# О создании нового учебного курса по верификации программ для студентов МФТИ

Андрей Михайлович Миронов

amironov66@gmail.com

Московский Государственный Университет
механико-математический факультет

доклад на семинаре в МФТИ
8 октября 2025

Слайд из заключительного доклада профессора, доктора наук Барта Пренеля «Обзор криптографического года», конференция Information Security Solutions Europe (ISSE) 2012, Брюссель, Бельгия, 24.10.2012 г.:

ISSE 2009

## Cryptography ≠ security

crypto is only a tiny piece of the security puzzle

– but an important one

most systems break elsewhere

– incorrect requirements or specifications

– implementation errors

– application level

– social engineering (layer 8)

Проблема **верификации** (т.е. доказательства правильности) программ занимает центральное положение в теории и практике разработки надёжного программного обеспечения.
Под **правильностью** программ понимается их соответствие различным условиям

- корректности,
- безопасности,
- устойчивости в случае непредусмотренного поведения окружения,
- эффективности использования ресурсов времени и памяти,
- оптимальности реализованных в программе алгоритмов, и т.п.

# Пример уязвимости протокола безопасности

Рассмотрим пример уязвимости протокола безопасности (ПБ), который использовался много лет в банковских транзакциях.
Данный протокол очень прост, он состоит всего из трёх действий, и сначала его правильность не вызывала сомнений. Однако после 15 лет его использования выяснилось, что он содержит уязвимость (которая была обнаружена автоматической системой верификации).
Этот ПБ называется протоколом Нидхема– Шредера (Needham-Schroeder, 1979), мы будем обозначать его записью NS.
Целью данного ПБ является взаимная аутентификация агентов $a$ и $b$.

# Описание протокола Нидхема– Шредера

Предполагается, что $a$ и $b$ используют общую ассимметричную систему шифрования (АСШ).
Протокол NS имеет следующий вид:

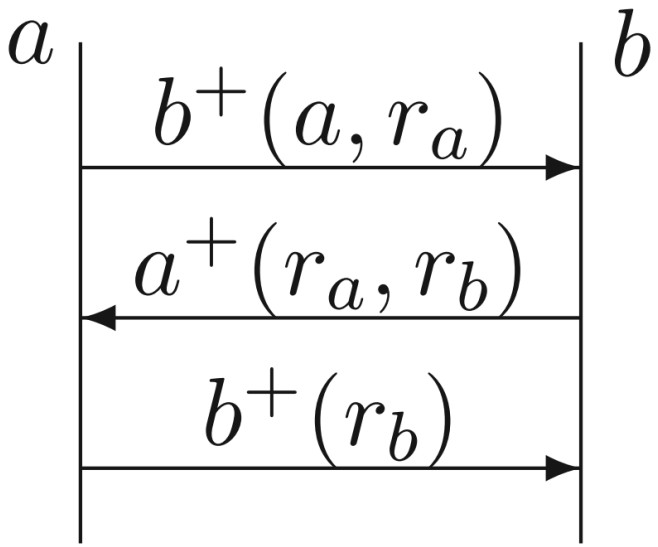
$$\begin{cases} a \to b : b^+(a, r_a) \\ b \to a : a^+(r_a, r_b) \\ a \to b : b^+(r_b) \end{cases}$$

где $r_a$ и $r_b$ – уникальные значения, называемые в литературе по протоколам безопасности нонсами (nonces, что является сокращением от number used once), каждое из которых представляет собой большую (несколько сотен битов) псевдослучайную строку, сгенерированную агентом $a$ и $b$ соответственно.
Мы предполагаем что все нонсы, сгенерированные в различных сеансах выполнения любого ПБ, являются различными и отличаются от других значений. Нонсы обозначаются символом $r$ (возможно, с индексами).

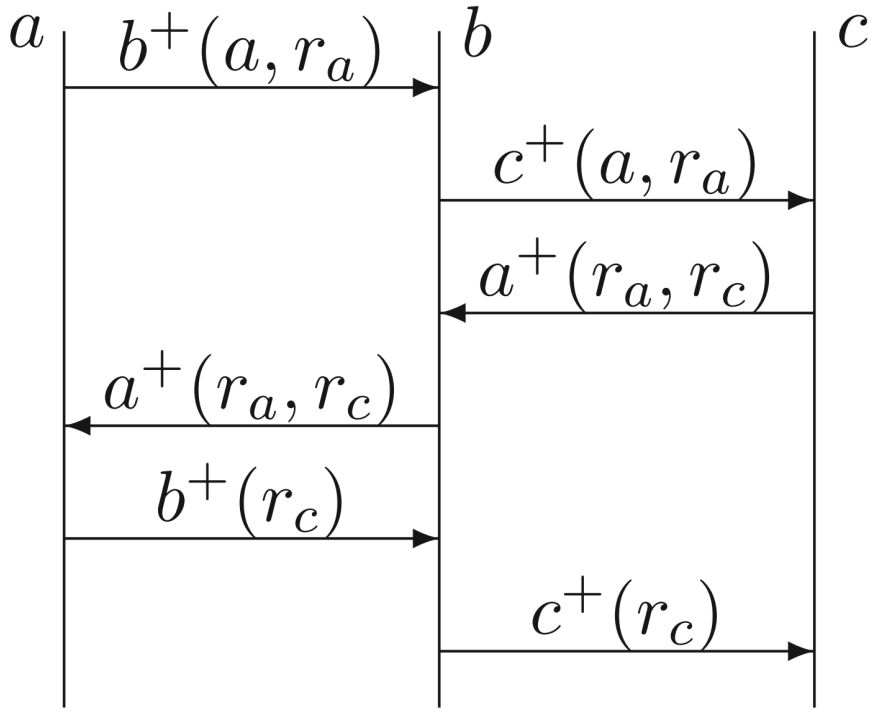
Действия, входящие в ПБ NS, имеют следующий смысл.

- Первое действие заключается в пересылке от $a$ к $b$ шифртекста (ШТ) $b^+(a, r_a)$, получаемого шифрованием на ключе $b^+$ пары, состоящей из имени агента $a$ и нонса $r_a$, который используется для того, чтобы дать возможность $b$ доказать свою подлинность путем извлечения данного нонса из полученного ШТ.
- Второе действие заключается в пересылке от $b$ к $a$ ШТ $a^+(r_a, r_b)$, получаемого шифрованием на ключе $a^+$ пары, состоящей из нонса $r_a$, который $b$ извлёк из полученного ШТ, и нонса $r_b$, сгенерированного агентом $b$. После его получения $a$ убеждается в подлинности $b$, т.к. никто кроме $b$ не может извлечь $r_a$. Нонс $r_b$ предназначен для того, чтобы дать возможность $a$ тоже доказать $b$ свою подлинность.
- Третье действие заключается в пересылке от $a$ к $b$ ШТ $b^+(r_b)$. После получения $r_b$ агент $b$ убеждается в подлинности агента $a$.

Таким образом, после успешного завершения данного протокола $a$ и $b$ будут убеждены в подлинности друг друга.
Для формальной записи ПБ используется также нотация в виде диаграмм. Каждое действие в ПБ, связанное с пересылкой сообщений, изображается в диаграмме горизонтальной стрелкой, помеченной пересылаемым сообщением. Диаграмма, соответствующая протоколу (5), имеет вид



Одна из уязвимостей данного ПБ связана с тем, что агент $b$ может использовать свой статус участника ПБ NS для того, чтобы во взаимодействии с другими агентами выдавать себя за агента $a$. Использование агентом $b$ своего статуса участника ПБ NS для совершения мошеннических действий можно изобразить следующей диаграммой:



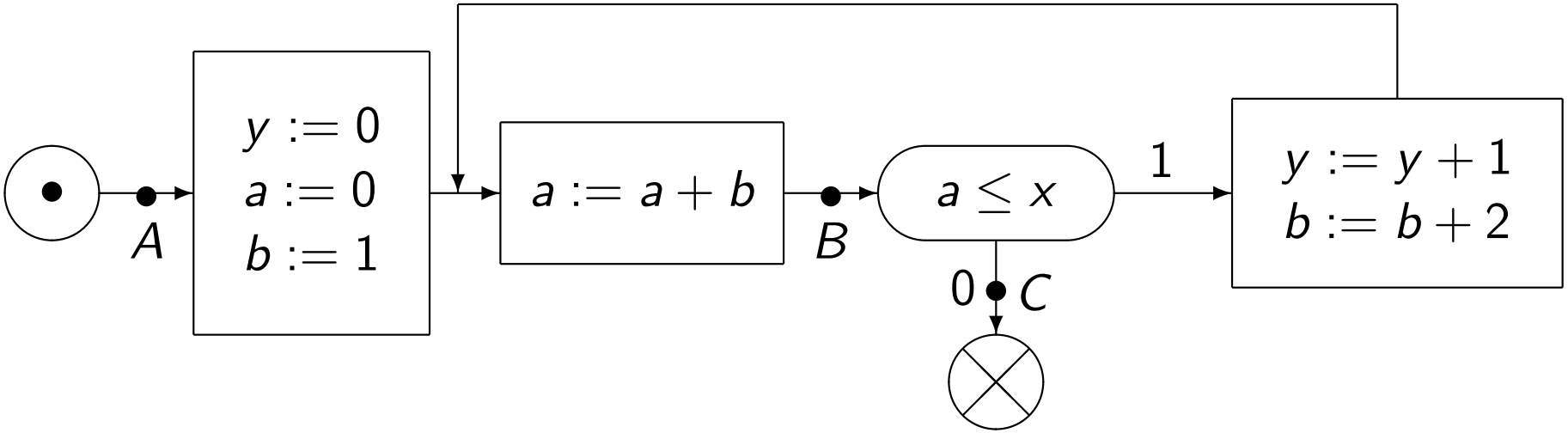
Данную диаграмму можно рассматривать как объединение двух диаграмм, изображающих выполнение двух сеансов выполнения ПБ NS: в первом сеансе участвуют агенты $a$ и $b$, а во втором – агенты $b$ и $c$. Действия $a$, $b$ и $c$ во время выполнения этих сеансов выглядят следующим образом.

1. После того, как $a$ и $b$ выполнят первое действие первого сеанса, агент $b$ создает с использованием полученного нонса $r_a$ ШТ $c^+(a, r_a)$, и посылает этот ШТ агенту $c$ в качестве своего первого действия во втором сеансе ПБ NS (выдавая себя за $a$).

2. Получив этот ШТ и дешифруя его, агент $c$ приходит к выводу, что этот ШТ был послан агентом $a$. Согласно протоколу NS, агент $c$ генерирует нонс $r_c$ и посылает ответный ШТ $a^+(r_a, r_c)$.
$b$ пересылает полученный от с ШТ $a^+(r_a, r_c)$ агенту $a$.
$a$ рассматривает полученный от $b$ ШТ как ответ, который $b$ должен послать ему в соответствии со вторым шагом ПБ NS, т.е. $a$ полагает, что извлечённый из полученного ШТ нонс $r_c$ был сгенерирован агентом $b$.
3. Согласно третьему шагу NS, $a$ посылает $b$ ШТ $b^+(r_c)$.
$b$ извлекает из полученного ШТ нонс $r_c$, и посылает агенту $c$ ШТ $c^+(r_c)$.
После этого $c$ верит, что тот агент, с которым он выполнял этот сеанс ПБ NS, является агентом $a$.

Пример дедуктивного решения задачи верификации

Верифицируем программу извлечения корня относительно предусловия $x \geq 0$ и постусловия $y = \lfloor \sqrt{x} \rfloor$.



Определим инварианты в выбранных точках следующим образом:

$$\varphi_A \stackrel{\text{def}}{=} (x \geq 0),\ \varphi_B \stackrel{\text{def}}{=} \left\{ \begin{array}{l} y \geq 0 \\ y^2 \leq x \\ a = (y+1)^2 \\ b = 2y+1 \end{array} \right\},\ \varphi_C \stackrel{\text{def}}{=} \left\{ \begin{array}{l} y \geq 0 \\ y^2 \leq x < (y+1)^2 \end{array} \right\}$$

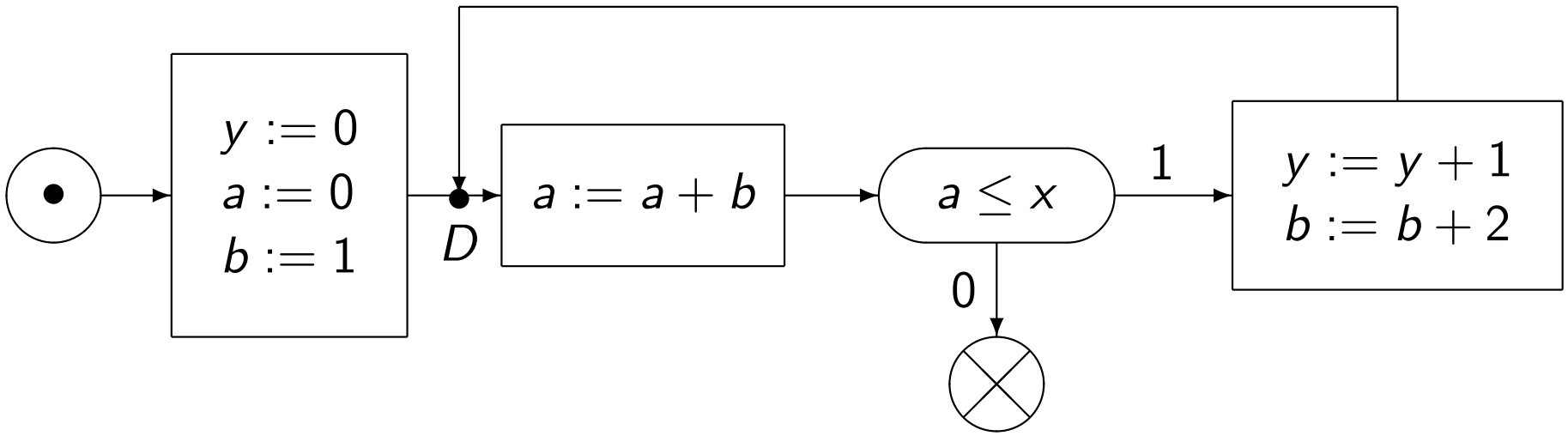
(нетрудно видеть, что $\varphi_C$ эквивалентна постусловию).
Докажем, что $\varphi_B$ определен правильно.

1. При первом входе в $B$ $\varphi_B$ имеет вид $\left\{\begin{array}{l} 0 \geq 0 \\ 0^2 \leq x \\ 1 = (0+1)^2 \\ 1 = 2 \cdot 0 + 1 \end{array}\right\}$, истинность данного соотношения следует из $\varphi_A$.
2. При переходе от $B$ к $B$ по циклу верно дополнительное условие $a \leq x$, с учетом которого импликация $\varphi_B \Rightarrow \varphi'_B$ имеет вид

$$\left\{\begin{array}{l} y \geq 0 \\ y^2 \leq x \\ a = (y+1)^2 \\ b = 2y+1 \\ a \leq x \end{array}\right\} \Rightarrow \left\{\begin{array}{l} y' \geq 0 \\ (y')^2 \leq x' \\ a' = (y'+1)^2 \\ b' = 2y'+1 \end{array}\right\}. \qquad (1)$$

   Учитывая соотношения $y' = y+1$, $b' = b+2$, $a' = a+b+2$, $x' = x$, нетрудно установить, что импликация (1) верна.
3. При переходе от $B$ к $C$ верно дополнительное условие $x < a$, что в сочетании с $\varphi_B$ влечёт $\varphi_C$.

