Алгоритм Раны.

**Коллегия**:

*Багров Никита*

**Василенко Анатолий**

Хахалин Анатолий

**Задача**: сформулировать инвариант, обосновать его и доказать с его помощью теорему:

*Теорема*: Алгоритм Раны является корректным алгоритмом обнаружения завершения вычислений

**Алгоритм оповещения о завершении**:

Так же есть ещё алгоритм Announce - который после запуска одним из узлов рассылает всем броадкастом сообщения stop, и после того как каждый получает stop от всех соседей, то он halt.

**Предикат term**:

term(γ) - истина на всякой конфигурации, на которой не может произойти ни одного события в базовом вычислении

(согласно теореме 1):

term(γ) ⇔ (∀p∈**P**: statep = passive) and (∀pq∈**E**: Mpq не содержит сообщений <mes>)

**Алгоритм Раны в 2-х словах**:

У каждого процесса есть часы (аппаратный Таймер или часики Лемпорта), если для процесса выполнен предикат quiet(p) ⇔ (statep == passive) and (unackp == 0) - т.е. процесс в пассиве и все посланные сообщения им уже доставлены, то этот процесс запускает волну (записав в неё своё время), которую поддержат все процессы, которые перешли в quiet не позже, чем время волны, иначе они гасят волну, и потом когда-нибудь запустят её сами.

Строгое описание алгоритма может быть найдено на слайдах лекции, оно состоит из 5-ти процедур:

1. S - (~~активный процесс становится пассивным только после осуществления внутреннего события~~) - посылка сообщения базового алгоритма (процесс в активном состоянии)
	1. θp = θp +1 увеличиваем счётчик времени (часики)
	2. **send** <**mes**, θp> шлём сообщения основного алгоритма (а также своё время)
	3. unackp = unackp + 1 +1 не подтверждённое сообщение
2. R - (~~процесс всегда становится активным после получения сообщения~~) - мы p получили сообщение от q
	1. **receive** <**mes**, θ> получили сообщение
	2. θp = max(θ, θp) + 1 пропатчили время часиков
	3. **send** <**ack**, θp> to q послали ack и время из часиков
	4. statep = active перешли в активное состояние
3. I - (~~внутренние события, приводящие к тому, что процесс становится пассивным - единственно возможные внутренние события в процессе p~~) - внутреннее состояние меняется на пассивное, в следствие того, что базовый алгоритм отработал что хотел и теперь лишь ждёт
	1. θp = θp + 1 увеличиваем счётчик времени (часики)
	2. statep = passive перешли в пассивное состояние
	3. if (unackp == 0) если мы больше не ждём ack, то:
		1. qtp = θp **send**(**tok**, θp, qtp, p) to Nextp запоминаем время, когда мы перешли в состояние quiet, и стартуем волну кольцевого алгоритма (*согласно слайдам*) за счёт посылки токена следующему по кольцу.
4. A - мы получили подтверждение на посланный пакет
	1. **receive** <**ack**, θ>
	2. θp = max(θ, θp) + 1 пропатчили время часиков
	3. unackp = unackp - 1 -1 не подтверждённое сообщение
	4. if (unackp == 0 and statep == passive) если мы больше не ждём ack и passive, то:
		1. qtp = θp **send**(**tok**, θp, qtp, p) to Nextp запоминаем время, когда мы перешли в состояние quiet, и стартуем волну кольцевого алгоритма (*согласно слайдам*) за счёт посылки токена следующему по кольцу.
5. T - к нам пришёл токен волны
	1. **receive** <**tok**, θ, qt, q>
	2. θp = max(θ, θp) + 1 - пропатчили время часиков
	3. if (quiet (p)) - если мы в состоянии quiet, то
		1. if (p == q) - т.е. если мы были инициатором волны, т.е. волна закончилась
		2. тогда Announce - запуск алгоритма оповещения о завершении
		3. иначе if (qt > qtp) - т.е. если та волна началась позже нашей волны, то
			1. **send**(**tok**, θp, qtp, p) to Nextp - то поддерживаем волну кольцевого алгоритма (*согласно слайдам*) за счёт посылки токена следующему по кольцу.

**Свойства ∀ волнового алгоритма**:

1. Вычисление конечно (т.е. алгоритм завершается через конечное число шагов (если все вершины поддерживают его))
2. Каждое событие содержит хотя бы одно событие решения
3. В любом вычислении всякому событию решения предшествует в причинно-следственном отношении хотя бы одно событие в каждом из процессов.

Корректный алгоритм обнаружения завершения вычисления, это алгоритм, который в конце имеет все процессы переведённые в заключительные состояния (**(A)** - придётся доказывать), при этом, алгоритм должен удовлетворять следующим пунктам:

1. (Невмешательство) Алгоритм обнаружения завершения вычисления не должен оказывать влияния на вычисления базового алгоритма.

