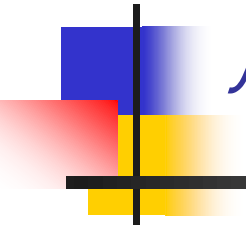


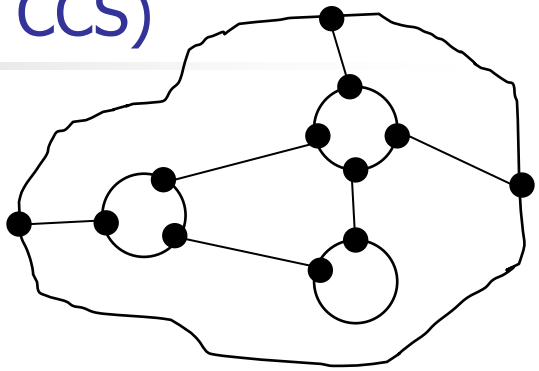
Лекция 3

Алгебры процессов

Calculus of Communicating Systems (CCS) Р.Милнера

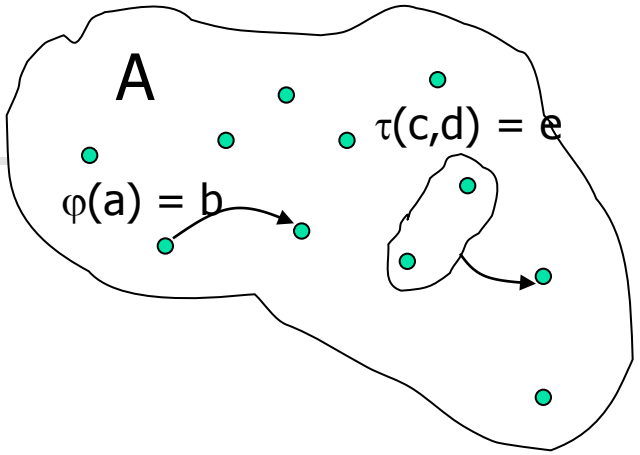
π - исчисление – базовая модель мобильной коммуникации

Исчисление взаимодействующих систем (Calculus of Communicating Systems - CCS)



- CCS – это формальная модель сети взаимодействующих процессов
  - – коммуникационная топология статична
- Цель
  - - разработать теорию, в рамках которой можно формально анализировать поведение параллельно протекающих взаимодействующих процессов
- Алгебраический подход
  - – система взаимодействующих процессов представлена термом (формулой)
  - – разработана теория манипуляций этими формулами
  - – введено понятие эквивалентности процессов как сохранение порядков внешне видимых действий
- Литература
  - Robin Milner. *Communication and Concurrency*, Prentice Hall, 1989.
  - Ю.Карпов. *Формальное описание и верификация протоколов на основе CCS*. Автоматика и вычислительная техника, N6, 1986

Алгебра – что это такое?



A=(A, Ω) – алгебра

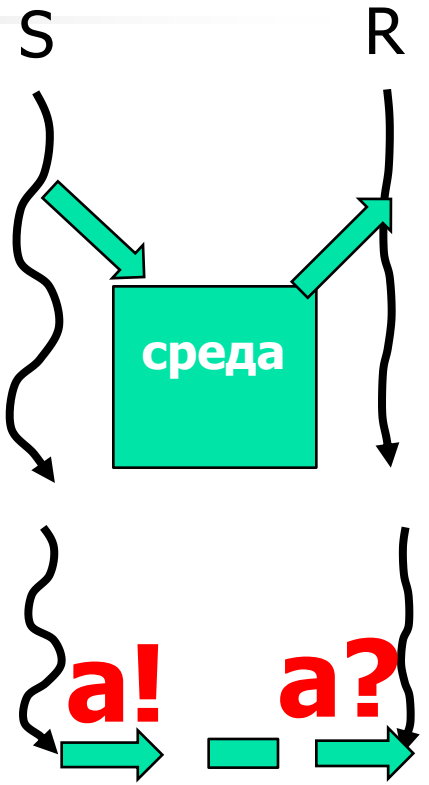
A – носитель алгебры {a, b, c, ...)

Ω- конечное множество операций { φ, τ, ...} из A×A×...×A в A

- Алгебра – это множество объектов одной природы и набор операций над этими объектами, результат которых – объект того же множества
- Идея Р.Милнера:
  рассматривать параллельную и последовательную композицию процессов P и Q как процесс. Этот новый процесс имеет свойства, зависящие от свойств исходных процессов и операции над ними. После выполнения взаимодействия со средой процесс остается процессом, но уже другим! Это унарная операция над процессом
- Т.е. операции || композиции, взаимодействия с внешними процессами и внутренние действия процессов дают в результате объект той же природы – процесс. Отсюда – и алгебра процессов
- Итак, множество функционирующих параллельных процессов имеют алгебраическую структуру

Взаимодействие процессов: хэндшейк

- Формализация взаимодействия – важнейший элемент теории
- Что такое взаимодействие?
  один процесс выдает информацию (Sender), другой ее получает (Receiver)
- Обычно - передача информации – через пассивную среду:
  - разделяемые переменные
  - ограниченное взаимодействие (через семафоры)
  - ограниченный буфер
  - неограниченный буфер
  - посылка сообщений
- Каждый тип взаимодействия – какое-нибудь ограничение. Какой тип взаимодействия выбрать в теории?
- Идея Робина Милнера:
  не различать активные объекты (процессы) и пассивные (среду коммуникации). Все-процессы! Среда-тоже процесс!



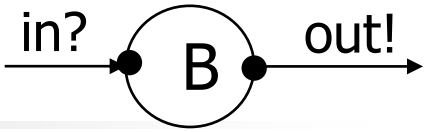
Два процесса, S и R, взаимодействуют, iff они могут выполнить действие и кодействие одновременно

Реализация хендшейка

- Реализация хендшейка в одном компьютере выполняется просто, с помощью аппаратных средств

- Реализация хендшейка в распределенной системе процессов требует специального протокола

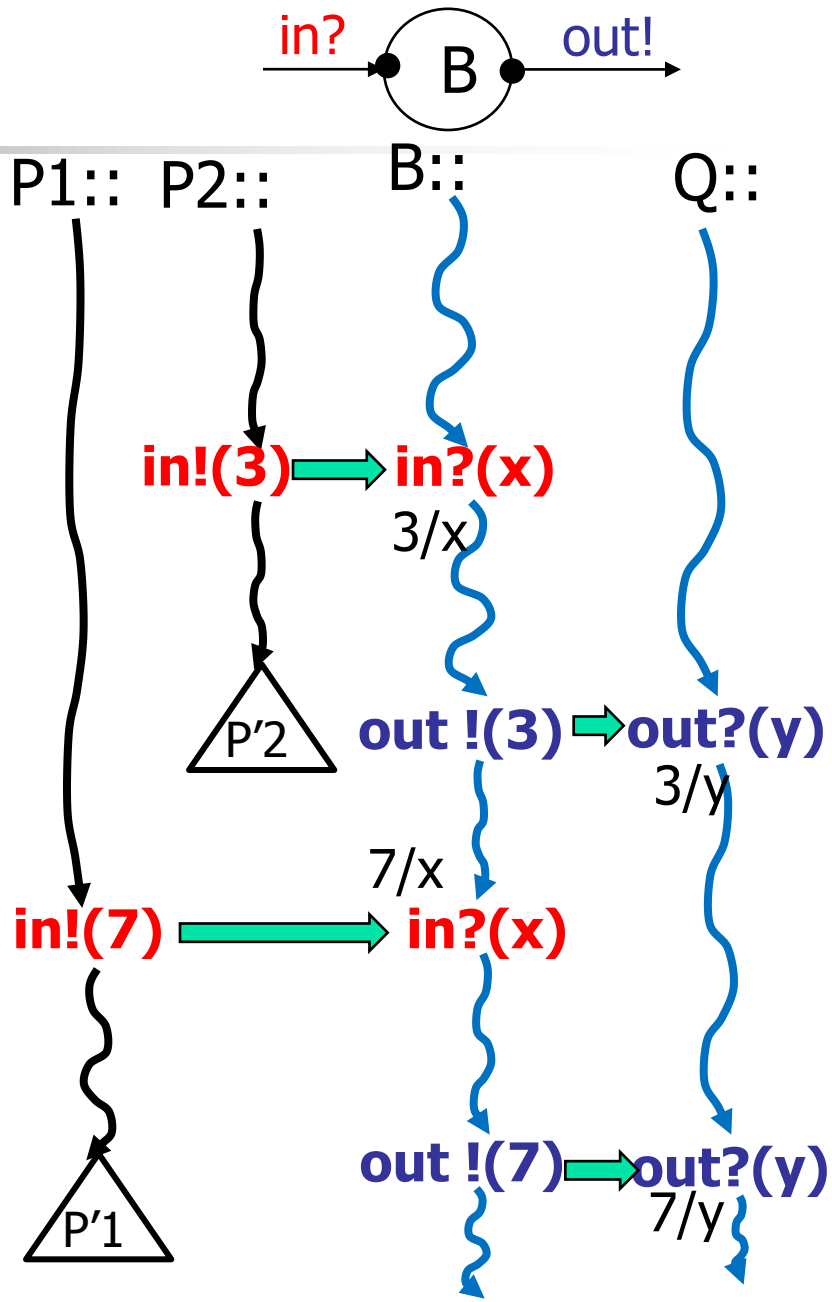
Простой пример



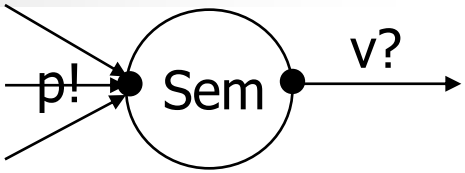
- *Процесс (агент) B*
  - агент выполняет действия (либо внешние взаимодействия, либо свои собственные внутренние действия)
- Поведение агента – это его (наблюдаемая извне) способность к последовательности взаимодействий с окружением
- Агент имеет помеченные порты
  - входные (у агента B: in?) и выходные (у агента B: out!)
- Поведение агента B
  - B := in?(x).B’ (x)
  - B’ (x) := out!(x) .B
- Поведение системы взаимодействующих процессов определяется системой взаимно рекурсивных уравнений
- Динамика процесса – выполнение действий: a . P –a-> P
  выполнив действие a, процесс a.P ведет себя, как процесс P

Описание поведения

- Имена агентов могут иметь параметры
- Поведение агента B
  - B := in?(x) . B’ (x)
  - B’ (x) := out!(x) . B
  - B := in?(x) . out!(x) . B
    Агент B принимает x по порту in, затем значение переменной x выдается по порту out, и затем ведет себя как агент B
- Если несколько агентов хотят доступиться к буферу, выполняя операции in!(x), то они смогут выполнить свои операции по очереди (в произвольном порядке)
  - P1:= in!(7) . P’1
  - P2:= in!(3) . P’2
  - Q := out?(y) . Q(y)



Семафор: примитив синхронизации



- Семафор

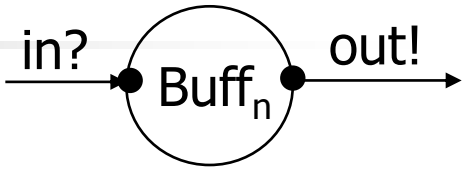
  Sem := p! . v? . Sem

- Если несколько рабочих процессов хотят закрыть (сделать взаимно исключенной) свою работу с ресурсом, то можно на ресурс “навесить” семафор
- Каждый рабочий процесс Qi для входа в критический интервал выполняет операции p и v (чистая синхронизация без передачи значений):

  Qi := … p? . τ . v! . …

- Действие τ обозначает внутреннее, невидимое снаружи действие процесса. Все такие внутренние действия для нас (наблюдателей, пользователей процесса) одинаковы, неразличимы

Пример: ограниченный буфер



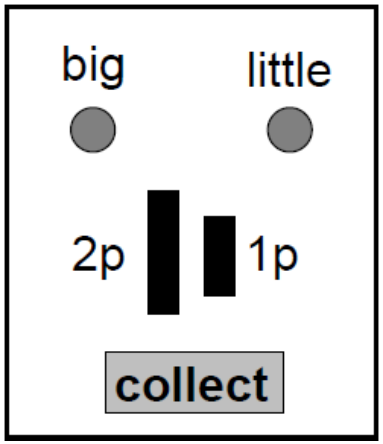
- Ограниченный буфер $Buff_n(s)$ (FIFO)

$Buff_n(\ ) := in?(x)\ .\ Buff_n(x)$
$Buff_n(v_n, \ldots, v_1) := out!(\ v_1\ )\ .\ Buff_n(v_n, \ldots, v_2\ )$
$Buff_n(v_k, \ldots, v_1) := in?(x).Buff_n(x, v_k, \ldots, v_1) + out!(\ v_1\ ).Buff_n(v_k, \ldots, v_2)\ (0<k<n)$

- Поведение буфера $Buff_n(s)$ – это очередь FIFO
- Первая базовая операция – (взаимо)действие.
- Вторая базовая операция – сумма (альтернатива) ‘+’
  - P+Q ведет себя, как P либо, как Q
  - Некоторые альтернативы могут быть запрещены
  - Если возможны обе альтернативы, выбор недетерминирован
- Задача: опишите формально буфер LIFO

Пример: Vending Machine

вид машины



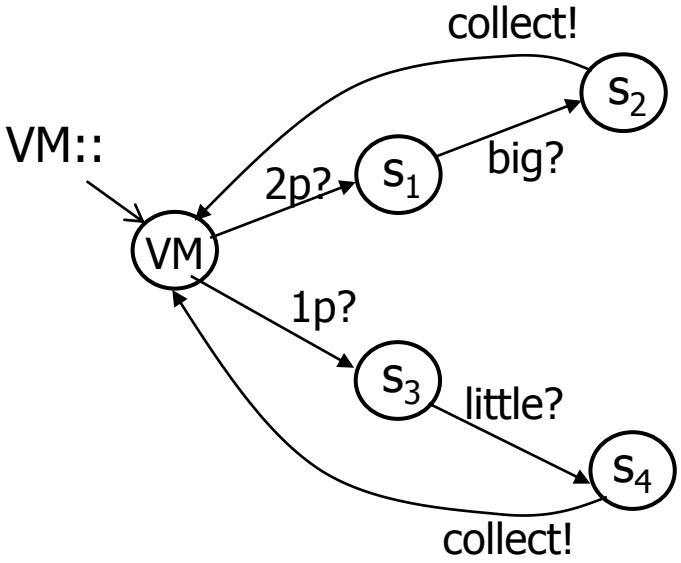
Большая шоколадка стоит два пенни, маленькая – одно пенни

уравнение (терм) поведения

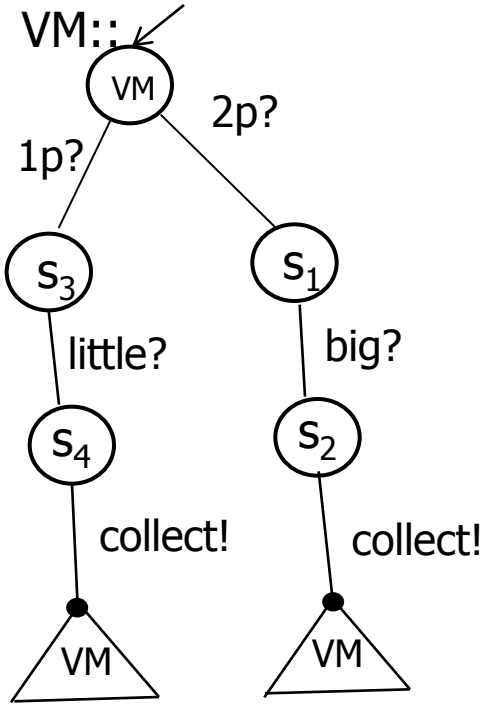
VM := 2p? . big? . collect!. VM +
1p? . little? . collect!. VM

Как *реализовать* поведение VM?

