announce the termination of basic computation.

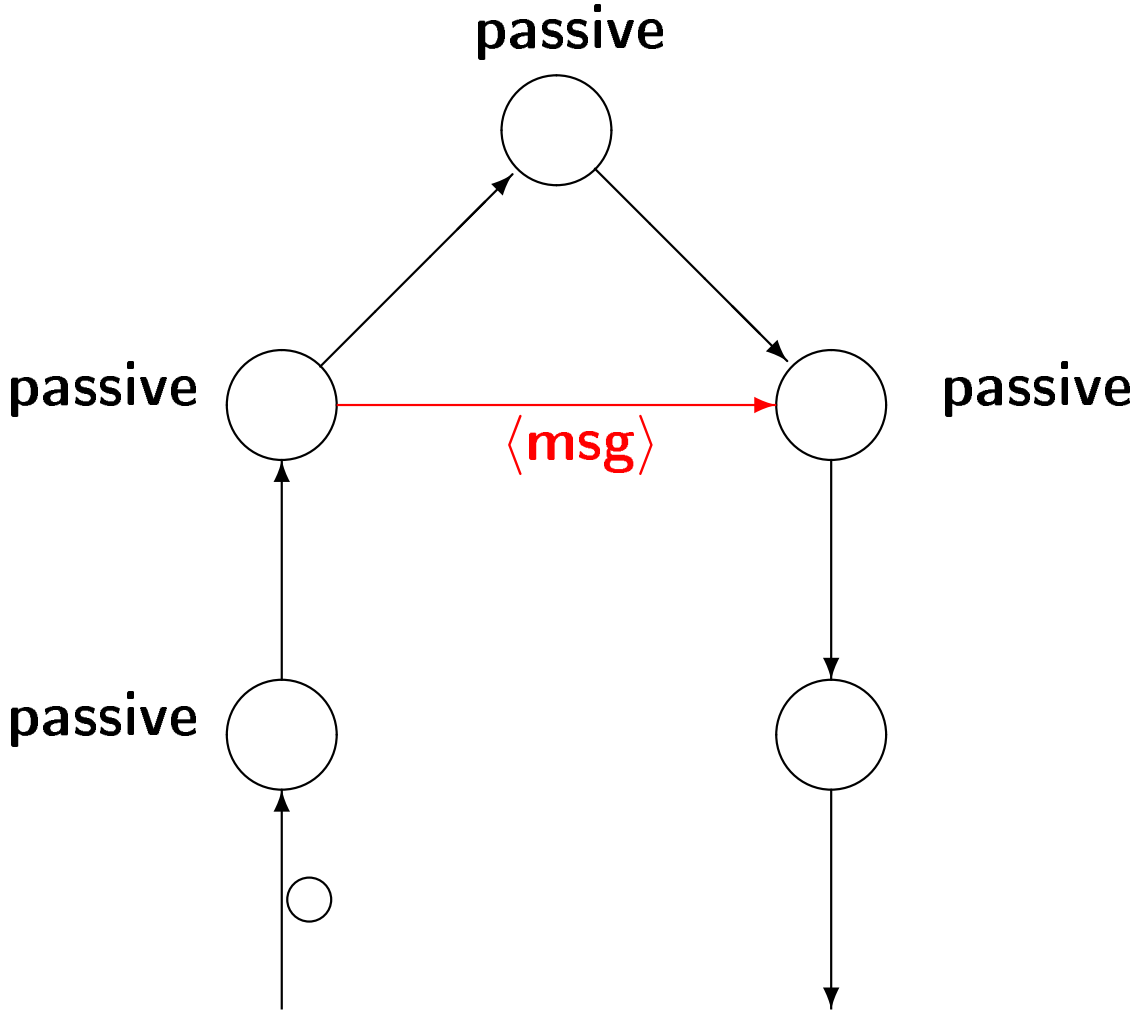
Safra's Algorithm

Alas, these principles are not enough for the correct operation of the algorithm.



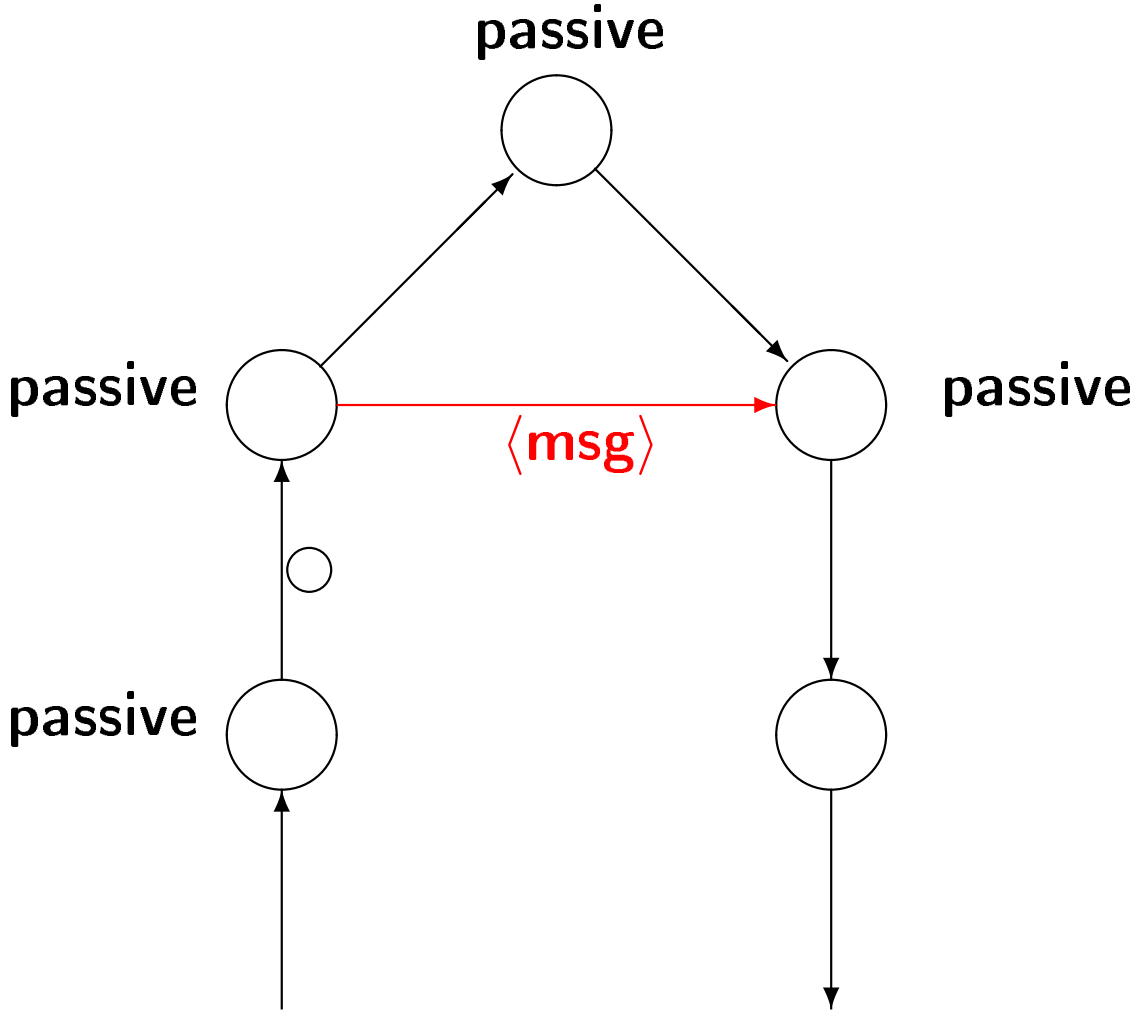
Safra's Algorithm

Alas, these principles are not enough for the correct operation of the algorithm.



