Распределенные алгоритмы

ЛЕКТОР: В.А. Захаров

Лекция 11.

Задача обнаружения завершения вычислений

Алгоритм Дейкстры–Шолтена

Алгоритм Шави–Франчеза

Алгоритм Сафры

Алгоритм возвращения кредитов

Алгоритм Раны

Задача обнаружения завершения вычислений

Вычисление распределенного алгоритма завершается, когда этот алгоритм достигает некоторой **заключительной конфигурации**, из которой нельзя сделать ни одного шага.

Вычисление завершается явно , если все процессы пребывают в заключительном состоянии . Тогда говорят, что произошло завершение работы алгоритма .

Вычисление завершается неявно , если все каналы пусты, но при этом не все процессы пребывают в заключительном состоянии, т.е. некоторые процессы ожидают поступления сообщений. Тогда говорят, что произошло завершение обмена сообщениями .

Нужно уметь преобразовывать алгоритмы, в которых вычисления оканчиваются по завершении обмена сообщениями, в алгоритмы, в которых вычисления оканчиваются завершением работы процессов.

Задача обнаружения завершения вычислений

Для этого потребуются два дополнительных алгоритма, которые взаимодействуют друг с другом, а также с заданным алгоритмом, имеющим неявное завершение вычислений.

Один из этих алгоритмов анализирует текущее вычисление исходного алгоритма и обнаруживает, что это вычисление достигло заключительной конфигурации. После этого вызывается второй алгоритм, который рассылает сообщение о завершении вычисления всем процессам, призывая их перейти в заключительное состояние.

Самым сложным оказывается алгоритм, обнаруживающий завершение вычисления. Процедура рассылки весьма проста.

Покажем, что обнаружение завершения возможно для всех классов сетей, для которых можно разработать волновой алгоритм..

Задача обнаружения завершения вычислений

Множество всех состояний Z_p процесса p разбивается на два подмножества активных и пассивных состояний.

Состояние c_p процесса p активно, если процесс p в состоянии c_p может выполнить внутреннее действие или отправить сообщение; иначе это состояние считается пассивным.

В пассивном состоянии c_p возможен только прием сообщений, или в нем вообще не выполняется никаких действий, и тогда c_p — одно из заключительных состояний процесса p .

Говорят, что процесс p активен, если он пребывает в одном из активных состояний; иначе процесс p называют пассивным.

Отправлять сообщения может только активный процесс, а пассивный процесс может стать активным только после получения сообщения. Активный процесс может стать пассивным, если он перейдет в пассивное состояние.

Задача обнаружения завершения вычислений

Сделаем ряд упрощений.

1. **Активный процесс становится пассивным только после осуществления внутреннего события.**
2. **Процесс всегда становится активным после получения сообщения.**
3. **Внутренние события, приводящие к тому, что процесс p становится пассивным, — это единственно возможные внутренние события в процессе p .**

   Внутренние события, переводящие процесс p из одного активного состояния в другое активное состояние, игнорируются: алгоритм обнаружения завершения не занимается выведыванием информации о локальных вычислениях процессов.

Задача обнаружения завершения вычислений

Базовый алгоритм.

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

S$_p$: { $state_p = active$ }
begin send $\langle$**mes**$\rangle$ **end**

R$_p$: { Сообщение $\langle$**mes**$\rangle$ доставлено процессу p }
begin receive $\langle$**mes**$\rangle$; $state_p := passive$ **end**

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

Вот так выглядит всякий процесс для алгоритма обнаружения завершения.

Задача обнаружения завершения вычислений

Будем полагать, что предикат **term**(γ) обращается в истину на всякой конфигурации γ , на которой не может произойти ни одного события в базовом вычислении.

Теорема 1.

$$\textbf{term}(\gamma) \iff (\forall p \in \mathbb{P} : state_p = passive) \wedge (\forall pq \in E : M_{pq} \text{ не содержит сообщения } \langle \textbf{mes} \rangle).$$

Доказательство. Если все процессы пассивны, то ни внутреннее событие, ни событие отправления сообщения невозможно. А если каналы пусты, то события приема сообщения также невозможны.

Если какой-нибудь процесс активен, то в нем может произойти внутреннее событие или событие отправления сообщения, а если в одном из каналов есть сообщение, то может осуществиться прием этого сообщения.

Задача обнаружения завершения вычислений

Задача состоит в том, чтобы присоединить к системе контрольный алгоритм , который приведет все процессы в заключительные состояния, после того как базовое вычисление достигнет заключительной конфигурации.

В контрольном алгоритме происходит обмен (контрольными) сообщениями. Эти сообщения могут быть отправлены пассивными процессами, и получение контрольного сообщения не будет переводить пассивный процесс в разряд активных.

Контрольный алгоритм состоит из алгоритма обнаружения завершения вычисления и алгоритма оповещения о завершении вычисления , который переводит все процессы в заключительные состояния.

Задача обнаружения завершения вычислений

Алгоритм обнаружения завершения вычисления должен удовлетворять следующим трем требованиям.

1. **Невмешательство.** Алгоритм обнаружения завершения вычисления не должен оказывать влияния на вычисления базового алгоритма.
2. **Живость.** Если выполняется условие **term**, то алгоритм оповещения *Announce* должен быть вызван спустя конечное число шагов.
3. **Безопасность.** Если вызван алгоритм *Announce*, то конфигурация должна удовлетворять условию **term**.

Задача обнаружения завершения вычислений

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

{Сообщение $\langle$**stop**$\rangle$ поступило процессу p}
begin receive $\langle$**stop**$\rangle$; $RecStop_p := RecStop_p + 1$;
Announce ;
if $RecStop_p = \#In_p$ **then halt**
end

Задача обнаружения завершения вычислений

Теорема 2.

Для всякого алгоритма обнаружения завершения вычисления существует такое базовое вычисление, осуществляющее M обменов базовыми сообщениями, для обнаружения завершения которого рассматриваемый алгоритм совершает не менее M обменов контрольными сообщениями.

Теорема 3.

Для обнаружения завершения децентрализованного базового вычисления в худшем случае требуется совершить обмен не менее чем W контрольными сообщениями, где W — коммуникационная сложность волнового алгоритма.

Алгоритм Дейкстры-Шолтена

Условия применения:

- базовый алгоритм централизованный;
- топология сети произвольная;
- сеть неориентированная;
- контрольный алгоритм централизованный и инициируется в том же процессе, что и базовый алгоритм.

Алгоритм Дейкстры-Шолтена

Алгоритм обнаружения завершения строит и постоянно обновляет дерево вычисления $T = (V_T, E_T)$, которое обладает следующими двумя свойствами.

1. Граф T либо является пустым, либо представляет собой ориентированное дерево, корнем которого является вершина p_0 .
2. Множество V_T включает все активные процессы и все базовые сообщения, находящиеся на этапе пересылки.

Инициатор p_0 вызывает процедуру *Announce*, если $p_0 \notin V_T$; согласно первому свойству граф T в таком случае является пустым, а согласно второму свойству это означает, что вычисление завершилось.

Алгоритм Дейкстры-Шолтена

Когда процесс p отправляет базовое сообщение $\langle \mathbf{mes} \rangle$, это сообщение добавляется к дереву в качестве вершины $\langle \mathbf{mes} \rangle$, причем родительской вершиной для нее становится процесс p .

Когда процесс p , не входящий в состав дерева, становится активным после получения сообщения от процесса q , вершина q становится родительской вершиной для процесса p .

Чтобы обозначить явно отправителя сообщения, всякое базовое сообщение $\langle \mathbf{mes} \rangle$, отправленное процессом q , будет обозначаться записью $\mathbf{cmes}_q$.

Алгоритм Дейкстры-Шолтена

Вершины из дерева T удаляются по двум причинам.

1) Базовое сообщение удаляется после его получения.

2) Чтобы контрольный алгоритм сработал, дерево должно исчезнуть после завершения базового алгоритма. Сообщения — листья дерева T ; в каждом процессе есть переменная для подсчета числа вершин-последователей этого процесса в дереве T . Удаление из дерева вершины-последователя процесса p осуществляется в другом процессе q в результате того, что вершина-последователь

a) либо соответствует сообщению, достигшему процесса q ,

b) либо соответствует самому процессу q .

Чтобы в процессе p не нарушался учет вершин-последователей, ему отправлется сигнальное сообщение $\mathbf{sig}_p$ при удалении вершины-последователя процесса p . Это сообщение замещает удаленную вершину-последователя процесса p , а удаление этой вершины из дерева осуществляется самим процессом p после получения указанного сигнала; при этом p уменьшает значение счетчика последователей на 1.

Алгоритм Дейкстры-Шолтена

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$_p$: {Сообщение $\langle$**mes**, $q\rangle$ достигло процесса 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**(* Удалить p из T *)
begin if $father_p = p$
then *Announce*
else send $\langle$**sig**, $father_p\rangle$ to $father_p$;
$father_p := udef$
end
end

Алгоритм Дейкстры-Шолтена

$\mathbf{A}_p$: { Сигнал $\langle \mathbf{sig}, p \rangle$ достигает 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

Алгоритм Дейкстры-Шолтена

Для обоснования корректности алгоритма нужно показать, что граф T является деревом и при этом становится пустым только в том случае, когда базовое вычисление завершилось.

Для всякой конфигурации γ определим два множества

$$\begin{aligned} V_T \;=\; & \{p : father_p \neq udef\} \cup \{\langle \mathbf{mes}, p\rangle \text{ на этапе пересылки}\} \\ \cup\; & \{\langle \mathbf{sig}, p\rangle \text{ на этапе пересылки}\} \end{aligned}$$

и

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

Алгоритм Дейкстры-Шолтена

Свойство безопасности алгоритма обеспечивается инвариантом P , который определяется следующим образом:

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

Алгоритм Дейкстры-Шолтена

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

Согласно определению множество вершин графа T включает все сообщения (как базовые, так и контрольные), а также, согласно соотношению (1), все активные процессы.

Алгоритм Дейкстры-Шолтена

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

Соотношение (2) играет вспомогательную роль; в нем утверждается, что T действительно является графом и все дуги направлены к процессам.

Алгоритм Дейкстры-Шолтена

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

Соотношение (3) гласит о том, что подсчет вершин-последователей проводится каждым процессом корректно

Алгоритм Дейкстры-Шолтена

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

в соотношении (4) утверждается, что T является деревом и процесс p_0 — это его корень.

Алгоритм Дейкстры-Шолтена

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

Соотношение (5) служит для того, чтобы показать, что это дерево действительно исчезает, если базовое вычисление завершится.

Алгоритм Дейкстры-Шолтена

Лемма 1.
Предикат P является инвариантом алгоритма Дейкстры—Шолтена.

Доказательство.

Самостоятельно.

Говорков

Алгоритм Дейкстры-Шолтена

Теорема 1.

Алгоритм Дейкстры—Шолтена является корректным алгоритмом обнаружения завершения вычисления с M обменами базовыми сообщениями.

