Лекция 7. Анализ программ на ошибки синхронизации.

Содержание

- Условия возникновения проблем.
- Динамический анализ. Его возможности и проблемы.
- Статический анализ. Его возможности и проблемы.

Условия возникновения тупиков

- Тупик — ситуация, возникающая когда два или более потока пытаются завладеть двумя или более ресурсами, проявляющаяся бесконечным взаимным ожиданием.
- При использовании совместного ресурса возникают фазы:
  - блокировки
  - использования
  - разблокировки

Состояние тупика

- Тупик — состояние взаимного ожидания параллельно исполняющихся вычислительных потоков, ожидающих разблокировку одних и тех же ресурсов.
- Тупики трудно диагностируются и появление состояния тупика в ядре операционной системы обычно означает отказ какого-то её компонента или всей системы.
- Процесс или поток находится в состоянии тупика, если он ожидает события, которое никогда не произойдёт.

Динамический анализ

- Производится во время исполнения алгоритма под управлением супервизора.
- Вероятностный.
- Требуется значительное количество прогонов.
- Ресурсоёмкий.
- Не даёт ложноотрицательных результатов.

Статический анализ

- Производится по тексту алгоритма.
- Требует анализа текста алгоритма
  - Синтаксический анализ
  - Семантический анализ
  - Построение модели
  - Верификация модели
- Не даёт ложноположительных результатов.

Статический анализ на достижимость состояния

- Задача: имеется набор недетерминированных активностей.
- Определить: существует ли такой порядок исполнения активностей, что возникнет состояние взаимного ожидания.
- Тупик — только один из примеров.
- Второй пример — потерянный сигнал.

Статический анализ

- Возможное решение: построить модель исполнения программы с моделированием активностей и определить свойства модели.