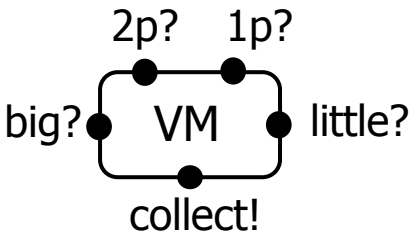
граф поведения



дерево поведения



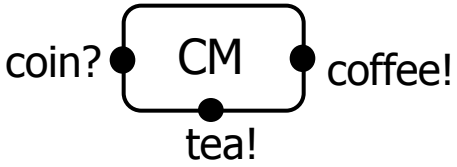
изображение агента



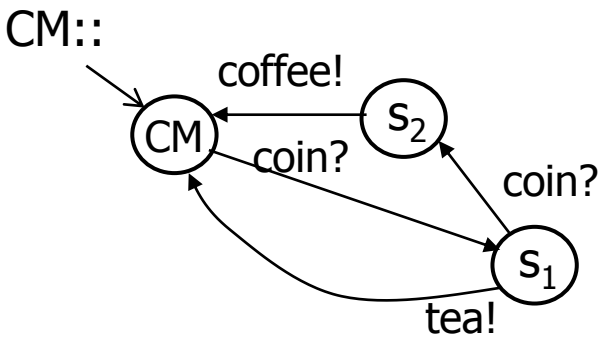
Процессы

- Что такое процесс?
- Мы говорим, что реагирующие системы взаимодействуют с окружением. Именно взаимодействие определяет процесс!

CM (Coffee Machine)::

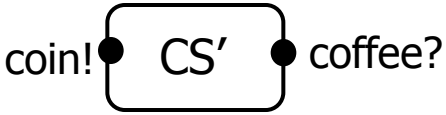


CM::

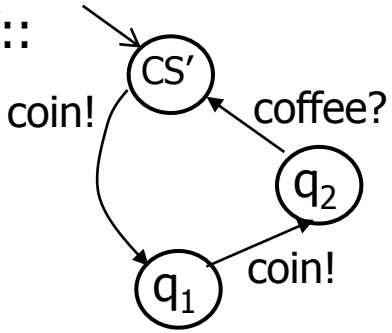


CM:=coin?.(coin?.coffee!+tea!).CM

Computer Scientist1



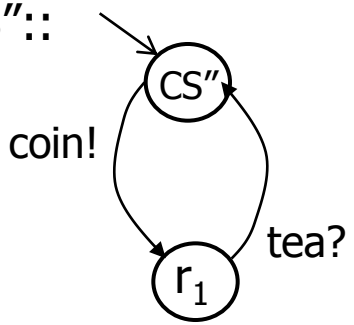
CS’::



CS’:=coin!.coin!.coffee?.CS’

Computer Scientist2

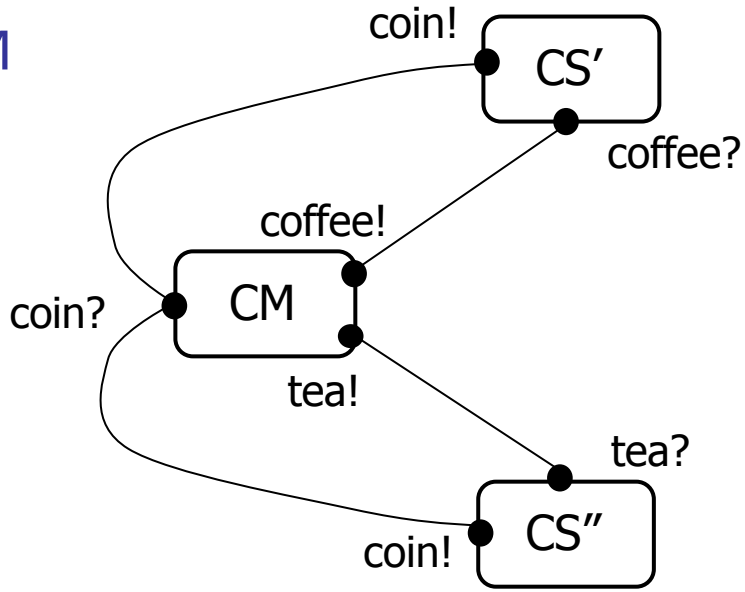
CS”::



CS”:=coin!.tea?.CS”

Композиция агентов

- Машина продажи и два ученых
  CM :=coin?.(coin?. coffee! + tea!).CM
  CS′ :=coin!.coin!.coffee?.CS′
  CS″ :=coin!.tea?.CS″



- Параллельная композиция агентов
  S := CM | CS′ | CS″
- Композиция коммутативна и ассоциативна
- Агенты могут работать независимо и взаимодействовать через комплементарные порты

- Существуют ли в этой системе некорректности?
  Как их формально определить?
  Кк их обнаружить?

Пример траектории поведения

- CCS синтаксис (грамматика):

P,Q ::= 0 | a?.P | a!.P | P|Q | P+Q | P\L

Процессы P и Q могут провзаимодействовать по порту a, если они к этому готовы

$$a?.P \mid a!.Q \xrightarrow{-\tau\rightarrow} P \mid Q$$

- Машина продажи и два ученых

  CM :=coin?.(coin?.coffee! + tea!).CM

  CS′ :=coin!.coin!.coffee?.CS′

  CS″ :=coin!.tea?.CS″

- S := CM | CS′ | CS″ =

одна из возможных траекторий поведения:

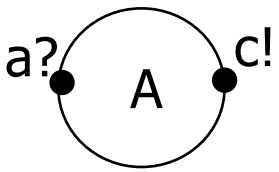
coin?.(coin?.coffee!+tea!).CM | coin!.coin!.coffee?.CS′ | coin!.tea?.CS″ $-\tau\rightarrow$
(coin?.coffee! + tea!).CM | coin!.coffee?.CS′ | coin!.tea?.CS″ $-\tau\rightarrow$
coffee!.CM | coffee?.CS′ | coin!.tea?.CS″ $-\tau\rightarrow$ CM | CS′ | CS″

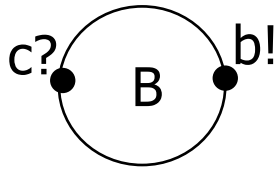
- Но в этой системе процессов возможен дедлок! (какой??)

Как формально определить все возможные траектории поведений?

Пример агентов и их композиции

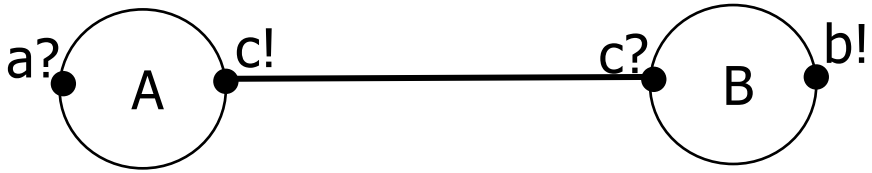
Агенты A и B:





A := a?. A′ ; A′ := c!. A; B := c? . B′ ; B′ := b! . B

Композиция агентов A|B:



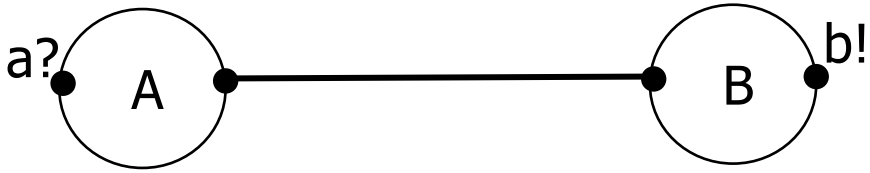
A–a?->A′ позволяет A|B –a?->A′| B

A′–c!->A позволяет A′|B –c!->A| B

A′–c!->A и B–c?->B′ позволяют A′|B–τ->A|B′ – τ-внутреннее мгновенное действие двух процессов

A и B по c могут взаимодействовать друг с другом или с окружением

Ограничение (A|B) \ {c}:

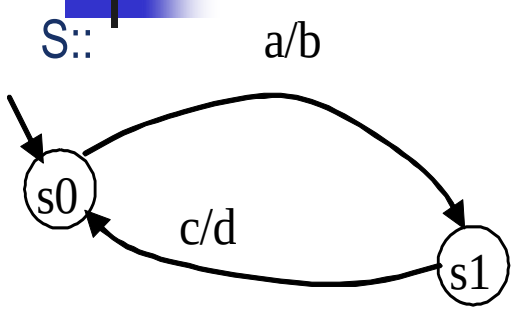


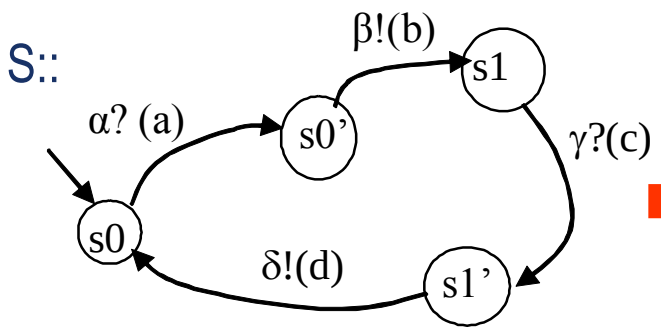
Если P –x->P′ то позволяется P\L –x-> P′\L, только если a∉L

A и B могут взаимодействовать только друг с другом по c

Скрытые (внутренние) действия τ не видны снаружи

Конечные автоматы как спецификаторы процессов (синронизация)





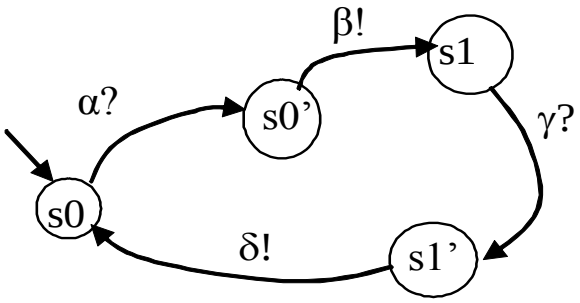
s0 = α?(a).s0';

s0'= β!(b).s1;

s1 = γ? (c).s1';

s1' = δ!(d).s0

Для простоты можно рассматривать только синхронизацию:



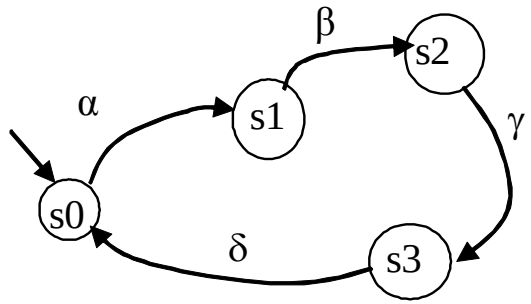
s0 = α?.s0';

s0'= β!.s1;

s1 = γ?.s1';

s1' = δ!.s0

Посылка и прием ничем не отличаются с точки зрения синхронизации:



s0 = α.s1;

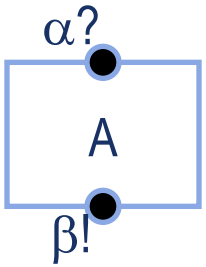
s1= β.s2;

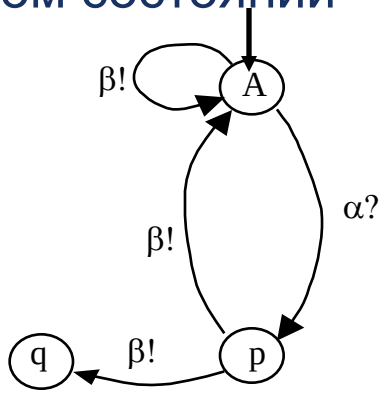
s2 = γ.s3;

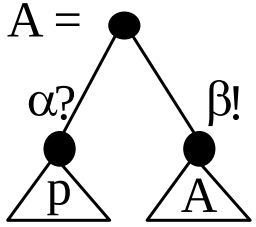
s3 = δ.s0

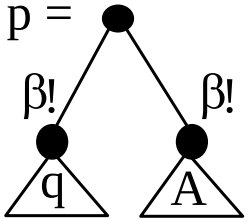
Описание агентов

Объект (агент) – описываем динамикой его поведения. Агент – это, фактически, возможные поведения процесса, находящегося сейчас в конкретном состоянии

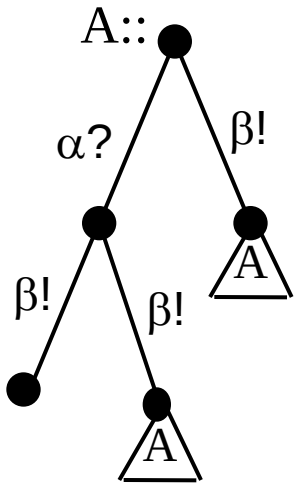




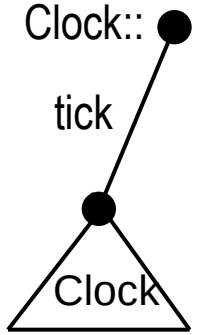




A := α?.p+β!.A;
p := β!.A + β!.q;
q := NIL



A= α? .(β!A + + β!.NIL) + β!A;



Clock = tick. Clock

Clock := tick. Clock
= tick. tick. Clock
= tick. tick. tick. Clock

Процесс представляем аналитически, множеством рекурсивных формул

CCS – исчисление взаимодействующих агентов

- Милнер дает пять базовых операций CCS:
- ***Префикс.*** В выражении a.A a называется префиксным действием агента A. Выражение a.A само описывает агента, который ведет себя как агент A *после того*, как он выполнит действие a. Действия – двух типов: a? и a!
- ***Сумма*** двух агентов P и Q представлена как P+Q. Агент P+Q недетерминированно выбирает одно из поведений – он ведет себя либо как P, либо как Q в зависимости от действия, предлагаемого окружением. Суммирование называют также оператором выбора.
- ***Параллельная композиция*** двух агентов P и Q записывается как P | Q. Смысл этой операции: P и Q функционируют независимо и параллельно, и взаимодействуют через комплементарные действия.
- ***Ограничение*** изолирует агент от его окружения, например, P\ L – это агент, который работает как P, но с тем ограничением, что ни одно действие из множества действий L не может быть видимо внешним наблюдателем
- ***Переименование***. Агент $P[a_1/b_1, \ldots, a_n/b_n]$ ведет себя как агент P, в котором действия $a_1, \ldots, a_n$ заменены соответственно действиями $b_1, \ldots, b_n$.
- Нульарный оператор 0 или *Nil* - агент (процесс), который ничего не делает

Формальная семантика операций CCS

Действие: a.E

$a.E \xrightarrow{a} E$

Сумма: $E_1 + E_2$

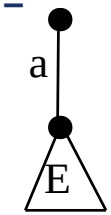
$$\frac{E_i \xrightarrow{a} E,\ i \in I}{\Sigma_{i \in I} E_i \xrightarrow{a} E}$$

Ограничение: $E \setminus L$

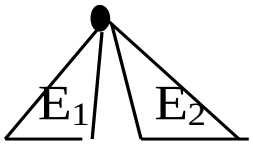
$$\frac{E \xrightarrow{b} E_1;\ b \notin L}{E \setminus L \xrightarrow{b} E_1 \setminus L}$$

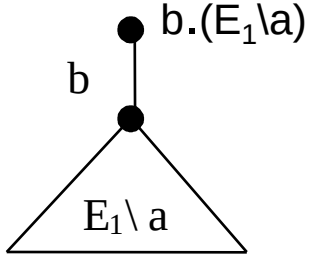
$(b.E_1) \setminus a = b.(E_1 \setminus a)$

a.E=



$E_1 + E_2$





Семантика процесса a.E – это выполнение a, а потом выполнение процесса E

Семантика суммы процессов:

Если процесс Ei может выполнить a, а потом вести себя, как E, то и сумма процессов, в которую входит Ei, может выполнить a, а потом вести себя, как E

Семантика ограничения:

Если процесс E может выполнить действие b, которое не входит в L, а потом вести себя, как E', то и процесс E, ограниченный по L, ведет себя так же

Семантика операций CCS: параллельная композиция процессов

Параллельная композиция: E | F

$$\frac{E \xrightarrow{a} E_i}{E \mid F \xrightarrow{a} E_i \mid F} \qquad \frac{F \xrightarrow{a} F_i}{E \mid F \xrightarrow{a} E \mid F_i} \qquad \frac{E \xrightarrow{a} E_i \;;\; F \xrightarrow{\bar{a}} F_k}{E \mid F \xrightarrow{\tau} E_i \mid F_k}$$

τ - внутреннее взаимодействие процессов

Семантика параллельной композиции процессов:
Если процесс E может выполнить a, а потом вести себя, как Ei, то и параллельная композиция процесса E и любого процесса F может выполнить a, а потом вести себя, как Ei