Докажем завершаемость данной программы. Положим $N \stackrel{\text{def}}{=} \{D\}$, где точка $D$ выбрана так, как показано на рисунке:




и, кроме того, $L \stackrel{\text{def}}{=} \mathbf{N}$, $u_D \stackrel{\text{def}}{=} x - a$. Для обоснования того, что

- при каждом проходе через $D$ текущая подстановка $\theta$ удовлетворяет условию $u_D^\theta \in \mathbf{N}$ и
- при переходе от $D$ к $D$ по циклу $u_D^\theta = x - a > x' - a' = u_D^{\theta'}$, где $\theta$ и $\theta'$ – текущие подстановки до и после прохода по циклу,

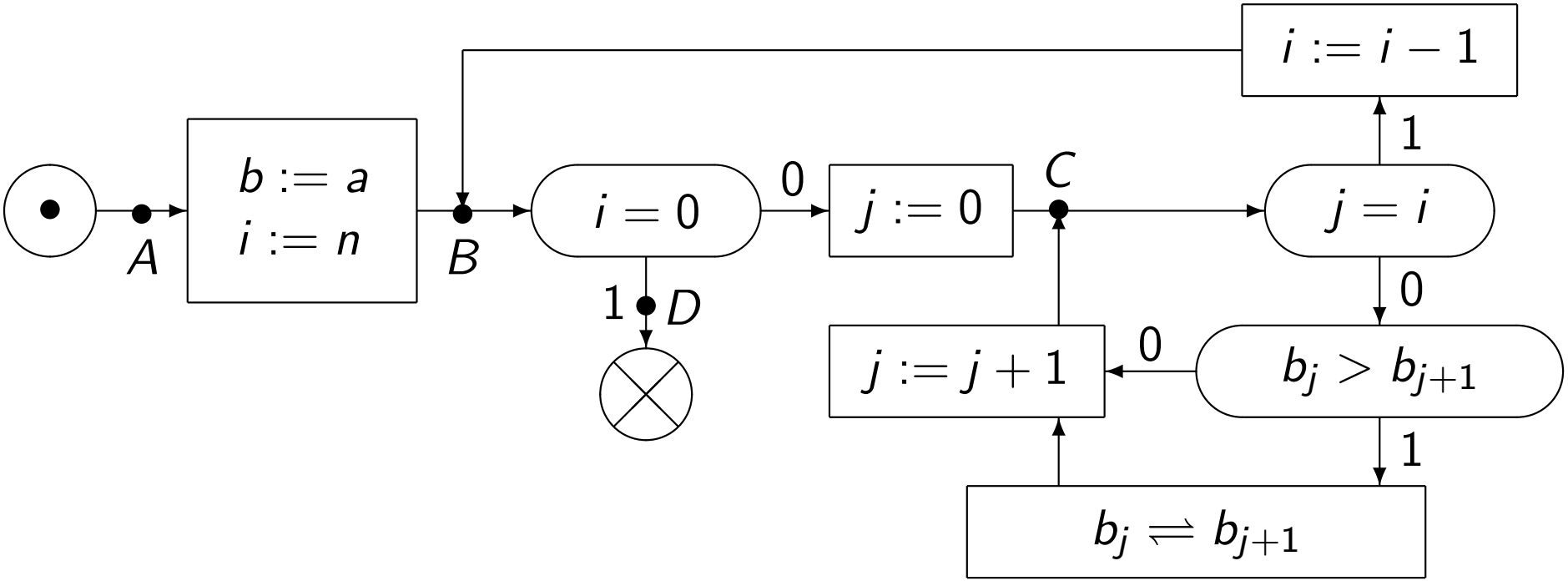
определим вспомогательный инвариант в $D$: $\varphi_D \stackrel{\text{def}}{=} (x \geq a) \wedge (b \geq 1)$.

Истинность $\varphi_D$ при каждом проходе через $D$ следует из того, что

- $\varphi_D$ верно при первом проходе через $D$ (это следует из $\varphi_A$) и
- при произвольном переходе от $D$ к $D$ по циклу верно соотношение, написанное в условном операторе программы ($a \leq x$).

Из истинности $\varphi_D$ при каждом проходе через $D$ следуют вышеупомянутые свойства, связанные с точкой $D$.

Другой пример - верификация программы пузырьковой сортировки. Исходный массив – $a = a_{0..n}$, результат – массив $b = b_{0..n}$.




Предусловие: $n \geq 0$, постусловие: $(b = perm(a)) \wedge ord(b)$.

Конъюнктивные члены в постусловии обосновываются раздельно. Обоснование утверждения $b = perm(a)$ заключается в том, что

- оно является верным после выполнения первого присваивания и
- все остальные действия в этой БС сохраняют его истинность, т.к. единственное действие в БС, которое изменяет массив $b$, это перестановка его соседних компонентов ($b_j \rightleftharpoons b_{j+1}$), однако эта перестановка не изменяет содержимое массива $b$.

Для обоснования утверждения $ord(b)$ определим инварианты в точках $A$, $B$, $C$, $D$:
Инвариантами $\varphi_A$ и $\varphi_D$ являются формулы $(n \geq 0)$ и $ord(b_{0..n})$ соответственно. Инварианты $\varphi_B$ и $\varphi_C$ имеют следующий вид:

$$\varphi_B \stackrel{\text{def}}{=} \left\{ \begin{array}{l} 0 \leq i \leq n \\ ord(b_{i..n}) \\ b_{0..i} \leq b_{i+1..n} \end{array} \right\}, \quad \varphi_C \stackrel{\text{def}}{=} \left\{ \begin{array}{l} 1 \leq i \leq n \\ 0 \leq j \leq i \\ ord(b_{i..n}) \\ b_{0..i} \leq b_{i+1..n} \\ b_{0..j-1} \leq b_j \end{array} \right\}.$$

Докажем завершаемость БС сортировки. Положим $N \stackrel{\text{def}}{=} \{C\}$, $L \stackrel{\text{def}}{=} \mathbf{N}^2$ (лексикографический порядок), $u_C \stackrel{\text{def}}{=} (i, i-j)$. Из $\varphi_C$ следует, что при проходе через $C$ текущая подстановка $\theta$ удовлетворяет условию $u_C^\theta \in \mathbf{N}^2$ и при переходе от $C$ к $C$ по верхнему циклу $i' = i - 1$, поэтому

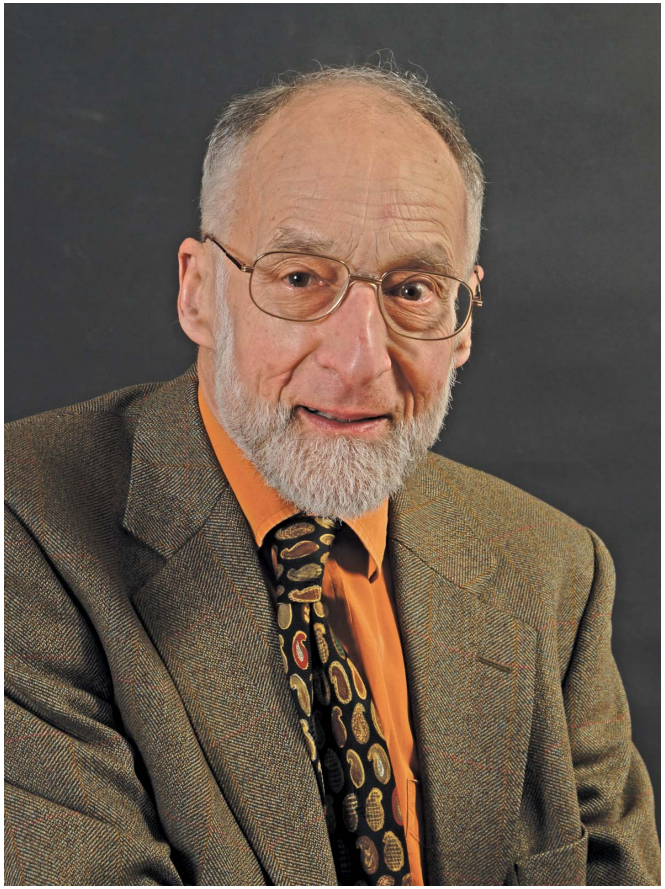
$$u_C^\theta = (i, i-j) > (i-1, \ldots) = u_C^{\theta'},$$

а при переходе от $C$ к $C$ по нижним циклам $i' = i$, $j' = j + 1$, поэтому

$$u_C^\theta = (i, i-j) > (i, i-(j+1)) = (i', i'-j') = u_C^{\theta'},$$

где $\theta$ и $\theta'$ – текущие подстановки до и после прохода по циклам.

**Теория процессов** – математическая основа формальных моделей и методов верификации свойств корректности и безопасности информационно-компьютерных систем



Робин Милнер (англ. Arthur John Robin Gorell Milner; 13 января 1934 - 20 марта 2010, Кембридж, Великобритания) - английский ученый в области теории вычислительных систем.

**Теория процессов** является одним из разделов математической теории программирования, который изучает математические модели поведения динамических систем, называемые **процессами**.
Говоря неформально, **процесс** представляет собой модель такого поведения, которое заключается в исполнении **действий**. Такими действиями могут быть, например,

- прием или передача каких-либо объектов, или
- преобразование этих объектов.

Основные достоинства теории процессов как математического аппарата, предназначенного для моделирования и анализа динамических систем, заключаются в следующем.

1. Аппарат теории процессов хорошо подходит для формального описания и анализа поведения **распределенных динамических систем**, т.е. таких систем, которые состоят из нескольких взаимодействующих компонентов, причем

- все эти компоненты работают параллельно, и
- взаимодействие компонентов происходит путем пересылки сигналов или сообщений от одних компонентов другим компонентам.

2. Методы теории процессов позволяют анализировать с приемлемой сложностью модели с очень большим и даже бесконечным множеством состояний. Это возможно благодаря разработанной в теории процессов технике символьных преобразований выражений, описывающих процессы.

3. Методы теории процессов хорошо подходят для изучения **иерархических** систем, т.е. таких систем, которые имеют многоуровневую структуру.
Каждая компонента таких систем рассматривается как подсистема, которая, в свою очередь, может состоять из нескольких подкомпонентов. Каждая из этих подкомпонентов может взаимодействовать

- с другими подкомпонентами, и
- с системами более высокого уровня.

Основными источниками задач и объектами применения результатов теории процессов являются распределенные компьютерные системы. Вместе с тем, теория процессов может использоваться также для моделирования и анализа поведения систем самой различной природы, важнейшими примерами которых являются **организационные системы**. К их числу относятся

- системы управления деятельностью предприятий,
- государственные структуры,
- системы организации коммерческих процессов (например, системы организации торговых сделок, аукционов, и т.п.)

Процессы, относящиеся к функционированию таких систем, принято называть **бизнес-процессами**.

# Верификация процессов

Наиболее важный класс задач, для решения которых предназначена теория процессов, связан с проблемой верификации процессов. Проблема **верификации** процесса заключается в построении формального доказательства того, что анализируемый процесс обладает заданными свойствами.

Для многих процессов задача верификации представляет исключительную актуальность. Например, безопасная эксплуатация таких систем, как

- системы управления атомными электростанциями,
- медицинские устройства с компьютерным управлением,
- бортовые системы управления самолетов и космических аппаратов,
- системы управления секретными базами данных,
- системы электронной коммерции

невозможна без удовлетворительного решения задачи верификации свойств корректности и безопасности процессов, функционирующих в таких системах, так как нарушения данных свойств в системах подобного типа могут привести к существенному ущербу для экономики и самой жизни людей.

Точная постановка задачи верификации состоит из следующих частей.

1. Построение процесса, представляющего собой математическую модель поведения анализируемой системы.
2. Представление проверяемого свойства в виде математического объекта (называемого **спецификацией**).
3. Построение **математического доказательства** утверждения о том, что построенный процесс удовлетворяет спецификации.

# Спецификация процессов

**Спецификация** процесса представляет собой описание свойств этого процесса в виде некоторого математического объекта.
Примером спецификации является требование надежности передачи данных через ненадежную среду. При этом не указывается, как именно должна обеспечиваться эта надежность.

В качестве спецификации могут выступать следующие объекты.

1. Логическая формула, выражающая некоторое требование к процессу. Например, таким требованием может быть условие того, что если процесс получил некоторый запрос, то через некоторое заданное время процесс выдаст ответ на этот запрос.
2. Представление анализируемого процесса на более высоком уровне абстракции. Данный вид спецификаций используется при многоуровневом проектировании процессов: реализацию процесса на каждом уровне проектирования можно рассматривать как спецификацию для реализации этого процесса на следующем уровне проектирования.
3. Некоторый эталонный процесс, относительно которого предполагается, что он обладает заданным свойством.
   В этом случае задача верификации заключается в построении доказательства эквивалентности эталонного и анализируемого процессов.

При построении спецификаций следует руководствоваться следующими принципами.

1. Одно и то же свойство процесса может быть выражено на разных языках спецификаций (ЯС), и

- на одном ЯС оно может иметь простую спецификацию,
- а на другом – сложную.

Например, спецификация, описывающая связь между входными и выходными значениями для программы, вычисляющей разложение целого числа на простые множители, имеет

- сложный вид на языке логики предикатов, но
- простой вид, если выразить эту спецификацию в виде некоторой эталонной программы.

Поэтому для представления свойства процесса в виде спецификации важно выбрать такой ЯС, на котором спецификация этого свойства имела бы наиболее ясный и простой вид.

2. Если свойство процесса изначально было выражено на естественном языке, то при переводе его в спецификацию важно обеспечить соответствие между

- естественно-языковым описанием этого свойства, и
- его спецификацией,

т.к. в случае несоблюдения этого условия результаты верификации не будут иметь смысла.

# Представление поведения динамических систем в виде процессов

Один из возможных методов математического моделирования поведения динамических систем заключается в представлении поведения этих систем в виде **процессов**.
Процесс, как правило, не учитывает всех деталей поведения анализируемой системы. Одно и то же поведение может быть представлено различными процессами, отражающими

- разную степень абстракции при построении модели этого поведения, и
- разные уровни детализации действий, исполняемых системой.

Если целью построения процесса для представления поведения некоторой системы является проверка свойств этого поведения, то выбор уровня детализации действий системы должен производиться с учетом тех свойств, которые необходимо проанализировать.

Построение процесса, представляющего поведение анализируемой системы, должно производиться с учетом следующих принципов.

1. Описание процесса не должно быть чрезмерно детальным, т.к. излишняя сложность этого описания может вызвать существенные вычислительные проблемы при формальном анализе этого процесса.
2. Описание процесса не должно быть чрезмерно упрощенным, оно должно
   - отражать те аспекты поведения моделируемой системы, которые имеют отношение к проверяемым свойствам, и
   - сохранять все свойства поведения этой системы, которые представляют интерес для анализа

   т.к. в случае несоблюдения этого условия результаты анализа такого процесса не будут иметь смысла.

Мы понимаем под процессом модель поведения динамической системы на некотором уровне абстракции. **Процесс** можно представлять себе как граф $P$, где

- Вершины графа $P$ называются **состояниями**, и изображают ситуации (или классы ситуаций), в которых может находиться моделируемая система в различные моменты своего функционирования. Одно из состояний является выделенным, оно называется **начальным состоянием** процесса $P$.
- Ребра графа $P$ имеют метки, изображающие **действия**, которые может исполнять моделируемая система.
- Функционирование процесса $P$ описывается переходами по ребрам графа $P$ от одного состояния к другому. Функционирование начинается из начального состояния. Метка каждого ребра изображает действие процесса, исполняемое при переходе от состояния в начале ребра к состоянию в его конце.

# Пример процесса

В качестве первого примера рассмотрим процесс, представляющий собой простейшую модель поведения некоторого торгового автомата. Мы будем представлять себе этот автомат как машину, которая имеет

- монетоприемник,
- кнопку, и
- лоток для выдачи товара.

Когда покупатель хочет приобрести товар, он

- опускает монету в монетоприемник,
- нажимает на кнопку

и после этого в лотке появляется товар.