Моделирование исполнения. Сети Петри

Обыкновенная сеть Петри

$$C = (P, T, F, \mu_0),$$

где

$$P - \text{множество позиций},$$

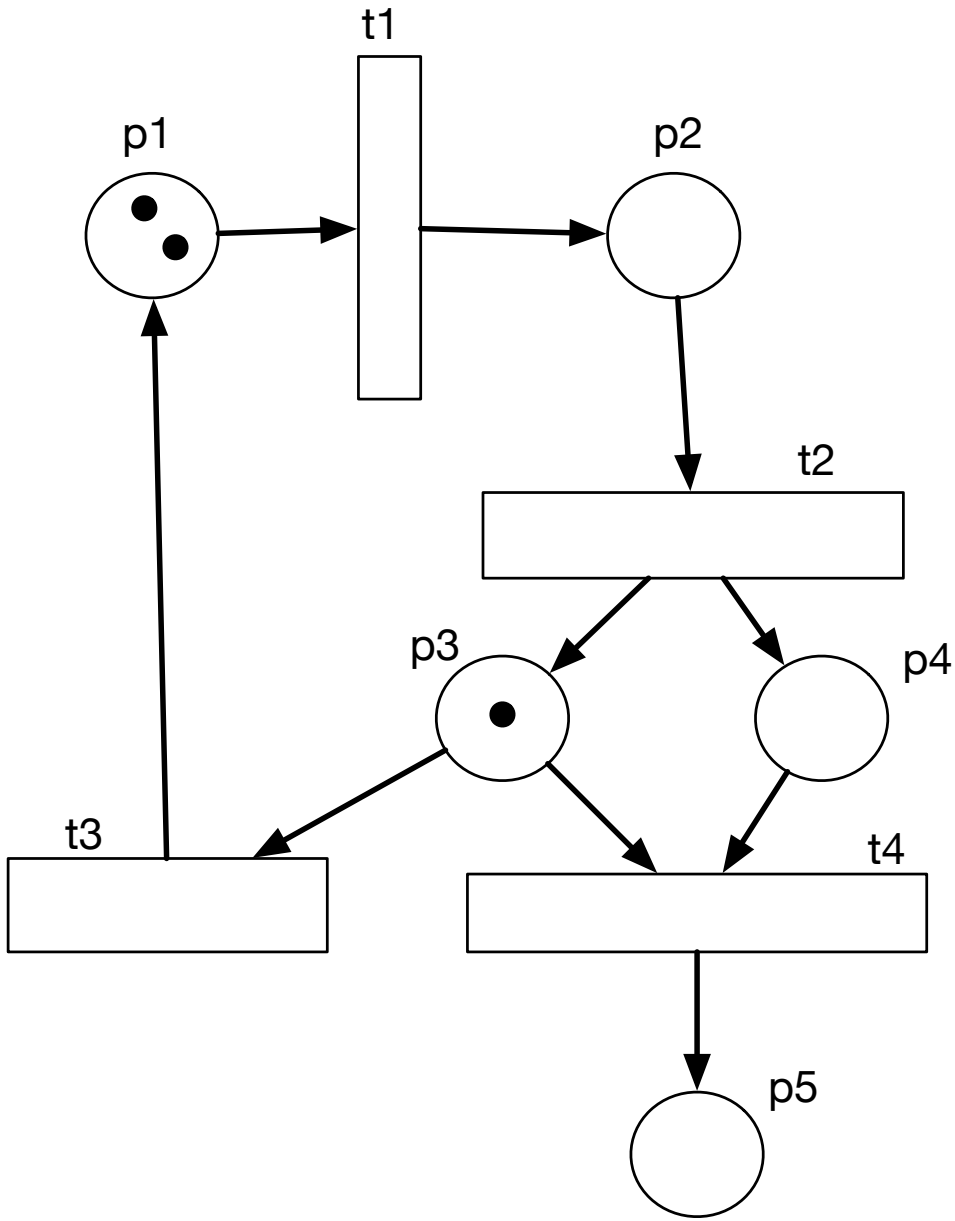
$$T - \text{множество переходов},$$

$$F \subseteq (P \times T) \cup (T \times P) - \text{множество направленных дуг},$$

$$\mu_0 : P \to Z_{\geqslant 0} - \text{начальная маркировка},$$

$$Z_{\geqslant 0} \in \{0..\infty\}$$

Пример сети Петри



Пример сети Петри

$$P = \{p1, p2, p3, p4, p5\}$$

$$T = \{t1, t2, t3, t4\}$$

$$F = \{\{p1, t1\}, \{t1, p2\}, \{p2, t2\}...\}$$

$$\mu_0 = \{2, 0, 1, 0, 0\}$$

Терминология сетей Петри

${}^\bullet p$ — множество входных переходов из позиции p.

$p^\bullet$ — множество выходных переходов из позиции p.

${}^\bullet t$ — множество входных позиций из перехода t.

$t^\bullet$ — множество выходных позиций из перехода t.

Терминология сетей Петри

Переход t разрешён или запускаем в μ_0 если для всех $p \in^\bullet t, \mu_0(p) \geq 1$.