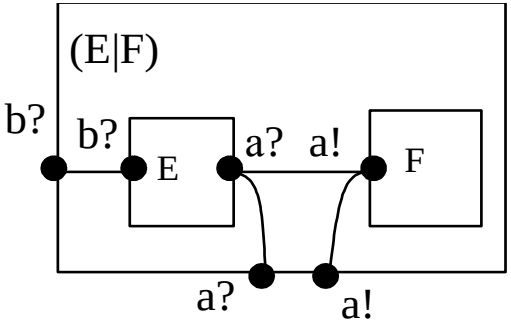
Если процесс F может выполнить a, а потом вести себя, как Fi, то и параллельная композиция любого процесса E и процесса F, может выполнить a, а потом вести себя, как Fi

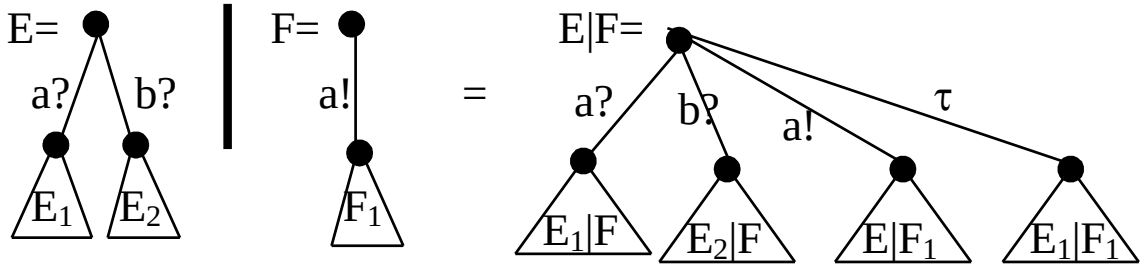
Если процесс E может выполнить действие a, а потом вести себя, как E_i, а процесс F может выполнить ко-действие ă, а потом вести себя, как F_k, то их параллельная композиция может выполнить внутренний переход τ (синхронизацию по a), а потом вести себя, как параллельная композиция $E_i|F_k$

Пример: операции алгебры процессов

E = a?.E1 +b?.E2;
F = a!. F1;

E|F = a?. (E1|F)
+ b?. (E2|F)
+ a! . (E|F1)
+ τ . (E1|F1);

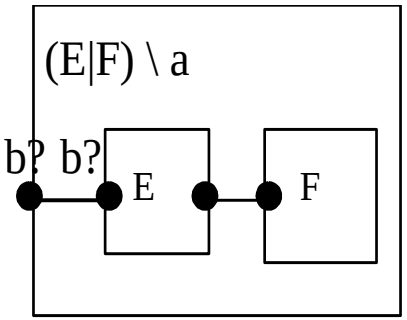




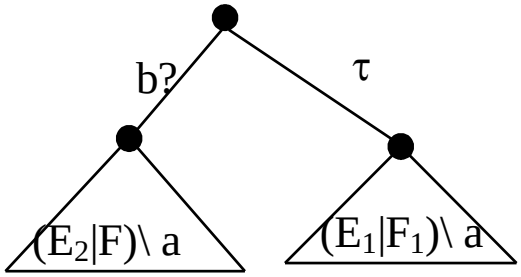
Ограничение: P \ L

(E|F) \{a} = b?. (E2|F) \{a}
+ τ . (E1|F1) \{a} ;

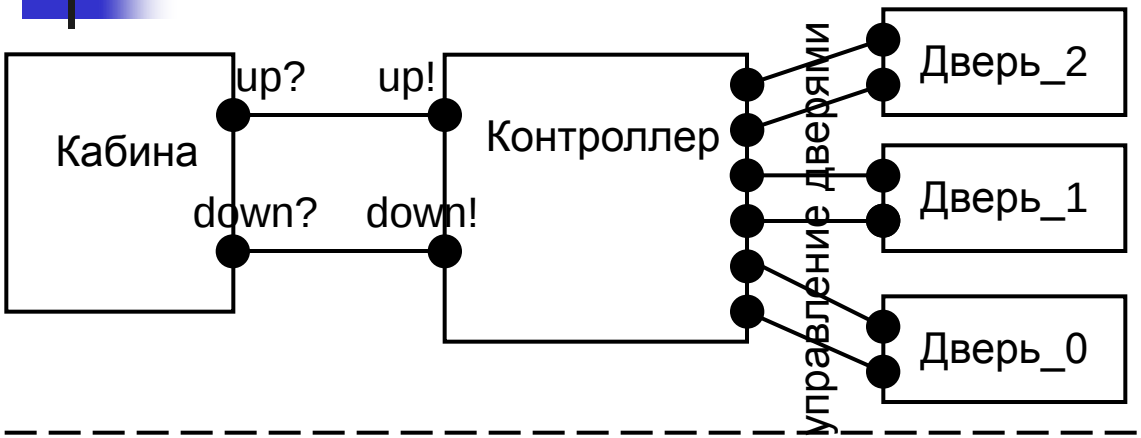
Ограниченные порты не видны извне, по ним нельзя взаимодействовать с внешней средой



((a?.E1+ b?.E2) | (a!. F1))\ a



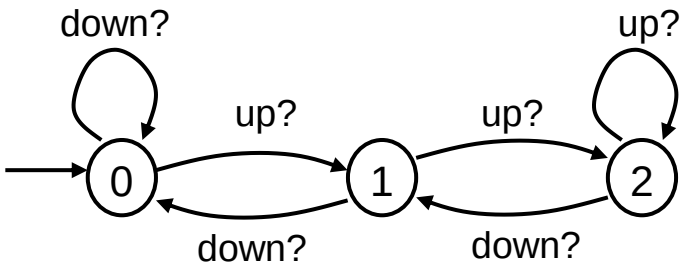
Пример: Система управления лифтом



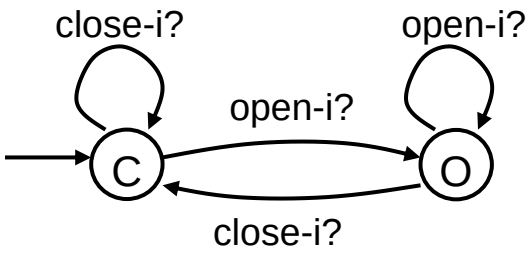
Представив поведение каждого процесса в алгебре CCS, мы можем проверить свойства всей системы

Кабина | Контроллер | Дверь_0 | Дверь_1 | Дверь_2

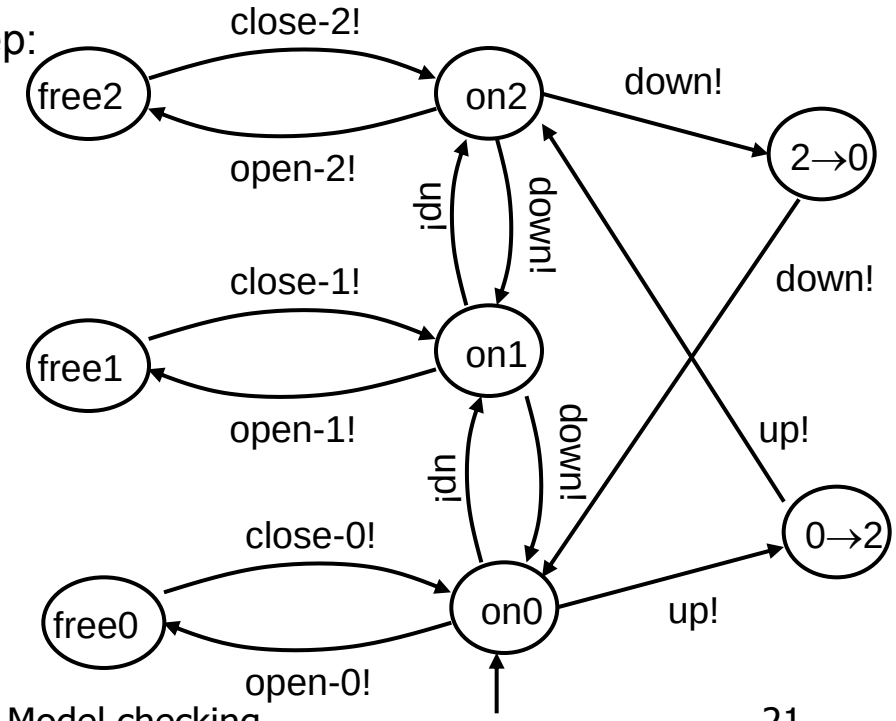
Кабина:



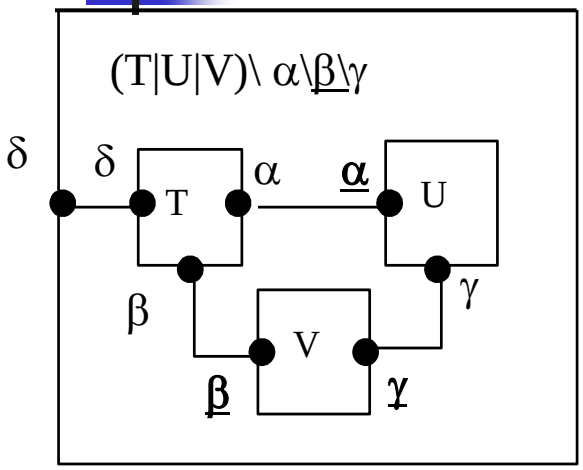
Двери:



Контроллер:

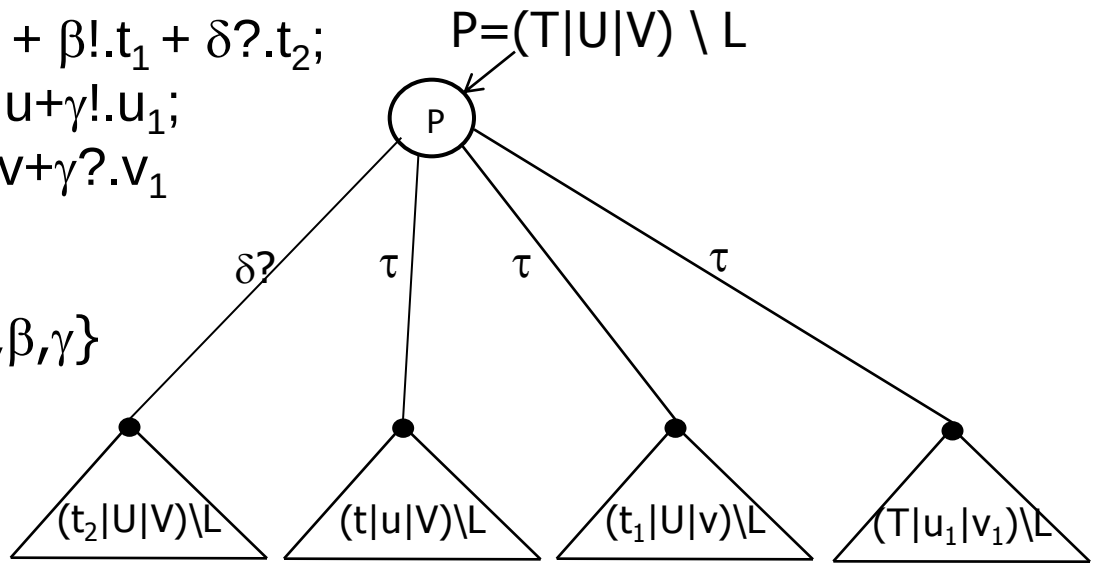


Пример: композиция и сокрытие портов дают новый процесс



$T = \alpha!.t + \beta!.t_1 + \delta?.t_2;$
$U = \alpha?.u + \gamma!.u_1;$
$V = \beta?.v + \gamma?.v_1$

$L = \{\alpha, \beta, \gamma\}$



Поведение всей системы

$P = (T \mid U \mid V)\setminus L =$

$= ((\alpha!t + \beta!t_1 + \delta?t_2) \mid (\alpha?u+\gamma!u_1) \mid (\beta?v+\gamma?v_1)) \setminus L =$

$= \delta?. (t_2 \mid (\alpha!u+\gamma!u_1) \mid (\beta?v+\gamma?v_1))\setminus L$ (действие δ - доступно извне)

$+ \tau . (t \mid u \mid (\beta?v+\gamma?v_1)) \setminus L$ (взаимодействие по скрытому порту α)

$+ \tau . (t_1 \mid (\alpha?u+\gamma?u_1) \mid v) \setminus L$ (взаимодействие по скрытому порту β)

$+ \tau . ((\alpha!t + \beta!t_1 + \delta?t_2) \mid u_1 \mid v_1)\setminus L =$ (взаимодействие по скрытому порту γ)

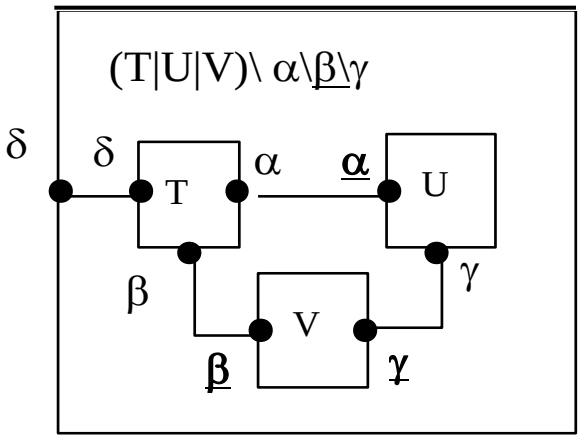
$= \ldots \ldots \ldots$

При заданных процессах t, t1, t2, u и т.д., можем аналитически раскрыть все шаги выполнения всей параллельной композиции процессов

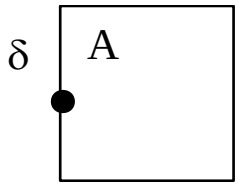
Основная идея верификации в CCS

Система –
параллельная композиция процессов

Желаемое внешнее поведение
системы процессов T, U и V



~ (?)

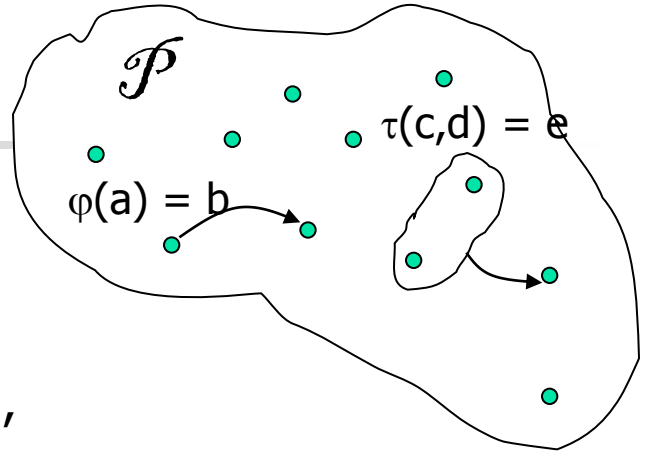


(T | U | V)\ α \ β \ γ ~ (?) A

Будет ли система процессов (T | U | V)\ α \ β \ γ эквивалентна (в каком-то смысле) процессу A с одним внешним портом δ?

Что значит “эквивалентна?”

Алгебра процессов



- Построили формулы, описывающие операции процессов, включая параллельную композицию процессов.
- Процессы описываются термами (выражениями), каждый представляет собой процесс
- Следовательно, множество процессов имеет алгебраическую структуру, процессы можно рассматривать, как элементы некоторой Алгебры процессов
- Основная задача алгебры – это проверка эквивалентности: будут ли, например, эквивалентны результаты при всех a и b

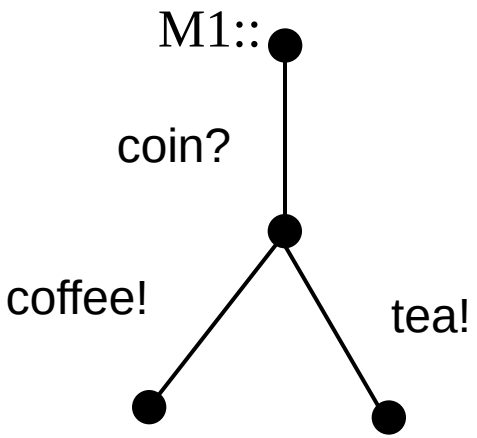
  в Булевой алгебре: эквивалентны ли $a \Rightarrow b$ и $\neg a \vee b$???

