

# Математические модели последовательных вычислений

mk.cs.msu.ru → Лекционные курсы  
→ Математические модели последовательных вычислений

## Блок 11

Другие виды сетей Петри

Лектор:

**Подымов Владислав Васильевич**

E-mail:

**valdus@yandex.ru**

# Вступление

Сети Петри широко применяются для моделирования систем в разнообразных областях

Но нередко возникает потребность в использовании варианта сетей, существенно отличающихся от рассмотренных (**обыкновенных**)

Это могут быть

- ▶ как более просто устроенные сети, для которых шире спектр разрешимых задач и их решение устроено проще,
- ▶ так и сложнее устроенные, в которых добавляются новые средства моделирования

# Обыкновенные сети Петри

Обыкновенная сеть Петри называется **ординарной**, если **вес** каждой дуги равен 1, то есть если в сети разрешены только дуги вида  $\xrightarrow{1}$ , они же просто  $\rightarrow$

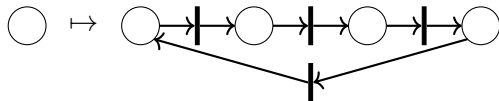
Хотя запрет на использование весов дуг и кажется существенно упрощающим модель, но всё же оказывается, что выразительные возможности сетей от этого не изменяются: можно преобразовать произвольную обыкновенную сеть Петри  $\pi$  в ординарную  $\pi'$  так, чтобы вычисления этих сетей были *по сути* одинаковы

Преобразование  $\pi = (P, T, E, W, M_0)$  в  $\pi' = (P', T', E', \_, M'_0)$  можно осуществить так

Пусть  $w$  — максимальный вес дуги в  $\pi$

1. Каждую позицию  $p$  «расщепим» на  $w$  копий  $(p[1], \dots, p[w])$ , соединив эти копии в кольцо переходами

**Например**, для  $w = 4$ :

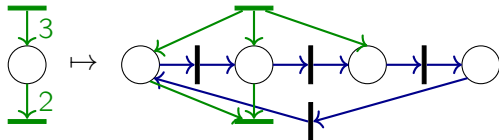


## Ординарные сети Петри

$$\pi = (P, T, E, W, M_0) \xrightarrow{W} \pi' = (P', T', E', \_, M'_0)$$

- Каждую дугу  $\_ \xrightarrow{n} p$  «расщепим» на  $\_ \rightarrow p[1], \dots, \_ \rightarrow p[n]$
- Каждую дугу  $p \xrightarrow{n} \_$  «расщепим» на  $p[1] \rightarrow \_, \dots, p[n] \rightarrow \_$

Например:



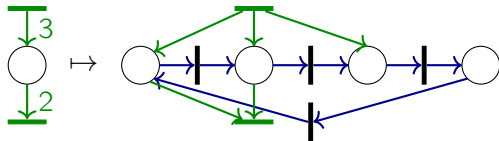
- Начальную разметку  $M'_0$  зададим так:

$$M'_0(p[1]) = M_0(p) \quad \text{и} \quad M'_0(p[2]) = \dots = M'_0(p[n]) = 0$$

*В итоге* получается ординарная сеть Петри, состоящая из **старых переходов** и **новых переходов** и произвольно распределяющая фишки среди копий позиции

# Ординарные сети Петри

$$\pi = (P, T, E, W, M_0) \xrightarrow{w} \pi' = (P', T', E', \_, M'_0)$$



**Утверждение.** Сеть Петри  $\pi$  имеет вычисление

$$M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \dots \xrightarrow{t_n} M_n$$

тогда и только тогда, когда ординарная сеть  $\pi'$  имеет вычисление

$$M'_0 \xrightarrow{t'_1} M'_1 \xrightarrow{t'_2} \dots \xrightarrow{t'_k} M'_k,$$

в котором

- ▶ подпоследовательность **старых переходов** последовательности  $t'_1, \dots, t'_k$  есть  $t_1, \dots, t_n$ , и
- ▶ для любой позиции  $p$  верно  $M_n(p) = M'_k(p[1]) + \dots + M'_k(p[w])$

## Ординарные сети Петри

$$\pi = (P, T, E, W, M_0) \xrightarrow{w} \pi' = (P', T', E', \_, M'_0)$$

Иными словами, сети  $\pi$  и  $\pi'$  взаимно моделируют вычисления друг друга

**Следствие.** Сеть  $\pi$  является живой (ограниченной) [консервативной] тогда и только тогда, когда сеть  $\pi'$  является живой (ограниченной) [консервативной]