**(+)** - выполнено, т.к. по условию алгоритм ни в одной из своих процедур не влияет на переменные и сообщения базового алгоритма.

Может быть лишь воздействие на саму волну, либо на часики.

1. (Живость) Если выполняется условие term, то алгоритм оповещения Announce должен быть вызван спустя конечное число шагов

**(B)** - придётся доказывать

1. (Безопасность) Если вызван алгоритм Announce, то конфигурация должна удовлетворять условию term.

**(C)** - придётся доказывать

**О всемогущий предикат**:

Pred = (1) (∀p∈**P**: unackp >= 0) and

 (2) (глобальная сумма всех unackp = глобальному количеству сообщений ack и mes в каналах) and

 (3) (∀t >= T (где T - ∀ момент когда выполнялся предикат term, и ни в одном из процессов процедура S, I, R, A, T не находится в недовыполненном состоянии): выполнен term) and

 (4) (∀**receive** <**tok**, θ, qt, q>: θ >= qt)

**Доказательство предиката**:

(1) - значение unack меняется только в процедуре S и A. Изначально unackp = 0. Чтобы доказать, что unack >= 0, достаточно показать, что процедура А выполнится не большее количество раз, чем S. Предположим обратное, т.е. процедура A в какой-то момент выполнилась на 1 раз больше, чем S, это значит, что A приняла n+1 сообщение ack, это значит, что эти сообщения были посланы, причём соседями, однако посылка сообщения ack происходит только в процедуре R, после получения сообщения mes, причём ack посылается по каналу в обратную сторону от сообщения mes, и раз было послано n+1 сообщение ack процессу p, это значит, что каждому сообщению ack соответствовало посланное сообщение mes от процесса p, однако сообщения mes посылаются только в процедуре S, т.е. процедура S выполнилась n+1 раз, а по предположению, она выполнилась на один раз меньше - n. Противоречие => наше предположение ложно => S всегда выполнится не меньшее число раз, что и A => unack >= 0 для любого p.

(2) - Доказательство по индукции.

Изначально, ∀p∈**P**: unackp = 0, а количество сообщений в каналах тоже = 0, т.е. предикат верен.

Допустим перед выполнением рассматриваемой процедуры предикат верен, докажем, что он будет верен и после её выполнения.

S - добавляет в канал mes (1.2), но увеличивает unack на +1 (1.3), предикат остаётся верен верен (остальные команды в процедуре не влияют на предикат)

R - вынимает одно сообщение из канала (2.1) и посылает другое сообщение в канал (2.3), остальные команды не меняют предикат, предикат остаётся верен,

I, T - не принимает и не посылает сообщений mes и ack и не меняет значение unack, предикат не меняется.

A - вынимает сообщение из канала (4.1) и уменьшает unack на -1 (4.3), остальные директивы не влияют на предикат => предикат остаётся верен.

Итак, по индукции следует, что предикат всегда верен после выполнения любой процедуры.

(3) - Докажем по индукции, что если term выполняется, то после выполнения любой из процедур в любом из процессов, term так же будет выполняться.

Базис индукции: Согласно условию в некоторый момент T term выполнен, и никакие процедуры не находятся на стадии выполнения.

Шаг индукции: term выполнен до начала выполнения какой-либо процедуры

S, R - выполнится не может, т.к. это посылка/принятие mes, а по условию term - не может произойти шага базового алгоритма.

I - не шлёт никаких mes в канал и statep = passive => согласно теореме 1, term выполняется после завершения процедуры (естественно для других процессов p всё и так выполнялось, согласно предположению индукции)

A, T - не шлёт никаких mes в канал и не меняет state, т.е. не влияет на условие term (согласно теореме 1) => term выполняется

Итак, согласно индукции если в какой-то момент выполнен term, то выполнение любой функции после этого не нарушает его, а значит выполнено (3).

(4) - **receive** <**tok**, θ, qt, q> - это некоторый **send** <**tok**, θ, qt, q>, т.е. нам нужно показать, что в любом send tok θ >= qt.

Докажем по индукции по посылаемым сообщениям.

Базис индукции: изначально в каналах нету send tok => (4) верно

Шаг индукции: предположим, что (4) истина до выполнения каждой из процедур, докажем, что истина и после них.

Процедуры S, R - не изменяют ничего

Процедуры I, A - 3.3.1 и 4.4.1 - θ == qt (присваивается в той же сточке), т.е. предикат верен для посылаемого send tok

Процедура T - 5.3.3.1 send tok выполнится только если выполнено условие qt > qtp, согласно предположению индукции θ >= qt и согласно 5.2 θp >= θ, т.е.