При этом в алгоритме Дейкстры—Шолтена происходит M обменов контрольными сообщениями.

Доказательство.

Самостоятельно.

Тутельян

Алгоритм Дейкстры-Шолтена

В алгоритме Дейкстры—Шолтена достигнут замечательный баланс между контрольными и базовыми сообщениями; для каждого базового сообщения, отправленного процессом p процессу q , рассмотренный алгоритм отправляет в точности одно сообщение от процесса q к процессу p .

Сложность по числу контрольных сообщений достигает нижней оценки, и поэтому данный алгоритм является оптимальным по сложности в наихудшем случае алгоритмом обнаружения завершения централизованных вычислений.

Алгоритм Шави—Франчеза

Этот метод обнаружения завершения обобщает алгоритм Дейкстры—Шолтена и может применяться для децентрализованных базовых вычислений.

В алгоритме Шави—Франчеза граф вычислений является лесом , каждое дерево которого имеет в качестве корня один из инициаторов базового вычисления.

Для обозначения дерева, корнем которого служит процесс p , будем использовать запись T_p .

Алгоритм Шави—Франчеза

В алгоритме Шави—Франчеза строится такой граф $F = (V_F, E_F)$, что

(1) либо F пуст, либо F представляет собой лес, каждое дерево которого имеет один из инициаторов в качестве корня,

(2) множество V_F содержит все активные процессы и все базовые сообщения, пребывающие на этапе пересылки.

Завершение базового вычисления будет обнаружено, когда указанный граф становится пустым.

Однако не просто убедиться в том, что этот граф стал пустым. Поскольку корнем дерева вычислений служит процесс p_0 , именно процесс p_0 может оценить, является ли дерево пустым, и вызвать в этом случае процедуру *Announce*.
Когда мы имеем дело с лесом, каждый инициатор может установить пустоту своего собственного дерева, но это еще не будет означать, что весь лес стал пустым графом.

Алгоритм Шави—Франчеза

Проверка того, что все деревья исчезли, проводится при помощи одной-единственной волны.

Лес F строится так, чтобы всякое дерево T_p , ставшее пустым, оставалось пустым и в дальнейшем. Это не препятствует процессу p вновь становиться активным, но если p становится активным после исчезновения его собственного дерева, то он заносится в дерево другого инициатора.

Каждый процесс принимает участие в распространении волны только тогда, когда его дерево исчезает; и если волна приводит к тому, что принимается решение, то вызывается процедура оповещения *Announce*.

Алгоритм Шави—Франчеза

Каждый процесс p располагает переменной $father_p$.

Значение этой переменной равно udef, если $p \notin V_T$.

Если же p является корнем дерева, то значением переменной $father_p$ является имя процесса p .

И наконец, если p является некорневой вершиной дерева T , то значением переменной $father_p$ служит имя родительской вершины данного процесса.

Переменная sc_p используется для подсчета числа последователей вершины p в дереве T .

Булева переменная $empty_p$ принимает значение true тогда и только тогда, когда дерево, в котором содержится вершина p , становится пустым.

Алгоритм Шави—Франчеза

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$_p$: {Сообщение $\langle$**mes**, $q\rangle$ поступило процессу 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** (* Удалить p из F *)
begin
if $father_p = p$ **then** $empty_p := true$ **else** send $\langle$**sig**, $father_p\rangle$
$father_p := udef$
end
end

Алгоритм Шави—Франчеза

$\mathbf{A}_p$: { Сигнал $\langle \mathbf{sig}, p \rangle$ достигает 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

Алгоритм Шави—Франчеза

Примечания.

1. В приведенном описании волновой алгоритм в явном виде не выделен.
2. Волновой алгоритм запускают только инициаторы базовых вычислений
3. Все процессы проводят параллельное выполнение волнового алгоритма, причем отправление сообщений или принятие решения разрешается только тем процессам p , у которых переменная $empty_p$ имеет значение true .
4. При осуществлении события *decide* вызывается процедура *Announce*.

Алгоритм Шави—Франчеза

Для всякой конфигурации γ определим следующее множество вершин:

$$V_T = \{p : father_p \neq udef\} \cup \{\langle \mathbf{mes}, p\rangle \text{ на этапе пересылки}\} \cup \{\langle \mathbf{sig}, p\rangle \text{ на этапе пересылки}\}$$

и дуг

$$E_F = \{(p, father_p) : father_p \neq udef \wedge father_p \neq p\} \cup \{(\langle \mathbf{mes}, p\rangle, p) : \langle \mathbf{mes}, p\rangle \text{ на этапе пересылки}\} \cup \{(\langle \mathbf{sig}, p\rangle, p) : \langle \mathbf{sig}, p\rangle \text{ на этапе пересылки}\}.$$

Алгоритм Шави—Франчеза

Достаточные условия корректности (так называемое свойство безопасности алгоритма) обеспечиваются инвариантом Q :

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

Алгоритм Шави—Франчеза

Лемма 2.
Предикат Q является инвариантом алгоритма Шави—Франчеза.

Доказательство.

Самостоятельно.

Романов

Алгоритм Шави—Франчеза

Теорема 2.

Алгоритм Шави—Франчеза является корректным алгоритмом обнаружения завершения вычисления; в нем используется $M + W$ обменов контрольными сообщениями (здесь W — коммуникационная сложность волнового алгоритма).

Доказательство.

Самостоятельно.

Колосов

Алгоритм Сафры

Условия применения:

- Базовый алгоритм децентрализованный.
- Каналы связи однонаправленные.
- В сети существует гамильтонов контур, по которому распространяются контрольные сообщения; базовые сообщения могут передаваться по любым каналам.
- Контрольный алгоритм — централизованный волновой алгоритм в однонаправленном кольце.

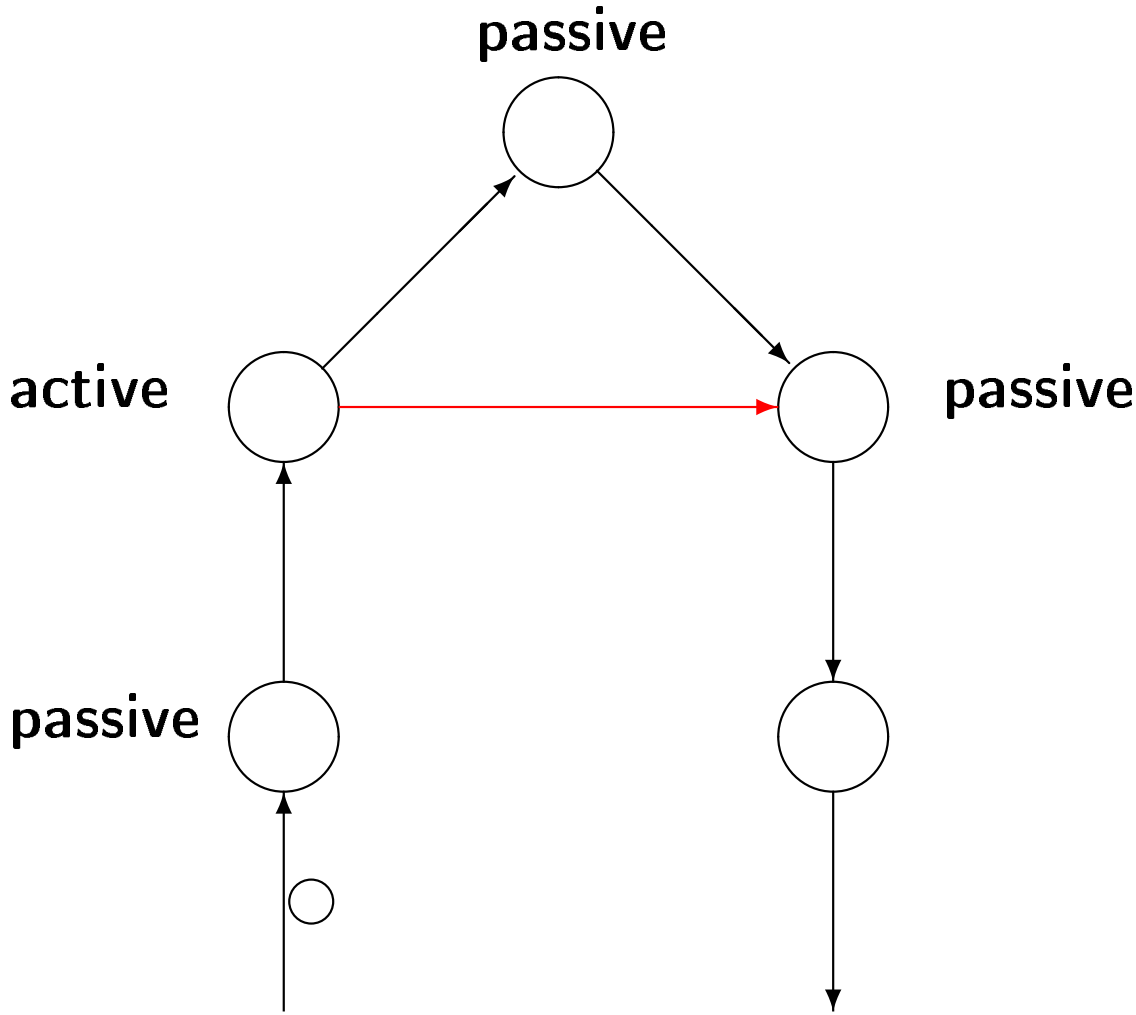
Алгоритм Сафры

Принципы работы.

- Каждый процесс может иметь некоторый цвет, который задается переменной *color* ; эта переменная может иметь одно из двух значений *white* или *black* .
- Контрольные сообщения (маркеры) также окрашиваются в один из двух цветов *white* или *black* .
- Маркер передается только пассивными процессами.
- Инициализатор контрольного алгоритима запускает маркер цвета *white* .
- После приема базового сообщения процесс окрашивается в цвет *black* .
- Процесс цвета *black* окрашивает маркер в цвет *black* .
- После передачи маркера процесс окрашивается в цвет *white* .
- Если инициализатор принимает маркер цвета *white* , то он объявляет о завершении базового алгоритма.