Значит, можно

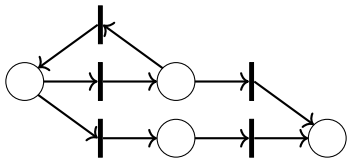
- ▶ для моделирования использовать любые обыкновенные сети Петри, и при этом
- ▶ разрабатывать алгоритмы анализа поведения только для ординарных сетей Петри

# Автоматные сети Петри

Сеть Петри называется **автоматной**, если для любого её перехода  $t$  верно  $|\bullet t| = |t\bullet| = 1$

Из определения следует, в частности, что любая автоматная сеть ординарна

**Например:**



Позиции автоматной сети отвечают состояниям автомата  
Срабатывание перехода автоматной сети перемещает фишку из одного состояния и отвечает выполнению перехода в автомате

Фишка в позиции  $p$  означает, что экземпляр автомата находится в состоянии  $p$

В сети может быть и несколько фишек — это отвечает одновременному выполнению нескольких автоматов в **семантике чередующихся**

**вычислений**

# Автоматные сети Петри

**Утверждение.** Любая автоматная сеть консервативна

**Утверждение.** Автоматная сеть жива  $\Leftrightarrow$  она представляет собой сильно связный граф

**Утверждение.** Существует полиномиальный по времени алгоритм проверки достижимости заданной разметки в заданной автоматной сети

Доказательство (этих трёх утверждений).

Можете попробовать самостоятельно

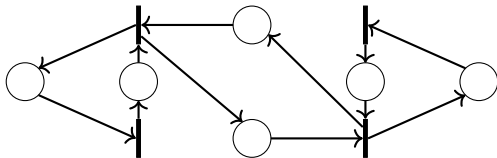


# Синхронизационные сети Петри

Сеть Петри называется **синхронизационной** (или, по-другому, — **синхрографом**), если для любой её позиции  $p$  верно  $|\bullet p| = |p\bullet| = 1$

Из определения следует, в частности, что любая синхронизационная сеть Петри ординарна

**Пример:**



Цикл в синхрографе может пониматься как циклический процесс (в «обыденном» понимании), и один виток перемещения фишки по циклу — одной итерации выполнения процесса

Переход может принадлежать нескольким процессам, обозначая точку синхронизации их выполнения

В такой сети может количество фишек может изменяться, но суммарное число фишек в каждом цикле (ресурсов процесса) остаётся неизменным

# Синхронизационные сети Петри

**Утверждение.** Для любых синхронизационной сети Петри, её простого цикла  $p_1 \rightarrow t_1 \rightarrow p_2 \rightarrow t_2 \rightarrow \dots \rightarrow p_n \rightarrow t_n \rightarrow p_1$  ( $p_1$  — позиция) и разметок  $M_1, M_2$ , таких что  $M_2$  достижима из  $M_1$ , верно  $\sum_{i=1}^n M_1(p_i) = \sum_{i=1}^n M_2(p_i)$

**Утверждение.** Синхронизационная сеть Петри жива  $\Leftrightarrow$  выполнены следующие два условия:

1. В позициях каждого простого цикла содержится хотя бы одна фишка
2. Каждая позиция достижима хотя бы из одного цикла

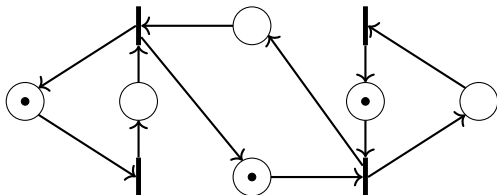
**Утверждение.** Живая синхронизационная сеть Петри ограничена  $\Leftrightarrow$  каждая позиция принадлежит хотя бы одному циклу

**Утверждение.** Живая синхронизационная сеть Петри безопасна  $\Leftrightarrow$  каждая позиция принадлежит хотя бы одному циклу, в позициях которого лежит ровно одна фишка

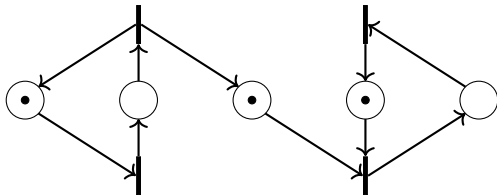
**Доказательство.** Можете попробовать самостоятельно