θp >= θ >= qt > qtp, значит (4) верен после выполнения процедуры T

Итак, из индукции следует, что (4) выполнено всегда

**Доказательство корректности алгоритма обнаружения завершения вычисления**:

**(B)**

“Если выполнено условие term” - имеется в виду, что в данный момент времени глобально все процессы удовлетворяют условию term, и ни один из них не находится в состоянии выполнения S, I, R, A или T.

“Если выполняется условие term” - т.е. согласно теореме 1, каждый процесс находится в состоянии passive и ни один из каналов не содержит сообщений.

∀p∈**P**: statep = passive

∀pq∈**E**: Mpq не содержит сообщений <mes>

Если выполнено term, то в этом случае, процедуры S, I - не могут произойти, т.к. процессы passive (I может только лишь “завершиться”, спровоцировав волну). R не может произойти, так как все каналы пусты на mes, и нет, сообщения mes, которое можно было бы принять, и согласно определению term, R уже не пошлёт нового сообщения mes (т.к. не может быть события в базовом алгоритме)

Могут случиться A или T.

Согласно (2) из предиката, с поправкой на отсутствие mes в каналах <глобальная сумма всех unackp = количество сообщений ack в каналах>, а значит ack будут досланы (они уже в каналах) (а это конечное число шагов, которые обязательно случатся) и выполнится <глобальная сумма всех unackp> процедур A, причём в результате во всех процессах будет unackp = 0, т.к. если где-то unackp > 0, то в другом процессе unackp будет < 0 (когда все ack дойдут) (ибо их общее количество совпадает), что противоречит (1), и в этом случае хотя бы раз выполнится 4.4 и 4.4.1 и процесс перейдёт в состояние quiet.

Таким образом, если глобально кол-во ack > 0, то они обязательно будут досланы, и выполнится A (конечное число раз), причём хотя бы раз в ней выполнится if (unack == 0 and state == passive) (про unack параграф выше, что все они станут когда-то == 0, про passive (3)) => произойдёт переход в состояние quiet, причём их будет конечное число > 0.

Если ack == 0 (а согласно (2) из предиката => unack == 0), и сейчас выполнен term, то ранее был момент когда до выполнения некоторой функции term был ложь, а после её выполнения стал истиной (ибо изначально когда базовый алгоритм только начинается есть хотя бы один active процесс). Это мог быть только момент 3.2 (только в этот момент одно из его условий теоремы 1 меняется с false на true (или в момент, когда условие, что <нету mes в каналах> меняется с false на true - это происходит при принятии сообщения, но процедура R не может выполняться, т.к. согласно определению term - не может произойти ни одного события в базовом вычислении, а принятие сообщения - это базовый алгоритм)). Значит был момент 3.2, когда последний процесс перешёл в состояние passive, а каналы были пусты от mes, и в этот момент либо глобальный unack > 0, и тогда по выше приведённым рассуждениям мы можем выбрать с того момента последний переход в состояние quiet, либо глобальный unack == 0, и тогда у процесса. у которого случилось 3.2 тоже unack == 0, и тогда выполнится 3.3, а значит произойдёт переход в состояние quiet.

Итак, если выполнен term, то у нас есть “последний по времени (т.е. переход quiet с максимально записанным счётчиком времени (когда говорится о времени всегда имеется ввиду именно это)) переход в состояние quiet”, произошедший в процессе s.

Перейдя в состояние quiet в 3.3.1 запускается tok, при этом, т.к. у этого tok максимальная метка времени, то в любой процесс p, до которого этот tok дойдёт, 5.3.3 будет истина и 5.3.3.1 всегда выполнится. (5.3 тоже всегда выполнится, т.к. state == passive согласно (3), а unack == 0, т.к. иначе - 5.3 не выполнится, однако время Лемперта θp обновится строкой 5.2, причём θp >= θ (из tok) >= qt (из tok) = qts (4), а согласно доказанному выше обязательно случится unackp == 0 и случится qtp = θp > qts, но мы же выбирали максимальный момент перехода в quiet - это противоречие, а значит unack == 0)

При этом tok от любой другой волны в процессе s будет в пункте 5.3.3 иметь меньшую метку, и потому волна будет прерываться (не произойдёт события решения и пересылки токена (нарушение 2-го пункта из свойств волнового алгоритма, а значит волновой алгоритм не завершится))

Всё это строилось на допущении, что алгоритм посылки tok - это волновой алгоритм, однако это

1. сказано, согласно формулировке алгоритма Раны.
2. можно доказать по сформулированным процедурам, процедуры идентичны (за исключением строки 5.3.3? прокомментированной выше, а часы Лемпорта в процедуре на условие посылки токена не влияют (они просто обновляются и посылаются)) процедурам кольцевого алгоритма, его формулировка и доказательство см. лекция 7.