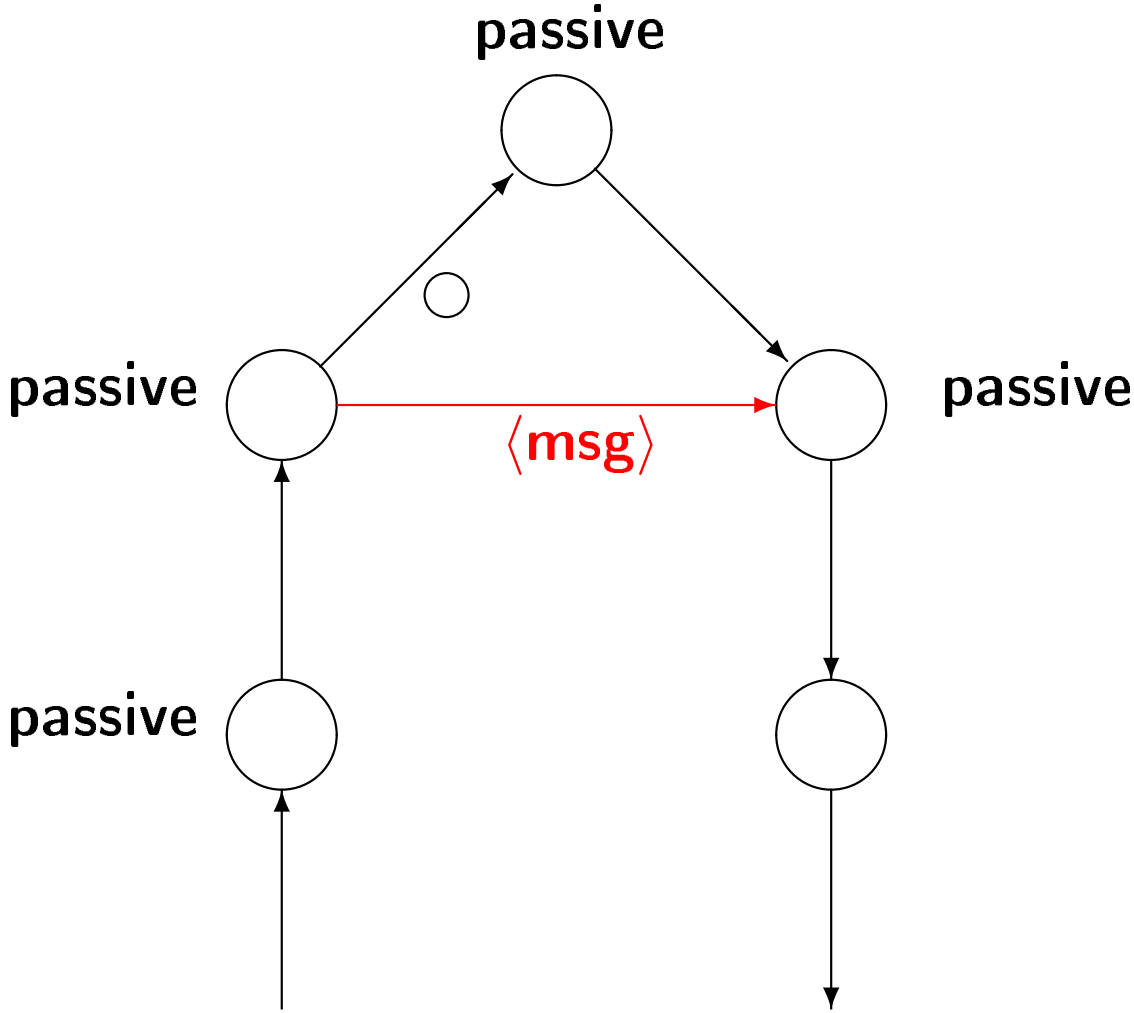
Safra's Algorithm

Alas, these principles are not enough for the correct operation of the algorithm.



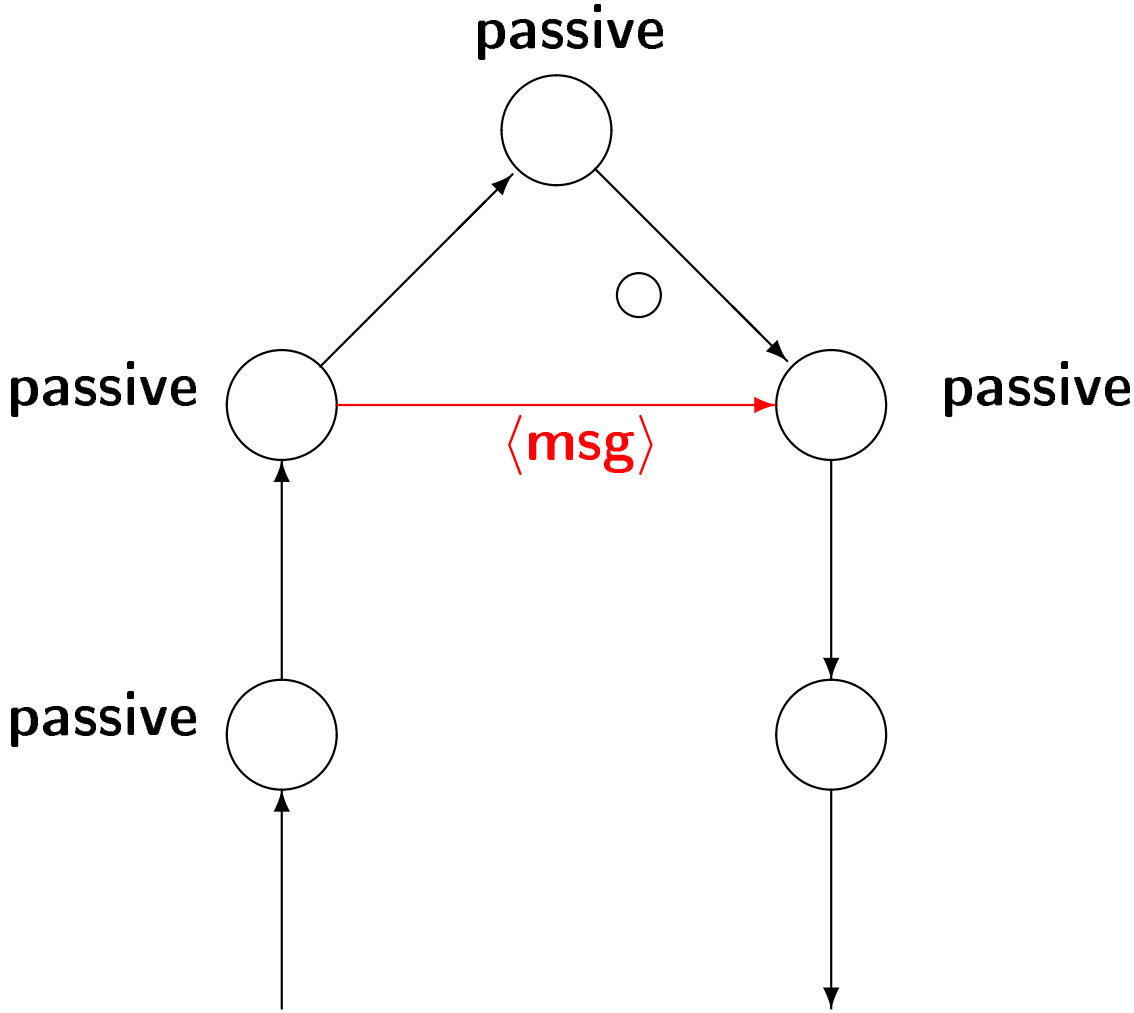
Safra's Algorithm

Alas, these principles are not enough for the correct operation of the algorithm.



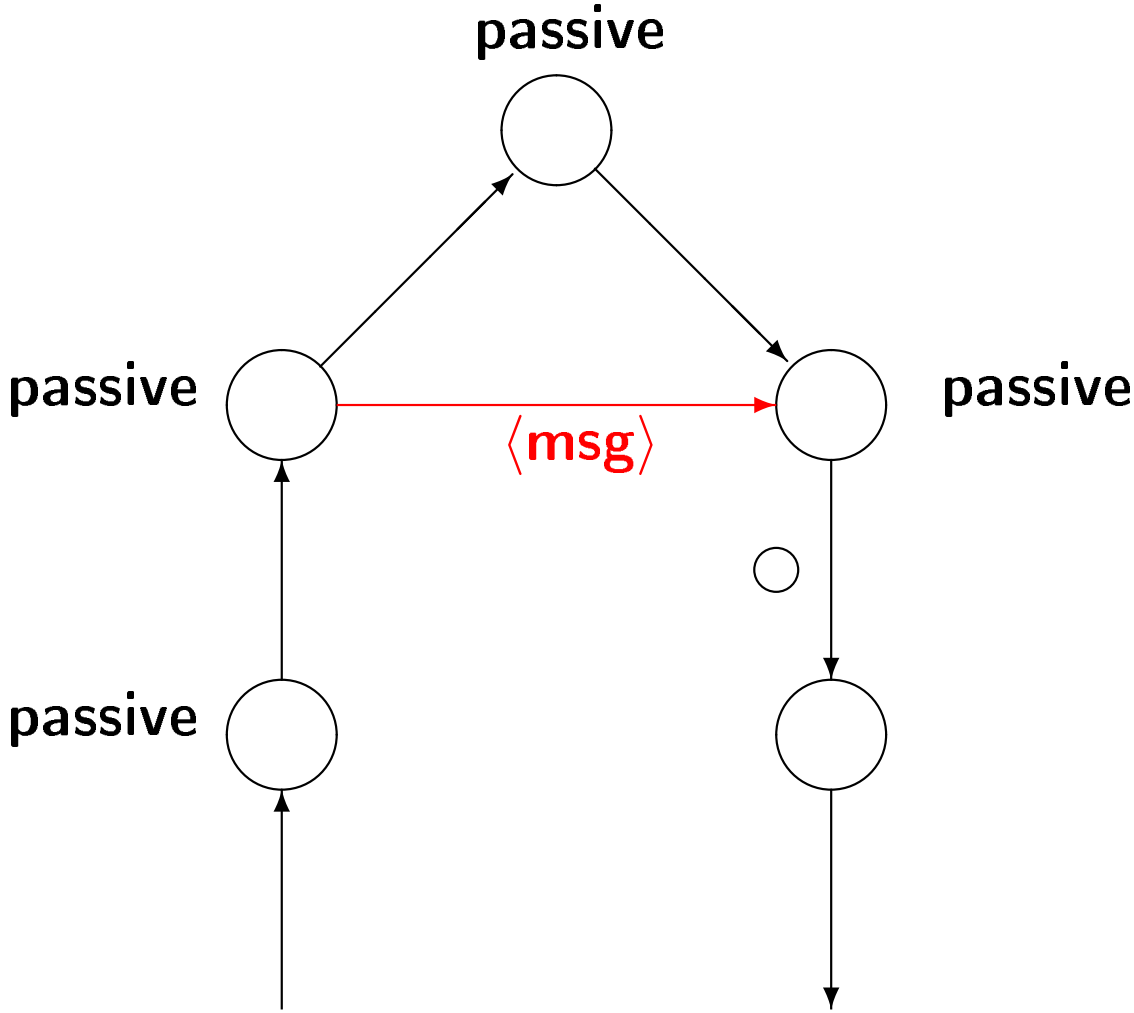
Safra's Algorithm

Alas, these principles are not enough for the correct operation of the algorithm.