Предположим, что наш автомат торгует шоколадками по цене 1 монета за штуку.

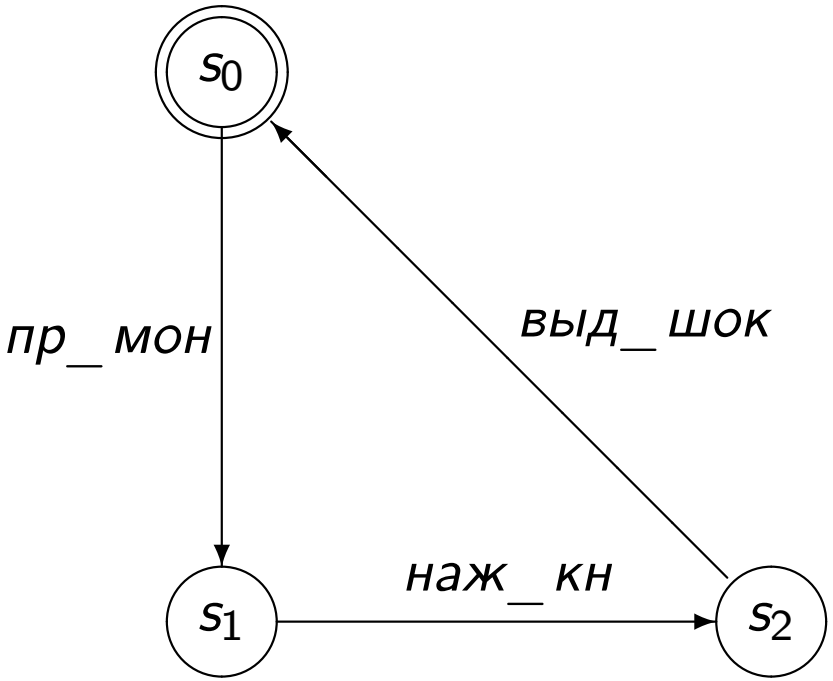
Опишем действия такого автомата.

- По инициативе покупателя, в автомате могут происходить следующие действия:
  - попадание в щель монеты, и
  - нажатие кнопки.
- В ответ автомат может осуществлять реакцию: выдавать в лоток шоколадку.

Обозначим действия короткими именами:

- прием монеты мы обозначим символом *пр_мон*,
- нажатие кнопки - символом *наж_кн*, и
- выдачу шоколадки - символом *выд_шок*.

Процесс нашего торгового автомата выглядит следующим образом:

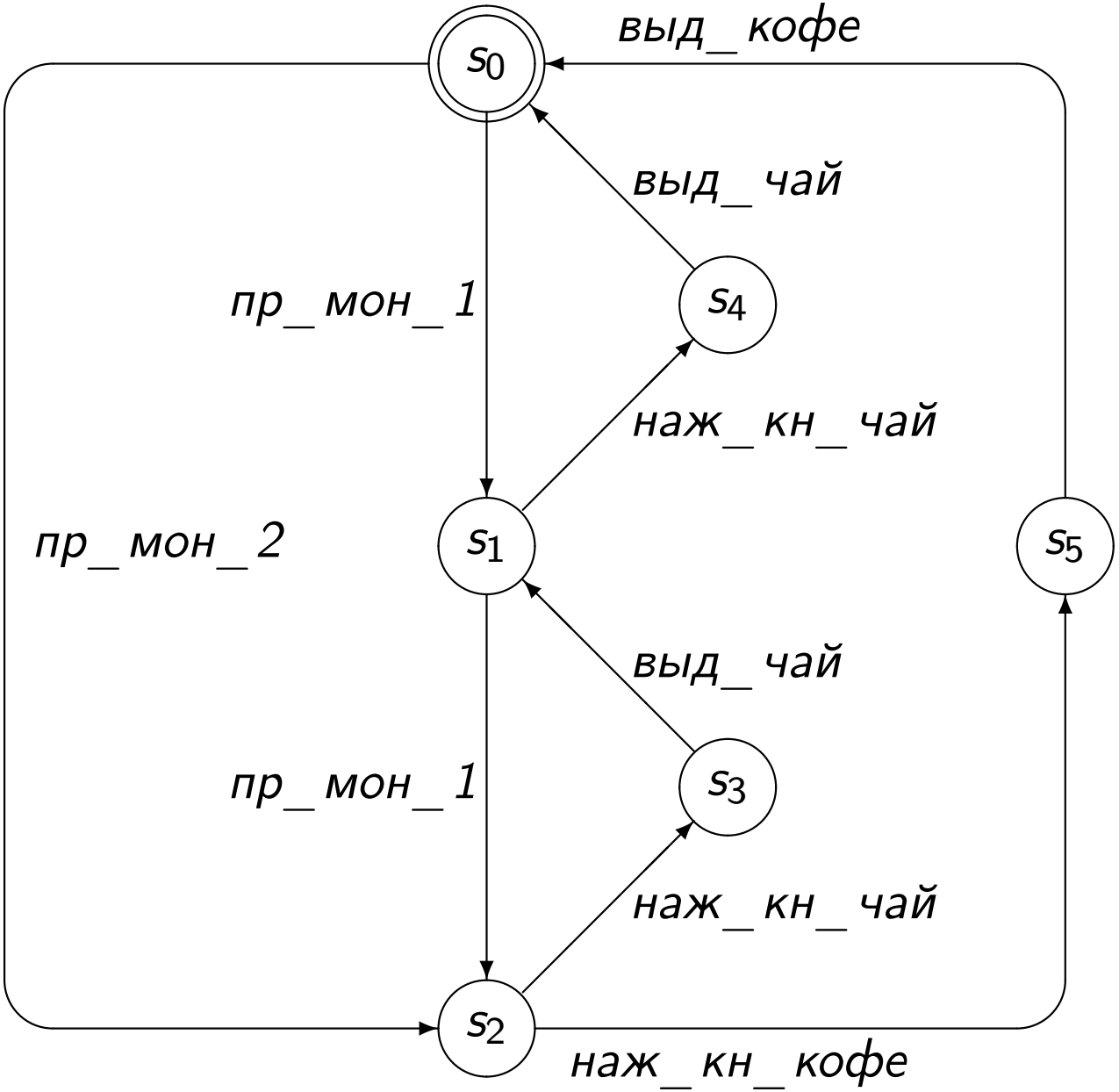


Данная диаграмма объясняет, как именно функционирует торговый автомат:

- вначале автомат находится в состоянии $s_0$, в этом состоянии он ожидает появления в приемнике монеты
  (то, что состояние $s_0$ является начальным, изображается на диаграмме двойным кружочком вокруг идентификатора этого состояния)
- когда монета появляется, автомат переходит в состояние $s_1$ и ждет нажатия на кнопку,
- после нажатия кнопки автомат
  - переходит в состояние $s_2$,
  - выдает шоколадку, и
  - возвращается в состояние $s_0$.

# Другой пример процесса

Рассмотрим более сложный пример торгового автомата, который отличается от предыдущего тем, что продает два вида товаров: чай и кофе, причем стоимость чая - 1 рубль, а стоимость кофе - 2 рубля. Автомат имеет две кнопки: одну - для чая, другую - для кофе. Покупатель может платить монетами достоинством в 1 рубль и 2 рубля. Данные монеты будут обозначаться знакосочетаниями *мон_1* и *мон_2* соответственно.
Если покупатель опустил в монетоприемник монету *мон_1*, он может купить только чай. Если же он опустил монету *мон_2*, он может купить кофе или два чая. Кофе можно купить также, опустив в монетоприемник две монеты *мон_1*.
Процесс такого торгового автомата выглядит следующим образом:


$s_0$
выд_кофе
выд_чай
пр_мон_1
$s_4$
наж_кн_чай
пр_мон_2
$s_1$
$s_5$
выд_чай
пр_мон_1
$s_3$
наж_кн_чай
$s_2$
наж_кн_кофе

# Действия

Для задания процесса $P$, представляющего собой модель поведения некоторой динамической системы, должно быть указано некоторое множество $Act(P)$ **действий**, которые может выполнять процесс $P$. Мы будем предполагать, что действия всех процессов являются элементами некоторого универсального множества $Act$ всех возможных действий, которые может выполнить какой-либо процесс, т.е. для любого процесса $P$

$$Act(P) \subseteq Act$$

Выбор множества $Act(P)$ действий процесса $P$ зависит от целей моделирования. В разных ситуациях для представления модели анализируемой системы в виде некоторого процесса могут выбираться разные множества действий.

Мы будем предполагать, что множество действий $Act$ делится на 3 следующих класса.

1. **Входные действия**, которые изображаются знакосочетаниями вида $\alpha?$. Действие вида $\alpha?$ интерпретируется как ввод в процесс некоторого объекта с именем $\alpha$.
2. **Выходные действия**, которые изображаются знакосочетаниями вида $\alpha!$. Действие вида $\alpha!$ интерпретируется как вывод из процесса некоторого объекта с именем $\alpha$.
3. **Внутреннее** (или **невидимое**) действие, которое обозначается символом $\tau$. Внутренним мы называем любое действие процесса $P$, которое не связано с его взаимодействием с **окружающей средой** (т.е. с процессами, являющимися внешними по отношению к процессу $P$, и с которыми он может взаимодействовать).
   Например, внутреннее действие может быть связано с взаимодействием компонентов процесса $P$.

Обозначим знакосочетанием *Names* совокупность имен объектов, которые могут вводиться в какой-либо процесс нли выводиться из него.

$$Act = \{\alpha? \mid \alpha \in Names\} \cup \{\alpha! \mid \alpha \in Names\} \cup \{\tau\} \tag{2}$$

Отметим, что объекты, которые вводятся в процесс и выводятся из него, могут иметь самую различную природу (как материальную, так и не материальную). Например, ими могут быть

- материальные ресурсы,
- люди,
- деньги,
- информация,
- энергия,
- и т.д.

Кроме того, сами понятия ввода и вывода могут иметь виртуальный характер, т.е. слова “ввод” и “вывод” могут использоваться лишь как метафоры, а в действительности никакого ввода или вывода какого-либо реального объекта может и не происходить. Говоря неформально, мы будем рассматривать действие процесса $P$ как

- **входное**, если его инициатором является процесс, внешний по отношению к $P$, и
- **выходное**, если оно не является внутренним, и его инициатором является сам процесс $P$.

Для каждого имени $\alpha \in Names$ действия $\alpha?$ и $\alpha!$ называются **комплементарными**.

# Определение понятия процесса

**Процессом** называется тройка $P$ вида $P = (S, s^0, R)$, где

- $S$ - множество, элементы которого называются **состояниями**.
- $s^0 \in S$ – некоторое выделенное состояние, называемое **начальным состоянием** процесса $P$.
- $R$ – подмножество вида $R \subseteq S \times Act \times S$. Элементы множества $R$ называются **переходами**.
  Если переход из $R$ имеет вид $(s_1, a, s_2)$, то
  - мы будем говорить, что этот переход является переходом из состояния $s_1$ в состояние $s_2$ с выполнением действия $a$,
  - состояния $s_1$ и $s_2$ называются **началом** и **концом** этого перехода соответственно, а действие $a$ называется **меткой** этого перехода, и
  - иногда, в целях повышения наглядности, мы будем обозначать данный переход знакосочетанием

$$s_1 \xrightarrow{a} s_2 \tag{3}$$

**Функционирование** процесса $P = (S, s^0, R)$ заключается в порождении последовательности переходов вида

$$s^0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \dots$$

и выполнении действий $a_0, a_1, a_2 \dots$, соответствующих этим переходам. Более подробно: на каждом шаге функционирования $i \geq 0$

- процесс находится в некотором состоянии $s_i$ ($s_0 = s^0$),
- если есть хотя бы один переход из $R$ с началом в $s_i$, то процесс
  - недетерминированно выбирает переход с началом в $s_i$, помеченный таким действием $a_i$, которое можно выполнить в текущий момент времени (если таких переходов нет, то процесс временно приостанавливает свою работу до того момента, когда появится хотя бы один такой переход)
  - выполняет действие $a_i$, и после этого переходит в состоние $s_{i+1}$, которое является концом выбранного перехода
- если в $R$ нет переходов с началом в $s_i$, то процесс заканчивает свою работу.

# Операции на процессах

Пусть заданы

- процесс $P = (S, s^0, R)$, и
- действие $a \in Act$.

Действие операции **префиксного действия** $a.$ на процесс $P$ заключается в том, что

- к множеству состояний $P$ добавляется новое состояние $s$, которое будет начальным состоянием нового процесса, и
- к множеству переходов добавляется переход

$$s \xrightarrow{a} s^0$$

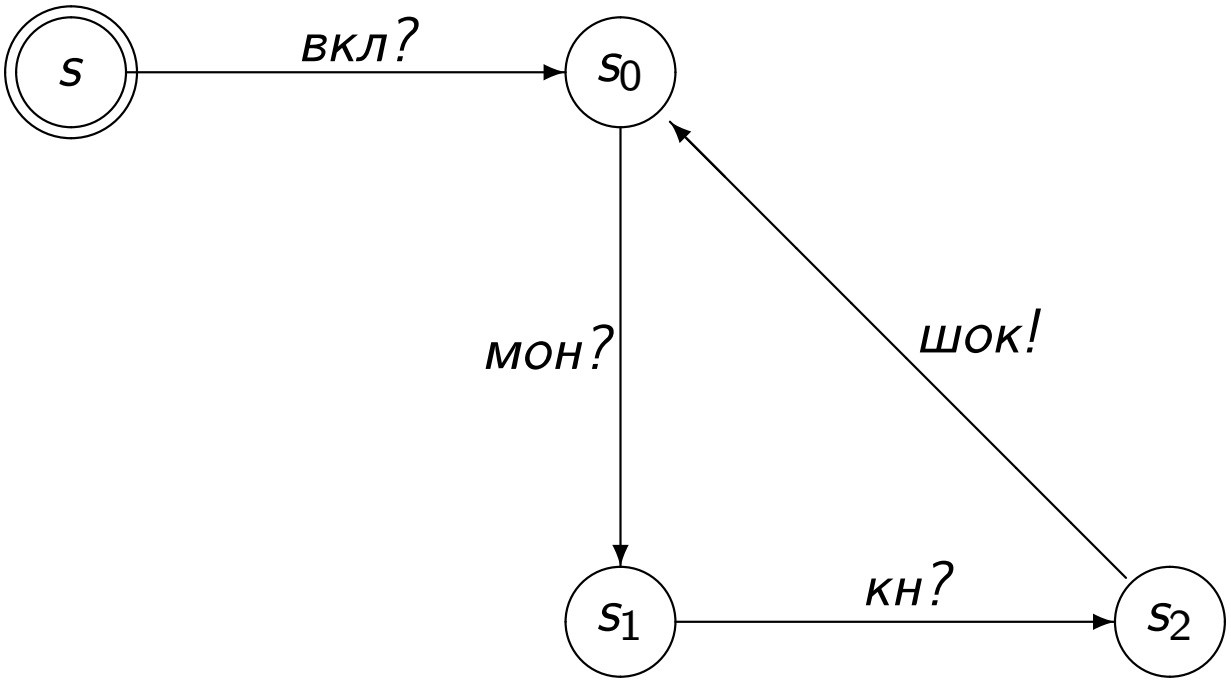
Получившийся процесс обозначается знакосочетанием

$$a.P$$

Проиллюстрируем действие данной операции на примере описанного выше торгового автомата. Обозначим процесс, представляющий поведение этого автомата, символом $P_{\text{та}}$.
Расширим множество действий данного автомата новым входным действием *вкл*?, которое будет означать включение этого автомата в сеть.
Процесс *вкл*?. $P_{\text{та}}$ представляет поведение нового торгового автомата, который в начальном состоянии не может

- ни принимать монет,
- ни воспринимать нажатия на кнопку,
- ни выдавать шоколадок.