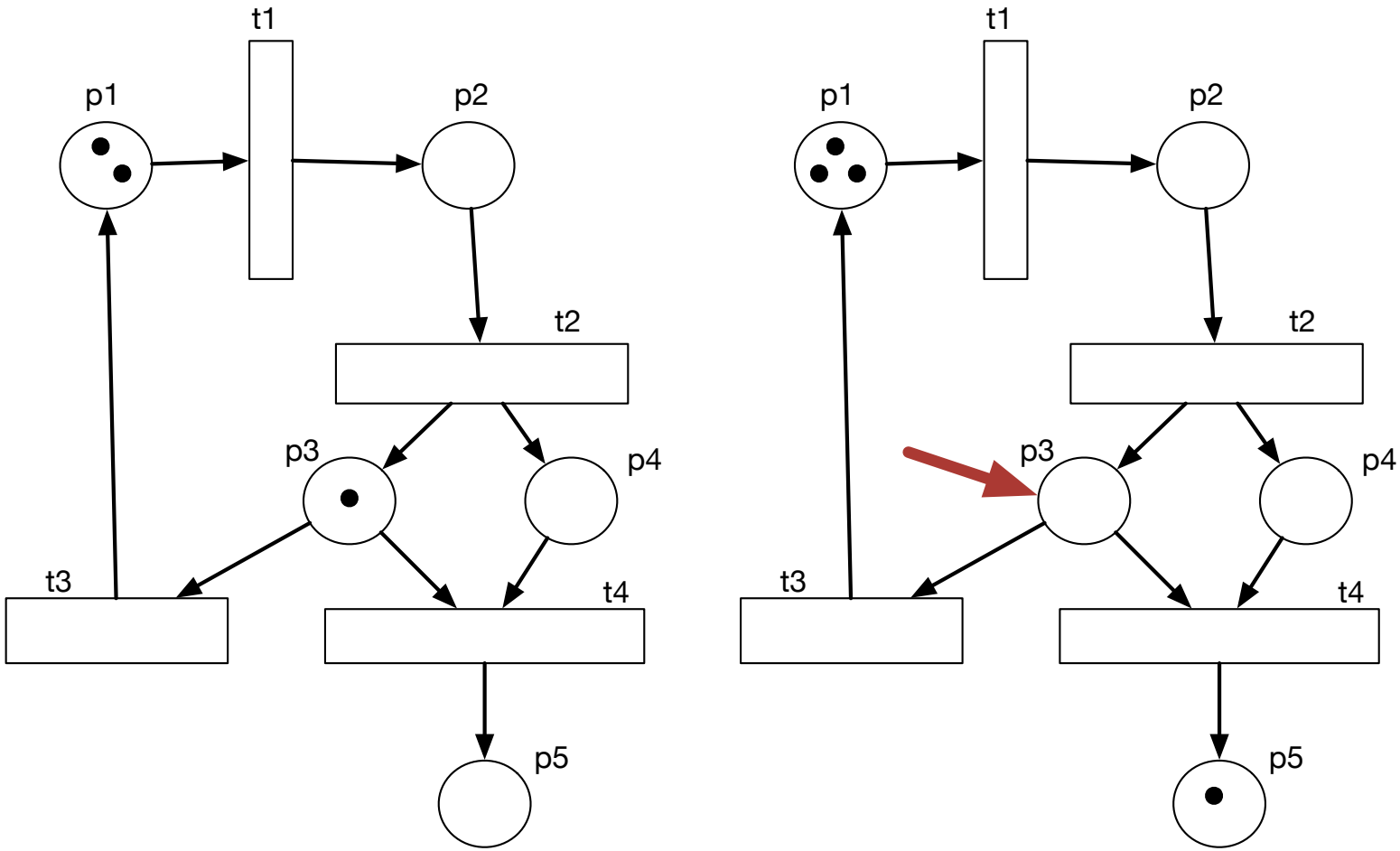
При запуске перехода t новая маркировка μ' производится удалением одного маркера из каждого $p_i, i \in^\bullet t$ и помещением одного маркера в каждый $p_j, j \in t^\bullet$.

Этот процесс обозначается как $\delta(\mu, t) > \mu'$, где μ — маркировка.

Множество допустимых в данной маркировке μ переходов обозначим как $D(\mu)$, количество таких переходов как $|D(\mu)|$.