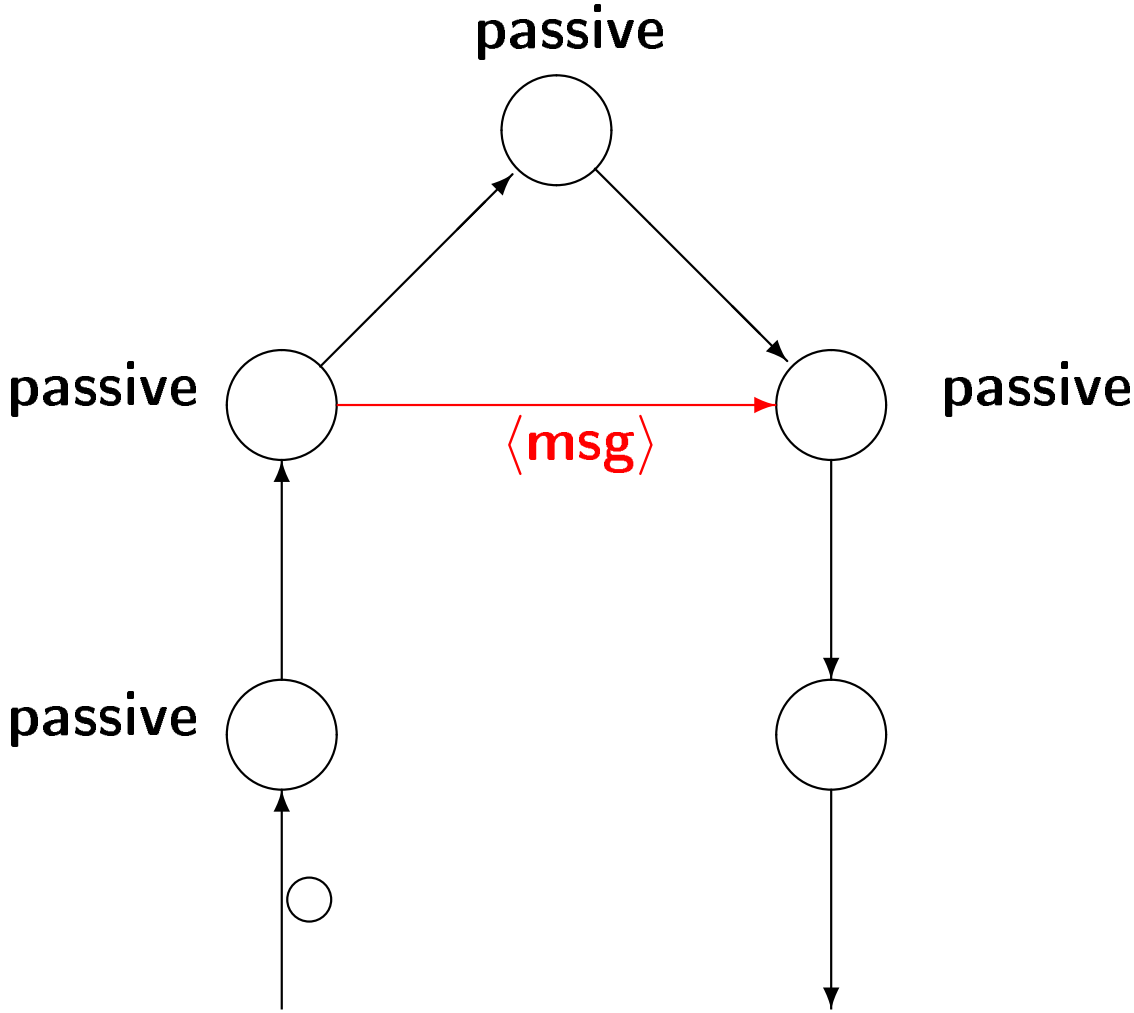
Алгоритм Сафры

Перечисленных принципов недостаточно для правильной работы алгоритма.



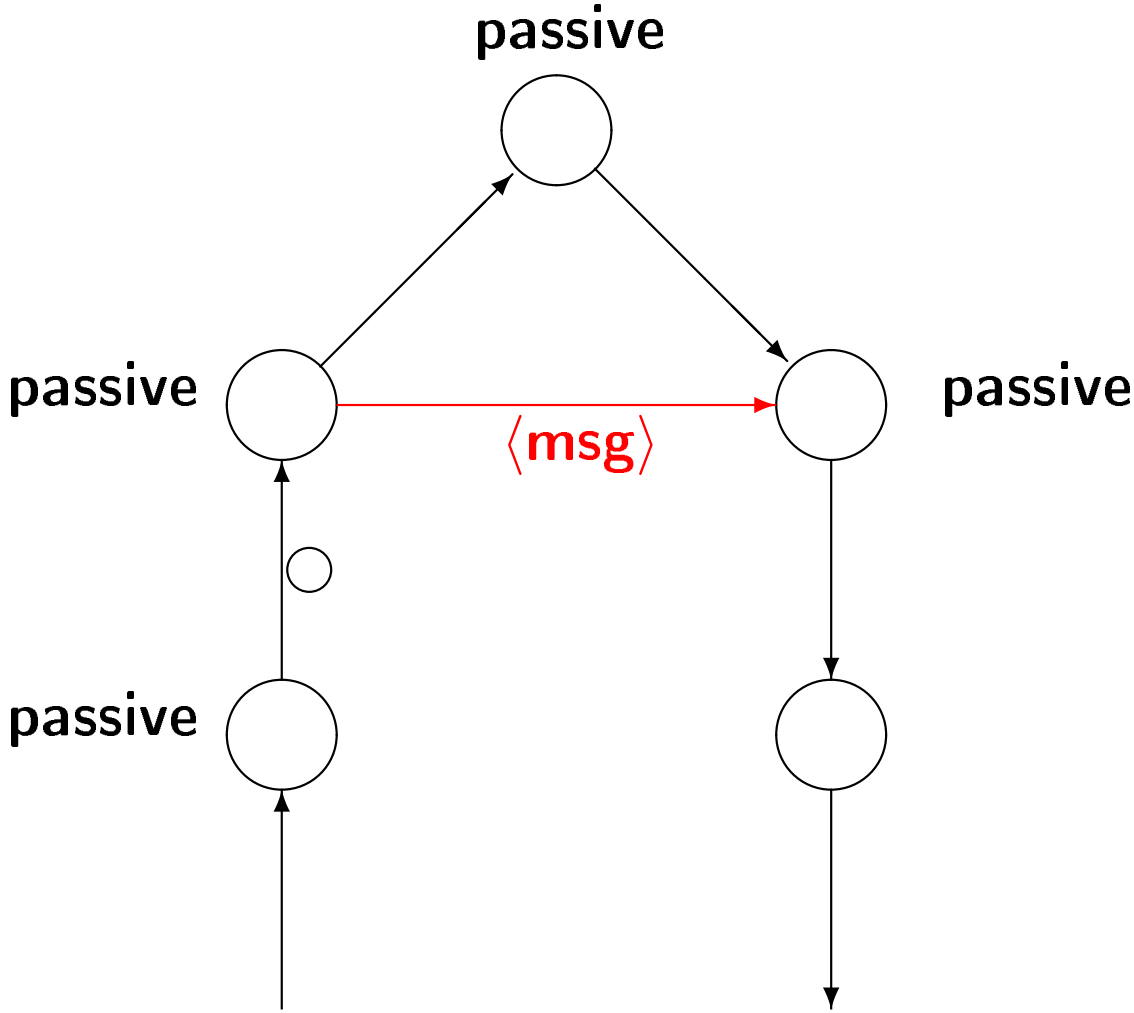
Алгоритм Сафры

Перечисленных принципов недостаточно для правильной работы алгоритма.



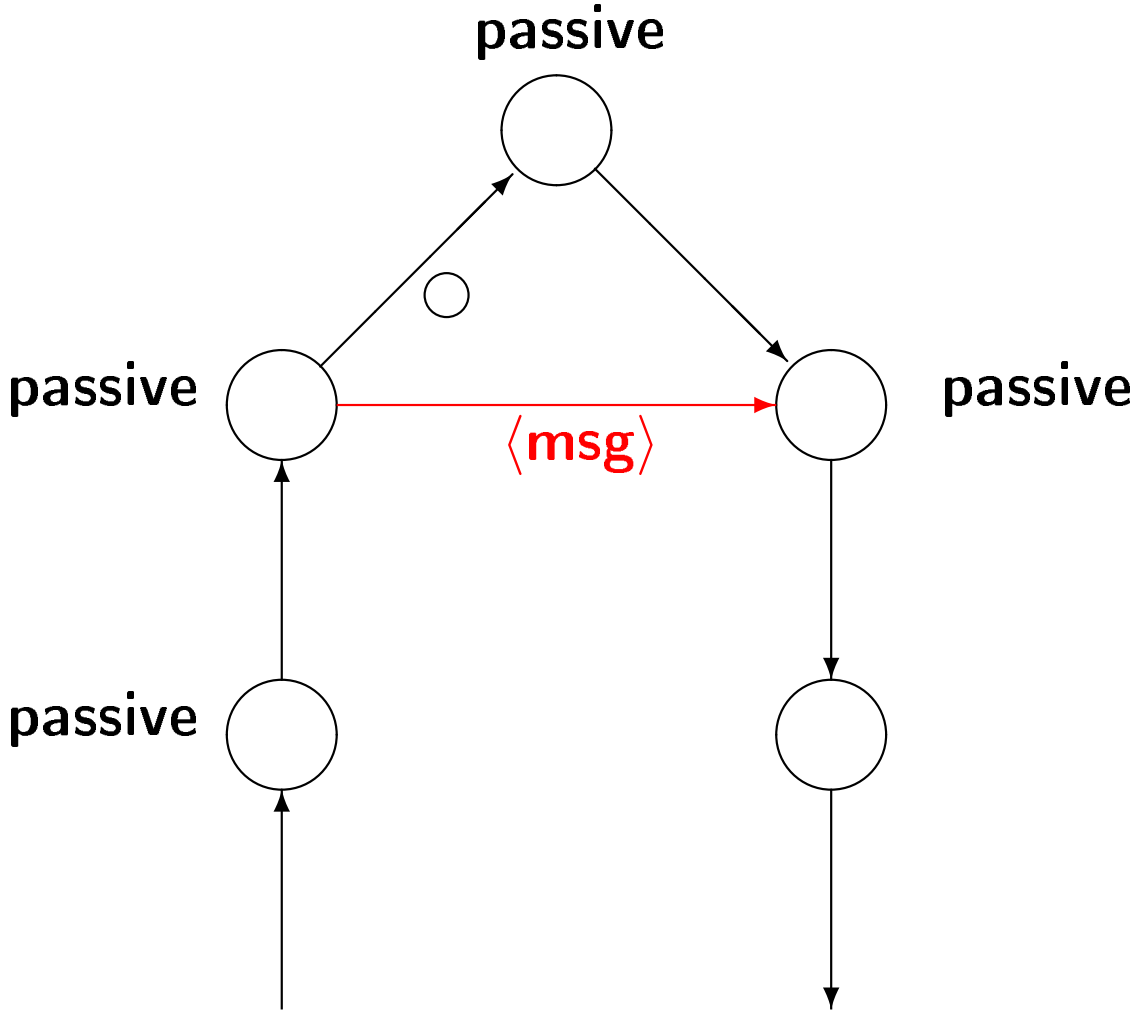
Алгоритм Сафры

Перечисленных принципов недостаточно для правильной работы алгоритма.