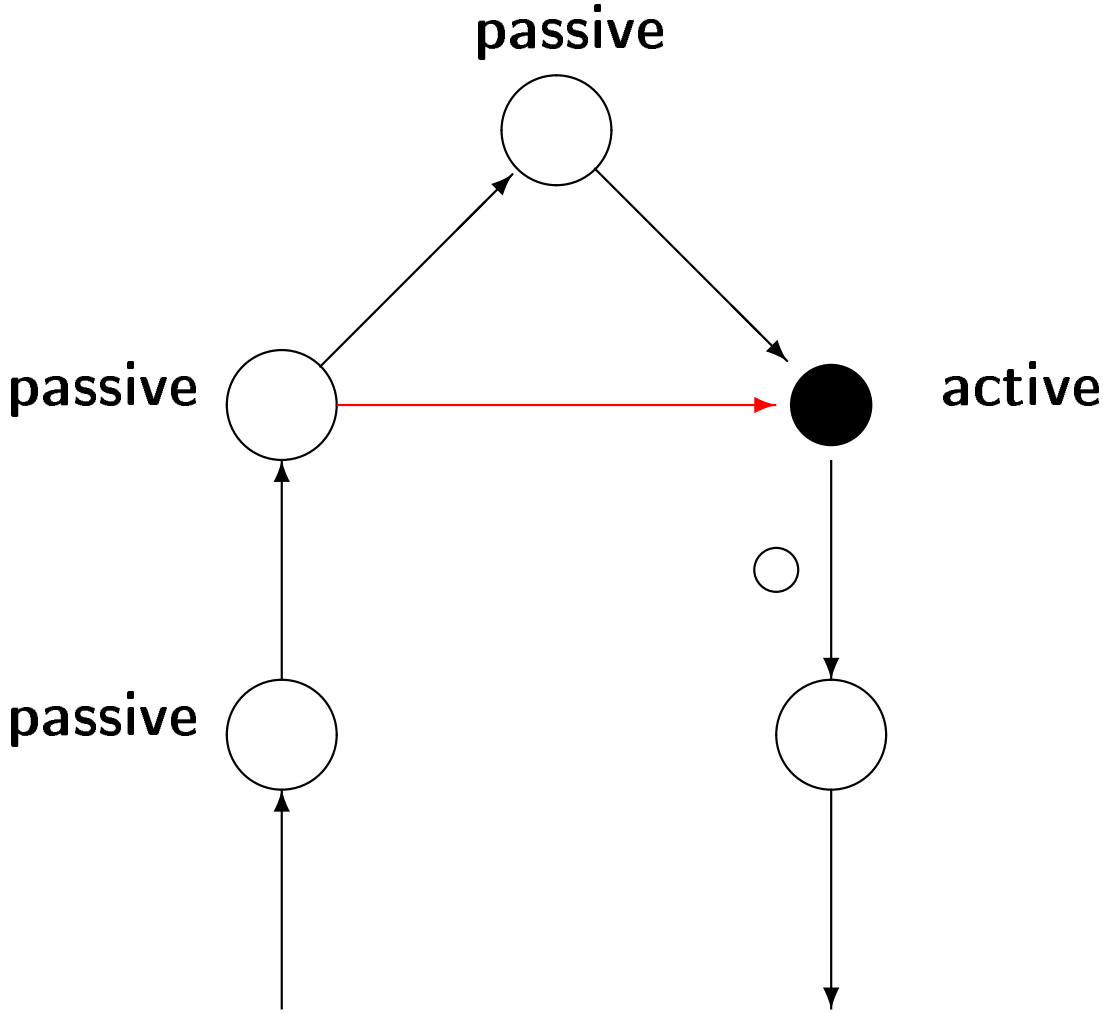
Safra's Algorithm

Alas, these principles are not enough for the correct operation of the algorithm.



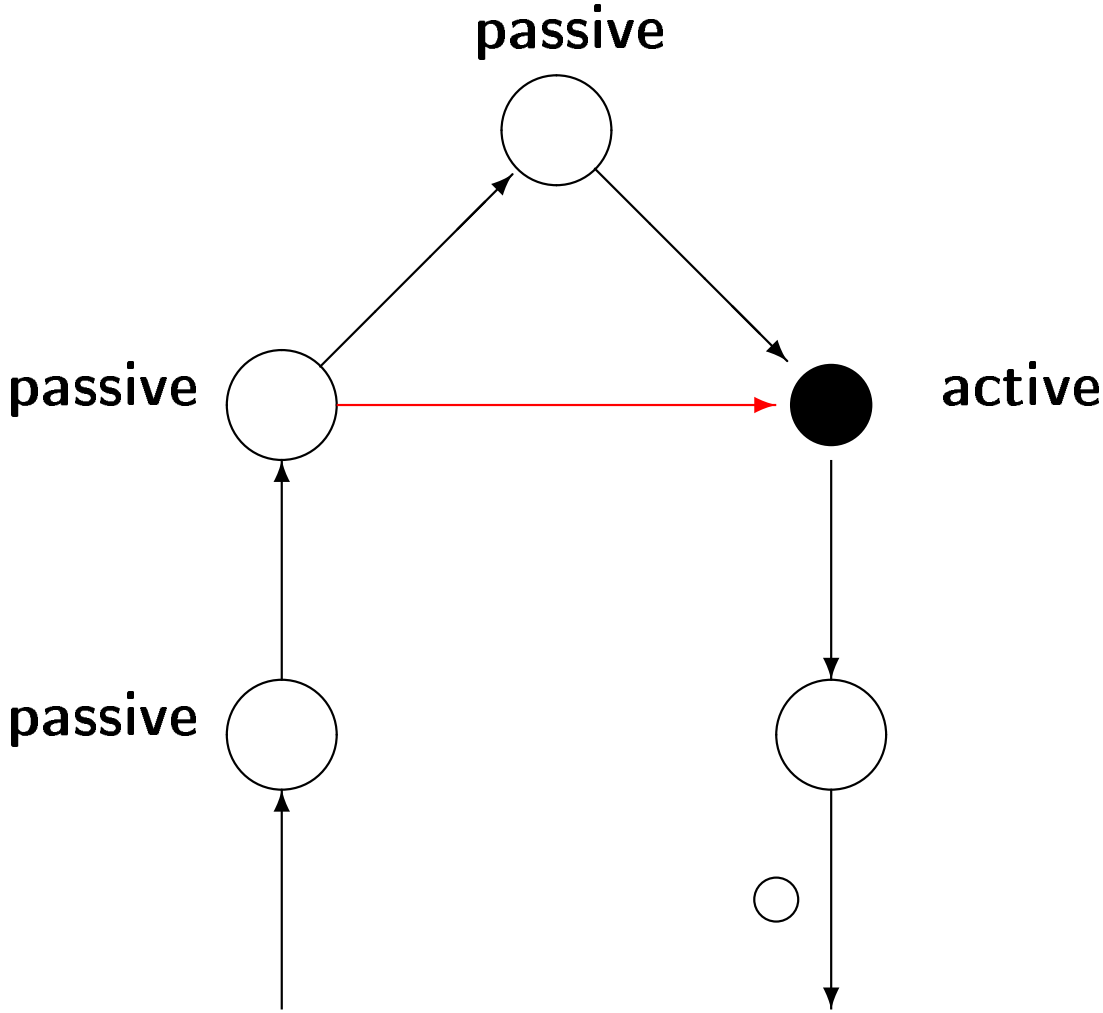
Safra's Algorithm

Alas, these principles are not enough for the correct operation of the algorithm.



Safra's Algorithm

Alas, these principles are not enough for the correct operation of the algorithm.



Safra's Algorithm

We need three more principles.

- Every process p is equipped with a counter of basic messages mc_p , which is incremented when the process sends a basic message and decremented when the process receives a basic message.

Safra's Algorithm

We need three more principles.