Единственное, что он может - это стать включенным. После этого его поведение ничем не будет отличаться от поведения исходного автомата.
Графовое представление процесса *вкл*?. $P_{\text{та}}$ выглядит следующим образом:


s
вкл?
s0
мон?
шок!
s1
кн?
s2

# Пустой процесс

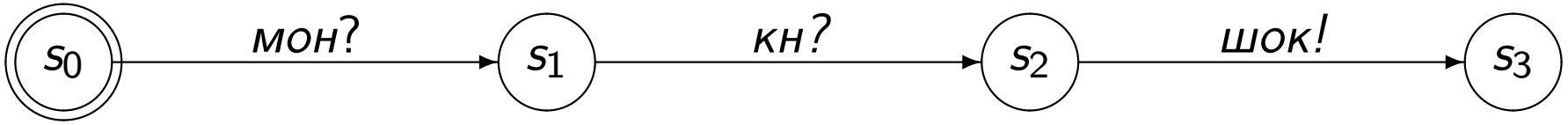
Среди всех процессов существует один наиболее простой. Этот процесс имеет всего одно состояние, и не имеет переходов. Для обозначения такого процесса мы будем использовать константу (т.е. нульарную операцию) $\mathbf{0}$.

Возвращаясь к примерам с торговыми автоматами, можно сказать, что процесс $\mathbf{0}$ представляет поведение сломанного автомата, то есть такого автомата, который вообще не может ничего делать.

Путем применения операций префиксного действия к процессу $\mathbf{0}$ можно определять поведение более сложных автоматов. Рассмотрим, например, такой процесс:

$$P = мон?.кн?.шок!.\mathbf{0}$$

Графовое представление этого процесса выглядит следующим образом:




Этот процесс задает поведение автомата, который обслуживает ровно одного покупателя, и после этого ломается.

# Альтернативная композиция

Следующая операция на процессах - это бинарная операция альтернативной композиции.
Данная операция используется в том случае, когда по паре процессов $P_1$ и $P_2$, надо построить процесс $P$, который будет функционировать

- либо как процесс $P_1$,
- либо как процесс $P_2$,

причем выбор процесса, в соответствии с которым $P$ будет функционировать, может определяться

- как самим $P$,
- так и окружающей средой, в которой функционирует $P$.

Например, если $P_1$ и $P_2$ имеют вид

$$P_1 = \alpha? \,.\, P_1', \quad P_2 = \beta? \,.\, P_2' \tag{4}$$

и в начальный момент времени окружающая среда

- может ввести в $P$ объект $\alpha$, но
- не может ввести в $P$ объект $\beta$

то $P$ должен выбрать то поведение, которое является единственно возможным в данной ситуации, т.е. работать так же, как процесс $P_1$.

Точное определение операции альтернативной композиции выглядит следующим образом.

Пусть процессы $P_1$ и $P_2$ имеют вид $P_i = (S_i, s_i^0, R_i) \quad (i = 1, 2)$, причем множества состояний $S_1$ и $S_2$ не имеют общих элементов.

**Альтернативной композицией** процессов $P_1$ и $P_2$ называется процесс

$$P_1 + P_2 = (S, s^0, R)$$

компоненты которого определяются следующим образом.

- $S$ получается добавлением к $S_1 \cup S_2$ нового состояния $s^0$, которое будет начальным состоянием процесса $P_1 + P_2$
- $R$ содержит все переходы из $R_1$ и $R_2$.
- для каждого перехода из $R_i$ $(i = 1, 2)$ вида

$$s_i^0 \xrightarrow{a} s$$

  $R$ содержит переход

$$s^0 \xrightarrow{a} s$$

# Параллельная композиция

Операция параллельной композиции используется для построения моделей поведения динамических систем, состоящих из нескольких взаимодействующих компонентов.
Параллельная композиция процессов $P_1 = (S_1, s_1^0, R_1)$ и $P_2 = (S_2, s_2^0, R_2)$ – это процесс

$$P_1 \mid P_2 = (S, s^0, R) \tag{5}$$

компоненты которого определяются следующим образом:

- $S \stackrel{\text{def}}{=} S_1 \times S_2 \stackrel{\text{def}}{=} \{(s_1, s_2) \mid s_1 \in S_1, s_2 \in S_2\}$, $s^0 \stackrel{\text{def}}{=} (s_1^0, s_2^0)$
- для
  - каждого перехода $s_1 \xrightarrow{a} s_1'$ из $R_1$, и
  - каждого состояния $s \in S_2$

  $R$ содержит переход $(s_1, s) \xrightarrow{a} (s_1', s)$
- для
  - каждого перехода $s_2 \xrightarrow{a} s_2'$ из $R_2$, и
  - каждого состояния $s \in S_1$

  $R$ содержит переход $(s, s_2) \xrightarrow{a} (s, s_2')$
- для каждой пары переходов с комплементарными метками

$$s_1 \xrightarrow{a} s_1' \quad \in R_1, \quad s_2 \xrightarrow{\overline{a}} s_2' \quad \in R_2$$

$R$ содержит переход $(s_1, s_2) \xrightarrow{\tau} (s_1', s_2')$

# Ограничение

Пусть $P = (S, s^0, R)$ некоторый процесс, и $L$ – произвольное подмножество множества $Names$.
**Ограничением** $P$ по $L$ называется процесс

$$P \setminus L = (S, s^0, R')$$

который получается из $P$ удалением тех переходов, которые имеют метки с именами из $L$, т.е.

$$R' \stackrel{\text{def}}{=} \left\{ \left( s \xrightarrow{a} s' \right) \in R \;\middle|\; \begin{array}{l} a = \tau, \quad \text{или} \\ name(a) \notin L \end{array} \right\}$$

Операция ограничения используется, как правило, совместно с операцией параллельной композиции для представления таких процессов, которые

- состоят из нескольких компонентов, и
- взаимодействие между этими компонентами должно удовлетворять некоторым ограничениям.

# Переименование

Следующая операция, которую мы рассмотрим - это унарная операция **переименования**.
Для задания этой операции необходимо определить функцию

$$f : Names \to Names$$

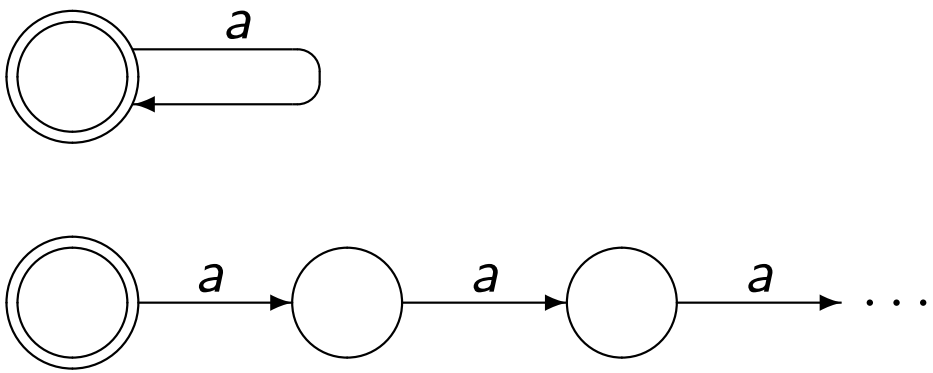
называемую **переименованием**.
Действие данной операции на процесс $P$ заключается в изменении меток переходов:

- метки вида $\alpha?$ заменяются на $f(\alpha)?$, и
- метки вида $\alpha!$ заменяются на $f(\alpha)!$

Получившийся процесс обозначается знакосочетанием $P[f]$.

# Понятие эквивалентности процессов и связанные с ним задачи

Одно и то же поведение может быть представлено различными процессами. Например, рассмотрим два процесса:




Хотя у первого процесса всего одно состояние, а у второго множество состояний бесконечно, но эти процессы представляют одно и то же поведение, которое заключается в постоянном выполнении одного и того же действия $a$.

Представляет интерес поиск подходящего определения эквивалентности процессов, согласно которому процессы эквивалентны тогда и только тогда, когда они имеют одно и то же поведение. Следует принять во внимание, что невидимое действие $\tau$ рассматривается как несущественное. Мы считаем две трассы одинаковыми, если одна может быть получена из другой путем вставок и/или удалений невидимых действий $\tau$.

Пусть $P$ и $P'$ – некоторые процессы.

1. Знакосочетание

$$P \xrightarrow{\tau^*} P' \tag{6}$$

означает, что или $P = P'$ или существует последовательность процессов

$$P_1, \ldots, P_n \quad (n \geq 2)$$

таких, что

- $P_1 = P, \quad P_n = P'$
- для каждого $i = 1, \ldots, n-1$

$$P_i \xrightarrow{\tau} P_{i+1}$$

(6) можно интерпретировать как утверждение о том, что процесс $P$, может незаметным для наблюдателя образом превратиться в процесс $P'$.

2. Для каждого действия $a \in Act \setminus \{\tau\}$ знакосочетание

$$P \xrightarrow{a_\tau} P' \tag{7}$$

означает, что существуют процессы $P_1$ и $P_2$ со следующими свойствами:

$$P \xrightarrow{\tau^*} P_1 \ , \qquad P_1 \xrightarrow{a} P_2 \ , \qquad P_2 \xrightarrow{\tau^*} P'$$

(7) можно интерпретировать как утверждение о том, что процесс $P$, может

- некоторым образом эволюционировать, так, что внешним проявлением этой эволюции будет лишь исполнение действия $a$,
- после чего вести себя как процесс $P'$.

Если имеет место (7), то мы будем говорить, что процесс $P$ может **наблюдаемо выполнить** $a$, и после этого вести себя как $P'$.

Понятие наблюдаемой эквивалентности основано на следующем понимании эквивалентности процессов: если мы рассматриваем процессы $P_1$ и $P_2$ как эквивалентные, то должны быть выполнены следующие условия.

1. - Если один из этих процессов $P_i$ может незаметно превратиться в некоторый процесс $P_i'$,
   - то и другой процесс $P_j \quad (j \in \{1,2\} \setminus \{i\})$ тоже должен обладать способностью незаметно превратиться в некоторый процесс $P_j'$, который эквивалентен $P_i'$.
2. - Если один из этих процессов $P_i$ может
     - наблюдаемо выполнить некоторое действие $a \in Act \setminus \{\tau\}$,
     - и после этого вести себя как некоторый процесс $P_i'$
   - то и другой процесс $P_j \quad (j \in \{1,2\} \setminus \{i\})$ должен обладать способностью
     - наблюдаемо выполнить то же действие $a$,
     - после чего вести себя как некоторый процесс $P_j'$, который эквивалентен $P_i'$.

Используя обозначения (6) и (7), вышеприведенное неформально описанное понятие наблюдаемой эквивалентности можно выразить равносильным образом как симметричное бинарное отношение $\mu$ на множестве всех процессов, обладающее следующими свойствами.

(1) Если $(P_1, P_2) \in \mu$, и для некоторого процесса $P_1'$ верно утверждение $P_1 \xrightarrow{\tau} P_1'$, то должен существовать процесс $P_2'$, такой, что выполнены условия

$$P_2 \xrightarrow{\tau^*} P_2' \qquad (P_1', P_2') \in \mu \tag{8}$$

(2) Если $(P_1, P_2) \in \mu$, и для некоторого процесса $P_1'$ верно утверждение $P_1 \xrightarrow{a} P_1'$, то должен существовать процесс $P_2'$, такой, что выполнены условия

$$P_2 \xrightarrow{a_\tau} P_2' \qquad (P_1', P_2') \in \mu$$

Если процессы $P_1$ и $P_2$ наблюдаемо эквивалентны, то этот факт обозначается знакосочетанием $P_1 \approx P_2$.

# Пример описания и верификации процесса: планировщик

Предположим, что имеется $n$ процессов

$$P_1, \ldots, P_n \tag{9}$$

и для каждого $i = 1, \ldots, n$ среди действий, которые может выполнить $P_i$, есть два служебных действия:

- действие $\alpha_i$?, которое представляет собой сигнал

$$P_i \textit{ начинает работу} \tag{10}$$

- действие $\beta_i$?, которое представляет собой сигнал

$$P_i \textit{ заканчивает работу} \tag{11}$$

Мы предполагаем, что все имена

$$\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n \tag{12}$$

различны, и для каждого $i = 1, \ldots, n$ имена из

$$names(Act(P_i)) \setminus \{\alpha_i, \beta_i\}$$

не совпадают ни с одним именем из (12). Обозначим множество имен из (12) символом $L$.
Для каждого $i = 1, \ldots, n$ действия из множества

$$Act(P_i) \setminus \{\alpha_i?, \beta_i?\}$$

называются **собственными действиями** процесса $P_i$.
Произвольная трасса каждого процесса $P_i$ может содержать действия $\alpha_i?$ и $\beta_i?$ в любом количестве и в любом порядке.

Мы хотели бы создать новый процесс $P$, в котором все процессы $P_1$, $\ldots$, $P_n$ работали бы совместно, причем эта совместная работа должна подчиняться определенному режиму. Процесс $P$ должен иметь вид

$$P = (P_1 \mid \ldots \mid P_n \mid Sch) \setminus L$$

где процесс $Sch$ называется **планировщиком** и предназначен для задания требуемого режима работы процессов $P_1$, $\ldots$, $P_n$. Процесс $Sch$ должен выполнять только действия из множества

$$\{\alpha_1!, \ldots, \alpha_n!, \beta_1!, \ldots, \beta_n!\} \tag{13}$$

В силу определения процесса $P$, для каждого $i = 1, \ldots, n$

- каждое исполнение процессом $P_i \in (9)$ действия $\alpha_i?$ или $\beta_i?$ в составе процесса $P$ может произойти только одновременно с исполнением комплементарного действия процессом $Sch$, и
- исполнение этих действий будет невидимо за пределами процесса $P$.

Говоря неформально, каждый процесс $P_i$, работающий в составе процесса $P$, может начать или закончить какой-либо сеанс своей работы тогда и только тогда, когда планировщик $Sch$ разрешит ему это сделать.

Режим, которому должна подчиняться работа процессов $P_1, \ldots, P_n$, заключается в следующих двух условиях.

1. Для каждого $i = 1, \ldots, n$ произвольная трасса процесса $P_i$, работающего в составе процесса $P$, должна иметь вид

$$\alpha_i? \ \ldots \ \beta_i? \ \ldots \alpha_i? \ \ldots \ \beta_i? \ \ldots$$

где точки изображают собственные действия процесса $P_i$, т.е. функционирование процесса $P_i$ должно представлять собой последовательность сеансов вида $\alpha_i? \ \ldots \ \beta_i? \ \ldots$, где каждый сеанс

- начинается с сигнала $\alpha_i?$ о начале сеанса
- далее выполняются некоторые собственные действия процесса $P_i$
- затем производится сигнал $\beta_i?$ о завершении сеанса,
- после чего процесс $P_i$ может производить некоторые собственные действия (например, связанные с подготовкой к следующему сеансу).

2. Процессы $P_i$ должны начинать новые сеансы только по очереди в циклическом порядке, т.е.

- сначала может начать свой первый сеанс только процесс $P_1$
- потом может начать свой первый сеанс процесс $P_2$
- ...
- затем может начать свой первый сеанс процесс $P_n$
- после этого может начать свой второй сеанс процесс $P_1$
- затем может начать свой второй сеанс процесс $P_2$
- и т.д.

Отметим, что мы не требуем, чтобы каждый процесс $P_i$ получал разрешение начать свой $k$-й сеанс только после того, как предыдущий процесс $P_{i-1}$ завершит свой $k$-й сеанс. Однако мы требуем, чтобы каждый процесс $P_i$ получал разрешение начать новый сеанс, только если он выполнил действие $\beta_i$?, сигнализирующее о завершении своего предыдущего сеанса.