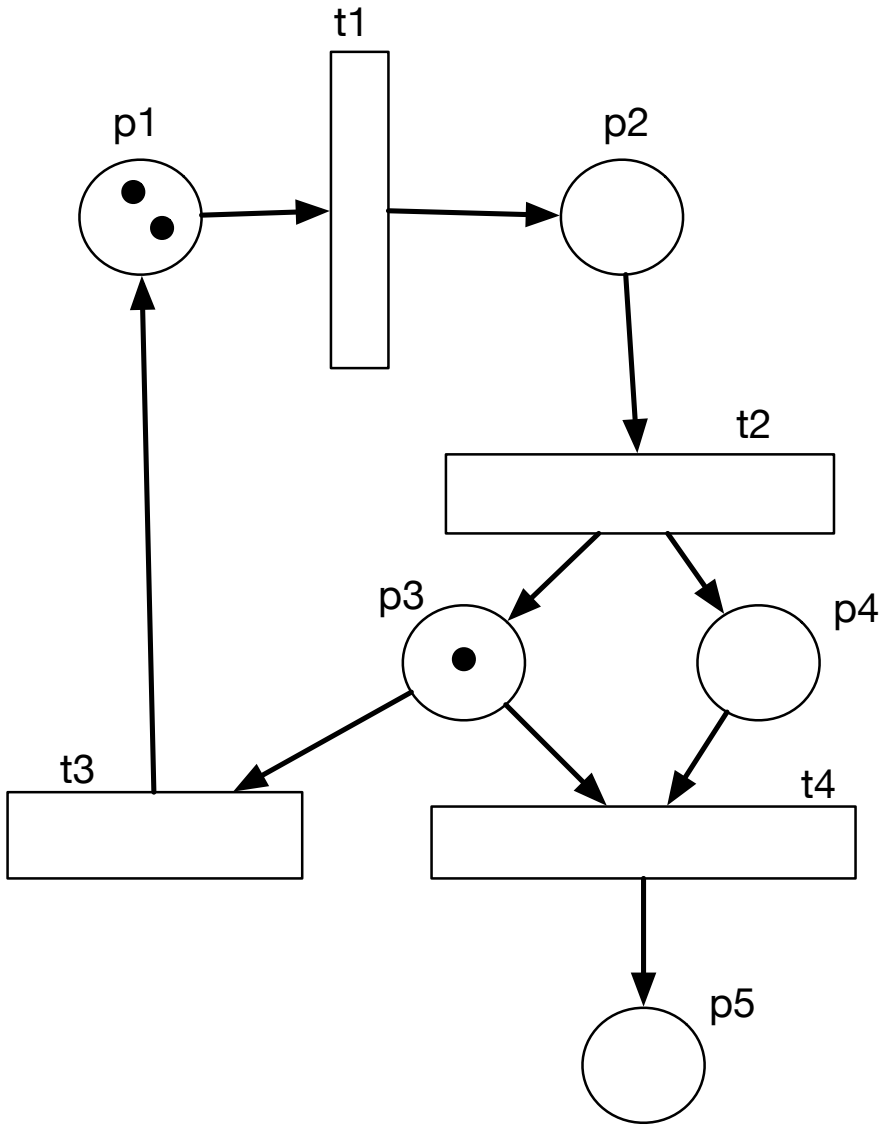
Пример сети Петри

Совершение перехода:



Терминология сетей Петри

Для рассмотренной сети:



$$
\begin{aligned}
{}^\bullet p_1 &= \{t_3\}, p_1^\bullet = \{t_1\} \\
{}^\bullet p_2 &= \{t_1\}, p_2^\bullet = \{t_2\} \\
{}^\bullet p_3 &= \{t_2\}, p_3^\bullet = \{t_3, t_4\} \\
{}^\bullet p_4 &= \{t_2\}, p_4^\bullet = \{t_4\} \\
{}^\bullet p_5 &= \{t_4\}, p_5^\bullet = \{\}
\end{aligned}
$$