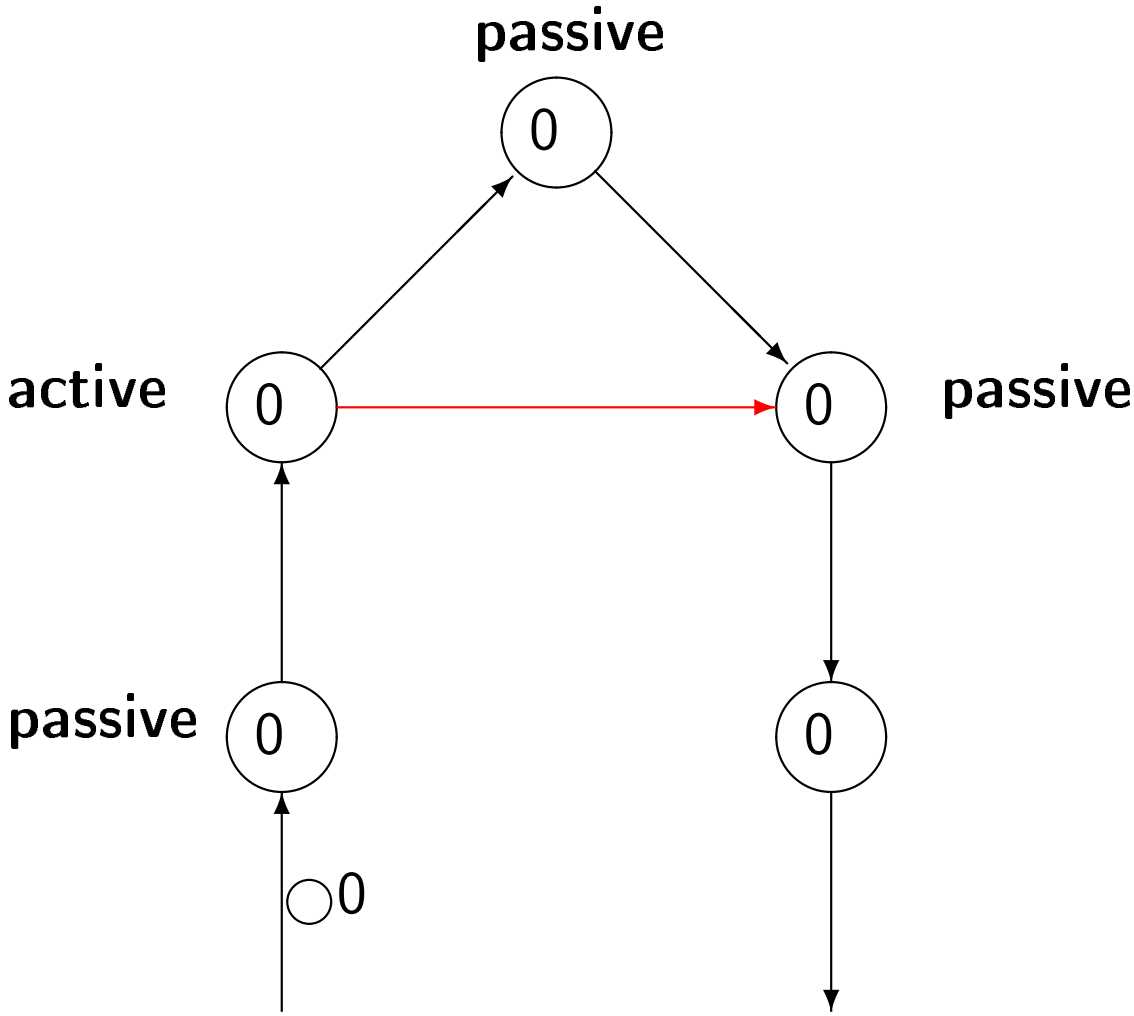
- Every process p is equipped with a counter of basic messages mc_p , which is incremented when the process sends a basic message and decremented when the process receives a basic message.
- The token during the passage of processes summarizes the values of their counters.

Safra's Algorithm

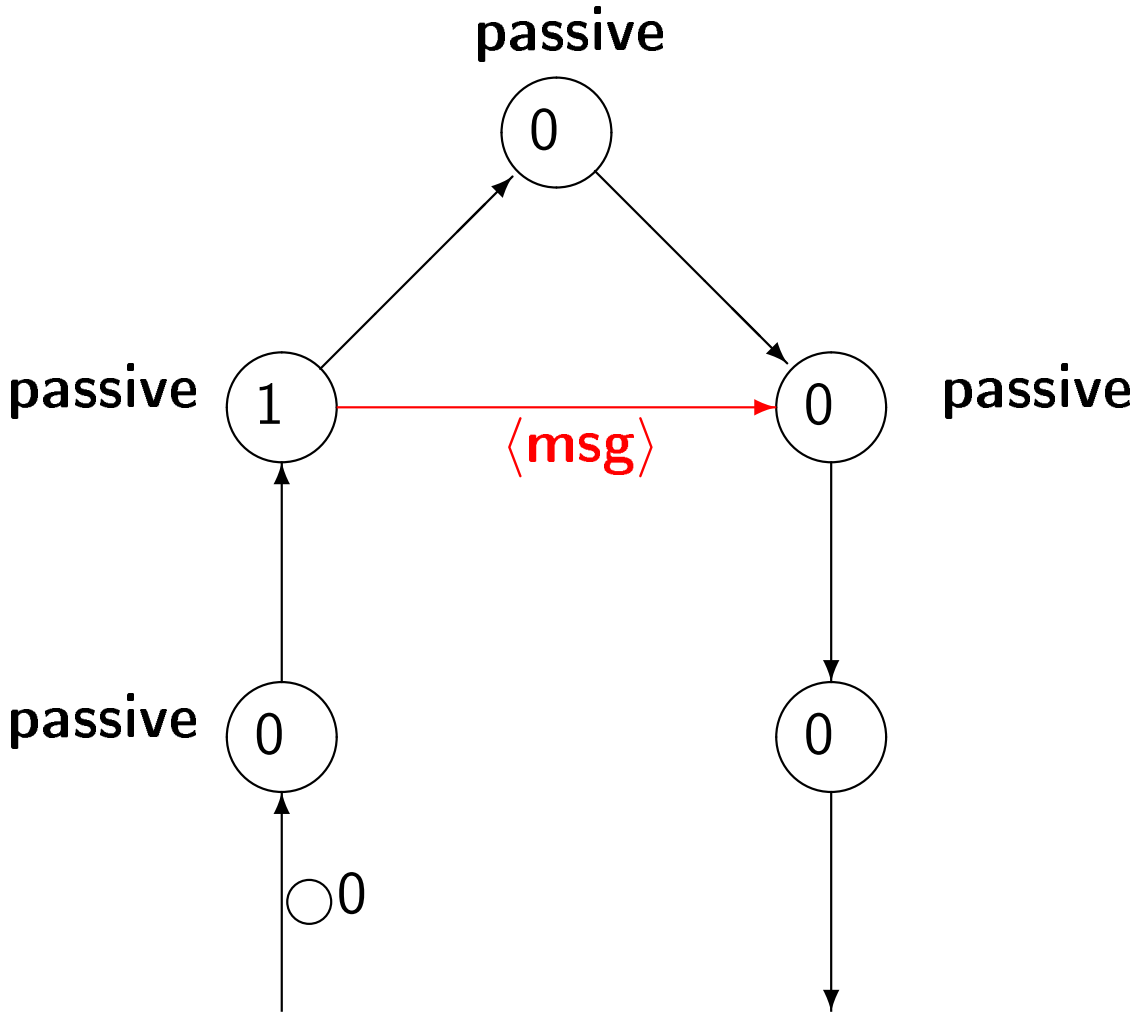
We need three more principles.

- Every process p is equipped with a counter of basic messages mc_p , which is incremented when the process sends a basic message and decremented when the process receives a basic message.
- The token during the passage of processes summarizes the values of their counters.
- The termination of basic computation is announced by the initiator when the sum of the counter readings collected by the token is 0 .