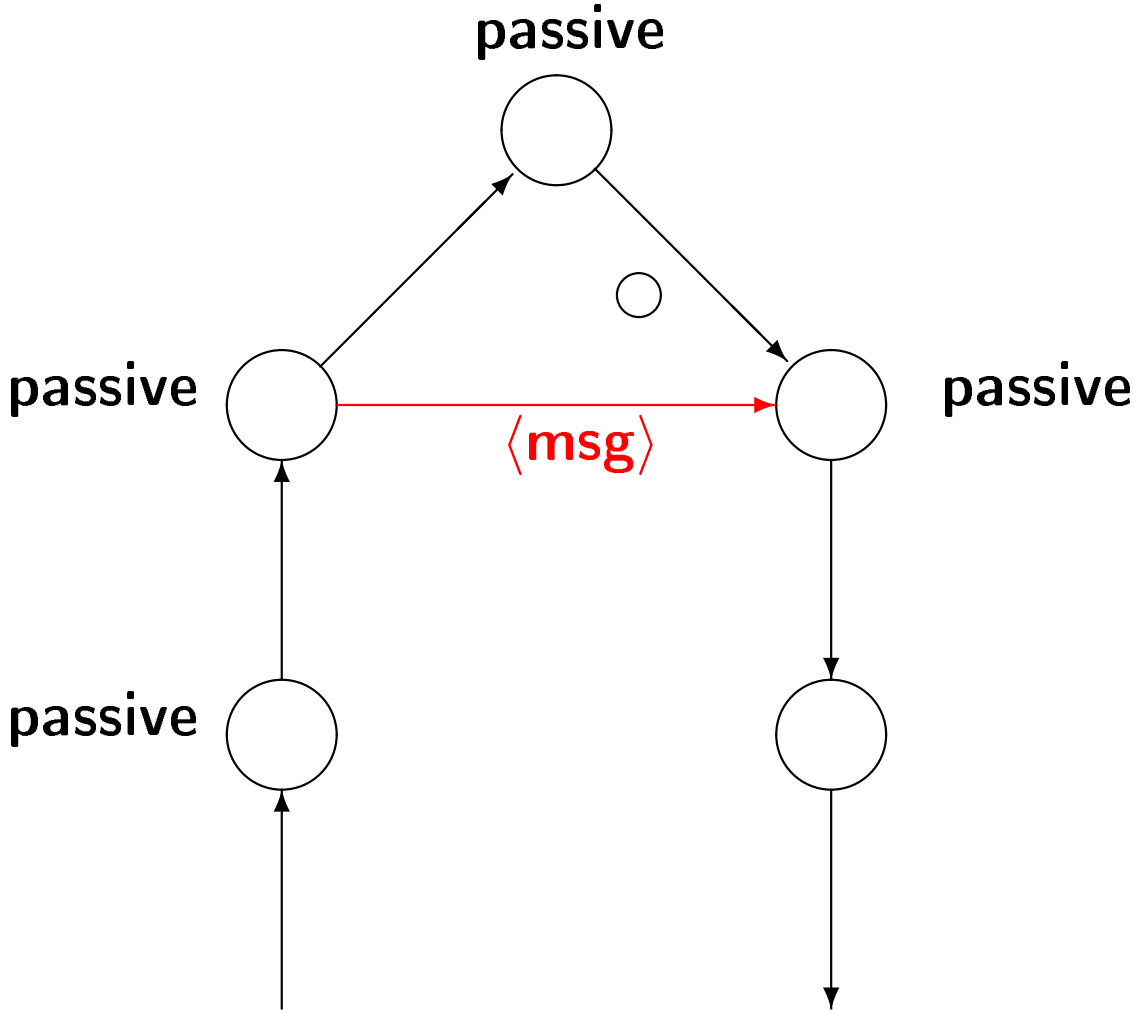
Алгоритм Сафры

Перечисленных принципов недостаточно для правильной работы алгоритма.



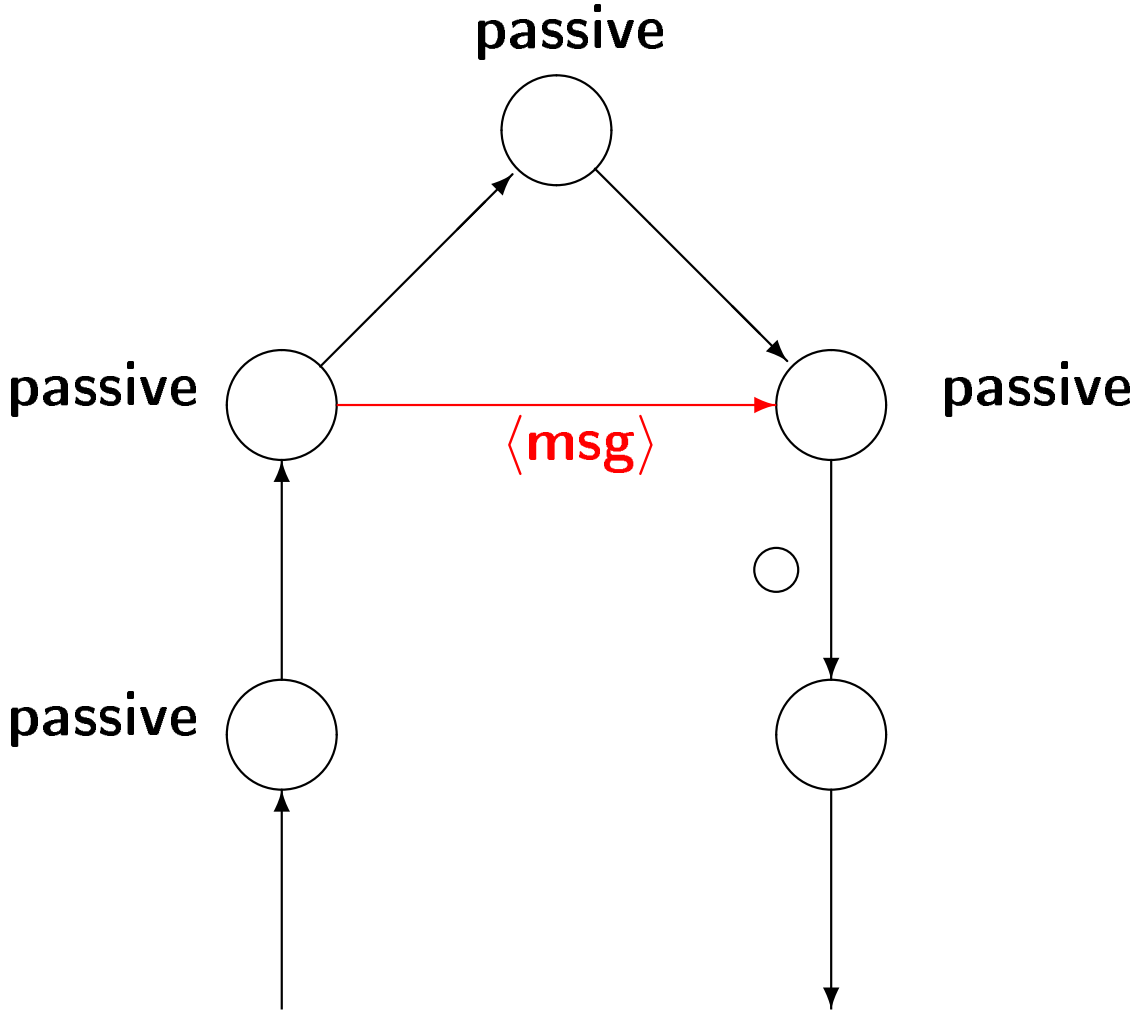
Алгоритм Сафры

Перечисленных принципов недостаточно для правильной работы алгоритма.



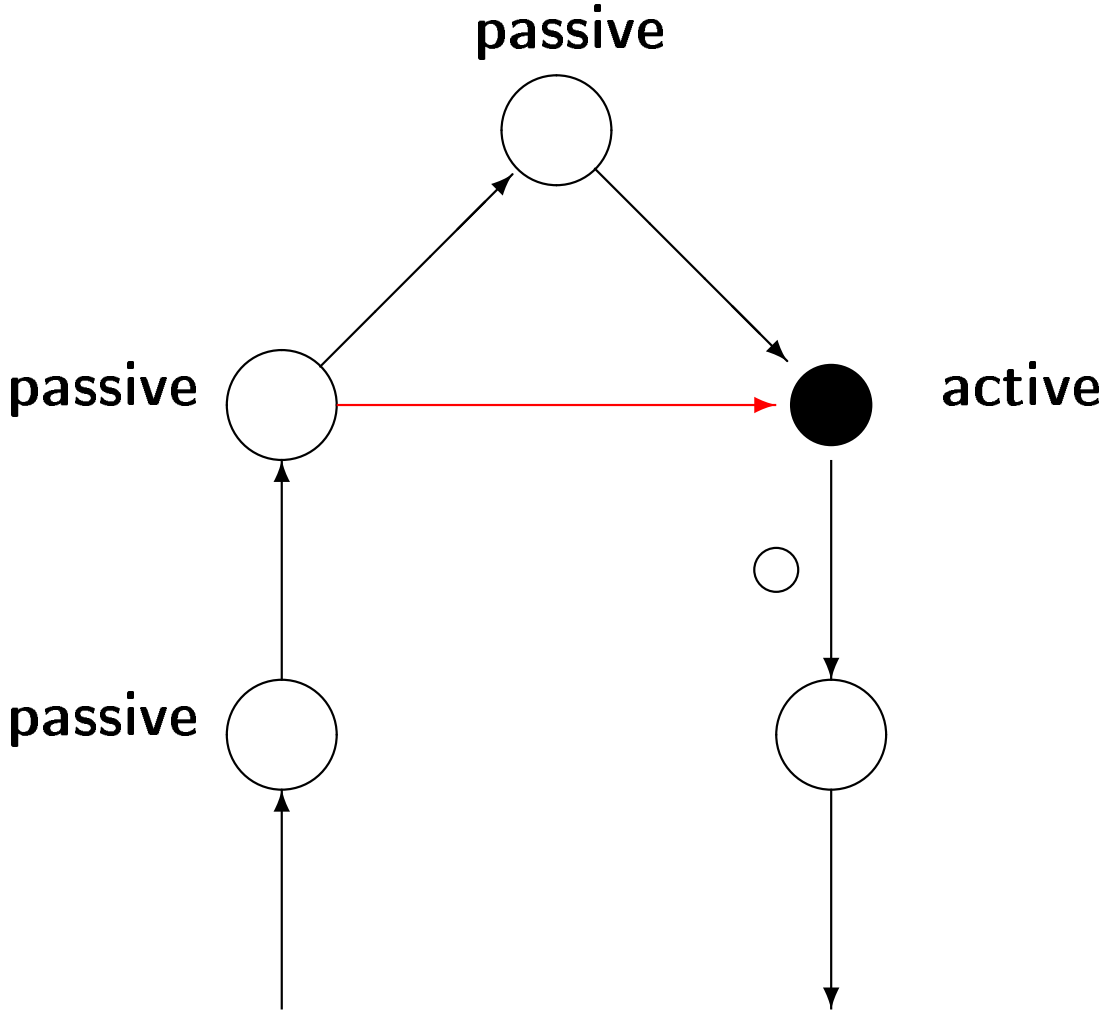
Алгоритм Сафры

Перечисленных принципов недостаточно для правильной работы алгоритма.