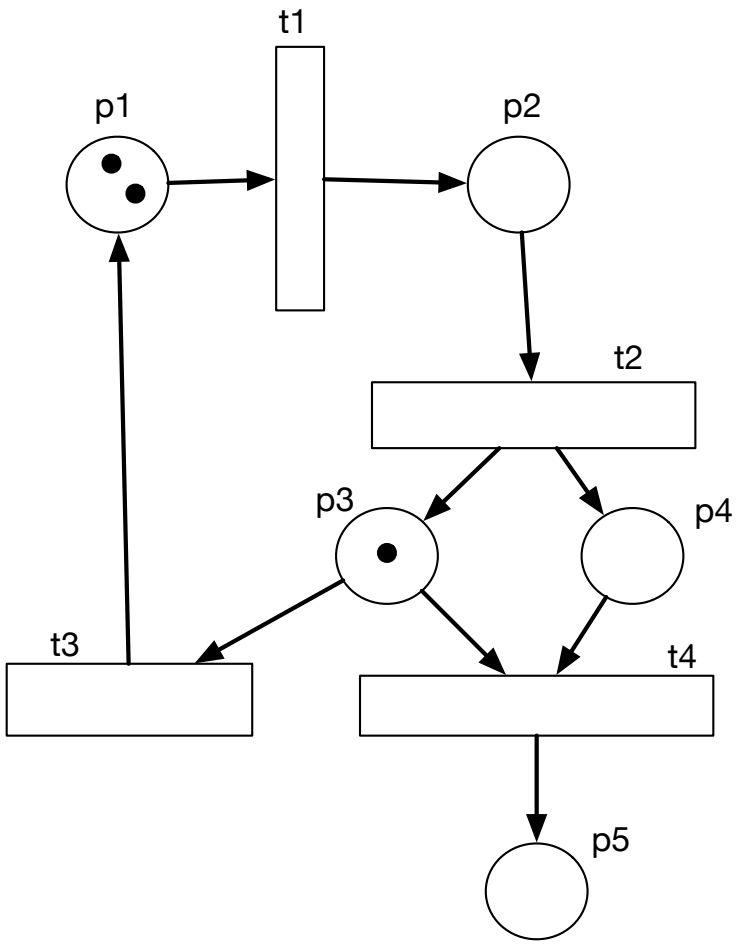
Терминология сетей Петри

Матрица инцидентности $C = [c_{ij}]$ такова, что

$$\begin{cases} c_{ij} = 1, & \text{если } t_j \in {}^\bullet p_i \setminus p_i{}^\bullet; \\ c_{ij} = -1, & \text{если } t_j \in p_j{}^\bullet \setminus {}^\bullet p_i; \\ c_{ij} = 0 & \text{в остальных случаях.} \end{cases}$$

Терминология сетей Петри

Для рассмотренной сети:



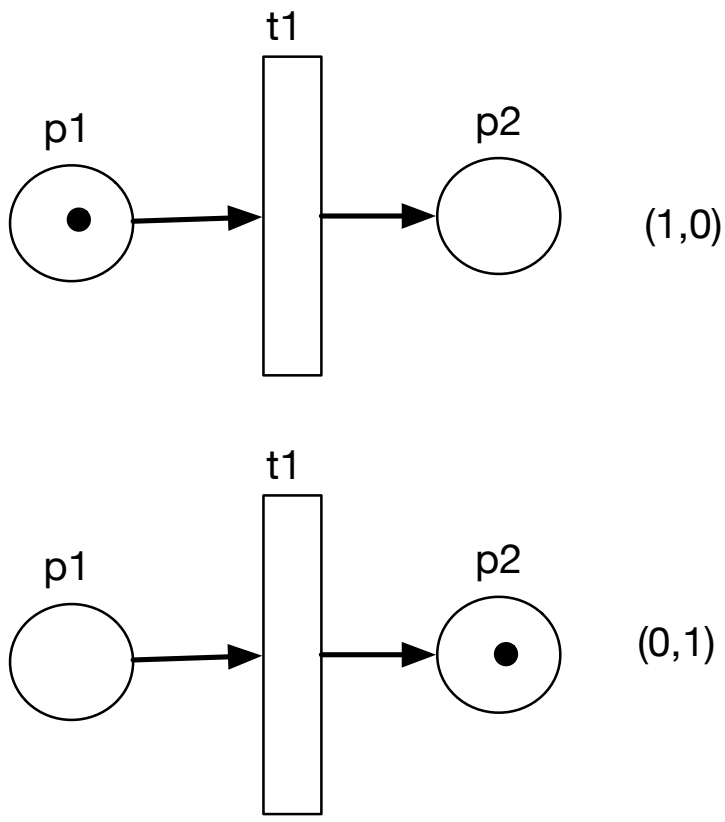
p/t	t_1	t_2	t_3	t_4	t_5	t_6
p_1	-1	0	1	0	0	0
p_2	1	-1	0	0	0	0
p_3	0	1	-1	-1	0	0
p_4	0	1	0	-1	0	0
p_5	0	0	0	1	0	0

Терминология сетей Петри

Для любого μ такого, что $\delta(\mu_0, \sigma) > \mu$, $\mu = \mu_0 + C\overline{\sigma}$, где $\overline{\sigma}$ называется вектором количества запусков — таким вектором, в котором $i-$й элемент обозначает количество вхождений t_i в σ.