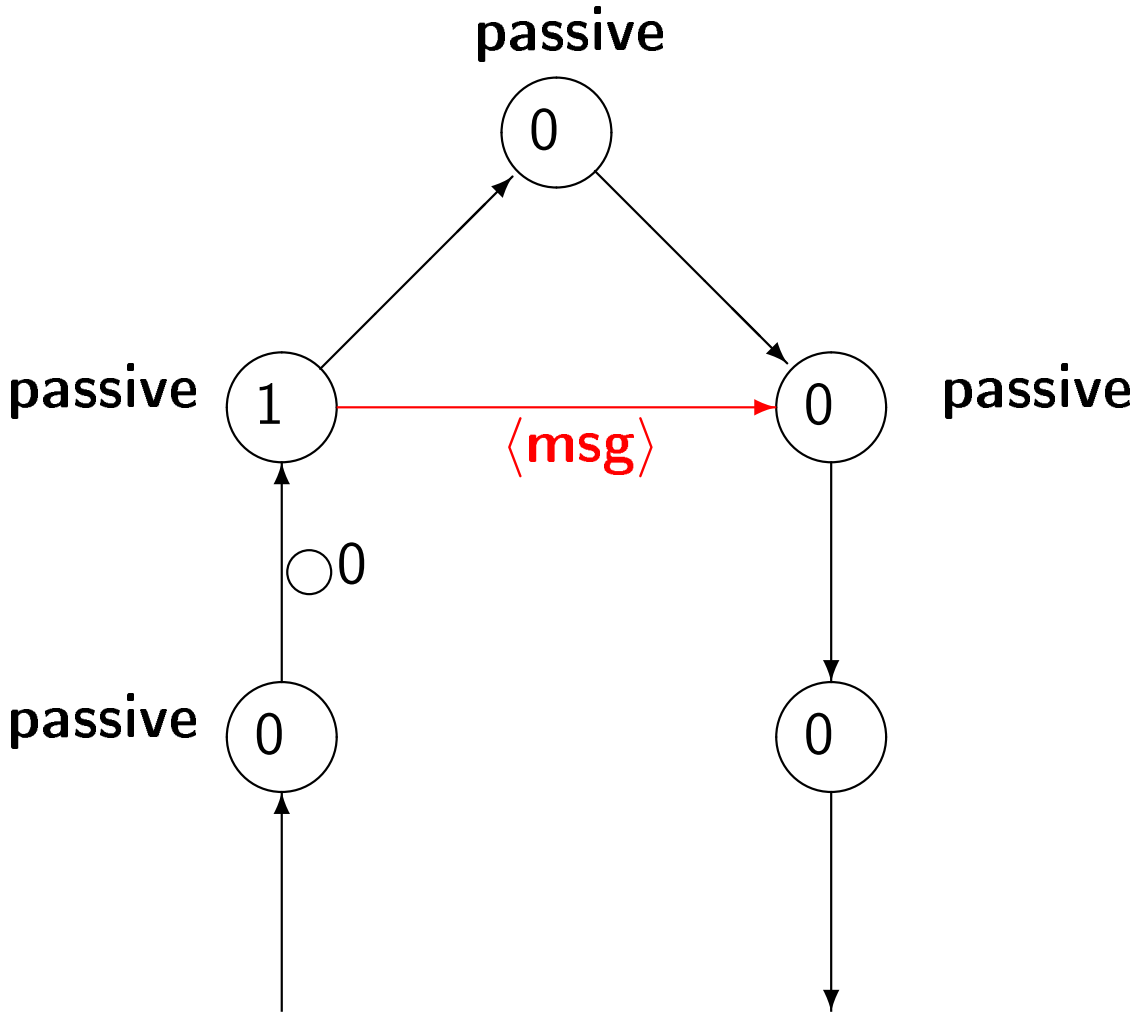
Safra's Algorithm



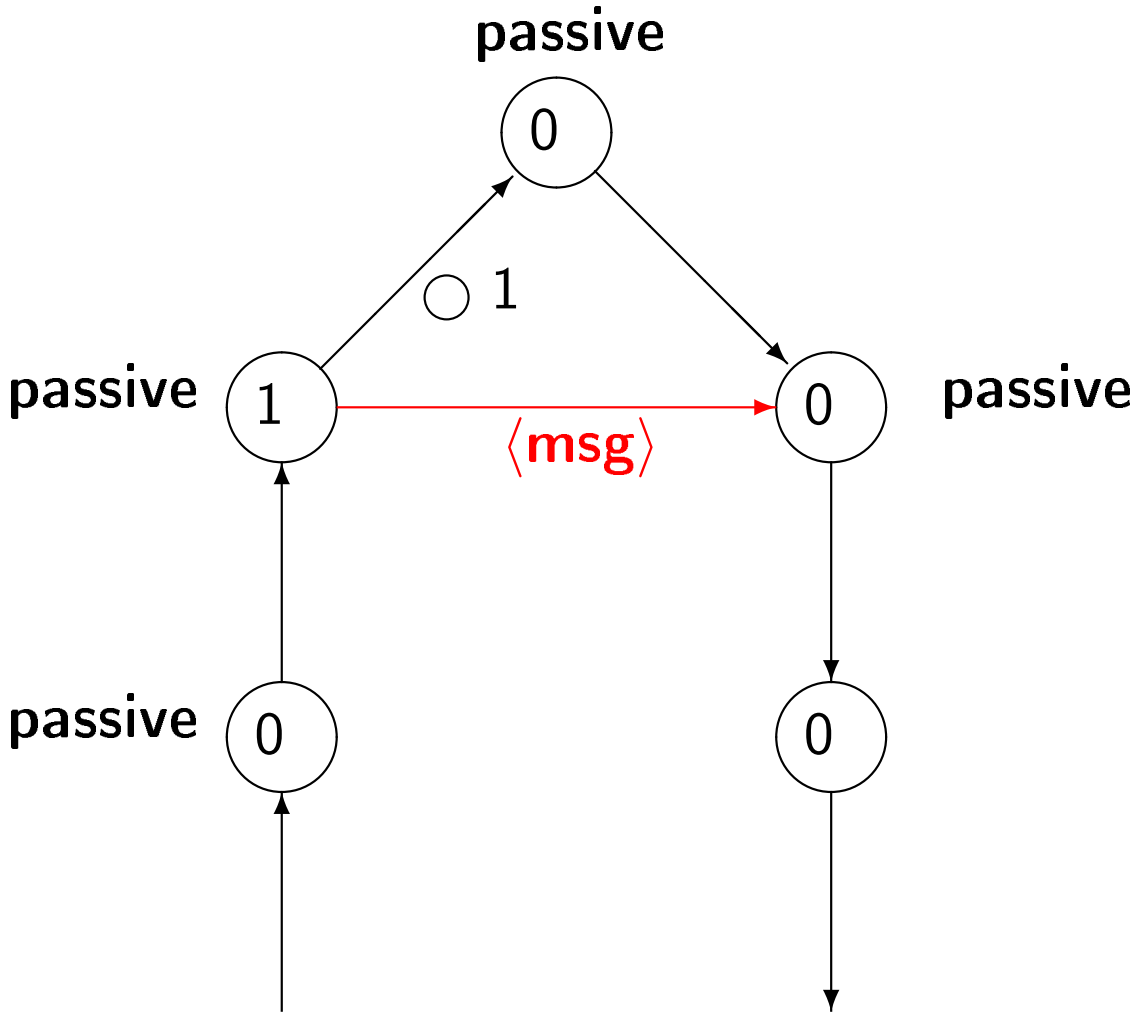
Safra's Algorithm



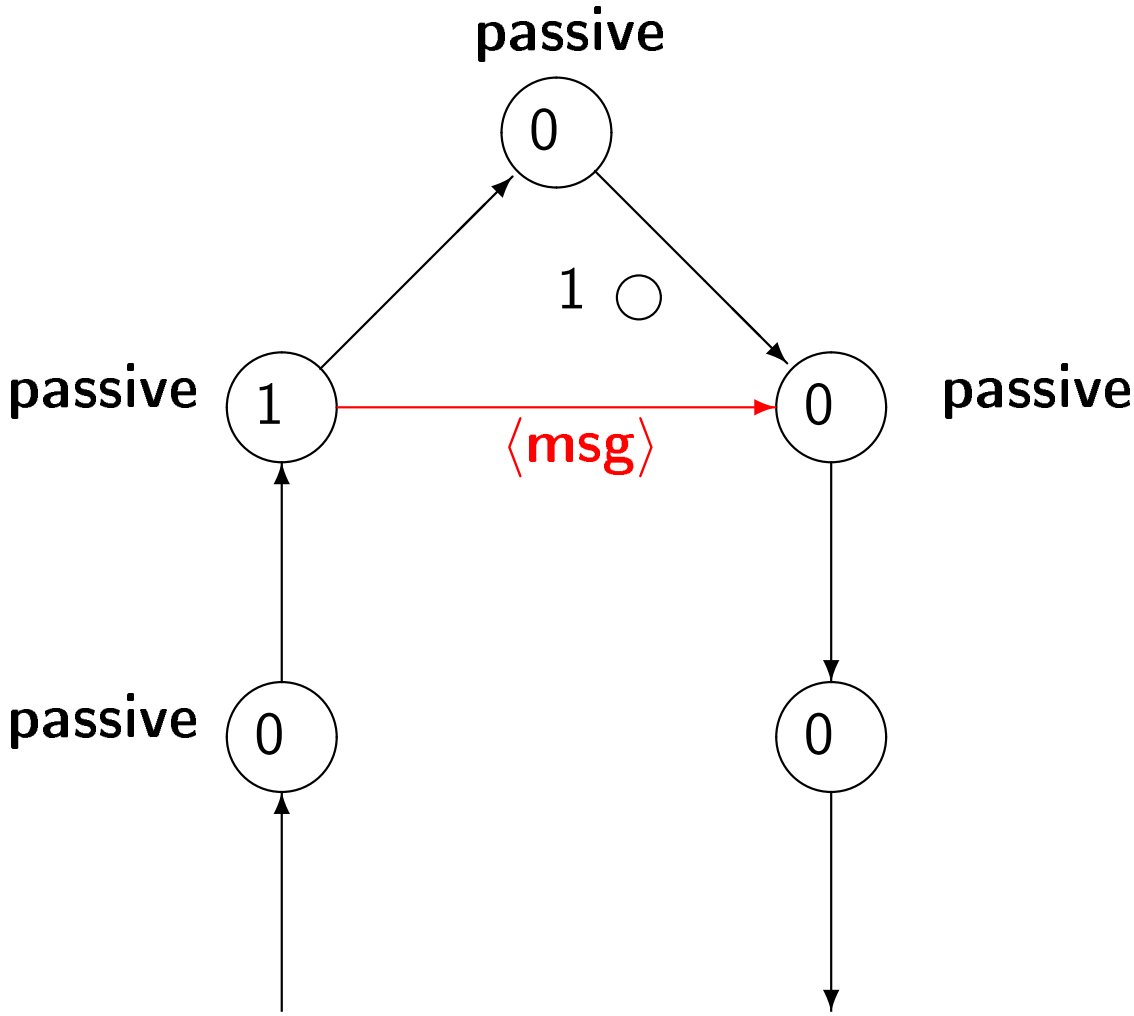
Safra's Algorithm



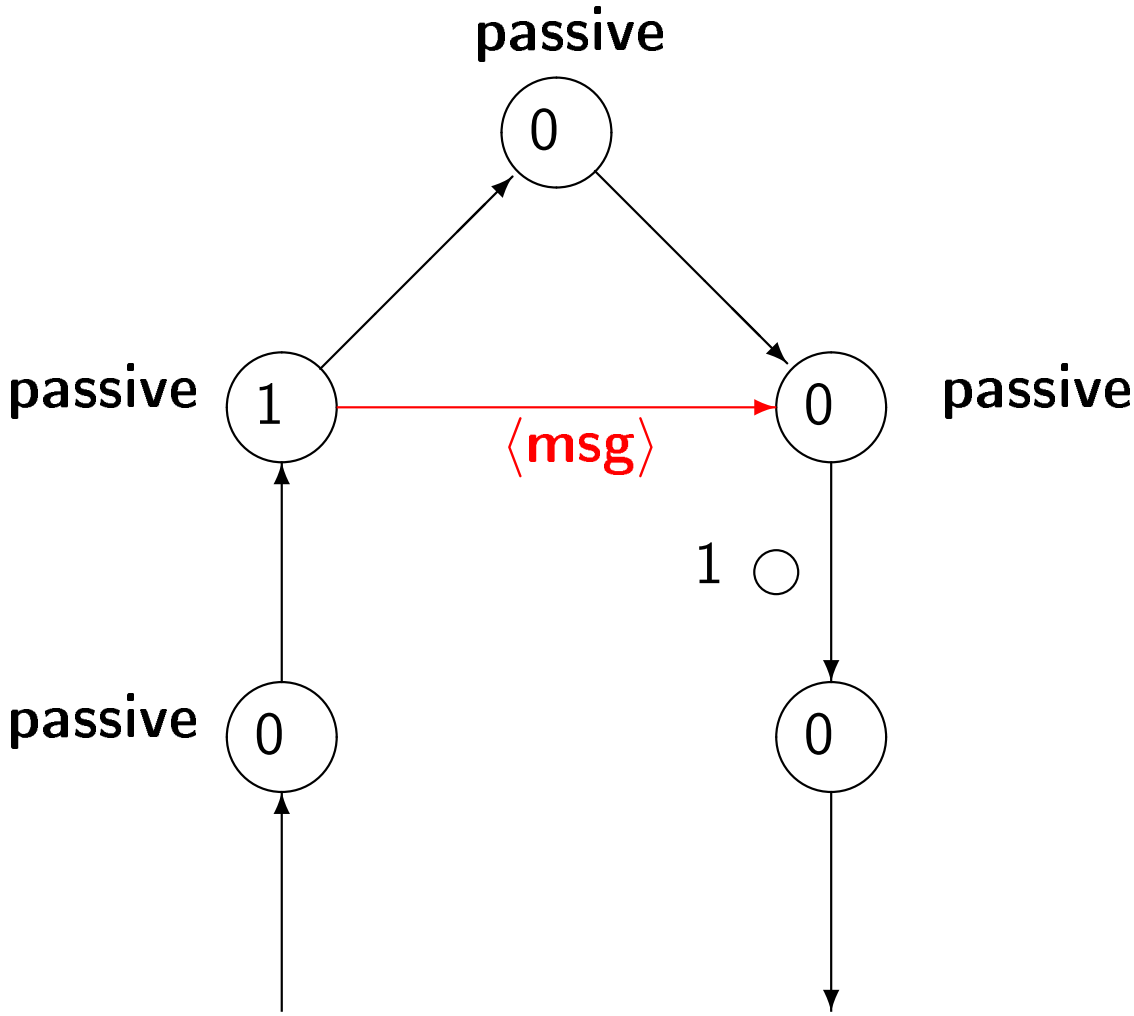
Safra's Algorithm



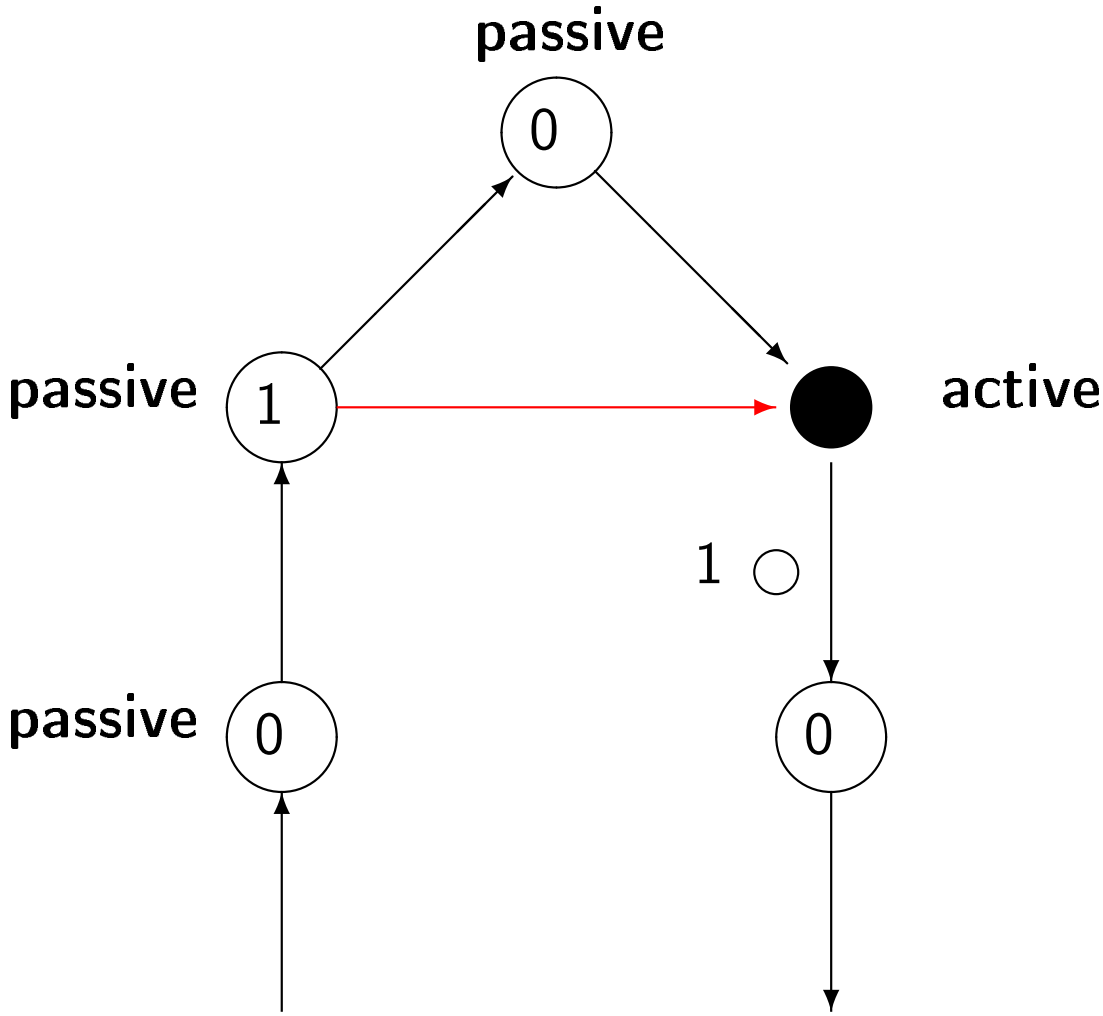
Safra's Algorithm



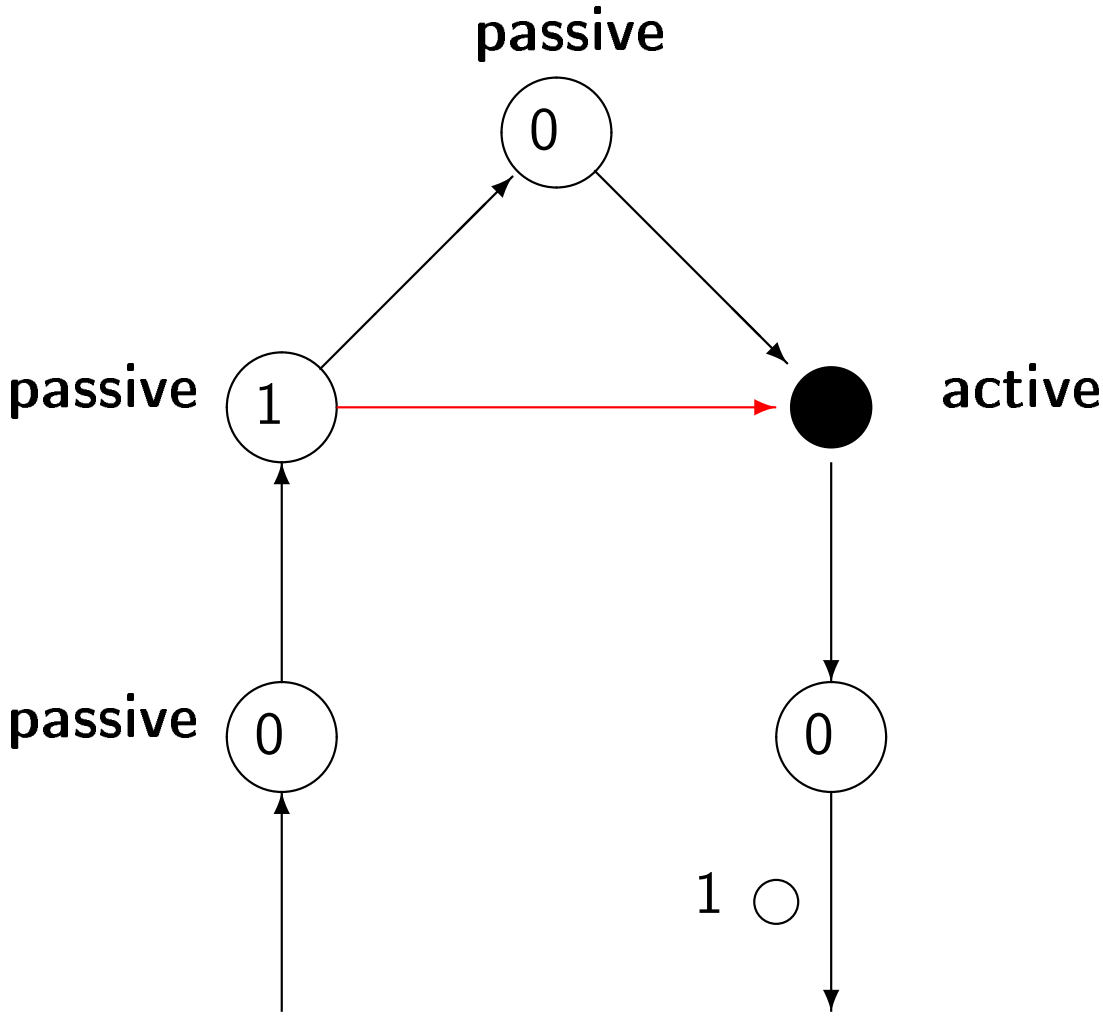
Safra's Algorithm



Safra's Algorithm



Safra's Algorithm



Safra's Algorithm

So, the operation of the algorithm is specified by the following rules.

Rule M: When process p sends a message, it increments its message counter; when process p receives a message, it decrements its message counter.

Safra's Algorithm

So, the operation of the algorithm is specified by the following rules.

Rule M: When process p sends a message, it increments its message counter; when process p receives a message, it decrements its message counter.

Rule 1: A receiving process becomes *black* .

Safra's Algorithm

So, the operation of the algorithm is specified by the following rules.

Rule M: When process p sends a message, it increments its message counter; when process p receives a message, it decrements its message counter.

Rule 1: A receiving process becomes *black* .

Rule 2: A process only handles the token when it is passive, and when a process forwards the token it adds the value of its message counter to q .

Safra's Algorithm

So, the operation of the algorithm is specified by the following rules.

Rule M: When process p sends a message, it increments its message counter; when process p receives a message, it decrements its message counter.

Rule 1: A receiving process becomes *black* .

Rule 2: A process only handles the token when it is passive, and when a process forwards the token it adds the value of its message counter to q .

Rule 3: When a *black* process forwards the token, the token becomes *black* .

Safra's Algorithm

So, the operation of the algorithm is specified by the following rules.

Rule M: When process p sends a message, it increments its message counter; when process p receives a message, it decrements its message counter.

Rule 1: A receiving process becomes *black* .

Rule 2: A process only handles the token when it is passive, and when a process forwards the token it adds the value of its message counter to q .