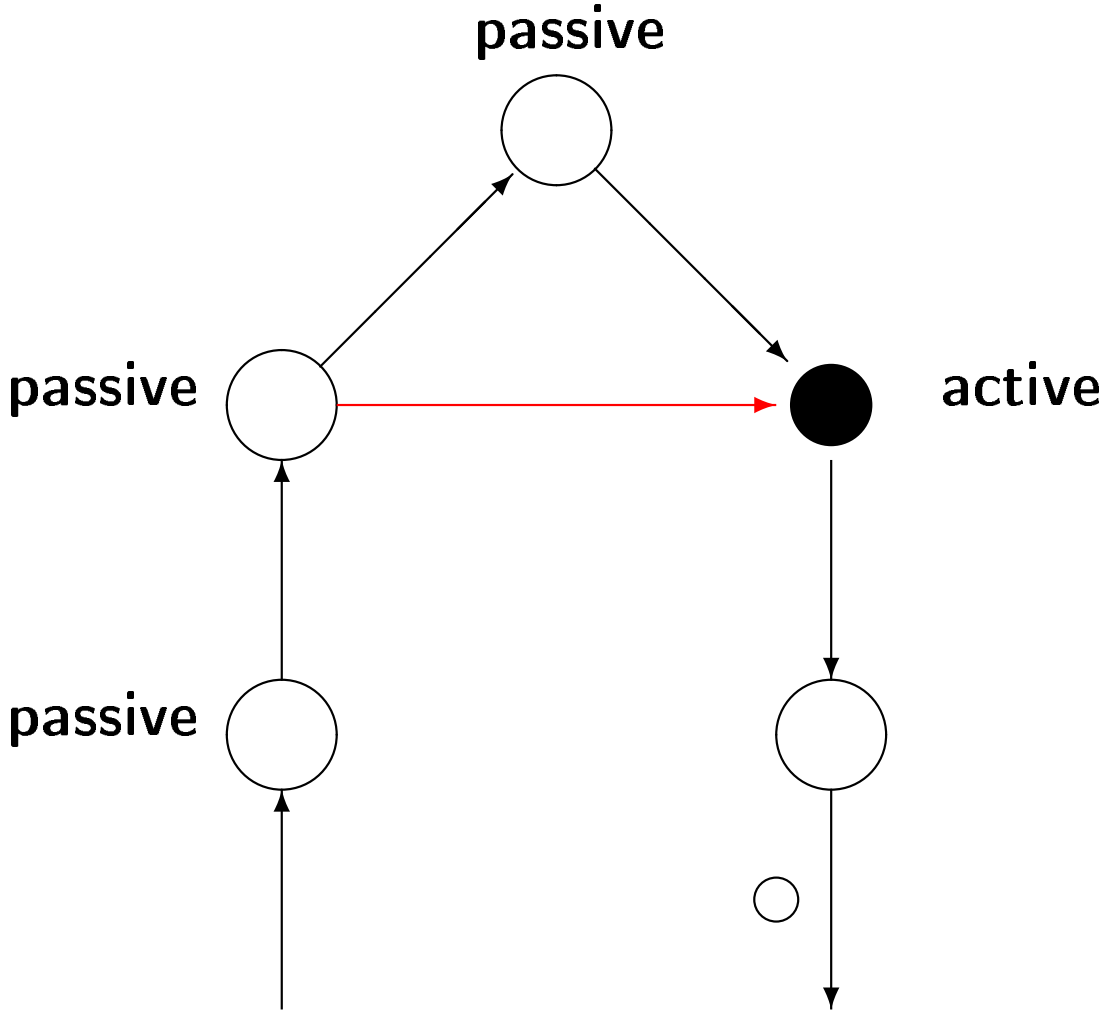
Алгоритм Сафры

Перечисленных принципов недостаточно для правильной работы алгоритма.



Алгоритм Сафры

Перечисленных принципов недостаточно для правильной работы алгоритма.

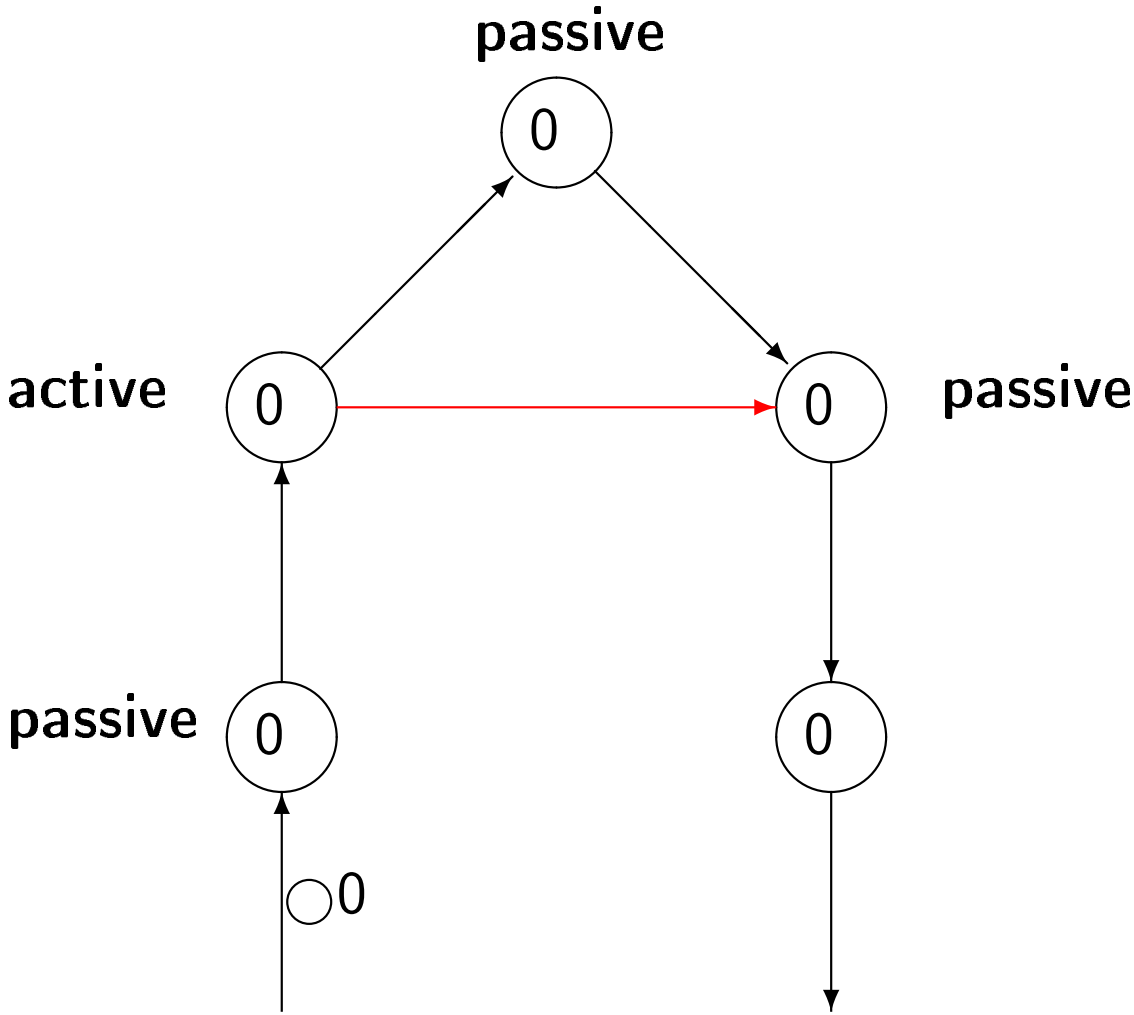


Алгоритм Сафры

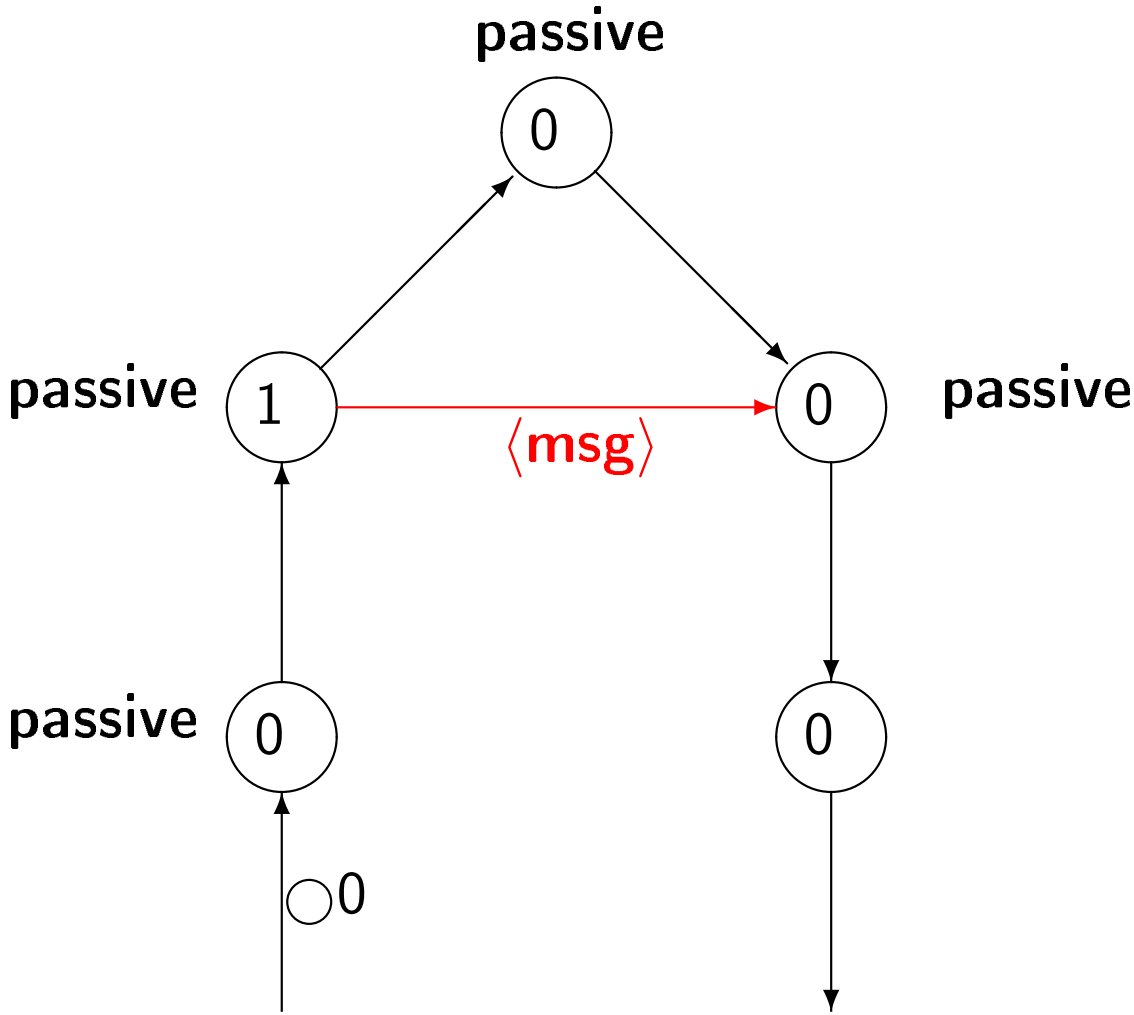
Еще три принципа работы.