Исполнение собственных действий процессами $P_i$ может производиться в произвольном порядке, в том числе допускается и взаимодействие между этими процессами во время их работы в составе процесса $P$.

Описанный режим можно равносильным образом выразить в виде следующих двух условий на произвольную трассу $tr \in Tr(Sch)$.
В формулировке этих условий мы будем использовать следующее обозначение: если $tr \in Tr(Sch)$ и $M \subseteq Act$, то знакосочетание $tr\,|_M$ обозначает последовательность действий, получаемую из $tr$ удалением всех действий, не принадлежащих $M$.

Условия, выражающие описанный выше режим, имеют следующий вид:

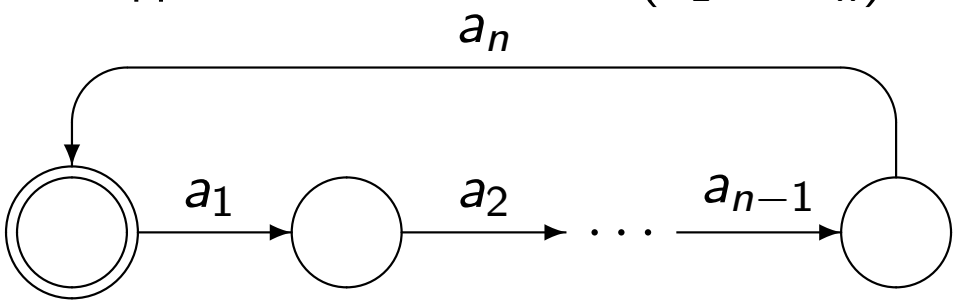
$$\begin{aligned} &\forall\ tr \in Tr(Sch),\ \forall\ i = 1, \ldots, n \\ &tr\,|_{\{\alpha_i, \beta_i\}} \ = \ (\alpha_i!\ \beta_i!\ \alpha_i!\ \beta_i!\ \alpha_i!\ \beta_i!\ \ldots) \end{aligned} \tag{14}$$

и

$$\begin{aligned} &\forall\ tr \in Tr(Sch) \\ &tr\,|_{\{\alpha_1, \ldots, \alpha_n\}} \ = \ (\alpha_1!\ \ldots\ \alpha_n!\ \alpha_1!\ \ldots\ \alpha_n!\ \ldots) \end{aligned} \tag{15}$$

Данные условия можно выразить равносильным образом в виде условия наличия наблюдаемой эквивалентности между некоторыми процессами. Чтобы определить эти процессы, мы введем вспомогательные обозначения.

1. Пусть $a_1 \ldots a_n$ – некоторая последовательность действий из $Act$. Тогда знакосочетание $(a_1 \ldots a_n)^*$ обозначает процесс




2. Пусть заданы процесс $P$, и множество действий

$$\{a_1, \ldots, a_k\} \subseteq Act \setminus \{\tau\} \tag{16}$$

Знакосочетание

$$hide\ (P, a_1, \ldots, a_k) \tag{17}$$

обозначает процесс $(\ P \mid (\ \overline{a_1}\ )^* \mid \ldots \mid (\ \overline{a_k}\ )^*\ ) \setminus names(\{a_1, \ldots, a_k\})$

Процесс (17) можно рассматривать как процесс, получаемый из $P$ заменой меток переходов: все метки переходов в $P$, принадлежащие множеству (16), заменяются на $\tau$.

Используя введенные обозначения, условие (14) можно выразить следующим образом: для каждого $i = 1, \ldots n$

$$hide \left( \begin{array}{ll} Sch, & \alpha_1!, \ldots, \alpha_{i-1}!, \alpha_{i+1}!, \ldots, \alpha_n! \\ & \beta_1!, \ldots, \beta_{i-1}!, \beta_{i+1}!, \ldots, \beta_n! \end{array} \right) \approx (\alpha_i!.\ \beta_i!)^* \tag{18}$$

Условие (15) можно выразить следующим образом:

$$hide\ (Sch, \beta_1!, \ldots, \beta_n!) \approx (\alpha_1!.\ \ldots \alpha_n!)^* \tag{19}$$

Нетрудно заметить, что существует несколько планировщиков, удовлетворяющих данным условиям. Например, данным условиям удовлетворяют следующие планировщики:

- $Sch\ =\ (\alpha_1!\ \beta_1!\ \ldots\ \alpha_n!\ \beta_n!)^*$
- $Sch\ =\ (\alpha_1!\ \ldots\ \alpha_n!\ \beta_1!\ \ldots\ \beta_n!)^*$

Однако такие планировщики налагают слишком жесткие ограничения на работу процессов $P_1, \ldots, P_n$.
Нам хотелось бы, чтобы планировщик допускал максимальную свободу в поведении процессов $P_1, \ldots, P_n$ в составе процесса $P$. Это означает, что если в какой-либо момент времени

- какой-либо процесс $P_i$ имеет намерение выполнить действие $a \in \{\alpha_i?, \beta_i?\}$, и
- это намерение процесса $P_i$ не противоречит описанному выше режиму

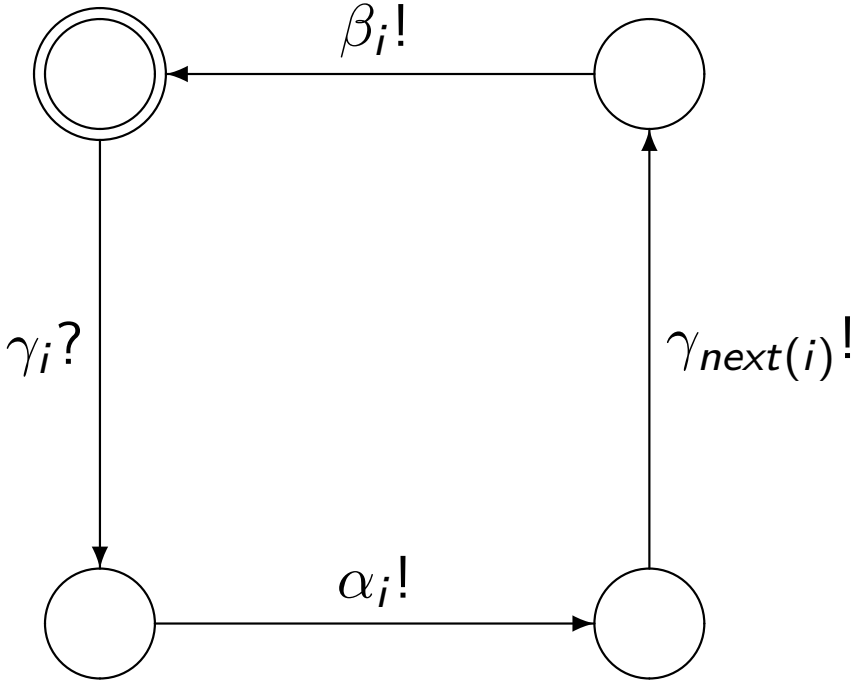
то планировщик не должен отказывать процессу $P_i$ в возможности выполнить это действие в текущий момент времени, т.е. среди действий планировщика, которые он может выполнить в текущий момент времени, должно присутствовать действие $\overline{a}$ (не факт, что именно это действие будет выполнено в текущий момент времени, но среди возможных действий оно должно присутствовать).

Для решения поставленной задачи мы определим планировщик $Sch$ путем задания некоторого выражения, которое определяет его в терминах композиции процессов достаточно простого вида.
В описании планировщика $Sch$ мы будем использовать новые вспомогательные имена $\gamma_1, \ldots, \gamma_n$. Обозначим множество этих имен символом $\Gamma$.
Процесс $Sch$ определяется следующим образом:

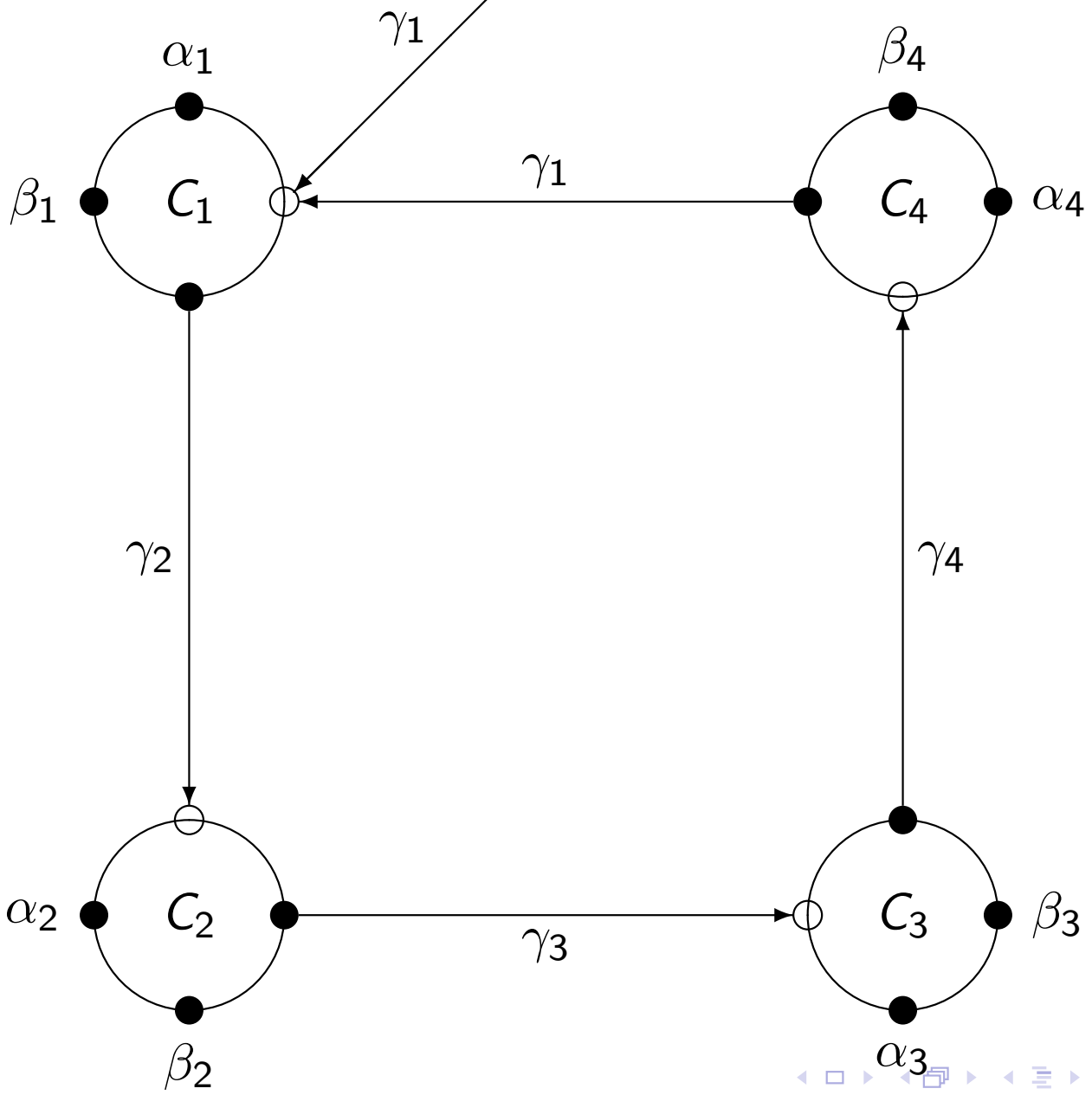
$$Sch \stackrel{\text{def}}{=} (Start \mid C_1 \mid \ldots \mid C_n) \setminus \Gamma \tag{20}$$

где

- $Start \stackrel{\text{def}}{=} \gamma_1!.\,\mathbf{0}$
- для каждого $i = 1, \ldots, n$ процесс $C_i$ имеет вид



Потоковый граф процесса $Sch$ в случае $n = 4$ имеет следующий вид:



Приведем неформальное пояснение функционирования процесса $Sch$. Назовем процессы $C_1, \ldots, C_n$, участвующие в определении планировщика $Sch$, **циклерами**. Циклер $C_i$ называется

- **выключенным**, если он находится в своем начальном состоянии, и
- **включенным**, если он находится не в начальном состоянии.

Процесс $Start$ включает первый циклер $C_1$ и после этого "умирает". Каждый циклер $C_i$ отвечает за работу процесса $P_i$. Циклер $C_i$

- включает следующий циклер $C_{next(i)}$ после того, как он дал разрешение процессу $P_i$ начать очередной сеанс работы, и
- выключается после того, как он дал разрешение процессу $P_i$ закончить очередной сеанс работы.

Докажем, что процесс (20) удовлетворяет условию (19) (проверку условия (18) мы опустим).
Согласно определению процесса (17), условие (19) имеет вид

$$(Sch \,|\, (\beta_1?)^* \,|\, \ldots \,|\, (\beta_n?)^*) \setminus B \approx (\alpha_1!.\ \ldots \alpha_n!)^* \tag{21}$$

где $B = \{\beta_1, \ldots, \beta_n\}$.
Обозначим $Sch' \stackrel{\text{def}}{=}$ левая часть (21).
Докажем, что

$$Sch' \approx \tau.\alpha_1!.\ \ldots \alpha_n!.\ Sch' \tag{22}$$

Отсюда, по свойству единственности (с точностью до $\approx$) решения уравнения

$$X \ = \ \tau.\alpha_1!.\ \ldots \alpha_n!.\ X$$

будет следовать соотношение

$$Sch' \ \approx \ (\tau\ \alpha_1!\ \ldots \alpha_n!\ )^*$$

из которого, в свою очередь, следует (21).

Мы будем преобразовывать левую часть соотношения (22) так, чтобы получилась правая часть этого соотношения.
Для этого мы будем использовать следующие свойства операций на процессах:

- $P \setminus L = P$, если $L \cap names(Act(P)) = \emptyset$
- $(P_1 \mid P_2) \setminus L = (P_1 \setminus L) \mid (P_2 \setminus L)$, если

$$L \cap names(Act(P_1) \cap \overline{Act(P_2)}) = \emptyset$$

- $(P \setminus L_1) \setminus L_2 = P \setminus (L_1 \cup L_2) = (P \setminus L_2) \setminus L_1$

Используя данные свойства, можно преобразовать левую часть соотношения (22) следующим образом.

$$Sch' = (Sch \mid (\beta_1?)^* \mid \ldots \mid (\beta_n?)^*) \setminus B = \tag{23}$$

где $C_i' = (C_i \mid (\beta_i?)^*) \setminus \{\beta_i\}$.