Rule 3: When a *black* process forwards the token, the token becomes *black* .

Rule 4: When the wave ends unsuccessfully, p_0 initiates a new one.

Safra's Algorithm

So, the operation of the algorithm is specified by the following rules.

Rule M: When process p sends a message, it increments its message counter; when process p receives a message, it decrements its message counter.

Rule 1: A receiving process becomes *black* .

Rule 2: A process only handles the token when it is passive, and when a process forwards the token it adds the value of its message counter to q .

Rule 3: When a *black* process forwards the token, the token becomes *black* .

Rule 4: When the wave ends unsuccessfully, p_0 initiates a new one.

Rule 5: Each process turns *white* immediately after sending the token.

Safra's Algorithm

var $state_p$: $(active, passive)$;
$color_p$: $(white, black)$;
mc_p : **integer init** 0 ;

S$_p$: { $state_p = active$ }
begin send $\langle$**mes**$\rangle$;
$mc_p := mc_p + 1$ (* Rule M *)
end

R$_p$: { Message $\langle$**mes**$\rangle$ has been delivered to p }
begin receive $\langle$**mes**$\rangle$; $state_p := active$;
$mc_p := mc_p - 1$; (* Rule M *)
$color_p := black$ (* Rule 1 *)
end

Safra's Algorithm

$\mathbf{I}_p$: { $state_p = active$ }
begin $state_p := passive$ **end**
Start the termination detection;
executed once by p_0:
begin send $\langle \mathbf{tok}, white, 0 \rangle$ to p_{N-1} **end**

$\mathbf{T}_p$: (* Process p handles the token $\langle \mathbf{tok}, c, q \rangle$*)
{ $state_p = passive$ } (* Rule 2 *)
begin if $p = p_0$
then if $(c = white) \wedge (color_p = white) \wedge (mc_p + q = 0)$
then *Announce*
else send $\langle \mathbf{tok}, white, 0 \rangle$ to p_{N-1} (* Rule 4 *)
else if $(color_p = white)$ (* Rule 2 and 3 *)
then send $\langle \mathbf{tok}, c, q + mc_p \rangle$ to $Next_p$
else send $\langle \mathbf{tok}, black, q + mc_p \rangle$ to $Next_p$;
$color_p := white$ (* Rule 5 *)
end

Safra's Algorithm

To prove the correctness of Safra's Algorithm we introduce the following terms and predicates:

- $B(\gamma)$ is the total number of basic messages in transit in configuration γ .

Safra's Algorithm

To prove the correctness of Safra's Algorithm we introduce the following terms and predicates:

- $B(\gamma)$ is the total number of basic messages in transit in configuration γ .
- $t(\gamma)$ is the ID (number) of a process to which the token is forwarded or which holds the token in configuration γ . (**Comment:** the token is forwarded to the processes in descending order of their numbers $p_0, p_{N-1}, p_{N-2}, \ldots, p_1, p_0, \ldots$).

Safra's Algorithm

To prove the correctness of Safra's Algorithm we introduce the following terms and predicates:

- $B(\gamma)$ is the total number of basic messages in transit in configuration γ .
- $t(\gamma)$ is the ID (number) of a process to which the token is forwarded or which holds the token in configuration γ . (**Comment:** the token is forwarded to the processes in descending order of their numbers $p_0, p_{N-1}, p_{N-2}, \ldots, p_1, p_0, \ldots$).
- $q(\gamma)$ is the value which is stored in the token in configuration γ .

Safra's Algorithm

To prove the correctness of Safra's Algorithm we introduce the following terms and predicates:

- $P_M(\gamma) \equiv (B(\gamma) = \sum_{p \in procs} mc_p)$.

Safra's Algorithm

To prove the correctness of Safra's Algorithm we introduce the following terms and predicates:

- $P_M(\gamma) \equiv (B(\gamma) = \sum_{p \in procs} mc_p)$.
- $P_0(\gamma) \equiv (\forall i\ (N > i > t(\gamma)) : state_{p_i} = passive) \wedge \left(q(\gamma) = \sum_{N>i>t} mc_{p_i}\right)$.

Safra's Algorithm

To prove the correctness of Safra's Algorithm we introduce the following terms and predicates:

- $P_M(\gamma) \equiv (B(\gamma) = \sum_{p \in procs} mc_p)$.
- $P_0(\gamma) \equiv (\forall i\ (N > i > t(\gamma)) : state_{p_i} = passive) \wedge \left(q(\gamma) = \sum_{N>i>t} mc_{p_i}\right)$.
- $P_1(\gamma) \equiv \left(\sum_{i \leq t(\gamma)} mc_{p_i} + q(\gamma)\right) > 0$.

Safra's Algorithm

To prove the correctness of Safra's Algorithm we introduce the following terms and predicates:

- $P_M(\gamma) \equiv (B(\gamma) = \sum_{p \in procs} mc_p)$.
- $P_0(\gamma) \equiv (\forall i\ (N > i > t(\gamma)) : state_{p_i} = passive) \wedge \left(q(\gamma) = \sum_{N>i>t} mc_{p_i}\right)$.
- $P_1(\gamma) \equiv \left(\sum_{i \leq t(\gamma)} mc_{p_i} + q(\gamma)\right) > 0$.
- $P_2(\gamma) \equiv \exists j\ (t(\gamma) \geq j \geq 0) : color_{p_j} = black$.