# Синхронизационные сети Петри

**Пример** живой безопасной (и ограниченной) синхронизационной сети:



**Пример** живой неограниченной синхронизационной сети:

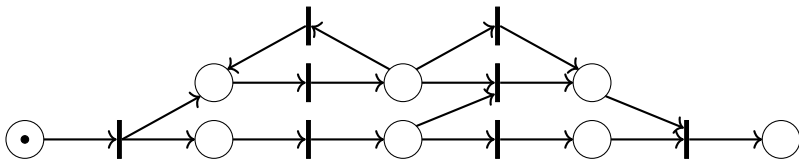


# Сети потоков работ

Сетью потоков работ (WF-сетью, workflow net) называется сеть Петри, устроенная следующим образом:

1. Ровно для одной позиции  $in$  (входной) верно  $\bullet in = \emptyset$
2. Ровно для одной позиции  $out$  (выходной) верно  $out \bullet = \emptyset$
3. Каждая вершина сети лежит на каком-либо пути из входа в выход
4. Начальная разметка содержит ровно одну фишку, и эта фишка лежит в  $in$

Пример:



# Сети потоков работ

WF-сети применяются при описании бизнес-процессов, планов работ больших проектов и аналогичных наборов действий со взаимозависимыми этапами выполнения

Начало работ отмечается фишкой во входной позиции в начальной разметке

Для обозначения полного/успешного/корректного завершения всех работ используется выходная позиция

Разметка WF-сети называется **заключительной**, если в выходной позиции лежит хотя бы одна фишка

Заключительная разметка называется **правильной**, если

- ▶ в выходной позиции лежит ровно одна фишка и
- ▶ в остальных позициях не лежит ни одной фишки

# Сети потоков работ

Для WF-сетей разумны такие свойства:

- ▶ Отсутствие **тупиков**: достижимых ситуаций, в которых невозможно завершить выполнение работ
- ▶ **Сбалансированность**: отсутствие незадействованных ресурсов по окончании работ
- ▶ Отсутствие **избыточности**: частей сети, не задействующихся ни в одном сценарии выполнения работ

WF-сеть называется **бездефектной**, если

- ▶ из любой достижимой разметки достижима заключительная разметка,
- ▶ достижима ровно одна заключительная разметка — правильная и
- ▶ для любого перехода существует вычисление, в котором срабатывает этот переход

**Утверждение.** WF-сеть  $\pi$  бездефектна  $\Leftrightarrow$  сеть, получающаяся из  $\pi$  добавлением перехода  $out \rightarrow | \rightarrow in$ , жива и ограничена

# Ингибиторные сети Петри

Согласно **свойству монотонности** сетей Петри, если в заданной разметке переход активен, то невозможно сделать его неактивным, добавляя фишки в позиции

Можно попытаться «устранить» монотонность сети, явно добавив возможность блокировать переход добавлением фишек

Эту возможность можно устроить, например, добавив в сеть особые дуги (**ингибиторные**) с таким смыслом:

- ▶ если в позиции в начале ингибиторной дуги есть хотя бы одна фишка, то переход в конце дуги считается неактивным
- ▶ иначе активность перехода определяется как обычно

# Ингибиторные сети Петри

Ингибиторная сеть Петри — это обыкновенная сеть Петри, в которую добавлено особое множество ингибиторных дуг  $E_0 \subseteq P \times T$

Ингибиторную дугу  $(p, t)$  принято изображать так:  $p \text{---} \circ \text{---} t$

Переход  $t$  в ингибиторной сети Петри **активен** в разметке  $M$ , если

- ▶  $E(\bullet, t) \preceq M$  (как в обыкновенной сети Петри) и
- ▶ для каждой ингибиторной дуги  $(p, t)$ , оканчивающейся в переходе  $t$ , верно  $M(p) = 0$

Оказывается, что добавление ингибиторных дуг в модель сети Петри делает эту модель алгоритмически полной



# Ингибиторные сети Петри

**Машина Минского** (счётчиковая машина) — это одна из известных алгоритмически полных вычислительных моделей

Машина Минского состоит из конечного набора ячеек памяти (счётчиков) с начальными значениями 0, способных хранить неотрицательные целые значения, и последовательности команд двух видов:

1.  $x := x + 1$ ; *goto*  $k$

- ▶ Увеличить значение счётчика  $x$  на 1 и перейти к выполнению  $k$ -й команды последовательности

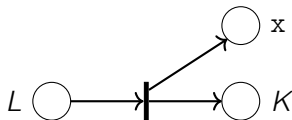
2. **if**  $x = 0$  **then** *goto*  $k_1$  **else**  $x := x - 1$ ; *goto*  $k_2$  **fi**