Заметим, что для каждого $i = 1, \ldots, n$ имеет место соотношение

$$C'_i \;\approx\; \gamma_i?.\, \alpha_i!.\, \gamma_{next(i)}!.\, C'_i \tag{24}$$

Действительно, по теореме о разложении

$$\begin{gathered} C'_i = \left(\left(\gamma_i?.\, \alpha_i!.\, \gamma_{next(i)}!.\, \beta_i!.\, C_i\right) \mid (\beta_i?)^*\right) \setminus \{\beta_i\} \sim \\ \sim \gamma_i?.\, \alpha_i!.\, \gamma_{next(i)}!.\, \tau.C'_i \;\approx\; \text{правая часть (24)} \end{gathered}$$

Используя данное замечание и теорему о разложении, цепочку равенств (23) можно продолжить следующим образом:

$$
\begin{aligned}
&(Start \mid C_1' \mid C_2' \mid \ldots \mid C_n') \setminus \Gamma \approx \\
&\approx (\underbrace{\gamma_1!.\,\mathbf{0}}_{=Start} \mid \underbrace{\gamma_1?.\,\alpha_1!.\,\gamma_2!.\,C_1'}_{\approx C_1'} \mid C_2' \mid \ldots \mid C_n') \setminus \Gamma \sim \\
&\sim \tau.\,(\mathbf{0} \mid \alpha_1!.\,\gamma_2!.\,C_1' \mid C_2' \mid \ldots \mid C_n') \setminus \Gamma = \\
&= \tau.\,(\alpha_1!.\,\gamma_2!.\,C_1' \mid C_2' \mid \ldots \mid C_n') \setminus \Gamma \sim \\
&\sim \tau.\,\alpha_1!.\,(\gamma_2!.\,C_1' \mid C_2' \mid \ldots \mid C_n') \setminus \Gamma \approx \\
&\approx \tau.\,\alpha_1!.\,(\gamma_2!.\,C_1' \mid \underbrace{\gamma_2?.\,\alpha_2!.\,\gamma_3!.\,C_2'}_{\approx C_2'} \mid \ldots \mid C_n') \setminus \Gamma \sim \\
&\sim \tau.\,\alpha_1!.\,\tau.\,(C_1' \mid \alpha_2!.\,\gamma_3!.\,C_2' \mid \ldots \mid C_n') \setminus \Gamma \sim \ldots \sim \\
&\sim \tau.\,\alpha_1!.\,\tau.\,\alpha_2!.\,\ldots \tau.\,\alpha_n!.\,(C_1' \mid \ldots \mid \gamma_1!.\,C_n') \setminus \Gamma \approx \\
&\approx \tau.\,\alpha_1!.\,\ldots \alpha_n!.\,(C_1' \mid \ldots \mid \gamma_1!.\,C_n') \setminus \Gamma \approx \\
&\approx \tau.\,\alpha_1!.\,\ldots \alpha_n!.\,(\underbrace{\gamma_1?.\,\alpha_1!.\,\gamma_2!.\,C_1'}_{\approx C_1'} \mid \ldots \mid \gamma_1!.\,C_n') \setminus \Gamma \sim \\
&\sim \tau.\,\alpha_1!.\,\ldots \alpha_n!.\,\underbrace{\tau.\,(\alpha_1!.\,\gamma_2!.\,C_1' \mid \ldots \mid C_n') \setminus \Gamma}
\end{aligned}
\tag{25}
$$

Выражение, подчеркнутое фигурной скобкой в последней строке данной цепочки, совпадает с выражением в четвертой строке этой цепочки, которое, в свою очередь, находится в отношении $\approx$ с $Sch'$. Мы получили, что последнее выражение в цепочке (25)

- находится в отношении $\approx$ с правой частью соотношения (22), и
- с другой стороны, данное выражение находится в отношении $\approx$ с левой частью соотношения (22)

Таким образом, соотношение (22) доказано. ■

# Вычисление квадрата

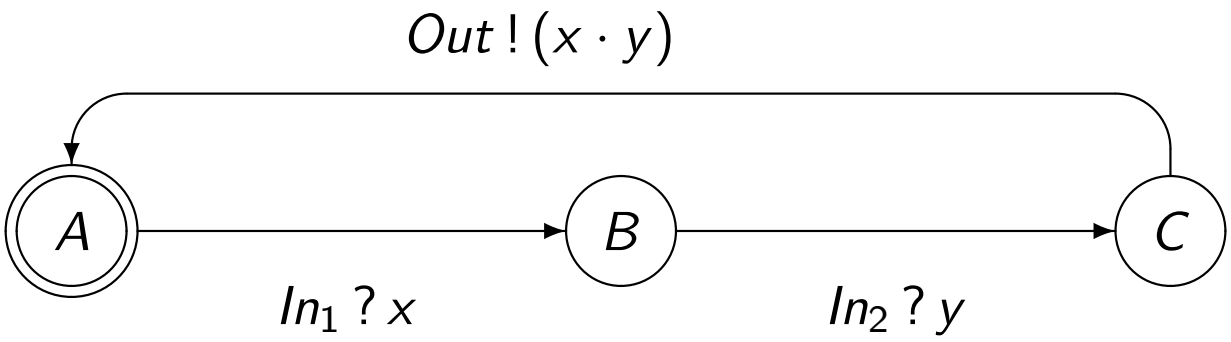
Предположим, что мы имеем систему “умножитель”, у которой есть

- два входных порта с именами $In_1$ и $In_2$, и
- один выходной порт с именем $Out$.

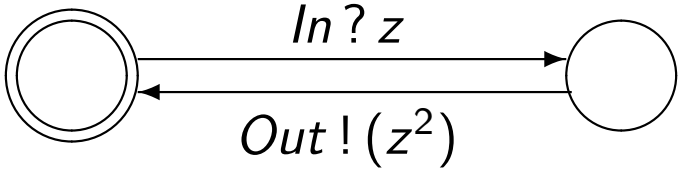
Работа умножителя заключается в том, что он периодически

- получает на свои входные порты два значения, и
- выдает на выходном порте их произведение.

Поведение умножителя описывается процессом $Mul$:



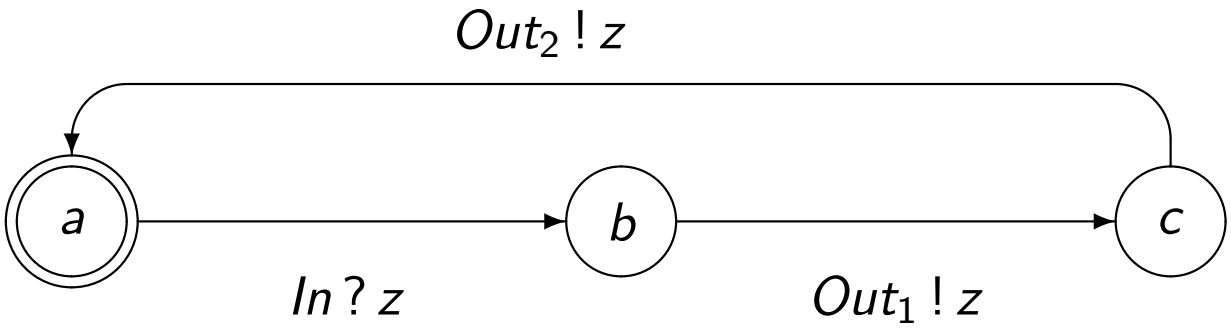
Используя этот умножитель, мы хотим построить систему “вычислитель квадрата”, поведение которой описывается процессом *Square_Spec*:



Искомую систему мы построим как композицию

- вспомогательной системы “дубликатор”, имеющей
  - входной порт $In$, и
  - выходные порты $Out_1$ и $Out_2$

  поведение которой описывается процессом $Dup$:



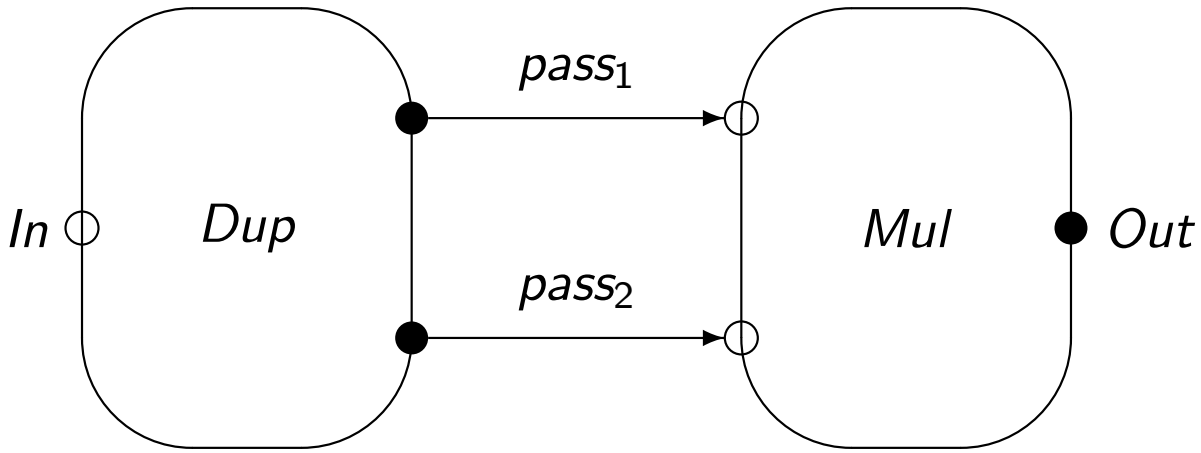

  т.е. дубликатор копирует свой вход на два выхода, и

- умножителя, который будет получать на свои входные порты те значения, которые будет выдавать дубликатор.

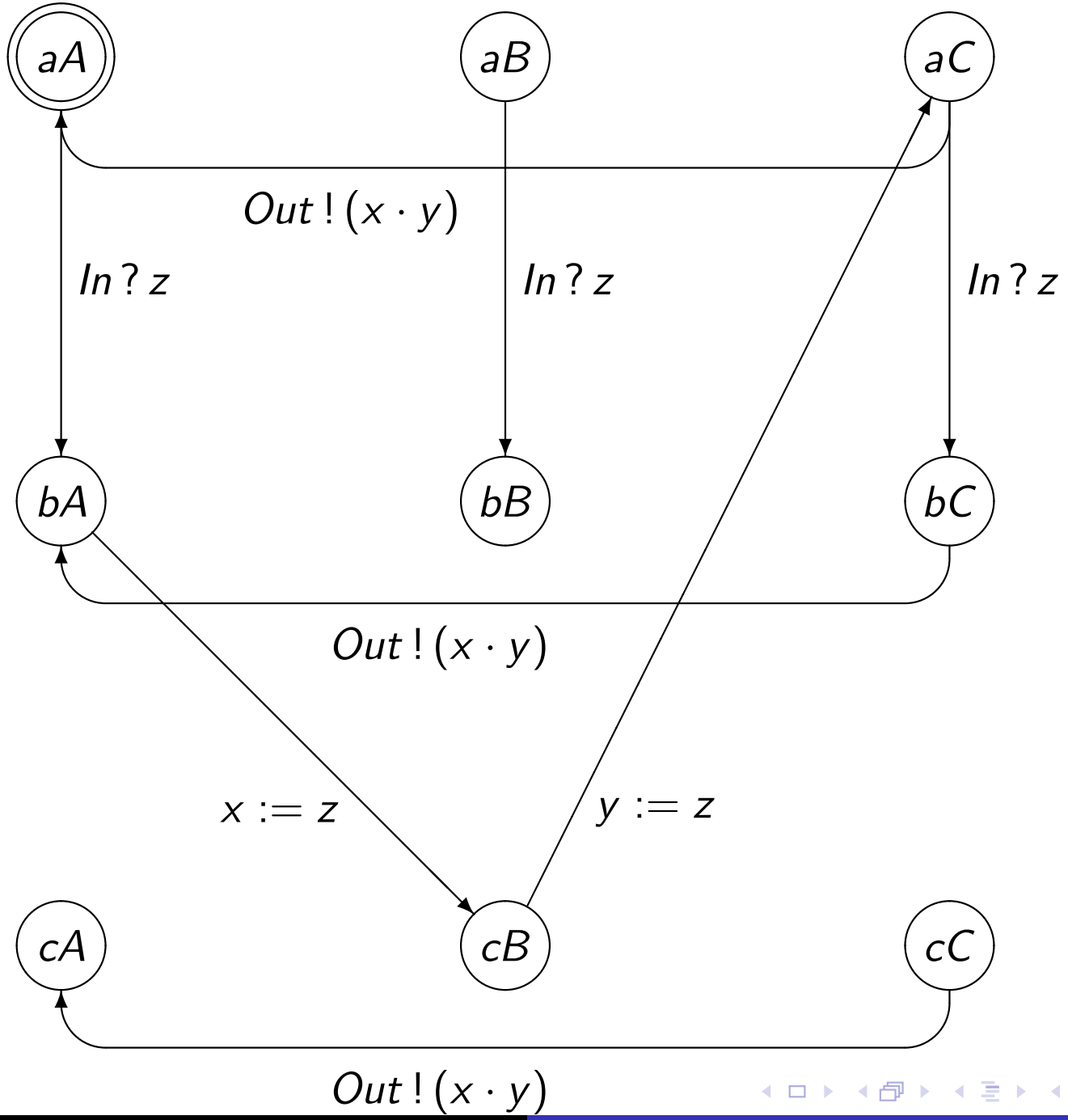
Процесс *Square*, соответствующий такой композиции, определяется следующим образом:

$$
\begin{aligned}
&Square \stackrel{\text{def}}{=} \\
&\stackrel{\text{def}}{=} \left( \begin{array}{l} Dup[pass_1/Out_1, pass_2/Out_2] \ \mid \\ \mid Mul[pass_1/In_1, pass_2/In_2] \end{array} \right) \setminus \{pass_1, pass_2\}
\end{aligned}
$$

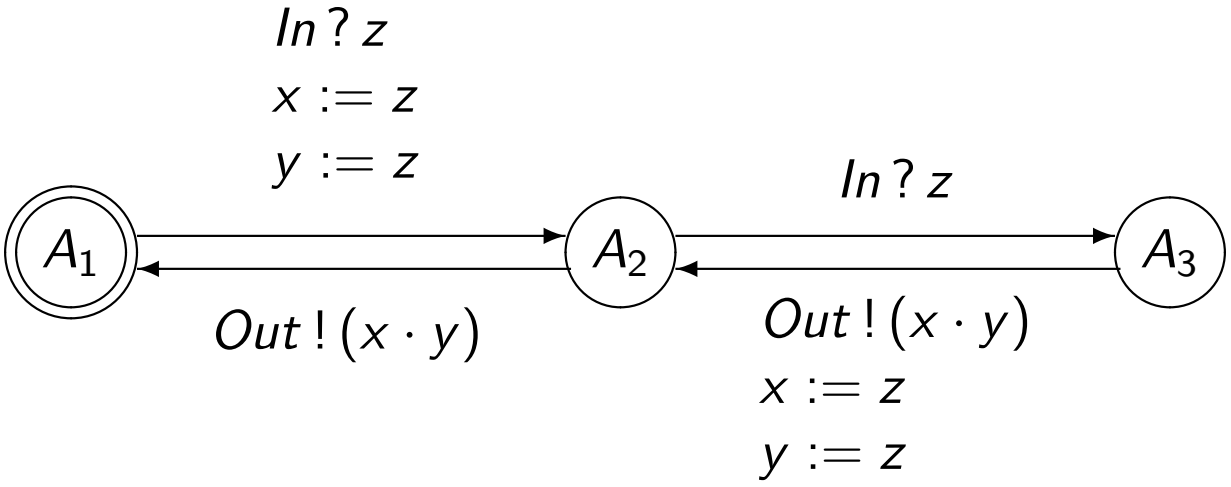
Потоковый граф процесса *Square* имеет вид



Однако процесс *Square* не соответствует своей спецификации *Square_Spec*. Данный факт нетрудно установить при помощи построения графового представления процесса *Square*, который, согласно определению операций параллельной композиции, ограничения и переименования, имеет следующий вид:


aA
aB
aC
Out ! (x · y)
In ? z
In ? z
In ? z
bA
bB
bC
Out ! (x · y)
x := z
y := z
cA
cB
cC
Out ! (x · y)

После редукции данного процесса мы получим диаграмму




(26)

из которой видно, что

- процесс *Square* может последовательно совершить два действия ввода, не выполняя между ними действия вывода,
- а процесс *Square_Spec* этого сделать не может.

# Параллельные программы

**Параллельные программы** – это программы, предназначенные для исполнения на многопроцессорных вычислительных системах (МПВС).

Проблема разработки корректных и безопасных параллельных программ представляет в настоящее время исключительно высокую актуальность. Формальное обоснование свойств корректности и безопасности параллельных программ (называемое также **верификацией** параллельных программ) является сложной математической задачей.

Существующие методы решения данной задачи пригодны лишь для достаточно ограниченного класса параллельных программ.

Одним из наиболее широко используемых языков для описания параллельных программ является язык MPI (Message Passing Interface).