  в алгебре вещественных чисел: эквивалентны ли $a+b+c$ и $2c+a+b-c$???
- У этих процессов, описанных аналитически, можно исследовать свойства
  - как и во всякой алгебре, можно ввести понятие эквивалентности:

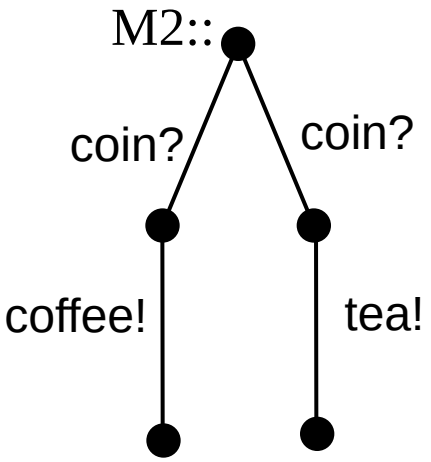
    когда два выражения, описывающие процессы, будут эквивалентны?

Как определить понятие эквивалентности в алгебре процессов?

Пример разных поведений

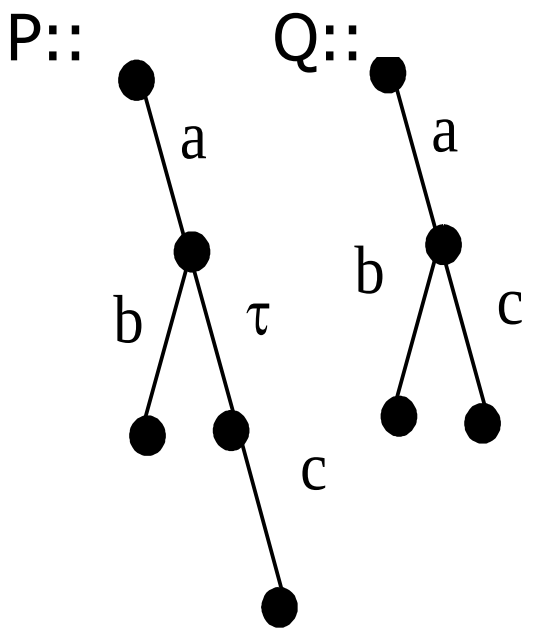


эквивалентны ли поведения?

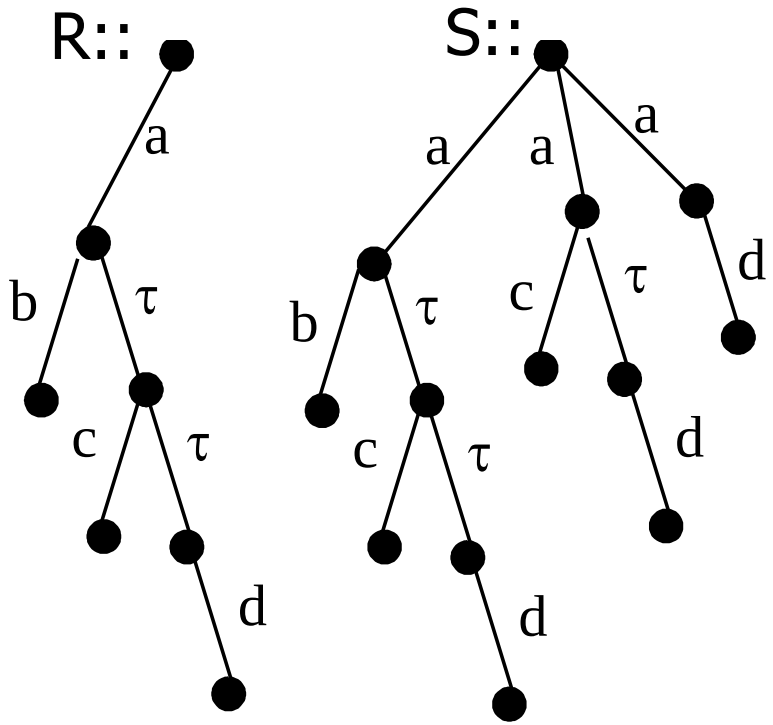


- Две машины, M1 и M2, с точки зрения возможных цепочек событий (языка) эквивалентны: две возможные цепочки событий:
  {<coin?, coffee!>; <coin?, tea!>}
- С точки зрения внешнего пользователя, машины M1 и M2 не эквивалентны: после опускания монеты, решение о том, пить кофе или чай:
  - M1 отдает решение внешнему пользователю, который решает это недетерминированно;
  - M2 принимает это решение сама недетерминированно

Эквивалентны ли процессы?



НЕ эквивалентны



эквивалентны

Наблюдаемая эквивалентность поведений

Пусть Δ – множество имен портов, а ***B*** – множество поведений.

Будем писать a.A —a — >A бинарное отношение на множестве поведений.

Определим для каждой видимой цепочки экспериментов $\alpha = a_1 a_2 \ldots a_n$ по портам Δ на множестве ***B*** бинарное отношение $=\alpha=>$ так:

$=\alpha=> = —\tau^{k0} — a_1 — \tau^{k1} — a_2 — \tau^{k2} — \ldots a_n — \tau^{kn} —>$ для некоторых $k_0, k_1, .. k_n$.

Отношение $=\alpha=>$ прозрачно для любого числа ненаблюдаемых действий τ, которые агент может выполнить в процессе наблюдаемого эксперимента α.

Определение (Observational Equivalence).

Два процесса P и Q являются наблюдаемо эквивалентными (находятся в отношении бисимуляции ~, обозначается P ~ Q), если любой внешний процесс не может их различить

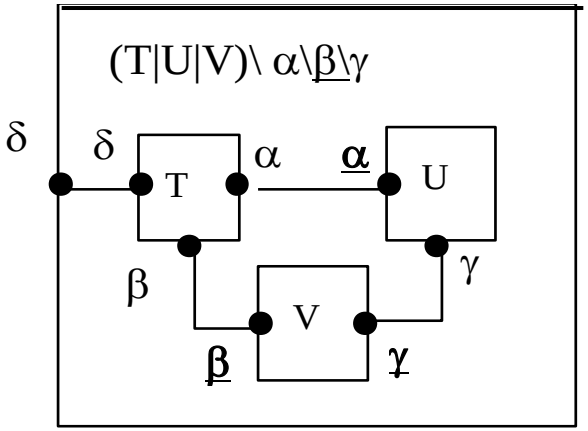
Формально: P~Q если для любой цепочки α внешних воздействий:

a) если $P =\alpha=> P'$, то существует такое Q', что $Q =\alpha=> Q'$ и P' ~ Q'

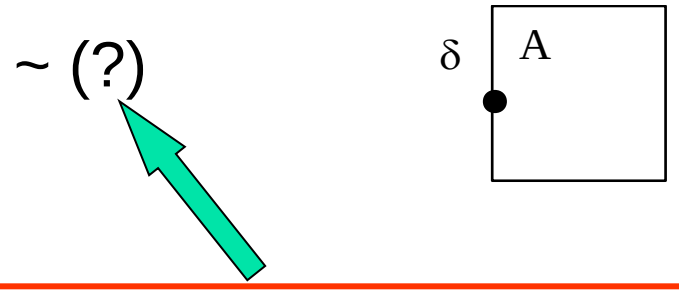
b) если $Q =\alpha=> Q'$, то существует такое P', что $P =\alpha=> P'$ и P' ~ Q'

Основная идея верификации в CCS

Система –
параллельная композиция процессов



Требуемое внешнее поведение системы процессов (в каком-то аспекте, в некоторой проекции)

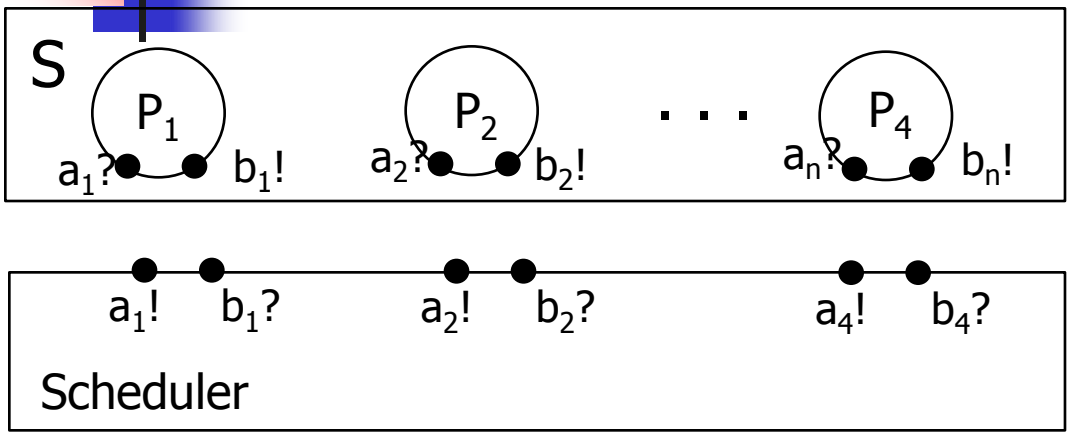


Наблюдаемая эквивалентность

(T | U | V)\ α \ β \ γ ~ (?) A

Наблюдаемая эквивалентность двух процессов P и Q понимается как невозможность для внешнего наблюдателя различить поведения P и Q

Пример: планировщик процессов (R.Milner)



Рабочий процесс

$P_i = a_i? \,.\, b_i! \,.\, P_i$

запуск - останов

Система процессов

$S = P_1 \mid P_2 \mid \ldots \mid P_n$

Построить планировщик Scheduler, запускающий процессы Pi в порядке номеров так, что повторный запуск Pi возможен только если он закончил свою работу

Требования к планировщику:

R1: Проекция поведения Планировщика на порты $\{a_1, \ldots, a_n\}$ цепочка $(a_1! \,.\, a_2! \,.\, \ldots \,.\, a_n!)^{\omega}$

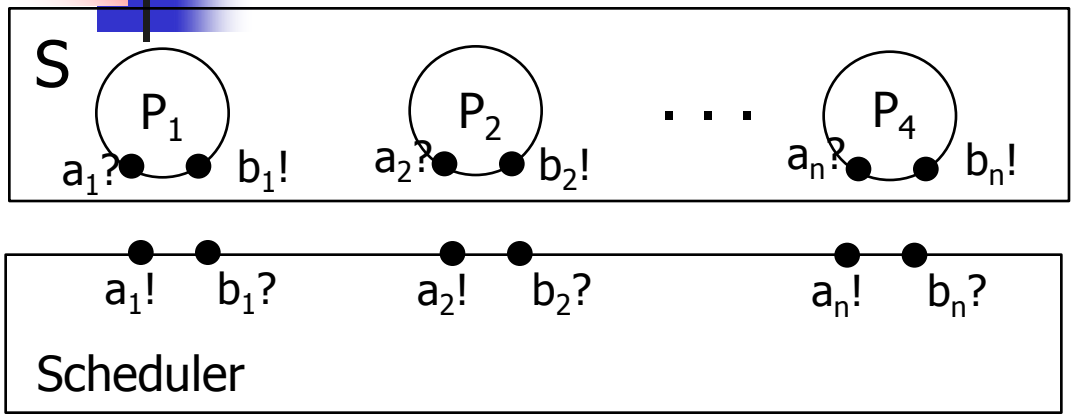
R2: Каждый процесс Pi видит последовательность взаимодействий с ним $(a_i \,.\, b_i \,.\,)^{\omega}$.

Формально требования к планировщику:

R1: Проекция поведения Планировщика на порты $\{a_1, \ldots, a_n\}$ - цепочка $(a_1! \,.\, a_2! \,.\, \ldots \,.\, a_n!)^{\omega}$

R2: Проекция поведения Планировщика на порты $\{a_i, b_i\}$ -цепочка $(a_i! \,.\, b_i?)^{\omega}$

Пример: планировщик процессов (R.Milner)



Рабочий процесс

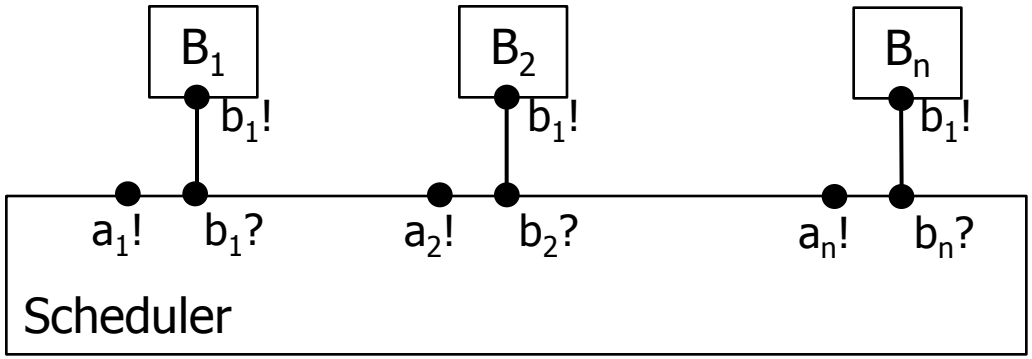
$P_i = a_i? \,.\, b_i! \,.\, P_i$

запуск - останов

Система процессов

$S = P_1 \mid P_2 \mid \ldots \mid P_n$

Если планировщик Scheduler построен, то проверка выполнения требований к нему:



Формально требование R1 к планировщику:

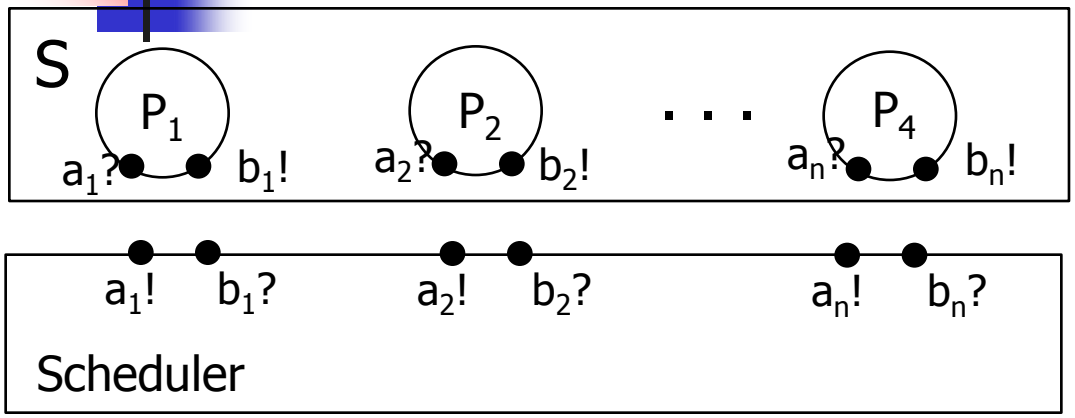
Проекция поведения Планировщика на порты $\{a_1, \ldots, a_n\}$ – цепочка $(a_1! \,.\, a_2! \,.\, \ldots \,.\, a_n!)^{\omega}$

Формальная проверка R1: выполняется ли эквивалентность?