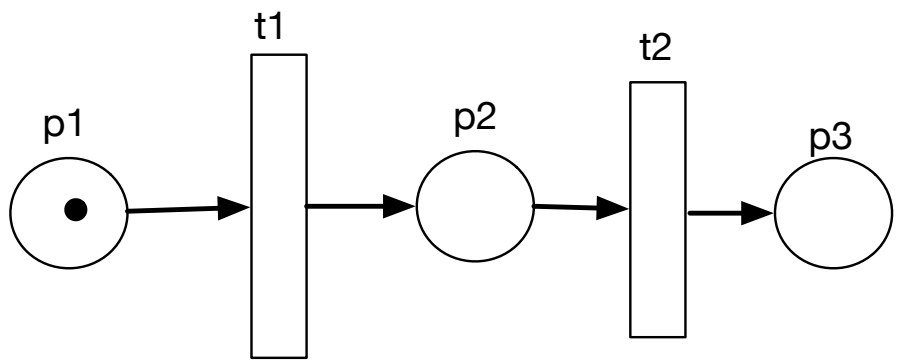
Свойства сетей Петри как модели параллельных процессов

- Любая позиция имеет 0 или более маркеров.
- Переход может совершиться только если имеется хотя бы один маркер в каждой входящей позиции.



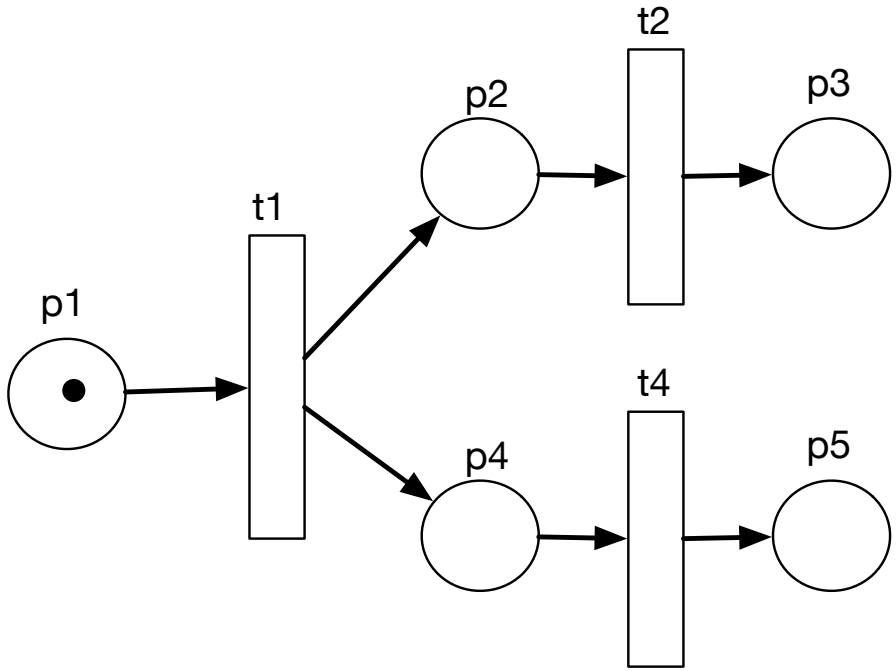
Свойства сетей Петри как моделей параллельных процессов

- Последовательное исполнение.