# Содержание доклада

В настоящей работе вводится новая математическая модель параллельных программ, на основе которой можно решать задачи верификации параллельных программ, представленных на некотором подмножестве MPI.

Введенная модель иллюстрируется применением к решению задачи верификации MPI-программы умножения матриц.

Наиболее важной особенностью предлагаемого подхода является возможность его применения для верификации параллельных программ, которые могут порождать неопределенное число процессов.

I am aware of one approach to proving the correctness of an MPI program for any number of processes: the tool is called ParTypes and there were a few papers written about it, here is one:

https://dl.acm.org/doi/10.1145/2858965.2814302

It requires the user to provide a protocol which specifies the communication pattern of an execution. Unfortunately, it has some limitations. In particular, it does not work for wildcard receives, which means it can't be applied to the matrix multiplication program you asked about.

There are several other approaches using symbolic execution but all of them, to the best of my knowledge, require some (small) bound on the number of processes.

# MPI программы

**MPI (Message Passing Interface)** представляет собой набор функций, типов и констант, позволяющих создавать программы (называемые **MPI–программами**) для МПВС.

Каждая MPI–программа представляет собой программу на языке Си, в которой могут использоваться функции, типы и константы из MPI.

Выполнение MPI–программы на МПВС заключается в том, что на каждом из узлов, входящих в эту систему, порождается вычислительный процесс, соответствующей этой MPI–программе.

Все процессы, порожденные MPI–программой на узлах МПВС, функционируют параллельно, и могут обмениваться информацией друг с другом посредством **передачи сообщений**.

Каждый из процессов, порожденных MPI–программой, имеет номер (называемый **рангом**), являющийся числом из множества $\{0, \ldots, m-1\}$, где $m$ – количество процессов, порожденных MPI–программой. Процесс с рангом 0 называется **корневым процессом**.

# MPI-программа умножения матриц

Задача умножения матриц заключается в том, чтобы по заданным матрицам $A$ и $B$ вычислить их произведение $C = AB$.

Неформально работу излагаемой в этом пункте MPI–программы для умножения матриц $A$ и $B$ можно описать следующим образом. Мы будем называть корневой процесс **менеджером**, а остальные процессы – **работниками**.

Работа менеджера состоит из следующих действий:

- рассылка второй матрицы ($B$) всем работникам,
- назначение задач работникам, и
- прием результатов от работников.

Каждая задача работнику заключается в вычислении одной строки матрицы $C = AB$.

# Менеджер и работники

Менеджер назначает задачу путем посылки работнику сообщения, содержащего одну строку матрицы $A$. Тег этого сообщения равен номеру посылаемой строки.

Как только менеджер получает от работника результат (т.е. сообщение с вычисленной строкой произведения, его тег равен номеру этой строки), он посылает этому работнику

- новую задачу (если еще есть неназначенные задачи), или
- сообщение с тегом 0 (если неназначенных задач нет).

# Спецификация MPI программы умножения матриц

Спецификация MPI- программы умножения матриц представляет собой следующее утверждение: после завершения выполнения этой программы должно быть верно утверждение $C = AB$, которое эквивалентно утверждению

$$\forall\, i = 1, \ldots, N \quad C_i = A_i B. \tag{27}$$

# MPI-программа умножения матриц

```
#define comm MPI_COMM_WORLD

int main(int argc, char *argv[])
{ int rank, nprocs, i, j;
  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(comm, &nprocs);
  MPI_Comm_rank(comm, &rank);

  if (rank == 0)
  { int count;
    double a[N][L], b[L][M], c[N][M], tmp[M];

    input(a, b);
    MPI_Bcast(b, L*M, MPI_DOUBLE, 0, comm);
    for (count = 0;
         count < nprocs-1 && count < N;
         count++)
      MPI_Send(&a[count][0], L, MPI_DOUBLE,
               count+1, count+1, comm);
    for (i = 0; i < N; i++)
    { MPI_Recv(tmp, M, MPI_DOUBLE, MPI_ANY_SOURCE, MPI_ANY_TAG, comm, &status);
      for (j = 0; j < M; j++)
        c[status.MPI_TAG-1][j] = tmp[j];
      if (count < N)
      { MPI_Send(&a[count][0], L, MPI_DOUBLE, status.MPI_SOURCE, count+1, comm);
        count++;
      }
    }
```

# MPI-программа умножения матриц

```
    for (i = 1; i < nprocs; i++)
      MPI_Send(NULL, 0, MPI_INT, i, 0, comm);
    output (c);
  }

  else
  { double b[L][M], in[L], out[M];

    MPI_Bcast(b, L*M, MPI_DOUBLE, 0, comm);
    while (1)
    { MPI_Recv(in, L, MPI_DOUBLE, 0, MPI_ANY_TAG, comm, &status);
      if (status.MPI_TAG == 0) break;
      vecmat(in, b, out);
      MPI_Send(out, M, MPI_DOUBLE, 0, status.MPI_TAG, comm);
    }
  }

  MPI_Finalize();
  return 0;
}
```

# Действия

**Элементарные действия (ЭД)** – это записи следующих видов:

$$c!e, \quad c?e, \quad e := e', \quad [\![\varphi]\!], \quad \text{где } c \in \mathcal{C},\ e, e' \in \mathcal{E}, \tau_e = \tau_{e'}, \varphi \in \mathcal{B},$$

которые называются **посылкой** сообщения $e$ в канал $c$, **приемом** сообщения $e$ из канала $c$, **присваиванием**, и **условным переходом**, соответственно.

**Действие** – это конечная последовательность ЭД, в которой имеется не более одного посылки или приема. Действие называется **посылкой** или **приемом**, если одно из входящих в него ЭД – посылка или прием, соответственно. Действие, не являющееся посылкой или приемом, называется **внутренним действием**.

# Понятие последовательного процесса

**Последовательный процесс (ПП)** – это тройка $(P, X, \varphi)$, компоненты которой имеют следующий смысл:

- $P$ – граф с выделенной вершиной $P^0$ (называемой **начальной вершиной**), каждому ребру которого сопоставлена метка $\alpha \in \mathcal{A}$,
- $X \subseteq \mathcal{X}$ – множество **входных переменных** ПП $P$, и
- $\varphi \in \mathcal{B}$ – **начальное условие** ПП $P$.

ПП является формальным описанием поведения динамической системы, работа которой заключается в последовательном выполнении действий, связанных с посылкой сообщений в каналы или приемом сообщений из каналов, а также с изменением значений невходных переменных.

**Состояние** ПП $P$ – это пара

$$s = (\theta^s, \{[c]^s \mid c \in \mathcal{D}_{\mathbf{C}}\}), \tag{28}$$

компоненты которой интерпретируются следующим образом:

- $\theta^s \in \Theta(\mathcal{X}_P)$ – связывание в $s$, причем $\forall\, x \in \mathcal{X}_P \;\; \mathcal{X}_{x^{\theta^s}} = \emptyset$,
- $\forall\, c \in \mathcal{D}_{\mathbf{C}} \;\; [c]^s \in \mathcal{D}^*$ – очередь непрочитанных значений в канале $c$ в состоянии $s$ ($[c]^s$ называется **содержимым** канала $c$ в $s$).

# Переходы в последовательных процессах

Утверждение $s \xrightarrow{\alpha} s'$ в том случае, когда $\alpha$ – ЭД, определяется отдельно для каждого возможного вида ЭД $\alpha$:

(a) если $\alpha = c!e$, то $\theta^{s'} = \theta^s$, и

$$[c^s]^{s'} = ([c^s]^s, e^s), \ \ \forall\, c' \in \mathcal{D}_{\mathbf{C}} \setminus \{c^s\} \ \ [c']^{s'} = [c']^s,$$

(b) если $\alpha = c?e$, то $[c^s]^s \neq \emptyset$ и

$$\begin{aligned} &\exists\, \theta \in \Theta(\hat{X}_P) : (e^\theta)^s = \mathit{head}([c^s]^s), \theta^{s'} = \theta\theta^s, \\ &[c^s]^{s'} = \mathit{tail}([c^s]^s), \ \ \forall\, c' \in \mathcal{D}_{\mathbf{C}} \setminus \{c^s\} \ \ [c']^{s'} = [c']^s \end{aligned}$$

(c) если $\alpha = (e := e')$, то

$$\begin{aligned} &\exists\, \theta \in \Theta(\hat{X}_P) : (e^\theta)^s = (e')^s, \theta^{s'} = \theta\theta^s, \\ &\forall\, c \in \mathcal{D}_{\mathbf{C}} \ \ [c]^{s'} = [c]^s, \end{aligned} \tag{29}$$

(d) если $\alpha = [\![\varphi]\!]$, то $\varphi^s = 1$, $\theta^{s'} = \theta^s$, и верно (29),

# Распределенные процессы

**Распределенный процесс (РП)** – это семейство ПП

$$\mathcal{P} = \{P_i \mid i \in I\}, \tag{30}$$

где компоненты семейства $\{\hat{X}_{P_i} \mid i \in I\}$ дизъюнктны и не пересекаются с множеством $X_{\mathcal{P}} \stackrel{\text{def}}{=} \bigcup_{i \in I} X_{P_i}$

# Понятие состояния распределенного процесса

**Состоянием** РП $\mathcal{P} = \{P_i \mid i \in I\}$ называется пара

$$s = (\theta^s, \{c^s \mid c \in \mathcal{D}_{\mathbf{C}}\}), \tag{31}$$

где $\theta^s \in \Theta(\mathcal{X}_{\mathcal{P}})$, и $\forall\, i \in I$

$$s_i \stackrel{\text{def}}{=} (\theta_i^s, \{c^s \mid c \in \mathcal{D}_{\mathbf{C}}\}) \in \Sigma_{P_i}$$

где $\theta_i^s \in \Theta(\mathcal{X}_{P_i})$, $\forall\, x \in \mathcal{X}_{P_i}\ \ x^{\theta_i^s} = x^{\theta^s}$.
$\forall\, s \in \Sigma_{\mathcal{P}}$, $\forall\, e \in e \in \mathcal{E}(\mathcal{X}_{\mathcal{P}})$ будем обозначать значение $e^{\theta^s}$ записью $e^s$.

Пусть $\mathcal{P} = \{P_i \mid i \in I\}$ – РП. $\forall\, s \in \Sigma_{\mathcal{P}}$, $\forall\, i \in I$
$s_i \stackrel{\text{def}}{=} (\theta_i^s, \{c^s \mid c \in \mathcal{D}_{\mathbf{C}}\}) \in \Sigma_{P_i}$, where $\theta_i^s \in \Theta(\mathcal{X}_{P_i})$, $\forall\, x \in \mathcal{X}_{P_i}\ \ x^{\theta_i^s} = x^{\theta^s}$.

**Переходом** в РП $\mathcal{P}$, соответствующим действию $\alpha$ ПП $P_i$, называется произвольная пара состояний $s, s' \in \Sigma_{\mathcal{P}}$, такая, что

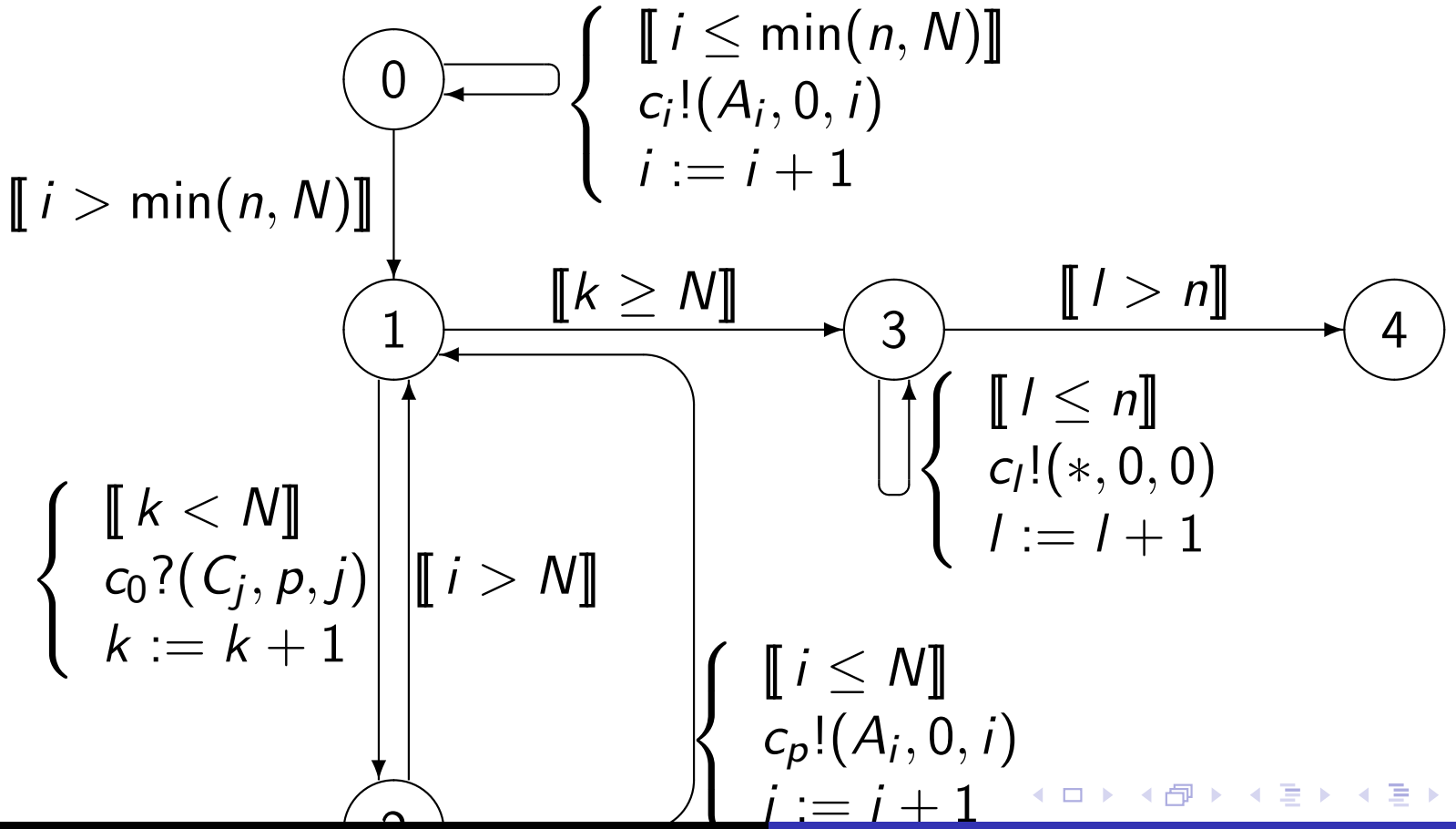
$$s_i \xrightarrow{\alpha} s_i',\ \forall\, i' \in I \setminus \{i\}\ \ s_{i'} \to s_{i'}'. \tag{32}$$

# Модель MPI-программы в виде РП

Определим РП $\mathcal{P}_\Pi = \{P_i \mid i \geq 0\}$, являющийся моделью MPI программы П.

Начальное условие: $\varphi_{P_0} = (N \geq 1) \wedge (i = 1) \wedge (k = 0) \wedge (l = 1)$.