$(Scheduler \mid B_1 \mid B_2 \mid \ldots \mid B_n) \setminus \{b1, b2, \ldots, bn\} \sim (?)\ A$

$A = a_1! \,.\, a_2! \,.\, \ldots \,.\, a_n!.A$

Пример: планировщик процессов (R.Milner)



Рабочий процесс

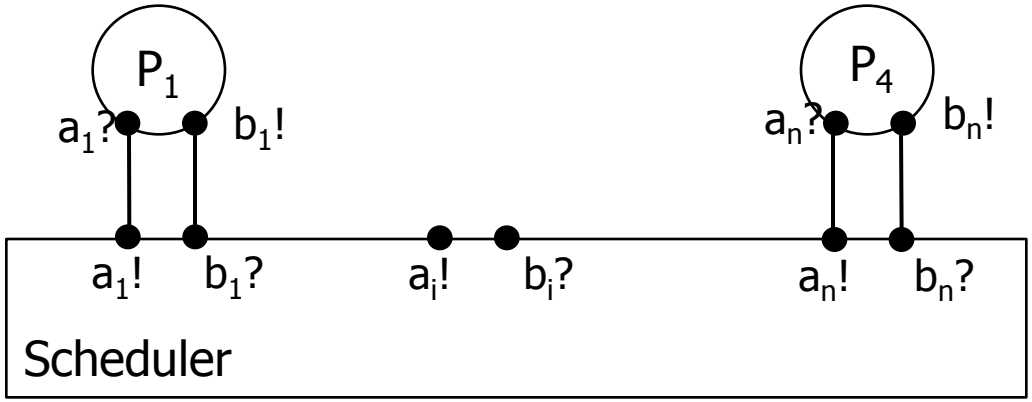
$P_i = a_i? \,.\, b_i! \,.\, P_i$

запуск - останов

Система процессов

$S = P_1 \mid P_2 \mid \ldots \mid P_n$

Если планировщик Scheduler построен, то проверка выполнения требований к нему:



Формально требование R2 к планировщику:

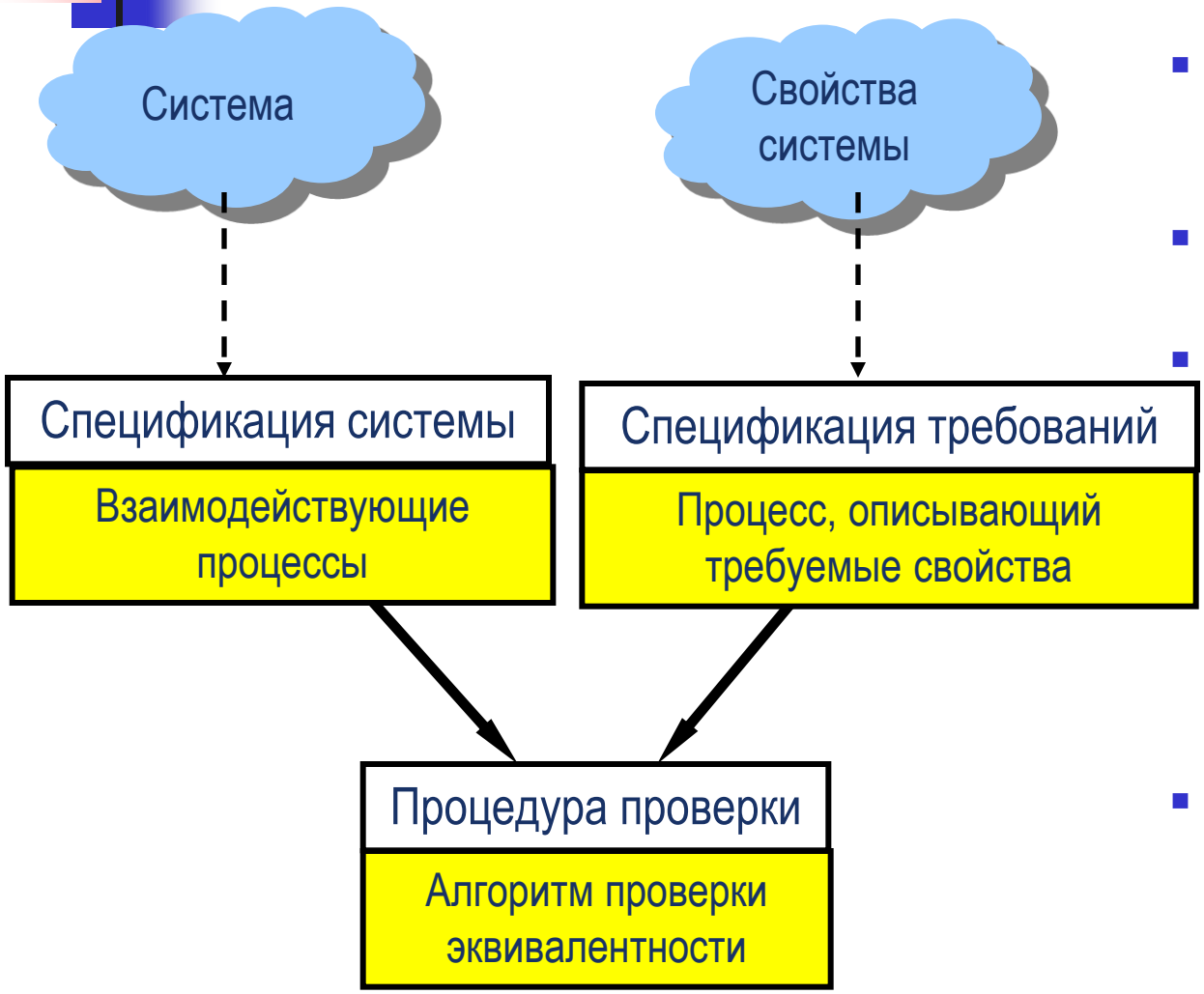
Проекция поведения Планировщика на порты $\{a_i, b_i\}$ – цепочка $(a_i! \,.\, b_i ?)^{\omega}$

Формальная проверка R1: выполняется ли эквивалентность?

$(\text{Scheduler} \mid P_1 \mid \ldots \mid P_{i-1} \mid \ldots \mid P_n) \setminus \{b1, b2, \ldots, bn\} \sim (?)\ B_i$

$B_i = a_i! \,.\, b_i? \,.\, B_i$

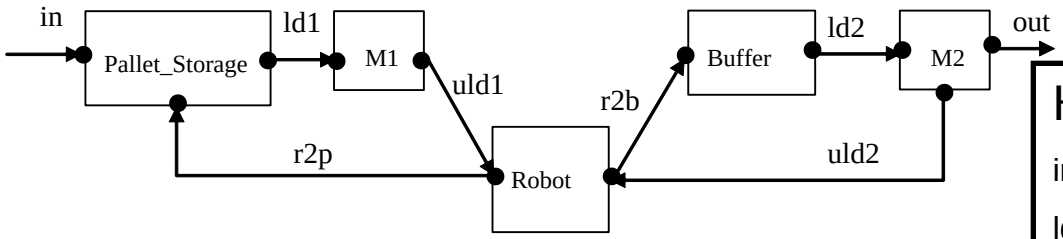
Подход к верификации в CCS



- Система
  - Несколько параллельных взаимодействующих процессов
- Свойства системы
  - Желаемое поведение на части входов/выходов
- Спецификация системы
  - Описание поведения составляющих процессов на CCS
- Спецификация требований
  - Для каждого свойства – процесс в CCS, который ведет себя на проекции части входов так, как хотел бы пользователь
- Процедура проверки
  - Алгоритм проверки эквивалентности композиции процессов и процесса, представляющего искомое свойство

Пример – проверка свойств manufacturing system

Производственная система состоит из 5 агентов:



Коммуникации:

in – новая заготовка входит в систему

ld1- загрузка в M1

uld1- разгрузка M1

r2b – продукт из руки робота попадает в буфер

r2p – помещение паллеты из руки робота в хранилище

Спецификации:

Pallet_Storage=in.ld1.r2p.Pallet_Storage

M1=ld1.uld1.M1

Robot = uld1.r2b.Robot+uld2.r2p.Robot

Buffer = r2b.ld2.Buffer

M2 = ld2.out.uld2.M2

Pallet_Storage2 = Pallet_Storage| Pallet_Storage

Pallet_Storage3 = Pallet_Storage2 | Pallet_Storage

Internals = {ld1, uld1,ld2, uld2, r2b, r2p}

Sys = (Pallet_Storage|M1|Robot|Buffer|M2)\Internals

Sys2 = (Pallet_Storage2|M1|Robot|Buffer|M2)\Internals

Sys3 = (Pallet_Storage3 |M1|Robot|Buffer|M2)\Internals

Внешняя спецификация:

Spec = in.out.Spec

Spec2 =in.Spec2'

Spec2' = out.Spec2 +in.out.Spec2

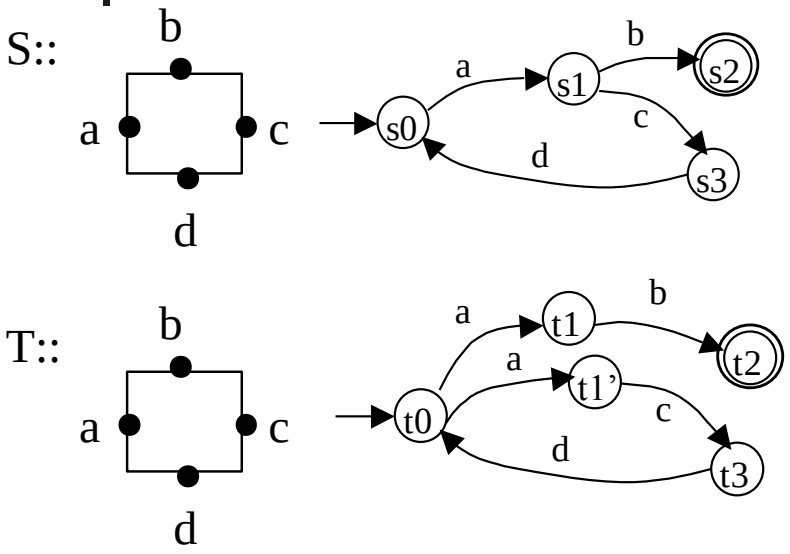
Анализ (Edinburgh Concurrency Workbench):

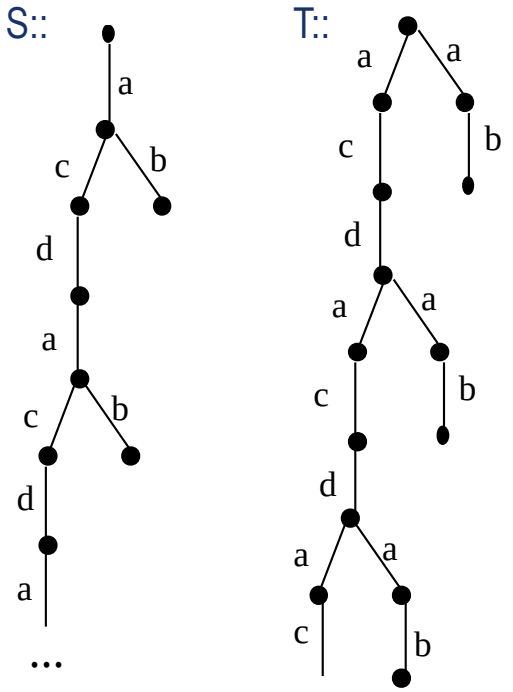
Spec ~ (?) Sys - Да

Spec2 ~ (?) Sys2 - Да

Sys3 - имеет дедлок

Проверка эквивалентности недетерминированных процессов





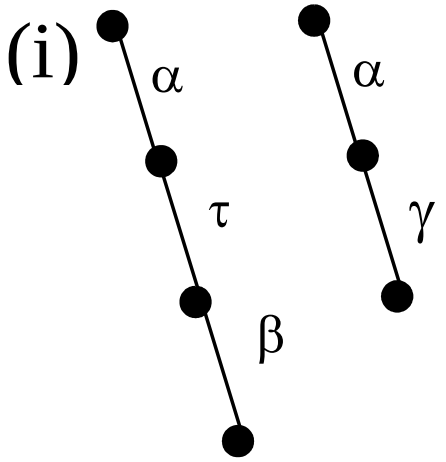
S и T допускают различные эксперименты, поскольку после подачи a агент T может не принять эксперимент c, а агент S всегда после подачи a принимает эксперимент c. Поэтому S и T не эквивалентны.

Как определить отношение наблюдаемой эквивалентности на деревьях экспериментов?

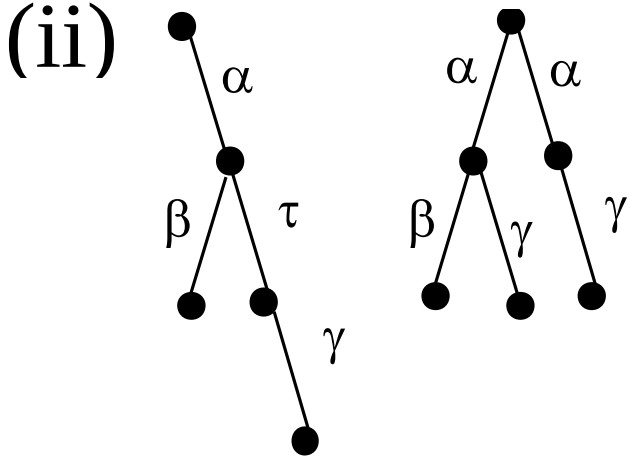
Р.Милнер представил деревья экспериментов (процессов) выражениями, ввел алгебру процессов и в ней определил отношение (наблюдаемой) эквивалентности

Эквивалентность деревьев поведений

Наша задача определить понятие эквивалентности процессов через их ВНЕШНЕЕ ПОВЕДЕНИЕ (функционально, не учитывая время)



НЕ эквивалентны



эквивалентны

Какой алгоритм позволит определить это отношение поведенческой эквивалентности (бисимуляции)?

Итеративный алгоритм проверки отношения наблюдаемой эквивалентности ‘~’

Отношение ‘~’ есть предел цепочки отношений $\sim_0, \sim_1, \sim_2, \ldots \sim_k \ldots$, где:

1) $\sim_0 =$ B^2, (т.е. любые два поведения находятся в отношении $\sim_0$)

2) $\sim_{k+1}$: $P \sim_{k+1} Q$ iff для любой цепочки $\alpha \in \Delta^*$:

a) если $P \Rightarrow_\alpha P'$, то существует такое Q', что $Q \Rightarrow_\alpha Q'$ и $P' \sim_k Q'$

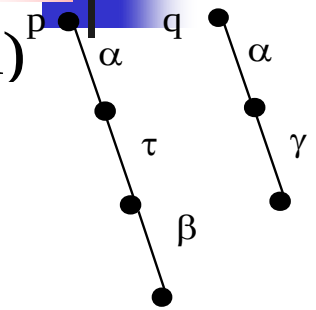
b) если $Q \Rightarrow_\alpha Q'$, то существует такое P', что $P \Rightarrow_\alpha P'$ и $P' \sim_k Q'$

Отношение языковой эквивилентности конечных автоматов – это отношение $\approx_1$

Фактически, проверяется не только допустимость цепочек экспериментов, но и эквивалентность всех промежуточных состояний

Примеры проверки эквивалентности

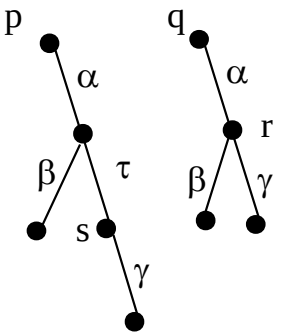
(i)



Верно, что $p \sim_0 q$ (любые два поведения находятся в этом отношении);

Неверно, что $p \sim_1 q$: они допускают разные языки

(ii)



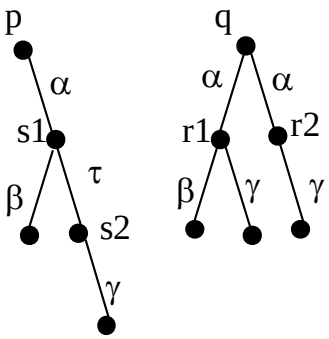
Верно, что $p \sim_0 q$,

Верно, что $p \sim_1 q$, оба допускают язык $\{\varepsilon, \alpha, \alpha\beta, \alpha\gamma\}$

Неверно, что $p \sim_2 q$: эксперимент α приводит p в состояние s, допускающее язык $\{\varepsilon, \gamma\}$, а единственное состояние r, в которое переходит q после подачи α, допускает язык $\{\varepsilon, \beta, \gamma\}$.

Следовательно, $(s \not\sim_1 r)$, и поэтому $(p \not\sim q)$

(iii)



Верно, что $p \sim_0 q$.

Верно, что $p \sim_1 q$, оба допускают язык $\{\varepsilon, \alpha, \alpha\beta, \alpha\gamma\}$.

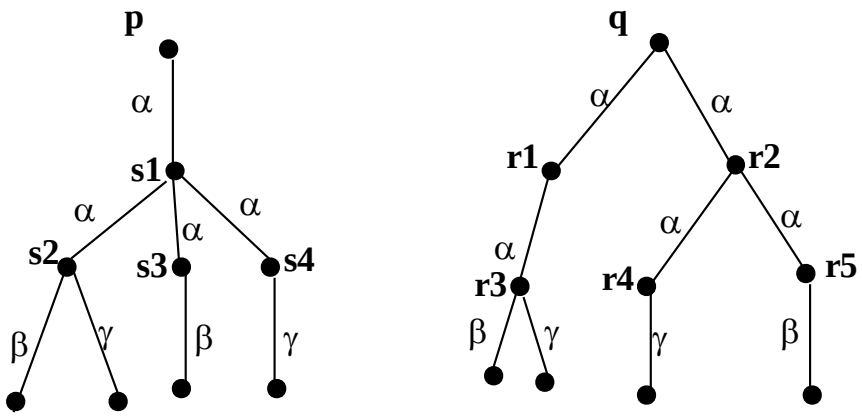
Верно, что $p \sim_2 q$: эксперимент α приводит p в состояния s1, для которого в процессе q после такого же эксперимента есть эквивалентное состояние r1; то же и для s2. Обратное тоже верно.