Таким образом, волновой алгоритм с максимальным по времени токеном завершится, т.к. он начнётся через конечное число шагов, после выполнения условия term (это доказано выше) + конечное число шагов выполнения волнового алгоритма (согласно его 1-му свойству), т.е. decide (Announce) - произойдёт обязательно и через конечное число шагов.

**(A)**

Если выполнен term (т.е. базовый алгоритм таки завершился, то согласно доказательству (B) Announce обязательно произойдёт, причём через конечное число шагов).

Осталось доказать, что Announce переводит все процессы в конечное состояние, для этого докажем, что в каждом процессе произойдёт событие решения halt этого алгоритма.

Начальный процесс s выполняет процедуру Announce, это приводит к тому, что он посылает сообщение stop всем своим соседям, каждый сосед получив это сообщение повторяет эту процедуру и посылает сообщение stop всем своим соседям. Так как граф связный, то найдётся путь от начальной вершины s до любой другой вершины p, и по этому пути будут посланы сообщения stop от одного процесса к другому процессу, однако если у одной из вершин q на этом пути произошло выполнение halt, то она больше не обрабатывает приходящие сообщения и следовательно не перешлёт stop, тем самым цепочка s -1-> q -> p нарушена, однако если в процессе q произошёл halt, то это значит, что она хоть раз приняла сообщение stop и сразу после этого, но перед halt выполнила Announce разослав stop своим соседям, таким образом существовала цепочка s -2-> q по которой распространилось сообщение stop а значит есть цепочка s -2-> q -> p от исходной вершины s до любой другой вершины p.

(рассуждения приведены для случая, если halt в цепочке случился только в одной вершине, однако аналогичным образом они распространяются и на k вершин (если в цепочке у k вершин случился halt, то есть пусть распространения stop до последней вершины, в которой случился halt, а оттуда путь до p))

Итак stop распространится по всем вершинам, это значит, что для любой вершины p все её соседи рано или поздно получат stop и выполнят Announce прислав вершине p сообщения stop, но тогда переменная RecStop из алгоритма будет после последнего stop == #In, а значит будет выполнена операция halt.

Таким образом, если Выполнен term, то через конечное количество шагов выполнится Announce, а Announce за конечное количество шагов (потому что каждый процесс может принять не более #In сообщений, после чего он выключится через halt) переведёт каждый процесс в конечное состояние.

**(С)**

Announce - является событием decide для волнового алгоритма используемого в алгоритме Раны в процессе s (причём в конкретно нашем случае - циклического волнового алгоритма). Это означает, что согласно пункту 3 волновых алгоритмов, в каждом процессе, ему предшествовало хотя бы одно событие связанное с этим алгоритмом (5.3.3.1), т.е. это процедура I (для процесса инициатора это ещё могла быть процедура A), т.е. для каждого процесса было выполнено условие 5.3.3 и 5.3, т.е. каждый процесс находился в состоянии quiet в момент передачи tok.

Пойдём от противного - допустим term не выполнен, в момент decide волнового алгоритма, это значит, что либо один из процессов active, либо было послано mes в канал, но mes послал кто-то - это процесс, который выполнил процедуру S, т.е. был active, причём это случилось в момент decide - т.е. после передачи tok этой волны, возьмём первый процесс из тех, которые передали tok, а после этого стали active, т.е. перестали удовлетворять quiet, этот процесс p - мог стать active только в результате получения сообщения mes в процедуру R (нету другого кода, меняющего состояние на active), однако это сообщение послал другой процесс q, и он ещё не передавал tok (т.к. если он уже передал tok, а после этого послал сообщение, т.е. стал active, то вместе с сообщением он послал своё времечко, => θp >= θq +1 => p - не самый первый процесс, который стал active после передачи tok, противоречие).

Таким образом p - первый процесс нарушитель передал tok процесса инициатора s с временем qts потом получил сообщение mes от процесса q с временем θq > θ, и отправил ему ack с временем θ > θq и θ > qts,процесс q, получив ack обновит своё время на новое θ+1, и т.к. он не находился в состоянии quiet, рано или поздно перейдёт в состояние quiet (ведь только в этом состоянии можно передать tok, который согласно утверждению, что decide случился и волновой алгоритм завершился, - должен быть передан по свойству волнового алгоритма), однако при переходе в quiet qtq >= θ+1 > qts, что приводит к невыполнению условия 5.3.3 и глушит волну.

Это противоречит тому, что волна закончилась => исходное предположение ложь, и term выполнен в момент decide волнового алгоритма.

всё, что нужно было доказать - доказали

ч.т.д.