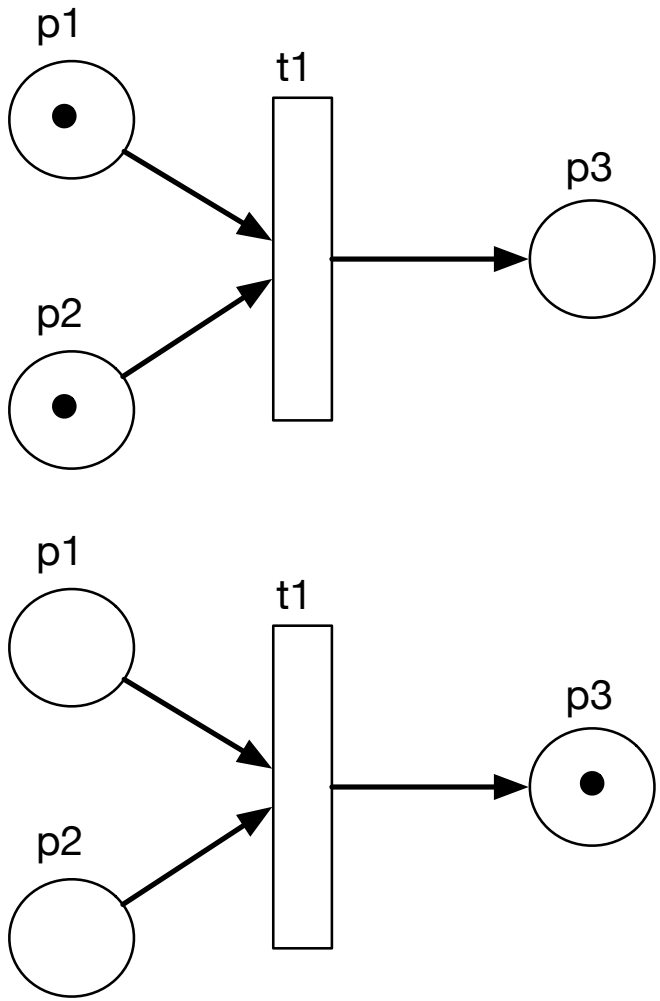
Свойства сетей Петри как моделей параллельных процессов

- Параллельное исполнение.



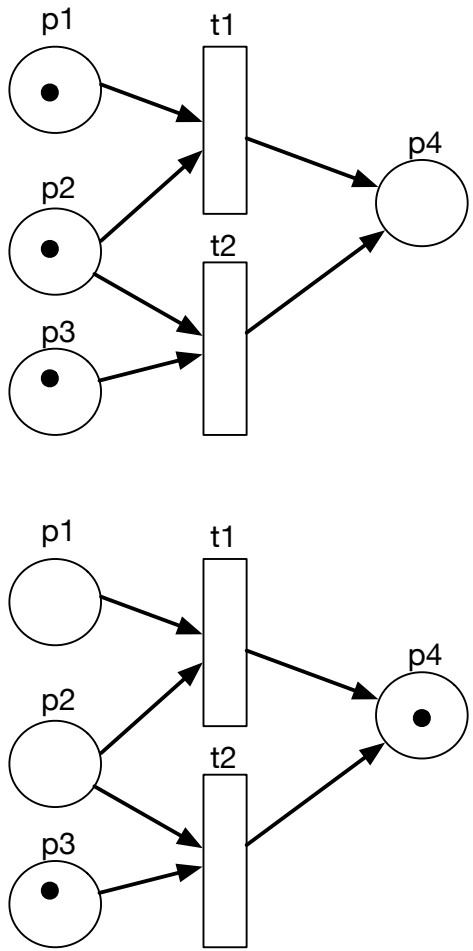
Свойства сетей Петри как моделей параллельных процессов

- Синхронизация и слияние.



Свойства сетей Петри как моделей параллельных процессов

- Моделирование конфликтов. Совершение любого из переходов t_1 или t_2 делает невозможным совершение другого.



Тупики в сетях Петри

- Определение наличия тупиков в сети Петри сводится к задаче определения одной из характеристик сети Петри — активности (`liveness`).
- Если в сети Петри C существует такая маркировка $\mu \in R(\mu_0)$, что $|D(\mu)| = 0$, то маркировка μ — тупиковая и сеть C содержит тупики.
- Обнаружение возможности наступления состояния тупика в сети Петри показывает наличие возможности состояния тупика и в моделируемой вычислительной среде.