Поэтому $p \sim q$

Примеры проверки наблюдаемой эквивалентности

Верно, что $p \sim_0 q$.

Верно, что $p \sim_1 q$, оба допускают язык $\{\varepsilon, \alpha, \alpha\alpha, \alpha\alpha\beta, \alpha\alpha\gamma\}$.



Верно, что $p \sim_2 q$:

Действительно:

Для $p \xrightarrow{\alpha} s1$ есть $q \xrightarrow{\alpha} r1$ и $s1 \sim_1 r1$.

Для $p \xrightarrow{\alpha\alpha} s2$ есть $q \xrightarrow{\alpha\alpha} r3$ и $s2 \sim_1 r3$.

Для $p \xrightarrow{\alpha\alpha} s3$ есть $q \xrightarrow{\alpha\alpha} r5$ и $s3 \sim_1 r5$.

Для $p \xrightarrow{\alpha\alpha} s4$ есть $q \xrightarrow{\alpha\alpha} r4$ и $s4 \sim_1 r4$.

Обратно:

Для $q \xrightarrow{\alpha} r1$ есть $p \xrightarrow{\alpha} s1$ и $s1 \sim_1 r1$.

Для $q \xrightarrow{\alpha} r2$ есть $p \xrightarrow{\alpha} s1$ и $s1 \sim_1 r2$.

Для $q \xrightarrow{\alpha\alpha} r3$ есть $p \xrightarrow{\alpha\alpha} s2$ и $s2 \sim_1 r3$.

Для $q \xrightarrow{\alpha\alpha} r4$ есть $p \xrightarrow{\alpha\alpha} s4$ и $s4 \sim_1 r4$.

Для $q \xrightarrow{\alpha\alpha} r5$ есть $p \xrightarrow{\alpha\alpha} s3$ и $s3 \sim_1 r5$.

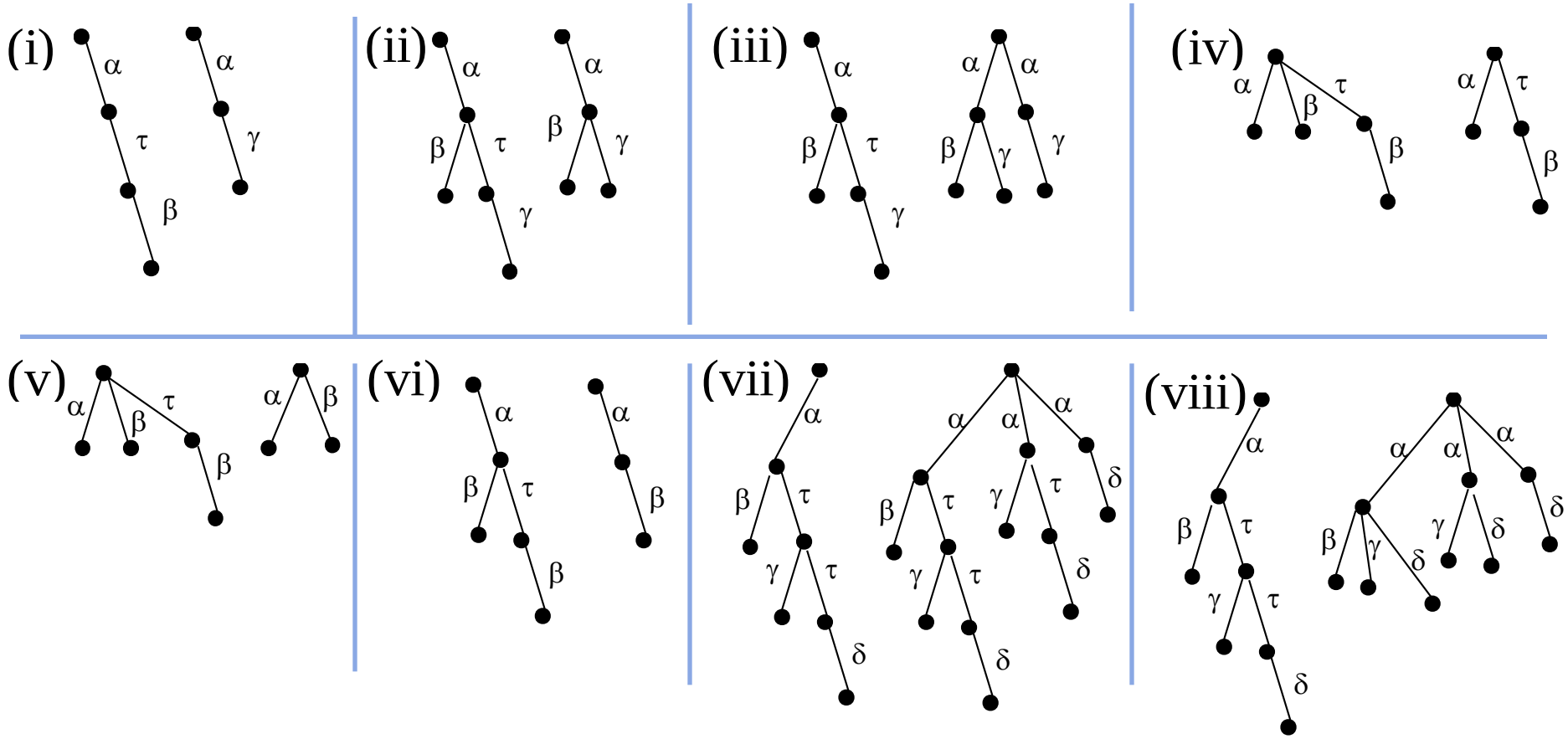
Но! $p \not\sim_3 q$: действительно:

$p \xrightarrow{\alpha} s1$, $q \xrightarrow{\alpha} r1$ и $q \xrightarrow{\alpha} r2$,
однако, $s1 \not\sim_2 r1$ и $s1 \not\sim_2 r2$.

Для каждого k можно построить агенты, которые k-наблюдаемо эквивалентны, но не являются k+1- наблюдаемо эквивалентными

Эквивалентность деревьев поведений (2)

Какие пары поведений наблюдаемо эквивалентны?



Первые три – проанализированы. На экзамене каждому будет дан какой-либо из оставшихся примеров

Синтаксис HML (Hennessy-Milner Logic)

- Формулы логики Хеннесси-Милнера (HML) над конечным словарем A строятся в соответствии со следующими правилами:
  - tt есть формула (имеет смысл True)
  - Если φ и ψ - формулы, то $\varphi \wedge \psi$ тоже формула
  - Если φ - формула, то $\neg\varphi$ тоже формула
  - Если φ - формула, $a \in A$, то $\langle a \rangle \varphi$ тоже формула
- Примеры формул: $\neg\langle a \rangle tt$; $\langle a \rangle \langle b \rangle tt \wedge \neg\langle c \rangle \neg tt$

Итак, HML – это обычная пропозициональная логика, в которой добавлена только одна конструкция: $\langle a \rangle \varphi$

Выводимые формулы HML:

$ff = \neg tt$

$\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi)$

$\varphi \Rightarrow \psi = \neg\varphi \vee \psi$

$[a]\varphi = \neg\langle a \rangle \neg\varphi$

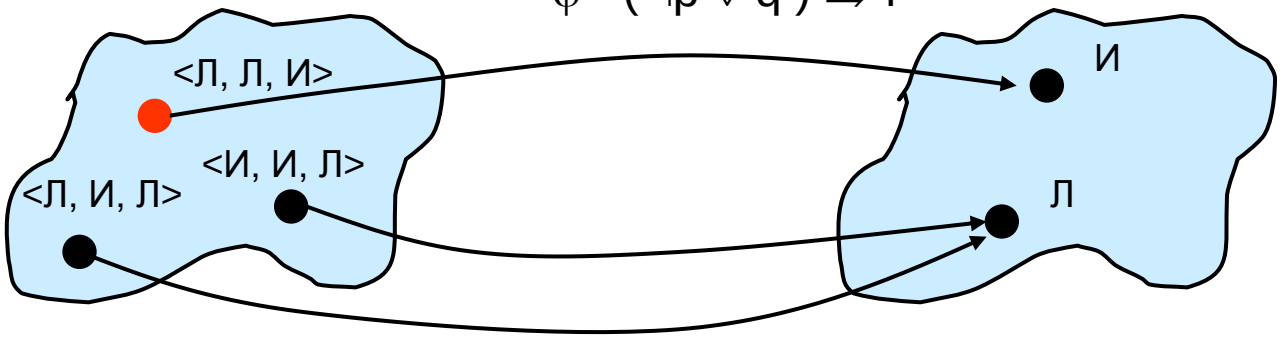
Структура (интерпретации) логики – помеченная система переходов. Атомарными формулами логики являются константы True и False.

M. Hennessy and R. Milner. On observing nondeterminism and concurrency. // In LNCS N 85, pp. 299–309. Springer-Verlag, 1980.

Интерпретации HML

Логика высказываний

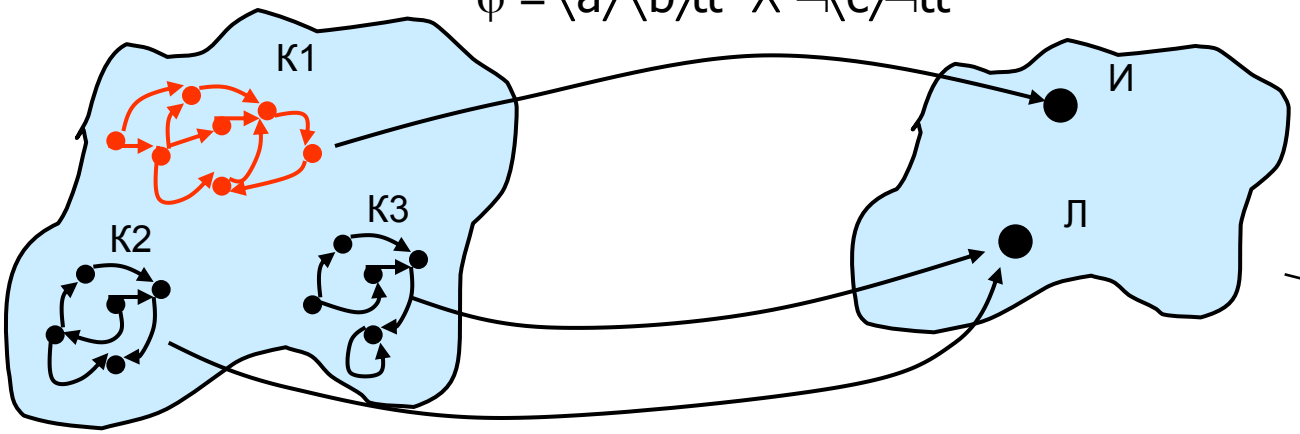
$\varphi = (\neg p \vee q) \Rightarrow r$



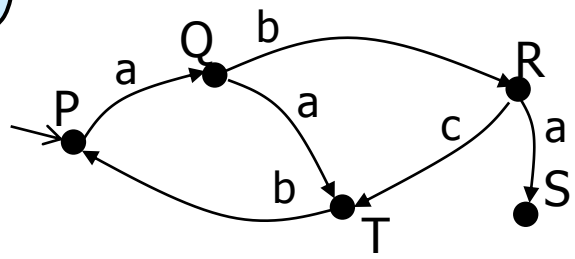
Интерпретации φ – наборы значений переменных p, q, r

Логика Хеннесси-Милнера

$\varphi = \langle a \rangle \langle b \rangle tt \wedge \neg \langle c \rangle \neg tt$



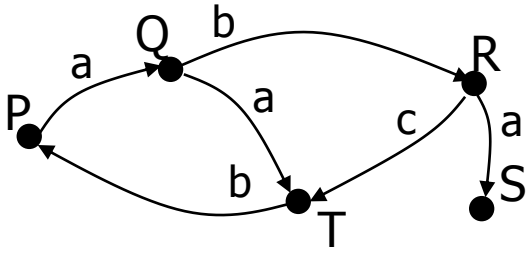
Интерпретации φ – состояния помеченных систем переходов



$P \models \langle a \rangle \langle b \rangle tt \wedge \neg \langle c \rangle tt$

Интерпретации HML (Hennessy-Milner Logic)

- Формулы HML интерпретируются на множествах вычислительных процессов, представленных помеченными системами переходов. Процессы сопоставляются состояниями системы переходов, переходы помечаются символами словаря A

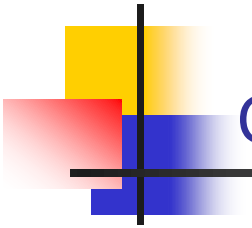


Пример:

- Процесс P может выполнить действие a и перейти в какое-нибудь состояние ($P \models \langle a \rangle tt$)
- Процесс R может выполнить действие a, или выполнить действие c, а затем действие b
  ($R \models \langle a \rangle tt \vee \langle c \rangle \langle b \rangle tt$)
- Процесс S не может выполнить никаких действий
  ($S \not\models \langle a \rangle tt \vee \langle b \rangle tt \vee \langle c \rangle tt$)

Интерпретации формул HML:

- операции $\wedge$, $\vee$, $\Rightarrow$ и $\neg$ обозначают обычные логические операции,
- $\langle a \rangle \varphi$ - модальная операция: формула $\langle a \rangle \varphi$ выполняется для процесса P тогда и только тогда, когда P может выполнить переход, помеченный a, после чего (новый процесс) будет удовлетворять формуле φ
- $[a]\varphi$ - модальная операция: формула $[a]\ \varphi$ выполняется для процесса P тогда и только тогда, когда P после выполнения всех переходов, помеченных a, новые процессы будут удовлетворять формуле φ (не существует такого перехода из P, помеченного a, после которого новый процесс НЕ удовлетворяет формуле φ)



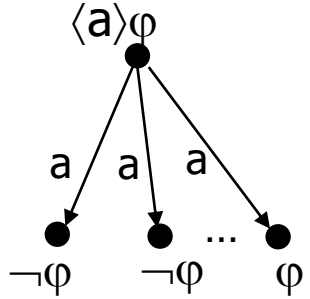
Семантика формул HML

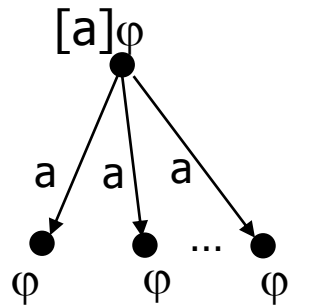
- Семантическая интерпретация формулы – множество состояний, удовлетворяющих ей для конкретной помеченной системы переходов
- Две формулы эквивалентны, если они эквивалентны на любой интерпретации. В HML две формулы эквивалентны на заданной интерпретации, если совпадают множества состояний, им удовлетворяющих

Формальная семантика Hennessy-Milner Logic

Логика Хеннесси-Милнера – это логика ветвящегося времени для помеченных систем переходов. Она позволяет делать заключения о возможных путях, которые ведут из конкретных состояния, и включает пометки на переходах

- Семантика.
  То, что процесс $P \in Pr$ удовлетворяет формуле φ (записывается $P \models \varphi$), формально определяется так:
  - $P \models true$
  - $P \models \neg\varphi$ iff $\neg(P \models \varphi)$
  - $P \models \varphi_1 \wedge \varphi_2$ iff $(P \models \varphi_1) \wedge (P \models \varphi_2)$
  - $P \models \langle a \rangle \varphi$ iff $(\exists Q \in Pr: P \xrightarrow{a} Q)\ Q \models \varphi$
  - $P \models \varphi_1 \vee \varphi_2$ iff $(P \models \varphi_1) \vee (P \models \varphi_2)$
  - $P \models \varphi_1 \Rightarrow \varphi_2$ iff $(P \models \varphi_1) \Rightarrow (P \models \varphi_2)$
  - $P \models [a] \varphi$ iff $(\forall Q \in Pr: P \xrightarrow{a} Q)\ Q \models \varphi$



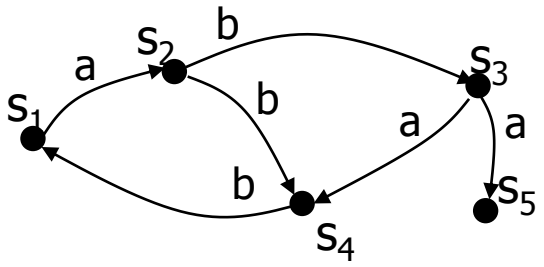


Множество всех возможных формул HML обозначим M

Оператор $\langle a \rangle$ отражает классическую модальность "возможность"
($\langle a \rangle \varphi$ - из s существует такой a-переход, после которого выполняется св-во φ)
Оператор [a] отражает классическую модальность "необходимость"
($[a]\varphi$ - из s после любого a-перехода выполняется свойство φ)