ПП $P_0$ имеет вид

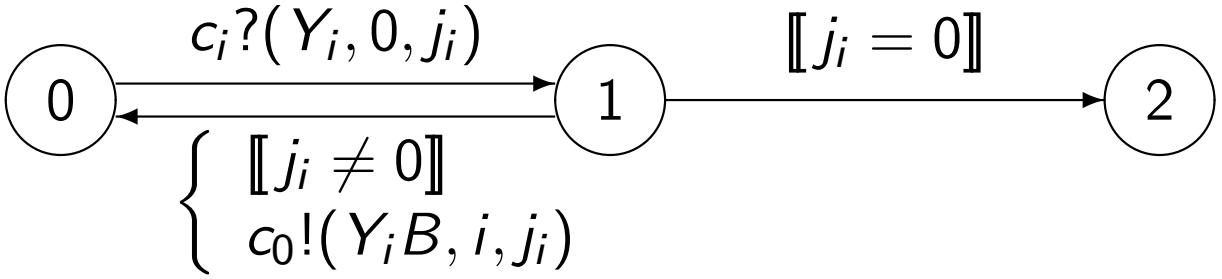



(33)

# ПП для работников

$P_i$ имеет следующие переменные: $X_{P_i} = \{B\}, \hat{X}_{P_i} = \{Y_i, j_i\}$, где значением $B$ является вторая матрица-сомножитель, и значениями $Y_i$ – строки действительных чисел. ПП

$P_i$ имеет следующий вид:




(34)

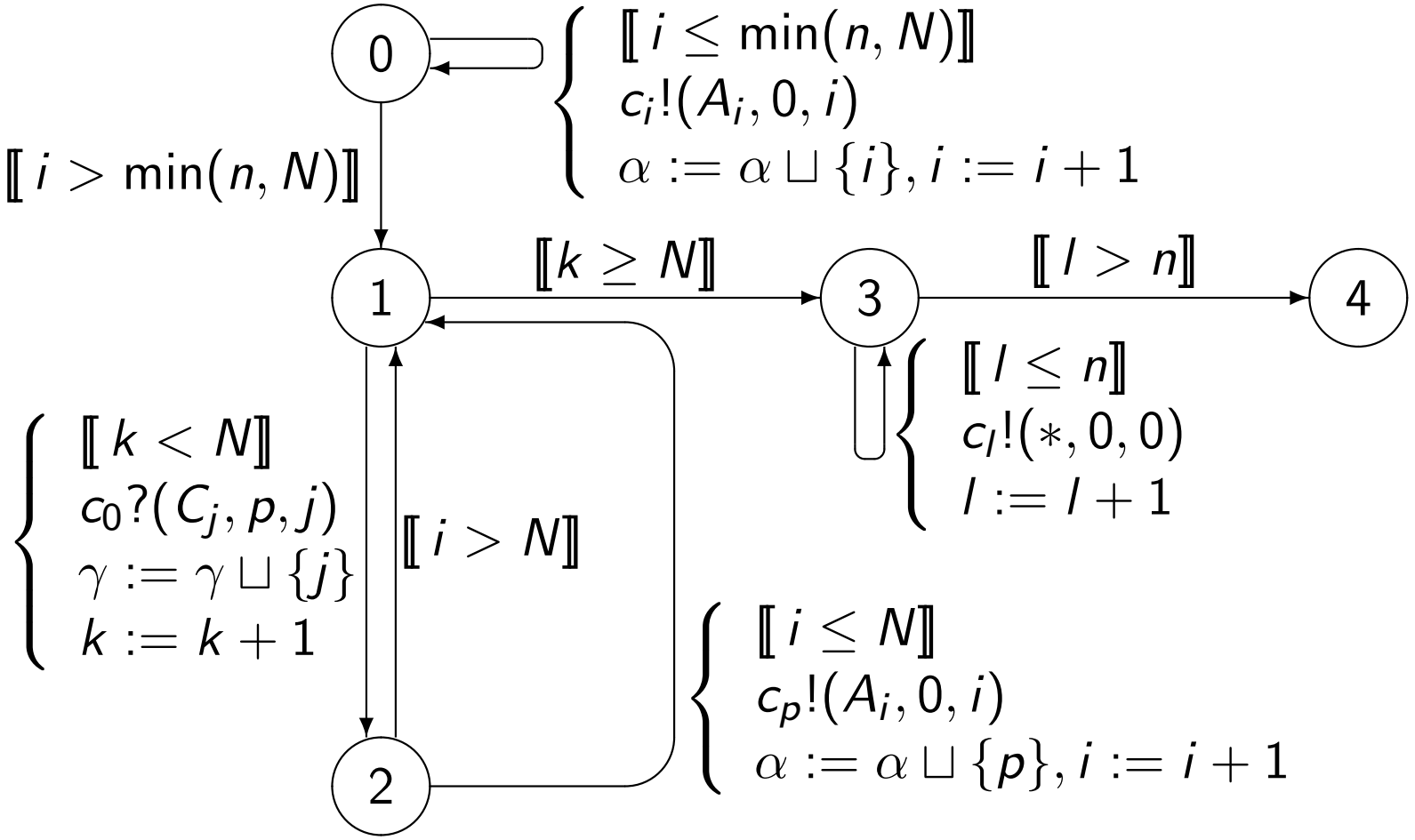
# Верификация распределенного процесса, моделирующего MPI-программу умножения матриц

Для доказательства утверждения о том, что РП $\mathcal{P}_\Pi$, удовлетворяет своей спецификации,(27), мы введем вспомогательные переменные $\alpha$, $\beta$, $\gamma$, (и связанные с ними действия, не влияющие на выполнение РП), значения которых будут иметь следующий смысл: их начальные значения равны $\emptyset$, и $\forall\, s \in \Sigma_{\mathcal{P}_\Pi}$

- $\alpha^s \subseteq \{1, \ldots, n\}$, $\alpha^s$ состоит из номеров каналов из $\{c_1, \ldots, c_n\}$ с непустым содержимым в состоянии $s$,
- $\beta^s \subseteq \{1, \ldots, N\}$, $\beta^s$ состоит из номеров строк матрицы $A$, для которых в состоянии $s$ вычисляется их произведение на $B$,
- $\gamma^s \subseteq \{1, \ldots, N\}$, $\gamma^s$ равно множеству номеров всех строк, которые $P_0$ записал в $C$ до момента прихода РП $\mathcal{P}_\Pi$ в состояние $s$.

# Дополненный ПП $P_0$

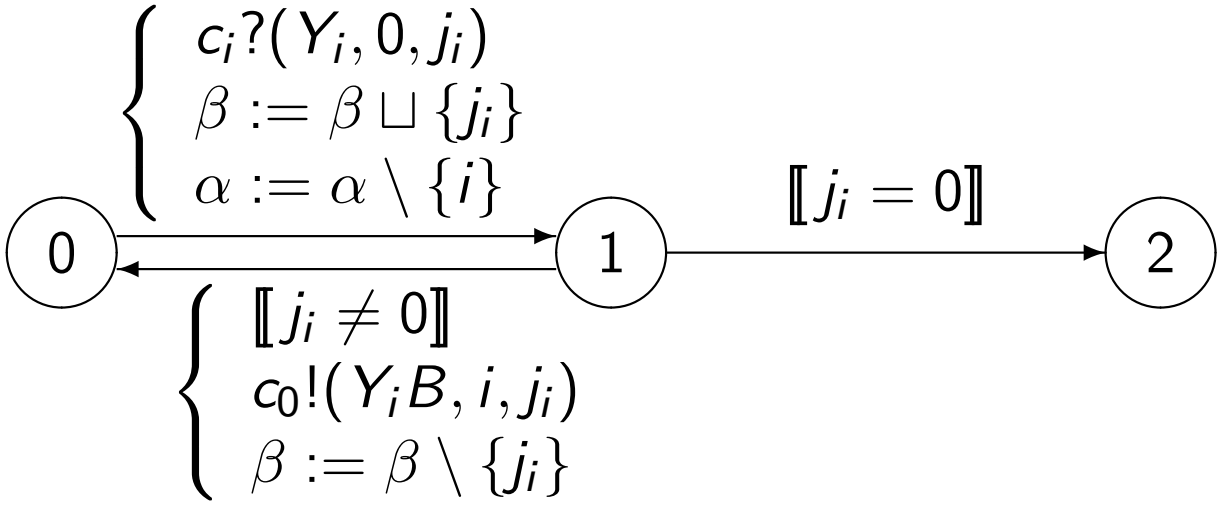
Дополненный ПП $P_0$ имеет вид




(35)

# Дополненный $P_i$

$\forall\, i = 1, \ldots, n$ дополненный ПП $P_i$ имеет вид




(36)

**Теорема 1**. $\forall\, s \in \Sigma_{\mathcal{P}_\Pi}$, если $at^s_{P_0} \neq 3, 4$, то

1. $\alpha^s \subseteq \{1, \ldots, i^s - 1\}$,
2. $i^s - 1 \leq N$
3. $|\gamma^s| = k^s \leq N$,
4. если $at^s_{P_0} = 1$ и $k^s < N$, то $k^s < i^s - 1$,
5. $[c_0]^s_2 \cap \alpha^s = \emptyset$,
6. $\forall\, i = 1, \ldots, n$
   1. $|[c_i]^s| = \begin{cases} 1, \text{ если } i \in \alpha^s, \text{ и} \\ 0 \text{ иначе,} \end{cases}$
   2. $at^s_{P_i} = 1 \;\Rightarrow\; (i \notin \alpha^s) \wedge (j^s_i \notin \gamma^s)$
7. $at^s_{P_0} = 2 \;\Rightarrow\; p^s \notin \alpha^s,\ p^s \in \{1, \ldots, i^s - 1\}$,
8. $[c_1]^s_3 \sqcup \ldots \sqcup [c_n]^s_3 \sqcup \beta^s \sqcup [c_0]^s_3 \sqcup \gamma^s = \{1, \ldots, i^s - 1\}$,
9. $\forall\, p \in \alpha^s \;\; [c_p]^s$ имеет вид $\{(A_i, 0, i)\}$, где $i \in \{1, \ldots, N\}$,
10. каждый элемент $[c_0]^s$ имеет вид $(A_i B, p, i)$, где $i \in \{1, \ldots, N\}$,
11. $\forall\, j \in \gamma^s \;\; C_j = A_j B$,
12. $k^s = N \;\Rightarrow\; \gamma^s = \{1, \ldots, N\}$.

**Теорема 2.** В $\Sigma_{\mathcal{P}_\Pi}$ нет тупиковых состояний.

**Доказательство**. Пусть $\Sigma_{\mathcal{P}_\Pi}$ содержит тупиковое состояние $s$. Нетрудно доказать, что $at^s_{P_0}$ не м.б. равно 0, 2, 3, и $\forall\, i = 1, \ldots, n$ $at^s_{P_i}$ не м.б. равно 1. Поэтому $at^s_{P_0}$ м.б. 1 или 4.

(1). Пусть $at^s_{P_0} = 1$. Из тупиковости $s$ следует, что $k^s < N$, откуда опять используя тупиковость $s$ получаем: $[c_0]^s = \emptyset$.
Из утверждения 4 в теореме 1 следует, что $k^s < i^s - 1$, откуда на основании утверждения 8 теоремы 1 и равенства $[c_0]^s = \emptyset$ получаем:

$$[c_1]^s_3 \sqcup \ldots \sqcup [c_n]^s_3 \sqcup \beta^s \neq \emptyset. \tag{37}$$

Если $\exists\, i \in \{1, \ldots, n\}$: $[c_i]^s \neq \emptyset$, то из тупиковости $s$ следует, что $at^s_{P_i} \neq 0$, поэтому $at^s_{P_i} = 2$, откуда нетрудно получить, что $at^s_{P_0} = 4$. Однако это возможно лишь в случае $k^s \geq N$, что противоречит установленному выше неравенству $k^s < N$.
Поэтому $\forall\, i \in \{1, \ldots, n\}$ $[c_i]^s = \emptyset$, и из (37) следует, что $\beta \neq \emptyset$. Однако анализом ПП $P_i$ $(i = 0, \ldots, n)$ нетрудно установить, что это возможно только если $\exists\, i \in \{1, \ldots, n\}$: $at^s_{P_i} = 2$. Как было установлено в предыдущем абзаце, данное равенство невозможно.

(2). Пусть $at^s_{P_0} = 4$. Тогда $k^s \geq N$, откуда следует, что $|\gamma^s| = k^s = N$. Пусть $s'$ – первое состояние на пути $\pi$ из $0_{\mathcal{P}_\Pi}$ в $s$, такое, что $k^{s'} = N$. Нетрудно видеть, что $at^{s'}_{P_0} = 2$. Из утверждений 2 и 8 теоремы 1 следует, что $i^{s'} - 1 = N$ и

$$[c_0]^{s'} = [c_1]^{s'} = \ldots = [c_n]^{s'} = \beta^{s'} = \emptyset,$$
$$at^{s'}_{P_1} = 0, \ldots, at^{s'}_{P_n} = 0.$$

Единственный переход из $s'$ соответствует действию $[\![i > N]\!]$ и имеет вид $P_0^{2\to1} : s' \to s''$, единственный переход из $s''$ соответствует действию $[\![k \geq N]\!]$ и имеет вид $P_0^{1\to3} : s'' \to s'''$. Нетрудно видеть, что все состояния в хвосте пути $\pi$, начинающемся с $s'''$, не являются тупиковыми.

В обоих случаях мы получили противоречие, которое является следствием предположения о том что в $\Sigma_{\mathcal{P}_\Pi}$ содержит тупиковое состояние. Следовательно, в $\Sigma_{\mathcal{P}_\Pi}$ нет тупиковых состояний. ■

**Теорема 3.**
Любое выполнение определенного выше РП $\mathcal{P}$ завершается после конечного числа шагов

**Доказательство**.
Пусть существует бесконечное выполнение $\pi$ РП $\mathcal{P}$.
Докажем, что число переходов в $\pi$, соответствующих действиям ПП $P_0$, конечно. Если бы число таких переходов было бесконечным, то

- в $\pi$ нет состояний $s$, обладающих свойством $at^s_{P_0} = 3$,
- существует состояние $s_1 \in \pi$, такое, что $at^{s_1}_{P_0} = 1$,
- каждый переход в $\pi$, начиная с $s_1$, соответствующий какому-либо действию $P_0$, имеет вид либо $P_0^{1\to 2} : s \to s'$, либо $P_0^{2\to 1} : s \to s'$, и число переходов вида $P_0^{1\to 2} : s \to s'$ бесконечно, что невозможно по причине того что при каждом таком переходе увеличивается значение переменной $k$, которое, согласно утверждению 3 теоремы 1, ограничено сверху значением $N$.

Пусть $s' \in \pi$ – состояние, начиная с которого $\pi$ не содержит переходов, соответствующих действиям $P_0$, и $\pi'$ – часть пути $\pi$, начинающаяся с $s'$. Нетрудно видеть, что

$$\exists\, i \in \{1, \ldots, n\}\colon \pi' \text{ содержит бесконечно много переходов вида } P_i^{0\to 1} : s \to s'. \tag{38}$$

Т.к. $\pi'$ не содержит переходов, соответствующих действиям $P_0$, то значение $|[c_i]^s|$ не может увеличиваться, и согласно (38) оно бесконечно уменьшается, что невозможно. ■

Из доказанных теорем следует, что каждое выполнение РП $\mathcal{P}_\Pi$ является конечным и завершается в некотором терминальном состоянии $s$. Из $at^s_{P_0} = 4$ следует, что $|\gamma^s| = k^s = N$, откуда согласно утверждениям 11 и 12 теоремы 1 следует, что РП $\mathcal{P}_\Pi$ удовлетворяет своей спецификации. ■

# Заключение

- Мы представили новую математическую модель параллельных программ
- предложили подход к верификации таких программ, и
- рассмотрели пример верификации MPI программы умножения матриц на основе предложенной модели.

Основное преимущество предложенного подхода – возможность его применения для программ, генерирующих неограниченный набор последовательных процессов.

Проблемы для дальнейших исследований, связанных с предлагаемой моделью, могут быть следующими:

- расширить предложенную модель, введя концепции для моделирования синхронной передачи сообщений и другие механизмы для организации параллельного выполнения,
- ввести язык спецификации свойств распределенных процессов, на котором свойства параллельных программ выражаются в терминах наблюдаемой эквивалентности.

Спасибо за внимание!

Вопросы?