- Каждый процесс p снабжен счетчиком базовых сообщений mc_p , который увеличивается на 1 при отправлении процессом базового сообщения и уменьшается на 1 при приеме базового сообщения.
- Маркер при прохождении через процессы суммирует показания их счетчиков.
- Завершение работы базового алгоритма регистрируется, если сумма собранных маркером показаний счетчиков равна 0.

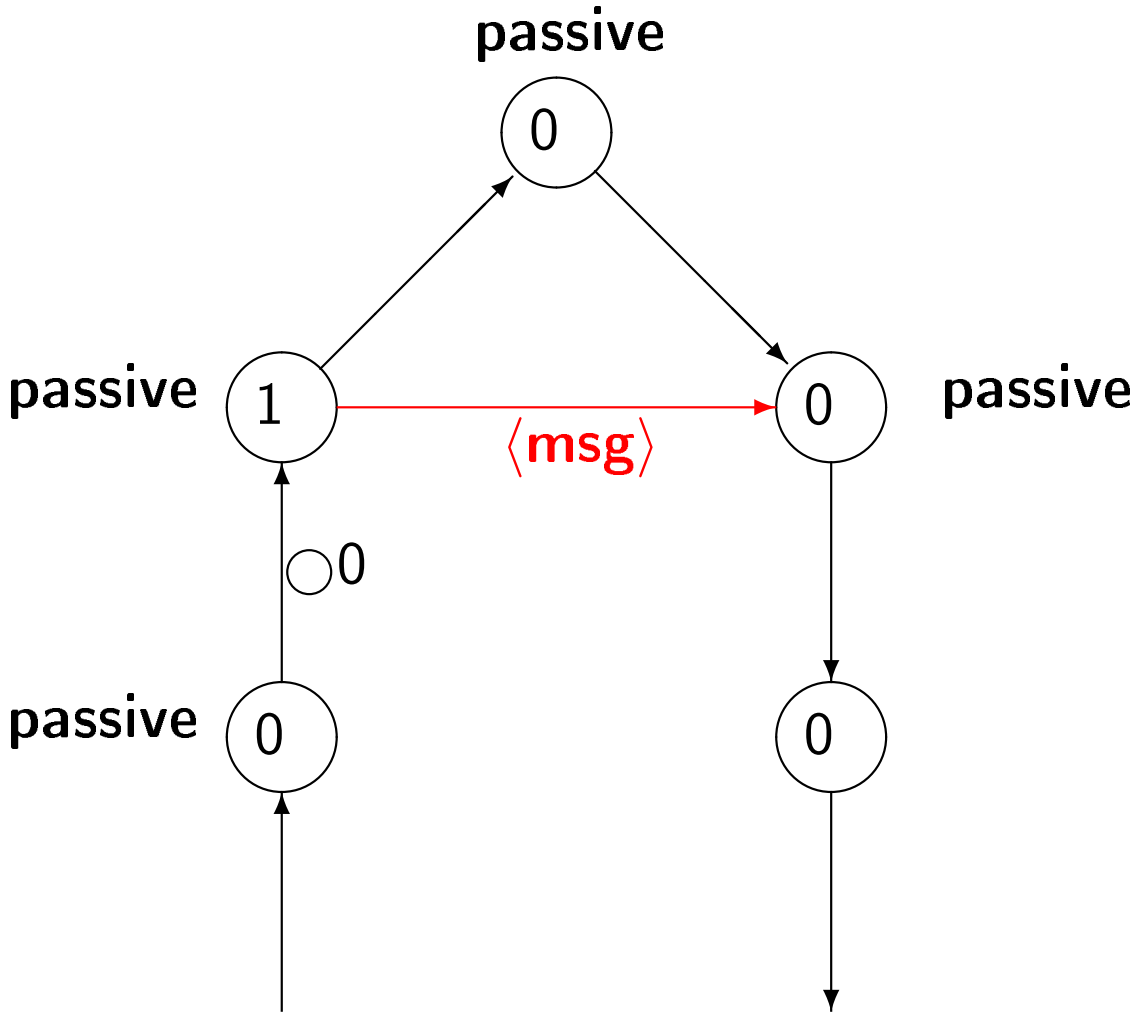
Алгоритм Сафры



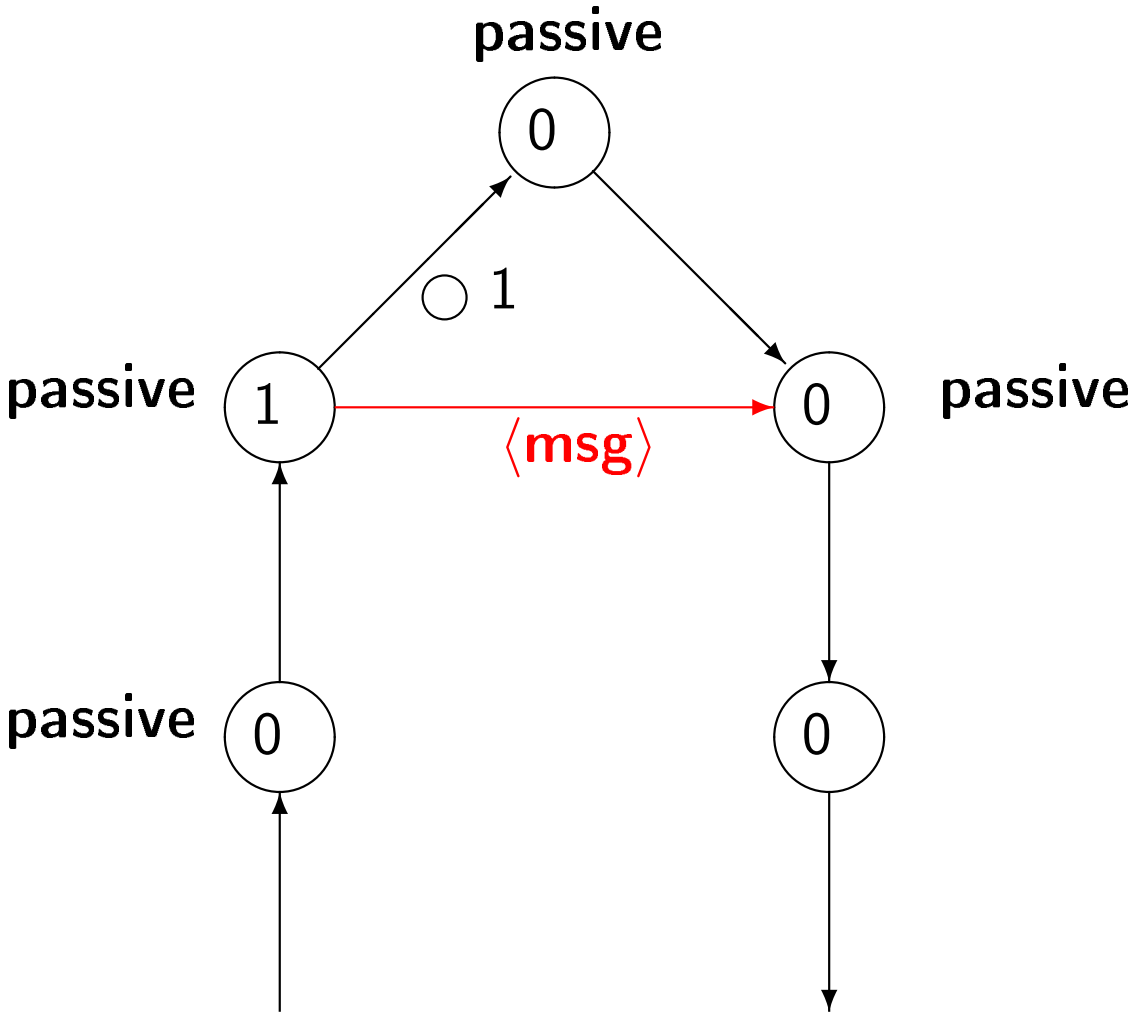
Алгоритм Сафры



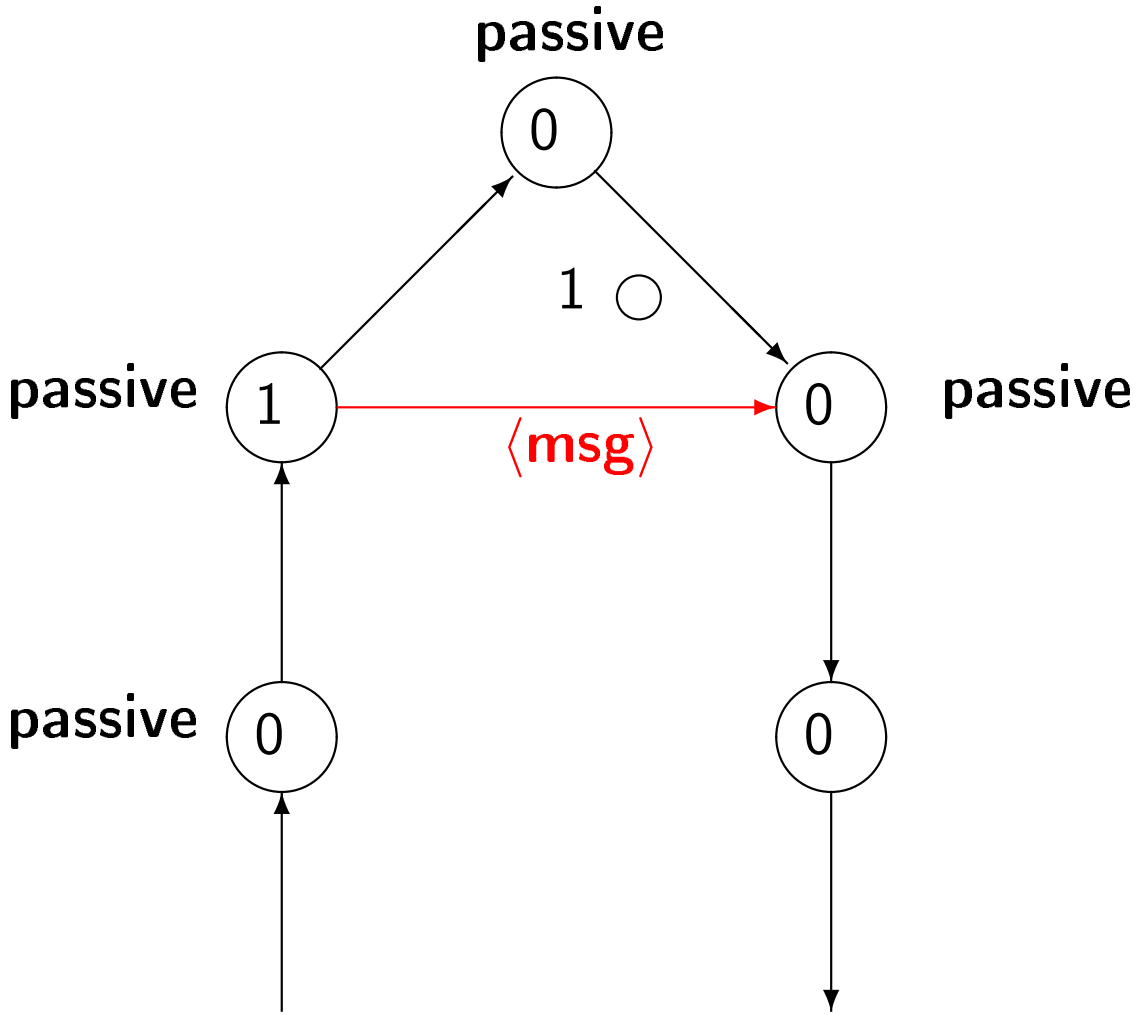
Алгоритм Сафры



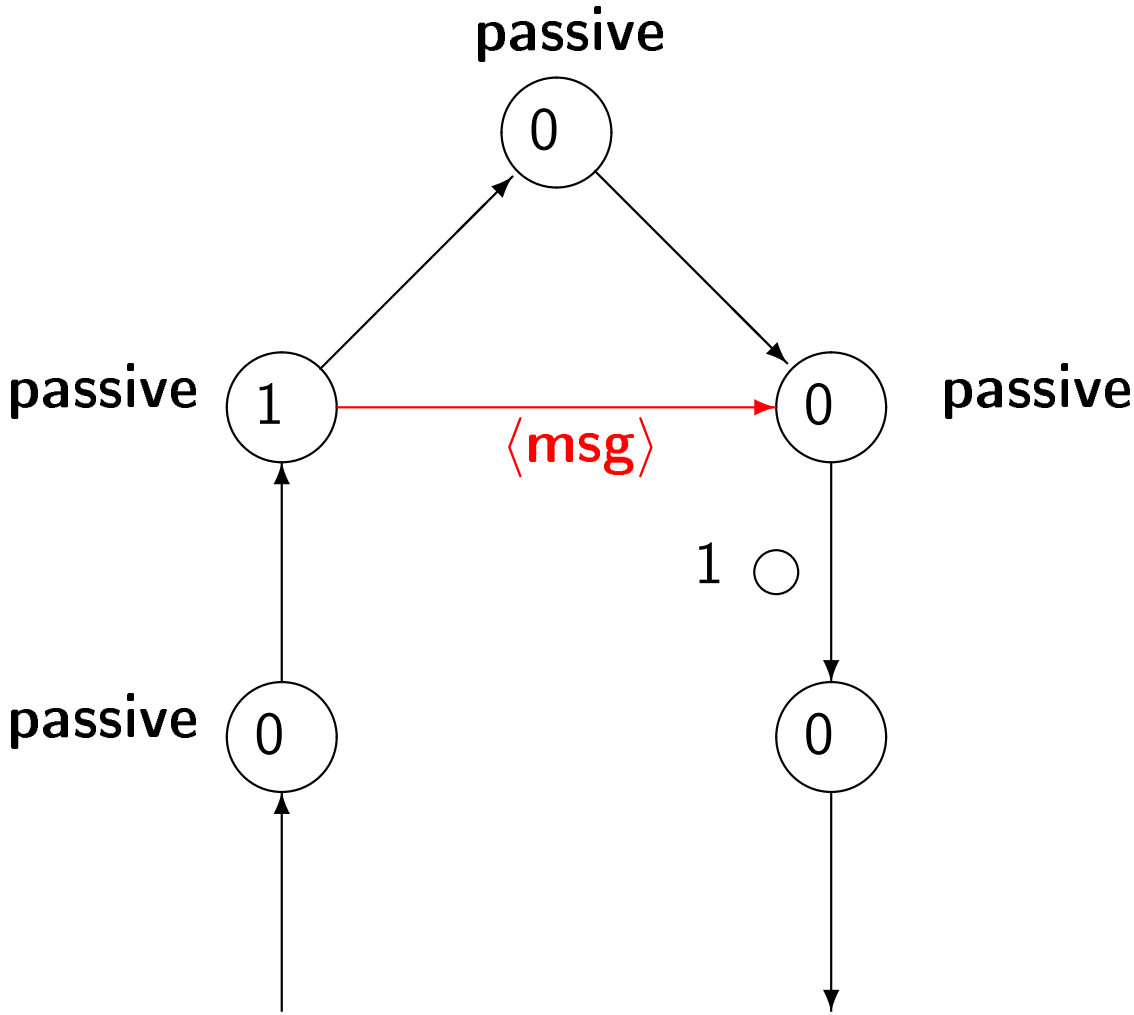
Алгоритм Сафры



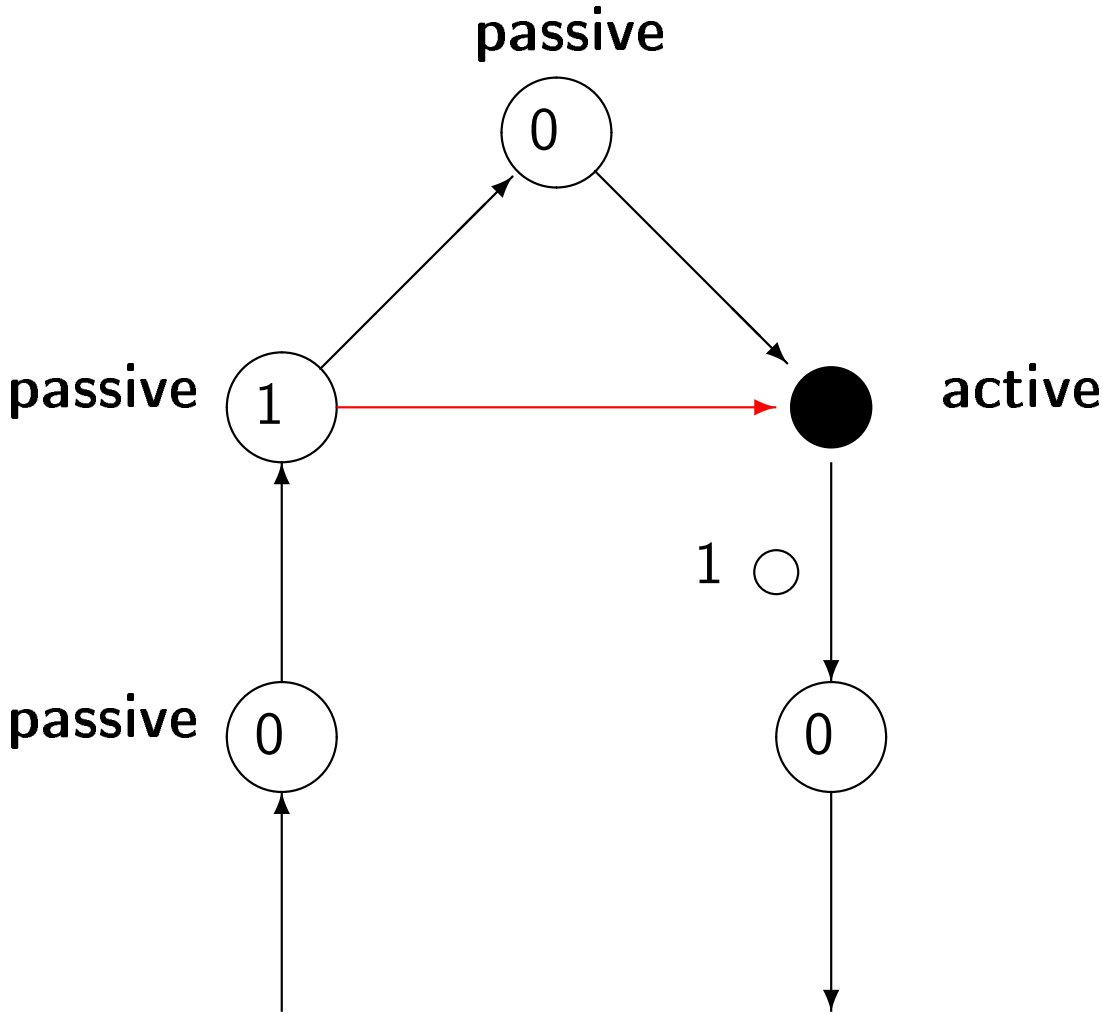
Алгоритм Сафры



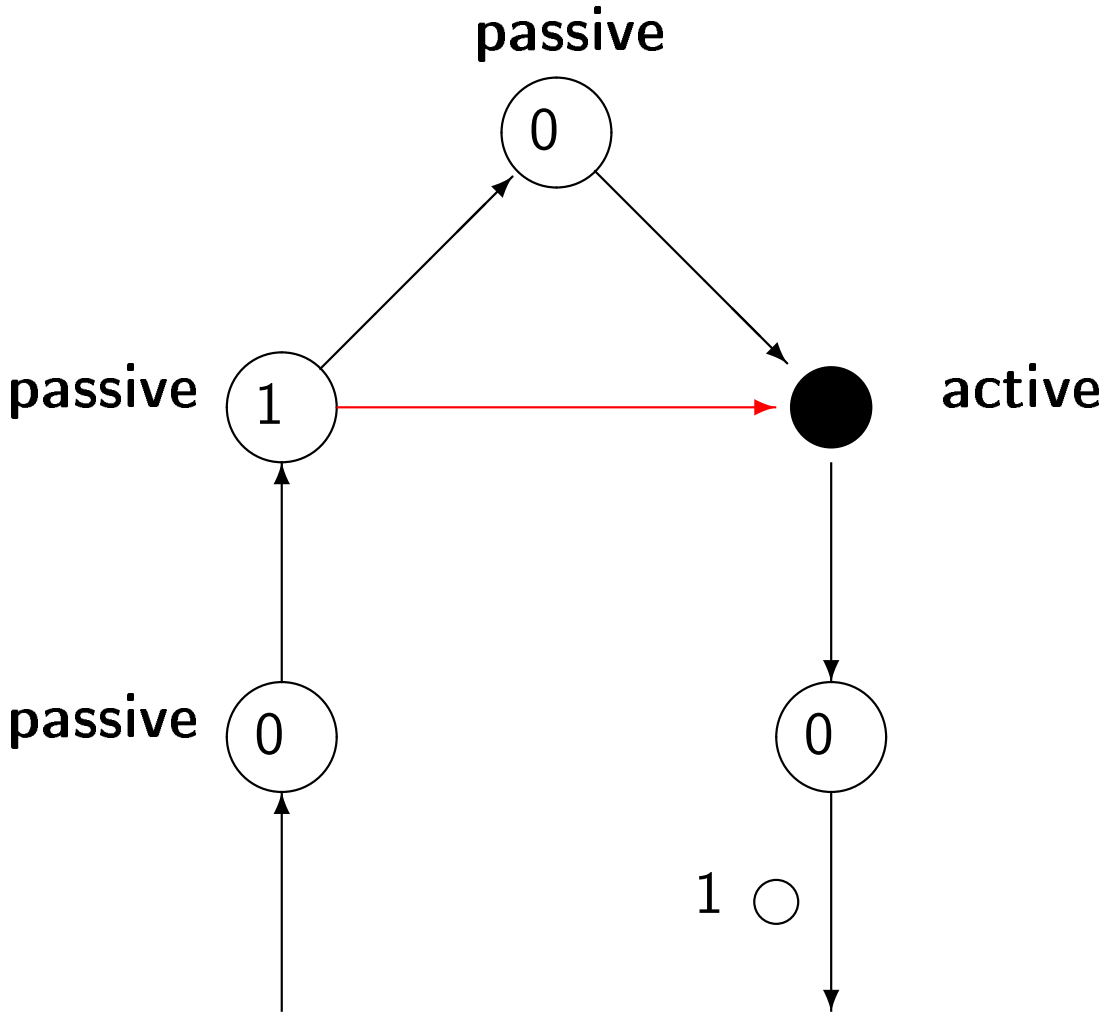
Алгоритм Сафры



Алгоритм Сафры



Алгоритм Сафры



Алгоритм Сафры

Итак, правила алгоритма.

Правило M: Когда процесс p отправляет сообщение, он увеличивает значение своего счетчика; когда процесс p получает сообщение, он уменьшает значение своего счетчика.

Правило 1: Процесс-получатель окрашивается в цвет *black* .

Правило 2: Только пассивный процесс может работать с маркером, и если процесс передает маркер, то он добавляет к пометке маркера q значение своего счетчика сообщений.

Правило 3: Когда процесс цвета *black* передает маркер, маркер приобретает окраску *black* .

Правило 4: Когда прохождение волны заканчивается неудачно, процесс p_0 запускает новую волну.

Правило 5: Сразу после передачи маркера всякий процесс окрашивается в цвет *white* .

Алгоритм Сафры

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

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

R$_p$: { Сообщение $\langle$**mes**$\rangle$ доставлено процессу p }
begin receive $\langle$**mes**$\rangle$; $state_p := active$;
$mc_p := mc_p - 1$; (* Правило M *)
$color_p := black$ (* Правило 1 *)
end

Алгоритм Сафры

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

Приступить к обнаружению завершения вычисления,
выполняется один раз процессом p_0:
begin send $\langle \mathbf{tok}, white, 0 \rangle$ to p_{N-1} **end**

$\mathbf{T}_p$: (* Процесс p обрабатывает маркер $\langle \mathbf{tok}, c, q \rangle$*)
{ $state_p = passive$ } (* Правило 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} (* Правило 4 *)
else if $(color_p = white)$ (* Правила 2 и 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 $Next_p$;
$color_p := white$ (* Правило 5 *)
end

Алгоритм Сафры

Для доказательства корректности введем следующие термы и предикаты:

- $B(\gamma)$ — число сообщений, находящихся на этапе пересылки в конфигурации γ .
- $t(\gamma)$ — номер процесса, которому направлен маркер или который владеет маркером в конфигурации γ .
  (**Замечание:** маркер переправляется процессам в порядке убывания номеров $p_0, p_{N-1}, p_{N-2}, \ldots, p_1, p_0, \ldots$).
- $q(\gamma)$ — число, которое хранит маркер в конфигурации γ .

Алгоритм Сафры

Для доказательства корректности введем следующие термы и предикаты:

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

Алгоритм Сафры

Лемма 3.
Предикат $P(\gamma)$ является инвариантом алгоритма Сафры.

Доказательство.

Самостоятельно.

Макарова, Шейдаев