Семантика формулы HML

- Для манипуляций с формулами необходимо установить “смысл” формулы, ее семантику. Сравнивать формулы по их виду нельзя!
- С каждым свойством (формулой HML) свяжем множество таких состояний (подмножество M), в которых это свойство выполняется



Свойство	Соответствующее множество
$\langle a \rangle tt$	$\{s_1, s_3\}$
$[a]\langle b \rangle tt$	$\{s_1, s_2, s_4, s_5\}$
$\langle a \rangle [b]\ [a]\ tt$	$\{s_1, s_3\}$

- Семантика формулы – это множество состояний, в которых данное свойство удовлетворяется: $\| \phi \| = \{s \in S \mid s \models \phi\}$:
  - $\| tt \| = \{s_1, s_2, s_3, s_4, s_5\}$
  - $\| ff \| = \{\ \}$
  - $\| \langle a \rangle \langle b \rangle tt \| = \{s_1, s_3\}$

- Две формулы равны, если их семантические значения совпадают
- Семантика формулы зависит от конкретной интерпретации (LTS), которую анализируем

Логика Хеннесси-Милнера (Hennessi-Milner Logic)

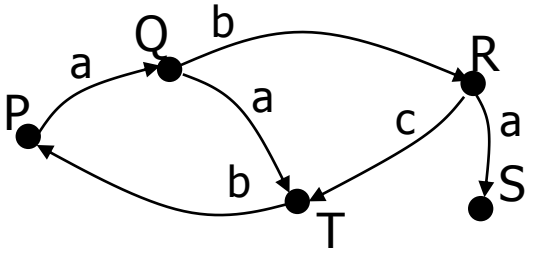
- Формулы логики Hennessy–Milner построены из множества S пометок a,b, ... В соответствии со следующей расширенной грамматикой:

$$f ::= tt \mid ff \mid \neg\varphi \mid \varphi\wedge\varphi \mid \varphi\vee\varphi \mid \varphi \Rightarrow\varphi \mid \langle a\rangle\varphi \mid [a]\,\varphi$$

Формулы HML интерпретируются на множестве LTS (Laballed Transitin Systems)

Семантика модальности $\langle a\rangle\varphi$ - в текущем состоянии возможно выполнить действие a с переходом в состояние, в котором выполняется φ

Семантика модальности $[a]\varphi$ - в текущем состоянии все действия a ведут в состояния, в которых выполняется φ



Интерпретация формулы φ обозначается $||\ \varphi\ ||$

$$||\ ff||\ = \varnothing$$
$$||\ tt\ ||\ = S$$
$$||\varphi_1\wedge\varphi_2|| = ||\varphi_1||\cap||\varphi_2||$$
$$||\varphi_1\vee\varphi_2|| = ||\varphi_1||\cup||\varphi_2||$$
$$||\ \langle a\rangle\ \varphi\ ||\ = \{\ s\in S \mid (\exists s'\colon s -a\rightarrow s'\ )\ \ s'\in||\ \varphi\ ||\ \}$$
$$||\ [a]\ \varphi\ ||\ = \{\ s\in S \mid (\forall s'\colon s -a\rightarrow s'\ )\ s'\in||\ \varphi\ ||\ \}$$

- Определение. В состоянии s выполняется формула f (т.е. $s \models \varphi$) iff $s \in ||\varphi||$

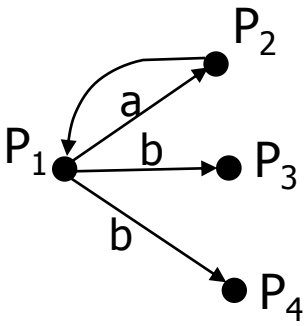
Формализация эквивалентности процессов в HML

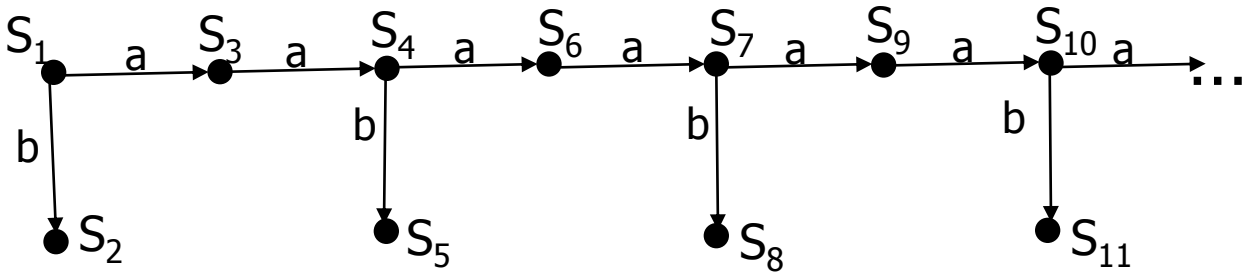
Теорема. Два процесса P и Q наблюдаемо эквивалентны, если для любой формулы ψ логики Хеннесси-Милнера (HML) выполняется:

$$P \models \psi \Leftrightarrow Q \models \psi$$

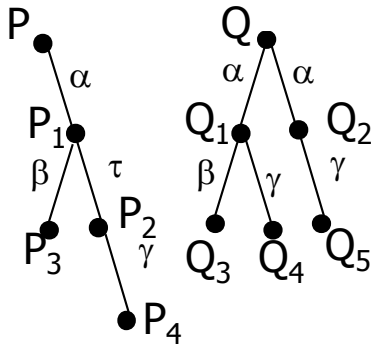
$$P \sim Q \quad \text{iff} \quad P \models \psi \Leftrightarrow Q \models \psi$$

Примеры

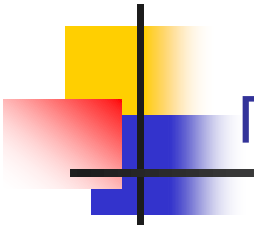




- Не все свойства состояний систем переходов являются модальными. P1 и S1 – находятся в отношении бисимуляции, но у них различны свойства
  - “иметь одинаковое число выходов”
  - “быть бесконечным графом”
- Эти свойства не являются бисимуляционными.

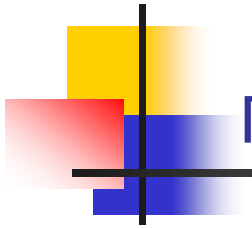


Процессы $\{P_3, P_4, Q_3, Q_4, Q_5\}$ все эквивалентны
Процессы $\{P_2, Q_2\}$ эквивалентны
Процессы $\{P_1, Q_1\}$ эквивалентны
… …
… …



Применение CCS

- Были разработаны автоматизированные системы, выполняющие анализ в рамках CCS – например, Edinburgh Concurrency Workbench
- Известно, что протоколы очень трудно проверять. Были разработаны три языка спецификации протоколов:
  - SDL – язык состояний и переходов (в двух эквивалентных - графической и программной - формах). Не является формальным
  - Language of Temporal Ordering Specifications (Lotos) стандарт ISO (1989г.), основан на CCS. Позволяет формально специфицировать поведение протокола на высоком уровне абстракции. Позволяет использовать множество правил трансформации поведения и отношение эквивалентности, можно проверить НЕКОТОРЫЕ свойства поведения. Но в общем проблема верификации систем, описанных на Lotos, неразрешима
  - Estelle – также формальный язык спецификации протоколов, основан на расширенной модели КА



Проблема для любознательных

- Освоить работу с инструментом верификации Concurrency Workbench и проверить корректность простых параллельных программ

  http://people.cs.aau.dk/~srba/courses/SV-08/material/03.pdf

π-calculus (pi-исчисление) – алгебра мобильных процессов (верификация мобильных приложений)

Р.Милнер расширил CCS для возможности формального анализа процессов мобильной коммуникации

- Мобильность процессов и изменение конфигурации взаимодействия систем НЕ ПОКРЫВАЕТСЯ CCS
- pi-calculus – это расширение исчисления CCS для того, чтобы работать с системами процессов с изменяемой конфигурацией (мобильными процессами и реконфигурируемыми связями)
- В дополнение к простейшему CCS, в пи-исчислении можно создавать новые имена каналов и передавать их другим процессам при коммуникации || процессов. Процесс, принявший имя канала, может взаимодействовать по этому новому каналу с окружением
- Это исчисление позволяет описывать распределенные и мобильные вычисления, оно включает λ-исчисление Черча и машины Тьюринга. Служит базой для других теорий. Создано множество расширений этого исчисления, в частности, для ООП

Robin Milner. Communicating and Mobile Systems: the π-calculus, 1999, Cambridge
D.Sangiorgi, D.Walker. The π-calculus: a Theory of Mobile Processes, 2001, Cambridge
B.Victor, F.Moller. The Mobility Workbench – a tool for the π-calculus. In CAV'94

CCS и π-Calculus

ППФ:

0 – пустой процесс, Nil

P, Q, ... - имена процессов (бесконечное множество имен)

P|Q – параллельная композиция

a(x).P или a(x)!.P – вывод по каналу a значения x

a(y).P или a(y)?.P – ввод по каналу a нового значения для y

{v/x} P – в терме P все свободные вхождения x заменяются на v

{new x} P – в процессе P имя x является локальным

Monadic calculus – передача одного значения

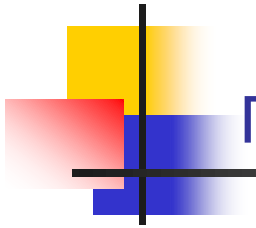
Poliadic calculus – передача вектора значений

CCS: $a!.\ P \mid a?.Q \xrightarrow{\tau} P|Q$

CCS с передачей значений: $a!(v).\ P \mid a?(x).Q \xrightarrow{\tau} \{v/x\}\ P|Q$

π-исчисление: $x!\ (y).\ P \mid x?\ (z).Q \xrightarrow{\tau} \{y/z\}\ P|Q$ но y и z теперь – и имена каналов

Пример выражения π-исчисления: $x!(z).Q \mid x?(y).\ y?(u).\ 0 \xrightarrow{\tau} Q \mid z?(u).\ 0$



Примеры

$x?(y).P \mid x!(z).\ Q \ \xrightarrow{-\tau}\ P\{z/y) \mid Q$

Процессы взаимодействуют по порту x, процесс x?(y).P принимает z, и это имя заменяет в P все свободные вхождения y

$x?(y).y!(u).0 \mid x!(z).\ Q \ \xrightarrow{-\tau}\ z!(u).\ 0 \mid Q$

Процессы взаимодействуют по порту x, процесс x?(y).P принимает z, и это имя заменяет в процессе y!(u).0 все свободные вхождения y, т.е. y!(u).0 становится процессом z!(u).0

Пример поведения с передачей значений

$x!(z).\,0 \mid x?(y).\,y!(x).\,x?(y).\,0 \mid z?(v).\,v!(v).0$

P1 P2 P3

Что происходит?

ШАГ 1. Первые два процесса взаимодействуют по каналу x.
В процессе P2: y:=z, т.е. переменная y - имя канала – приняла значение z.

Поведение всей системы после первого шага:

$0 \mid z!(x).x?(z).0 \mid z?(v).v!(v).0$ первый процесс стоит

ШАГ 2. Передача теперь идет через z. Процесс P2 передает по z значение x, а P3 принимает это значение, присваивая переменной v значение x, теперь в P3 v=x

Поведение после второго шага:

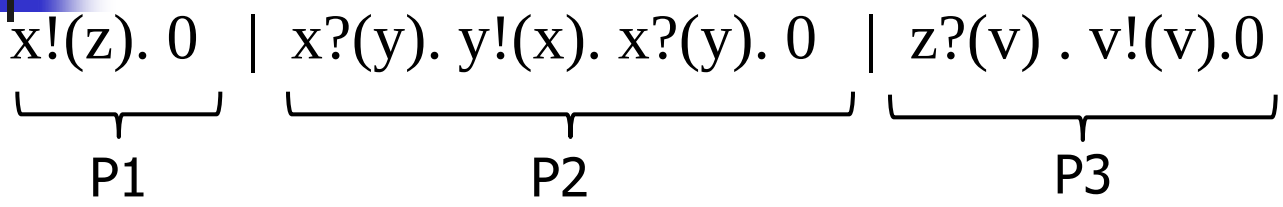
$0 \mid x?(z).0 \mid x!(x).0$

3. Последний шаг в функционировании процессов – это взаимодействие второго и третьего процессов по каналу x, передается имя x этого канала. В процессе P2 оно присваивается переменной z:

Поведение после третьего шага:

$0 \mid 0 \mid 0$ все процессы стоят

Пример анализа формулы пи-исчисления



- x!(z) . 0 | x?(y) . z! (x) . x?(y) . 0 | z?(v) . v! (v) . 0

Шаг 1: P1 передает имя z по порту x и становится 0
P2 принимает по x имя z, заменяет им имя порта y

- 0 | z! (x) . x? (z) . 0 | z?(v) . v! (v) . 0

Шаг 2: Взаимодействие по z, *x* заменяет v

- 0 | x? (z) . 0 | x! (x) . 0

Шаг 3: По каналу x передается имя x, которое заменяет имя z

- 0 | 0 | 0

Пример: обслуживание серверами клиентских процессов

Имеем пул серверов Si и множество клиентских процессов Pk, которым время от времени нужно обслуживание. Каждый сервер может обслужить любой процесс, если свободен.

Как организовать работу?

РЕШЕНИЕ:

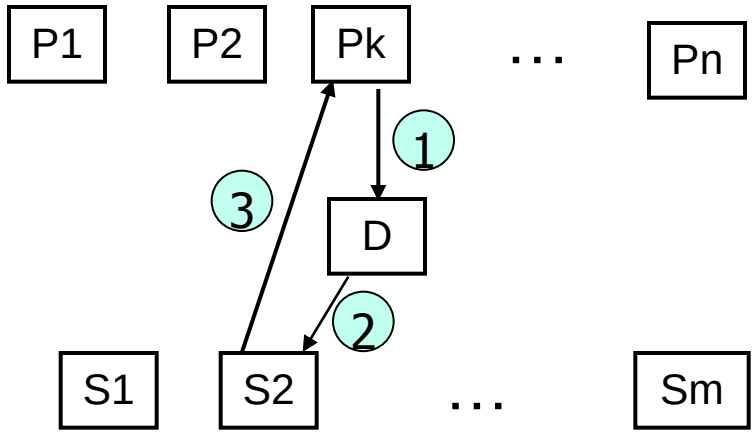
Вводим дополнительный процесс-диспетчер.