- ▶ Если счётчик  $x$  имеет значение 0, то перейти к выполнению  $k_1$ -й команды, а иначе уменьшить значение этого счётчика и перейти к выполнению  $k_2$ -й команды

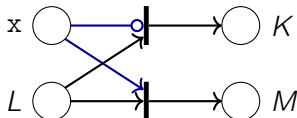
# Ингибиторные сети Петри

Команды несложно моделируются переходами ингибиторных сетей Петри

$L : x := x + 1; \text{ goto } K$



$L : \text{ if } x = 0 \text{ then goto } K \text{ else } x := x - 1; \text{ goto } M \text{ fi}$

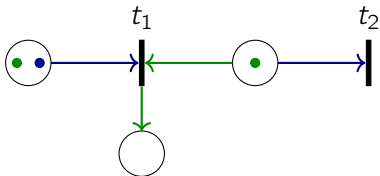


# Раскрашенные сети Петри

Фишки обыкновенной сети Петри неотличимы друг от друга, и поведение сети определяется только их количеством в каждой позиции. Но можно представить себе, что в одной позиции могут находиться фишки разного вида и на переходах явно говорится, фишки каких видов в каких позициях требуются на переходе.

Разные виды фишек можно воспринимать как раскрашенные в разные **цвета** из заранее заданного множества цветов  $C$ , или, более точно, фишки являются цветами.

Переходы для раскрашенных фишек тогда могут быть устроены так:



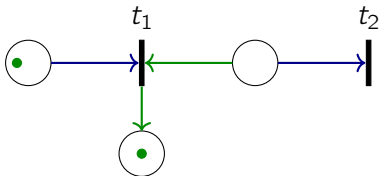
Переход  $t_1$  удаляет одну синюю и фишку из левой позиции и одну зелёную из правой и добавляет одну зелёную фишку в нижнюю позицию. Переход  $t_2$  удаляет одну синюю фишку из правой позиции.

# Раскрашенные сети Петри

Фишки обыкновенной сети Петри неотличимы друг от друга, и поведение сети определяется только их количеством в каждой позиции. Но можно представить себе, что в одной позиции могут находиться фишки разного вида и на переходах явно говорится, фишки каких видов в каких позициях требуются на переходе.

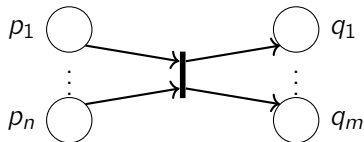
Разные виды фишек можно воспринимать как раскрашенные в разные **цвета** из заранее заданного множества цветов  $C$ , или, более точно, фишки являются цветами.

Переходы для раскрашенных фишек тогда могут быть устроены так:



Переход  $t_1$  удаляет одну синюю и фишку из левой позиции и одну зелёную из правой и добавляет одну зелёную фишку в нижнюю позицию. Переход  $t_2$  удаляет одну синюю фишку из правой позиции.

# Раскрашенные сети Петри



Цветов может быть и бесконечно много, как и способов преобразования их переходами

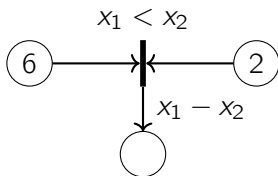
Если позиции  $\bullet t$  и  $t \bullet$  упорядочены  $(p_1, \dots, p_n$  и  $q_1, \dots, q_m)$ , то переход, при срабатывании

- ▶ удаляющий наборы фишек, являющиеся элементами множества  $\Phi \subseteq C^n$ , то есть предиката  $\Phi(x_1, \dots, x_n)$  над  $C$ , и
- ▶ для удалённого набора  $(x_1, \dots, x_n)$  кладущий в  $q_i$  фишку  $f_i(x_1, \dots, x_n)$

можно пометить множеством  $\Phi$  и набором функций  $(f_1, \dots, f_m)$

# Раскрашенные сети Петри

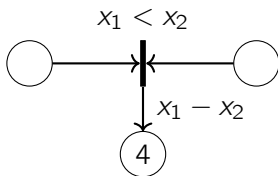
Пример:



**Для размышлений:** можете попробовать показать, что если множество цветов конечно, то раскрашенная сеть Петри моделируется обыкновенной

# Раскрашенные сети Петри

Пример:



**Для размышлений:** можете попробовать показать, что если множество цветов конечно, то раскрашенная сеть Петри моделируется обыкновенной