Алгоритм Сафры

Задачи.

Какова сложность алгоритма Сафры по числу обменов сообщениями?

А можно ли адаптировать алгоритм Сафры для проверки завершения работы базового алгоритма в произвольной сильно связной сети?

Алгоритм Сафры

Теорема 3.

Алгоритм Сафры является корректным алгоритмом обнаружения завершения вычисления.

Доказательство.

Самостоятельно с использованием инварианта $P(\gamma)$.

Шпилевой

Алгоритм возвращения кредита

Маттерн предложил алгоритм, позволяющий обнаруживать завершение вычисления спустя не более одной единицы времени, после того как это случилось.

Алгоритм обнаруживает завершение централизованного вычисления при условии, что каждый процесс способен отправить сообщение непосредственно самому инициатору вычисления (т. е., сеть содержит подграф, имеющий форму звезды, в центре которой расположен инициатор).

Алгоритм возвращения кредита

В этом алгоритме каждому сообщению и каждому процессу вручается некоторый кредит ; его значением служит вещественное число, расположенное между 0 и 1 (включительно). Алгоритм стремится сохранить инвариантность следующих утверждений.

S1. Суммарный кредит, выданный сообщениям и процессам, равен 1.

S2. Всякому базовому сообщению вручается положительный кредит.

S3. Всякому активному процессу вручается положительный кредит.

Процессы, имеющие положительный кредит и не подпадающие под перечисленные законы (т. е. пассивные процессы), возвращают кредит инициатору. Инициатор поступает подобно банку и собирает все кредиты, которые ему отправляют, в переменной *ret* (возвращенные кредиты).

Когда инициатор владеет всеми кредитами, то он сообщает о завершении базового алгоритма.

Алгоритм возвращения кредита

Правило 1. Если $ret = 1$, то инициатор вызывает процедуру *Announce*.

Правило 2. Как только процесс становится пассивным, он отправляет свой кредит инициатору.

Правило 3. Когда активный процесс p отправляет сообщение, имеющийся в его распоряжении кредит разделяется между самим процессом p и этим сообщением.

Правило 4. Когда процесс активизируется, ему вручается кредит, которым обладало активизировавшее его сообщение.

Правило 5. Как только активный процесс получает базовое сообщение, кредит, которым обладало это сообщение, добавляется к тому кредиту, которым располагал сам процесс.

Алгоритм возвращения кредита

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

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

R$_p$: { сообщение $\langle$**mes**, $c\rangle$ поступило процессу p }
begin receive $\langle$**mes**, $c\rangle$; $state_p := active$;
$cred_p := cred_p + c$ (* Правила 4 и 5 *)
end

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

A$_{p_0}$: { Сообщение $\langle$**ret**, $c\rangle$ поступило процессу p_0 }
begin receive $\langle$**ret**, $c\rangle$; $ret := ret + c$;
if $ret = 1$ **then** *Announce* (* Правило 1 *)
end

Алгоритм возвращения кредита

Теорема 4.

Алгоритм возвращения кредита является корректным алгоритмом обнаружения завершения вычислений.

Доказательство.

Самостоятельно сформируйте нужный инвариант и проведите доказательство.

Чупахин

Алгоритм Раны