Сценарий:

1. Клиентский процесс для получения обслуживания обращается к диспетчеру и регистрируется - передает свое имя (адрес) и данные

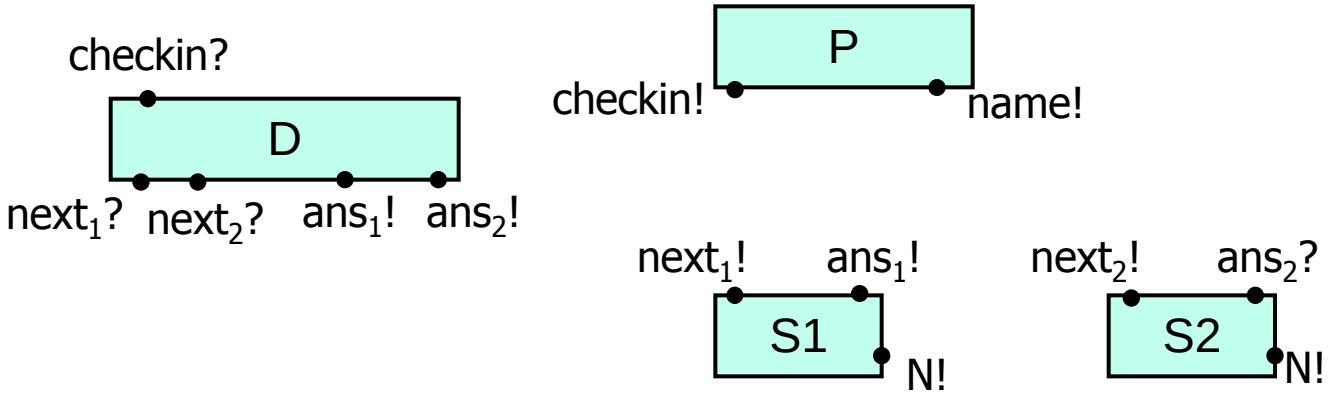
2. Диспетчер запрашивает, какой из серверов свободен, и передает ему имя клиента и его данные для обработки

3. Сервер связывается с клиентом по его имени и возвращает ему обработанную информацию, после чего снова становится готовым к обслуживанию



Формальное описание системы с двумя серверами

В CCS такая реконфигурация процессов невозможна



Диспетчер D ждет сигнала от очередного клиента с приемом его имени (N) и запроса (s), ждет сигнала от готового сервера (1 или 2), передает ему имя и запрос очередного клиента

$$D = checkin?(N, s) \,.\, (next_1?.\, ans_1!(N,s) \,.\, D + next_2?.\, ans_2!(N,s) \,.\, D)$$

i-й сервер Si посылает информацию о готовности, получает в ответ имя клиентского процесса и запрос, после чего выдает ***этому клиенту*** требуемые им данные

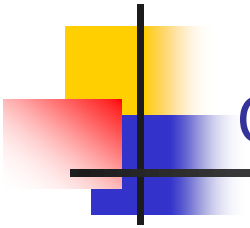
$$S_i = next_i!.\, ans_i\,!\,(N, s) \,.\, N!(rez(s)\,) \,.\, S_i$$

Клиентский процесс посылает свое имя и запрос (data), получает на свое имя результат

$$P(name, data) = checkin!\,(name, data) \,.\, name?(x) \,.\, P'(x)$$

CSP – Communicating Sequential Processes (Xoap)

- Язык для описания взаимодействующих процессов, “наследник” CCS. CSP поддержан элегантной математической теорией и инструментами верификации. Удобен для моделирования и анализа систем, которые включают сложный обмен сообщениями, в частности, криптопротоколов
- Книга *Communicating Sequential Processes* опубликована в 1985. В 2006 эта книга была третьей по числу ссылок на нее в области Computer Science
- Программа на CSP описывает параллельно функционирующие последовательные процессы, взаимодействующие рандеву
- Применение CSP для разработки safety critical systems:
  - использован для верификации процессорного конвейера и Virtual Channel Processor
  - Bremen Institute for Safe Systems и Daimler-Benz Aerospace построили модель отказоустойчивой системы управления (23 000 строк кода) на языке CSP для использования в Международной космической станции. Модель была проанализирована с целью проверки того, что в системе нет блокировок и ливлоков. Верификатор вскрыл множество ошибок, которые было трудно обнаружить тестированием
- Язык CSP использован в транспьютерах, на нем основан язык Оккам



OCCAM и транспьютеры

- В 1984 г. фирма INMOS Ltd. (Бристоль, Англия) объявила о выпуске микропроцессоров нового типа для параллельного программирования – транспьютеров (transistor computer), с языком программирования OCCAM (язык интерпретировался аппаратно)
- Уильям О́ккам (*William of Ockham)* ~1285 философ, францисканский монах из Оккама (Южная Англия). «Бритва О́ккама» — методологический принцип: *«Не следует привлекать новые сущности без крайней необходимости»*.
- Транспьютеры – параллельно функционирующие процессоры- сложный суперскалярный конвейерный процессор
- OCCAM – это язык, реализующий операции и идеологию алгебр процессов CCS Милнера и CSP Хоара. Процессы взаимодействуют по рандеву
- Реализация взаимодействия: если процессы в разных процессорах, то взаимодействие между ними реализуется по реальным физическим каналам, если процессы в одном процессоре, то взаимодействие между ними моделируется виртуальными каналами (то же в языке Ада)

Язык OCCAM: структура программы

- Примитивные процессы

```
keyboard ? x          /* ввод с клавиатуры значения в переменную x  */
screen ! y+x          /* вывод на экран значения выражения y+x  */
x:=x+1                /* оператор присваивания  */
```

- Конструкторы SEQ, PAR, ALT

```
SEQ                   /* процессы, выполняющиеся последовательно */
   buffer.in   ?   x
   buffer.out  !   x
```

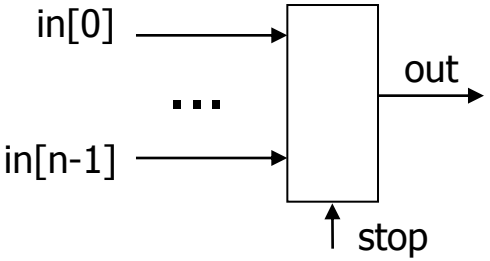
-

```
PAR                   /* процессы, выполняющиеся параллельно   */
   x := x+1
   z :=x*x
```

-

```
ALT                   /* несколько guarded commands – защищенных процессов */
  count1 < 100  &  c1 ? data
    SEQ
       count1 := count1 + 1
       merged ! data
  count2 < 100 & c2 ? data
    SEQ
       count2 := count2 + 1
       merged ! data
  status ? request
    PAR
       out ! count
       in ?  x
```

Простой пример программы на языке Оккам



Буфер с несколькими входами и одним выходом

Принимает по любому входу, следующим действием выдает принятое значение по порту out

```
VAR going :
SEQ
    going := TRUE
    WHILE going
        VAR x :                      /* описание переменной x  */
        ALT
          ALT i = [ 0 FOR n ]
            SEQ
                 in [ i ] ? x        /* принимает по любому готовому входу,
                out ! x                передает принятое значение на выход  */
          stop ? ANY                /* до тех пор, пока не будет принят сигнал
            going := FALSE            синхронизации по входу stop  */
```

Повторение имеет форму: [*первый* FOR *количество*]

ANY – фиктивная переменная, используется для синхронизации по событиям

Язык OCCAM: пример программы мультиплексора

По нескольким входным каналам параллельно поступают потоки байтов.

Как только по какому-нибудь из них поступит полная пачка (слово) (например, 100 байтов), ее нужно передать по выходному каналу

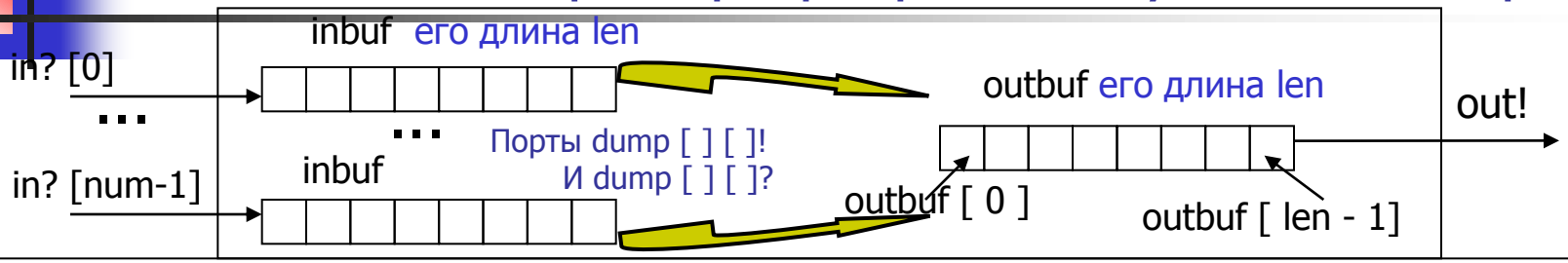
Как реализовать такую программу???

Конечно, это должны быть параллельно функционирующие процессы, между ними должна быть синхронизация

Построить такую программу на обычном языке довольно сложно

На Оккаме это делается очень изящно

Язык OCCAM: пример программы мультиплексора




```
DEF num = 8, len = 100
PROC multiplexor ( CHAN in [ num ],  out )
    CHAN dump [ num ] [ len ]     /* матрица каналов, при каждом i это массив входных и вых портов */
    PAR                           /* параллельно: 1) прием в inbuf по in и выдача в outbuf  по dump!  */
                                  /*              2) прием в outbuf  по dump? и выдача в out          */
      PAR i = [ 0 FOR num ]       /* параллельно по i: прием в inbuf [ i ] по in? и выдача по dump[i]!  */
        VAR inbuf [ len ] :
        WHILE TRUE                /*  бесконечный цикл приема в inbuf, потом выдачи из него *
Для каждого i     SEQ
параллельно         SEQ  j = [ 0 FOR len ]   /*последовательно вводим в i-ый inbuf */
                      in [ i ] ? inbuf [ j ]
                    PAR j = [ 0 FOR  len ]   /* когда все в i-ю строку набрались (SEQ!!), || выводим*/
                      dump [ i ] [ j ] ! inbuf [ j ]  /* из i-линейки inbuf по i-й линейке портов dump! */
      WHILE TRUE                  /* бесконечный цикл приема по dump? в outbuf и выдачи из outbuf по out! */
        VAR outbuf [ len ] :
        ALT i = [ 0 FOR num ]     /* альтернативно выбираем какую-нибудь готовую строку */
Выбираем  dump [ i ] [ 0 ]  ?  outbuf [ 0 ]  /* если готов 0-й (хоть какой!), то готова вся строка  */
готовую           SEQ
строку              PAR k = [ 1 FOR len-1 ]     /* параллельно, начиная с 1-го, 0-й уже получен */
                      dump [ i  ] [ k ] ? outbuf [ k ] /* из выбранного в outbuf,  а потом*/
                    SEQ j = [ 0 FOR len ]         /* последовательно выводим из outbuf по out   */
                      out ! outbuf [ j ]
```

Заключение (алгебры процессов)

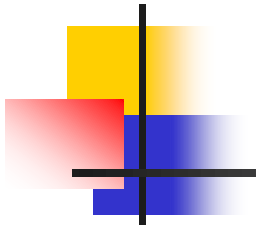
- CCS – первая попытка формализации параллельных взаимодействующих процессов на основе логико-алгебраического подхода для верификации, отражающей параллелизм, коммуникацию, недетерминизм, а не операционного исполнимого графического формализма, как СП)
- Разработаны системы для проверки наблюдаемой эквивалентности (бисимуляции) заданных процессов.
- Для CCS разработано расширение с введением формализмов, описывающих передачу значений при взаимодействии. Инструментов, поддерживающих верификацию систем с передачей значений, нет
- Был стандартизирован язык Lotos, основанный на CCS, для формальной спецификации и анализа протоколов
- Взаимодействие процессов по рандеву используется во многих моделях представления параллельных процессов, в частности в языке PROMELA
- В π-исчислении вдобавок к простейшему CCS, можно создавать новые имена каналов и передавать их другим процессам, в связи с чем можно описывать мобильные процессы, процессы с реконфигурацией и т.п.
- π-исчисление сейчас интенсивно исследуется. На нем основано множество расширений, например, формализм спецификации и анализа криптографических протоколов

Персоналии: Робин Милнер

- Robin Milner (1934 -2010) – профессор Эдинбургского университета, английский учёный в области теории вычислительных систем
- Милнер разработал систему автоматических доказательств теорем LCF (Logic for Computable Functions). Для этой системы им был создан функциональный язык программирования ML. Позже Р. Милнер разработал исчисление взаимодействующих систем (CCS), теоретическую основу для анализа взаимодействующих систем, а также его расширение — пи-исчисление.
- Милнер является обладателем девяти почётных докторских титулов от разных университетов.

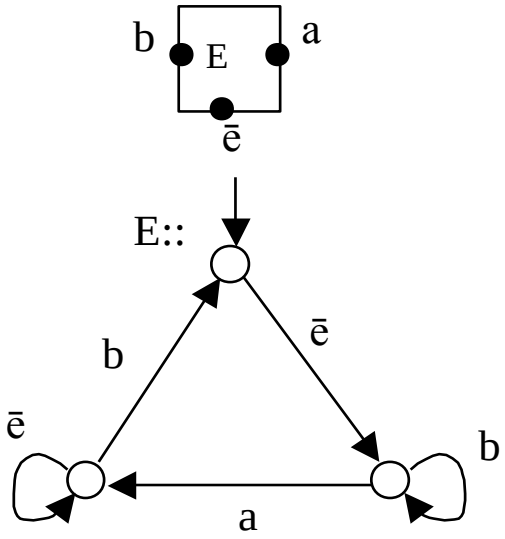
Награды

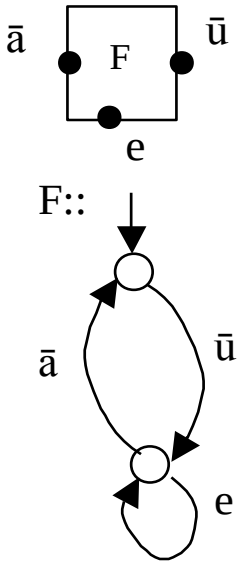
1988 — почётное членство в Лондонском королевском обществе
1991 — премия Тьюринга за LCF, ML и CCS
1994 — членство в ACM
2004 — королевская медаль от Эдинбургского королевского общества

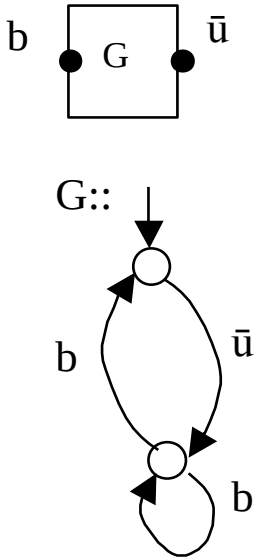


Спасибо за внимание

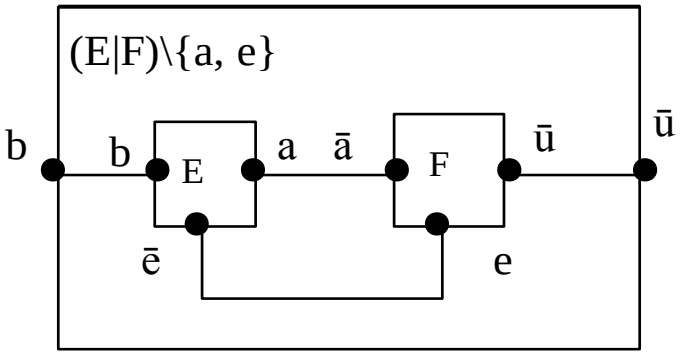
Пример задачи на экзамене



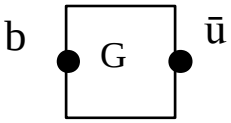




- Заданы три процесса, E, F и G.
- Проверить, будет ли параллельная композиция (E|F)\ {a,e} наблюдаемо эквивалентна процессу G



≈ (?)



Задача на π-исчисление

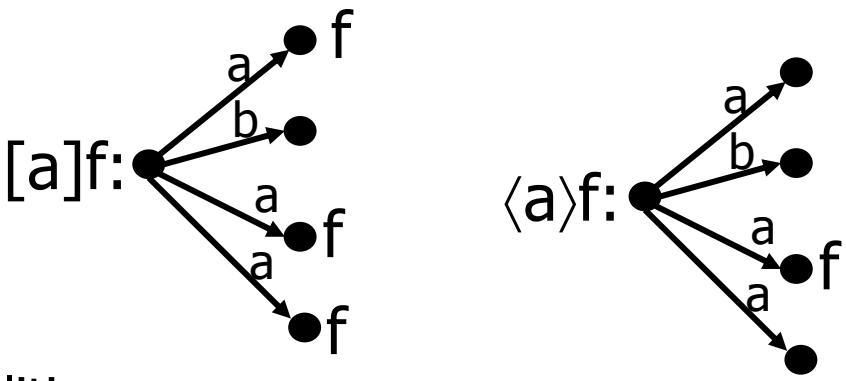
Проанализировать работу системы, описывающей произвольное число серверов (модификация предыдущей задачи)

$D = checkin\,(n, s)\,.next\,(a).\underline{a}\,(n,s)\,.\,D$

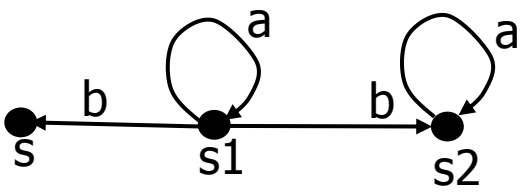
$S_i = \underline{next}\,(ans_i)\,.\,ans_i\,(n, s).\,\underline{n}\,(rez(s)\,)\,.\,S_i$

$P(name, data) = \underline{checkin}\,(name, data)\,.\,name\,(x)\,.\,P'$

- Branching-time modalities



- Примеры branching-time modalities



Для каких множеств состояний X выполняется следующая формула

$X = \langle a \rangle \text{ true} \vee [b]X$

Для каких множеств состояний X выполняется следующая формула

$X = \langle a \rangle \text{ true} \vee ([b]X \wedge \langle b \rangle \text{ true})$

В множестве: {s2}