Safra's Algorithm

To prove the correctness of Safra's Algorithm we introduce the following terms and predicates:

- $P_M(\gamma) \equiv (B(\gamma) = \sum_{p \in procs} mc_p)$.
- $P_0(\gamma) \equiv (\forall i\ (N > i > t(\gamma)) : state_{p_i} = passive) \wedge \left(q(\gamma) = \sum_{N>i>t} mc_{p_i}\right)$.
- $P_1(\gamma) \equiv \left(\sum_{i \leq t(\gamma)} mc_{p_i} + q(\gamma)\right) > 0$.
- $P_2(\gamma) \equiv \exists j\ (t(\gamma) \geq j \geq 0) : color_{p_j} = black$.
- $P_3(\gamma) \equiv$ **the token is black** .

Safra's Algorithm

To prove the correctness of Safra's Algorithm we introduce the following terms and predicates:

- $P_M(\gamma) \equiv (B(\gamma) = \sum_{p \in procs} mc_p)$.
- $P_0(\gamma) \equiv (\forall i\ (N > i > t(\gamma)) : state_{p_i} = passive) \wedge \left(q(\gamma) = \sum_{N>i>t} mc_{p_i}\right)$.
- $P_1(\gamma) \equiv \left(\sum_{i \leq t(\gamma)} mc_{p_i} + q(\gamma)\right) > 0$.
- $P_2(\gamma) \equiv \exists j\ (t(\gamma) \geq j \geq 0) : color_{p_j} = black$.
- $P_3(\gamma) \equiv$ **the token is black** .
- $P(\gamma) \equiv P_M(\gamma) \wedge (P_0(\gamma) \vee P_1(\gamma) \vee P_2(\gamma) \vee P_3(\gamma))$.

Safra's Algorithm

Lemma 3.

Assertion $P(\gamma)$ is an invaruiant of Safra's Algorithm.

Proof.

This is Your HOMETASK 3.

Safra's Algorithm

Theorem 6.

Safra's Algorithm is a correct termination detection algorithm.

Proof.

Follows from Lemma 3 .

Credit-Recovery Algorithm

Mattern has proposed an algorithm that detects termination very fast, namely, within one time unit after its occurrence

The algorithm detects the termination of a centralized computation and assumes that each process can send a message to the initiator of the computation directly (i.e., the network contains a star with the initiator as the center).

Credit-Recovery Algorithm

In the algorithm each message and each process are assigned a credit value, which is always between 0 and 1 (inclusive), and the algorithm maintains the following assertions as invariants.

S1. The sum of all credits (in messages and processes) equals 1 .

S2. Every basic message has a positive credit.

S3. Every active process has a positive credit.

Processes holding a positive credit when this is not prescribed according to these rules (i.e., passive processes) send their credit to the initiator. The initiator acts as a bank, collecting all credits sent to it in a variable *ret* (for returned credit).

When the initiator collects all the credits it announces the termination of basic computation.

Credit-Recovery Algorithm

Rule 1. If $ret = 1$ then the initiator calls the procedure *Announce*.

Credit-Recovery Algorithm

Rule 1. If $ret = 1$ then the initiator calls the procedure *Announce*.

Rule 2. When a process becomes passive, it sends its credit to the initiator.

Credit-Recovery Algorithm

Rule 1. If $ret = 1$ then the initiator calls the procedure *Announce*.

Rule 2. When a process becomes passive, it sends its credit to the initiator.

Rule 3. When an active process p sends a message, its credit is divided among p and the message.

Rule 4. When a process is activated it is given the credit of the message activating it.

Credit-Recovery Algorithm

Rule 1. If $ret = 1$ then the initiator calls the procedure *Announce*.

Rule 2. When a process becomes passive, it sends its credit to the initiator.

Rule 3. When an active process p sends a message, its credit is divided among p and the message.

Rule 4. When a process is activated it is given the credit of the message activating it.

Rule 5. When an active process receives a basic message, the credit of that message is added to the credit of the process.

Credit-Recovery Algorithm

var $state_p$: $(active, passive)$ **init if** $p = p_0$ **then** $active$ **else** $passive$;
$cred_p$: fraction **init if** $p = p_0$ **then** 1 **else** 0 ;
ret : fraction **init** 0 ; for p_0 only

$\mathbf{S}_p$: { $state_p = active$ } (* Rule 3 *)
begin send $\langle \mathbf{mes}, cred_p/2 \rangle$; $cred_p := cred_p/2$ **end**

$\mathbf{R}_p$: { message $\langle \mathbf{mes}, c \rangle$ has been delivered to p }
begin receive $\langle \mathbf{mes}, c \rangle$; $state_p := active$;
$cred_p := cred_p + c$ (* Rule 4 and 5 *)
end

$\mathbf{I}_p$: { $state_p = active$ }
begin $state_p := passive$;
send $\langle \mathbf{ret}, cred_p \rangle$ to p_0 ; $cred_p := 0$ (* Rule 2 *)
end

$\mathbf{A}_{p_0}$: { Message $\langle \mathbf{ret}, c \rangle$ has been delivered to p_0 }
begin receive $\langle \mathbf{ret}, c \rangle$; $ret := ret + c$;
if $ret = 1$ **then** *Announce* (* Rule 1 *)
end

Credit-Recovery Algorithm

Theorem 7.
The Credit-Recovery Algorithm is a correct termination detection algorithm.

Proof.
HOMETASK 4: find a suitable invariant and use it to prove the Theorem.

Solutions Using Timestamps

To solve the problem of termination detection time stamps can be exploited. It is assumed that all processes are equipped with a clock; for this purpose both hardware timers and Lamport logic clocks are suitable.
Rana's solution is based on a local predicate $quiet(p)$ which for every process p satisfies the condition:

$$quiet(p) \implies state_p = passive \wedge \\ \wedge \text{ no basic message sent by } p \\ \text{is in transit.}$$

As it can be seen from this specification: $(\forall p\ quiet(p)) \implies \mathbf{term}$.
Actually, $quiet$ can be defined as follows:

$$quiet(p) \equiv (state_p = passive \ \wedge \ unack_p = 0),$$

where $unack$ is the number of messages whose receipt is not acknowledged yet.

Rana's algorithm

The algorithm aims to check whether, for a certain point in time t , all processes were quiet at time t ; termination at time t follows.

This is done by a wave, which asks each process to confirm that it was quiet at that time and later; a process that was not quiet does not respond to messages of the wave, effectively extinguishing the wave.

The visit of a wave to process p does not affect the variables of process p used for termination detection. As a consequence the correct operation of the algorithm is not disturbed by the concurrent execution of several waves.

Rana's algorithm

Process p when becoming quiet, stores the time qt_p when this happens and launches a wave to check if all processes are quite since qt_p .

If this is the case, termination is detected. Otherwise, there will be a process that becomes quiet later, and a new wave will be started.

Rana's algorithm (part 1)

var $state_p$: ($active$, $passive$) ;
θ_p : **integer init** 0 ; (* Logical clocks *)
$unack_p$: **integer init** 0 ; (* Number of unacknowledged messages *)
qt_p : **integer init** 0 ; (* Time of most recent transition to *quiet*

S$_p$: { $state_p = active$ }
begin $\theta_p := \theta_p + 1$; send $\langle$**mes**, $\theta_p\rangle$; $unack_p := unack_p + 1$ **end**
R$_p$: { Message $\langle$**mes**, $\theta\rangle$ from q arrives at p }
begin receive $\langle$**mes**, $\theta\rangle$; $\theta_p := \max(\theta_p, \theta) + 1$;
send $\langle$**ack**, $\theta_p\rangle$ to q ; $state_p := active$
end
I$_p$: { $state_p = active$ }
begin $\theta_p := \theta_p + 1$; $state_p := passive$;
if $unack_p = 0$ **then** (* p passes to quite *)
begin $qt_p := \theta_p$; send $\langle$**tok**, $\theta_p, qt_p, p\rangle$ to $Next_p$ **end**
end

Rana's algorithm (part 2)

$\mathbf{A}_p$: { Acknowledgment $\langle \mathbf{ack}, \theta \rangle$ has been handled to p }
begin receive $\langle \mathbf{ack}, \theta \rangle$; $\theta_p := \max(\theta_p, \theta) + 1$;
$unack_p := unack_p - 1$;
if $unack_p = 0$ **and** $state_p = passive$ **then** (* p passes to *quiet*
begin $qt_p := \theta_p$; send $\langle \mathbf{tok}, \theta_p, qt_p, p \rangle$ to $Next_p$ **end**
end

$\mathbf{T}_p$: { Token $\langle \mathbf{tok}, \theta, qt, q \rangle$ arrives at p }
begin receive $\langle \mathbf{tok}, \theta, qt, q \rangle$; $\theta_p := \max(\theta_p, \theta) + 1$;
if $quiet(p)$ **then**
if $p = q$ **then** *Announce*
else if $qt \geq qt_p$ **then** send $\langle \mathbf{tok}, \theta_p, qt, q \rangle$ to $Next_p$
end

Rana's Algorithm

Theorem 8.
The Rana's Algorithm is a correct termination detection algorithm.

Proof.
HOMETASK 5: find a suitable invariant and use it to prove the Theorem.

END OF LECTURE 10.