Для решения задачи обнаружения завершения вычисления можно использовать штемпели времени. Предполагается, что все процессы снабжены часами; для этой цели пригодны как аппаратные таймеры, так и логические часы Лампорта.
В основу алгоритма положен локальный предикат $quiet(p)$, определенный для каждого процесса p так, что

$$quiet(p) \implies state_p = passive \wedge$$
$$\wedge \text{ никакое базовое сообщение, отправленное } p$$
$$\text{не пребывает на этапе пересылки,}$$

откуда следует импликация $(\forall p\ quiet(p)) \implies \mathbf{term}$.
Условие $quiet$ определяется следующим соотношением

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

Алгоритм Раны

Цель алгоритма — проверить, верно ли, что в определенный момент времени t все процессы удовлетворяют условию *quiet* .

Это осуществляется при помощи волны, по ходу которой каждому процессу задается вопрос о том, достиг ли этот процесс в данный момент состояния *quiet* и будет ли он оставаться в этом состоянии в дальнейшем. Всякий процесс, не пребывающий в состоянии *quiet* , не реагирует на сообщения волны, тем самым эффективно гася саму волну. В качестве волнового алгоритма можно использовать кольцевой алгоритм.

Взаимодействие волны с процессом p не оказывает влияния на переменные процесса p , используемые для обнаружения завершения вычисления. Вследствие этого правильность работы алгоритма не нарушается при параллельном распространении нескольких волн.

Алгоритм Раны

Процесс p , перейдя в состояние *quiet* , сохраняет в памяти тот момент времени qt_p , когда произошло это событие, и запускает волну, чтобы проверить, все ли процессы уже перешли в состояние *quiet* к моменту времени qt_p .

Если это так, то завершение вычисления считается обнаруженным. В противном случае существует процесс, который перейдет в состояние *quiet* позже, и он запустит новую волну.

Алгоритм Раны

var $state_p$: ($active$, $passive$) ;
θ_p : **integer init** 0 ; (* Логические часы *)
$unack_p$: **integer init** 0 ; (* Число неподтвержденных сообщений
qt_p : **integer init** 0 ; (* Время недавно случившегося переход

S$_p$: { $state_p = active$ }
begin $\theta_p := \theta_p + 1$; send $\langle$**mes**, $\theta_p\rangle$; $unack_p := unack_p + 1$ **end**
R$_p$: { Сообщение $\langle$**mes**, $\theta\rangle$ из q достигло 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 переходит в состояние quite *)
begin $qt_p := \theta_p$; send $\langle$**tok**, $\theta_p, qt_p, p\rangle$ to $Next_p$ **end**
end

Алгоритм Раны (продолжение)

$\mathbf{A}_p$: { Подтверждение $\langle \mathbf{ack}, \theta \rangle$ достигло 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 переходит в *qui*
begin $qt_p := \theta_p$; send $\langle \mathbf{tok}, \theta_p, qt_p, p \rangle$ to $Next_p$ **end**
end

$\mathbf{T}_p$: { Маркер $\langle \mathbf{tok}, \theta, qt, q \rangle$ достиг 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

Алгоритм Раны

Теорема 5.

Алгоритм Раны является корректным алгоритмом обнаружения завершения вычислений.

Доказательство.

Самостоятельно сформируйте нужный инвариант и проведите доказательство.

Михеев, Шер

КОНЕЦ ЛЕКЦИИ 11.