Моделирование с помощью сетей Петри

- Создание моделирующей машины сети Петри.
- Описание потоков и примитивов синхронизации в модели сети Петри.
- Синтаксический и семантический анализ языка описания и генерация сети Петри для модели.

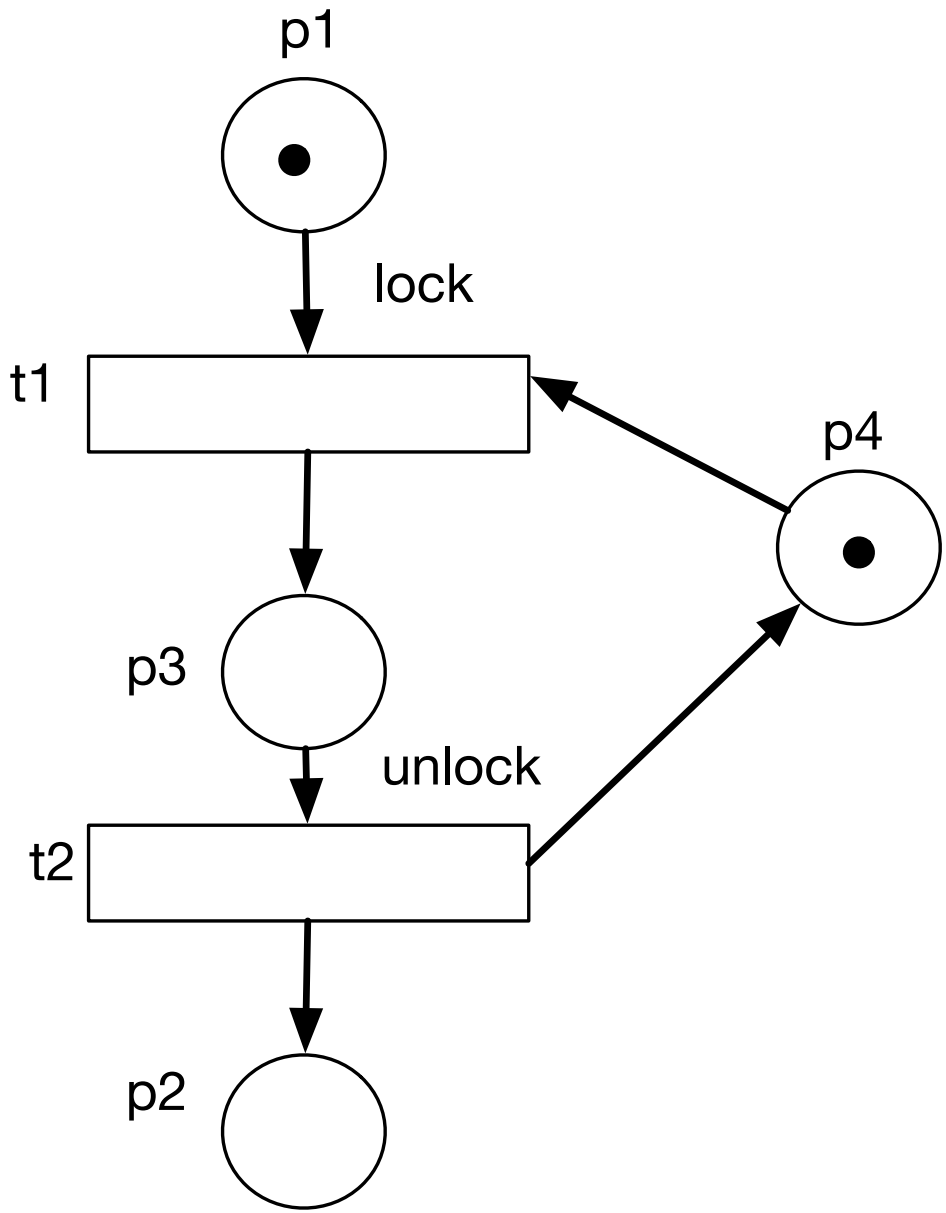
Моделирующая машина сети Петри

- Модели управляющих конструкций и примитивов синхронизации.
  - `Mutex`
  - `Event`
  - `Thread`
  - `Action`

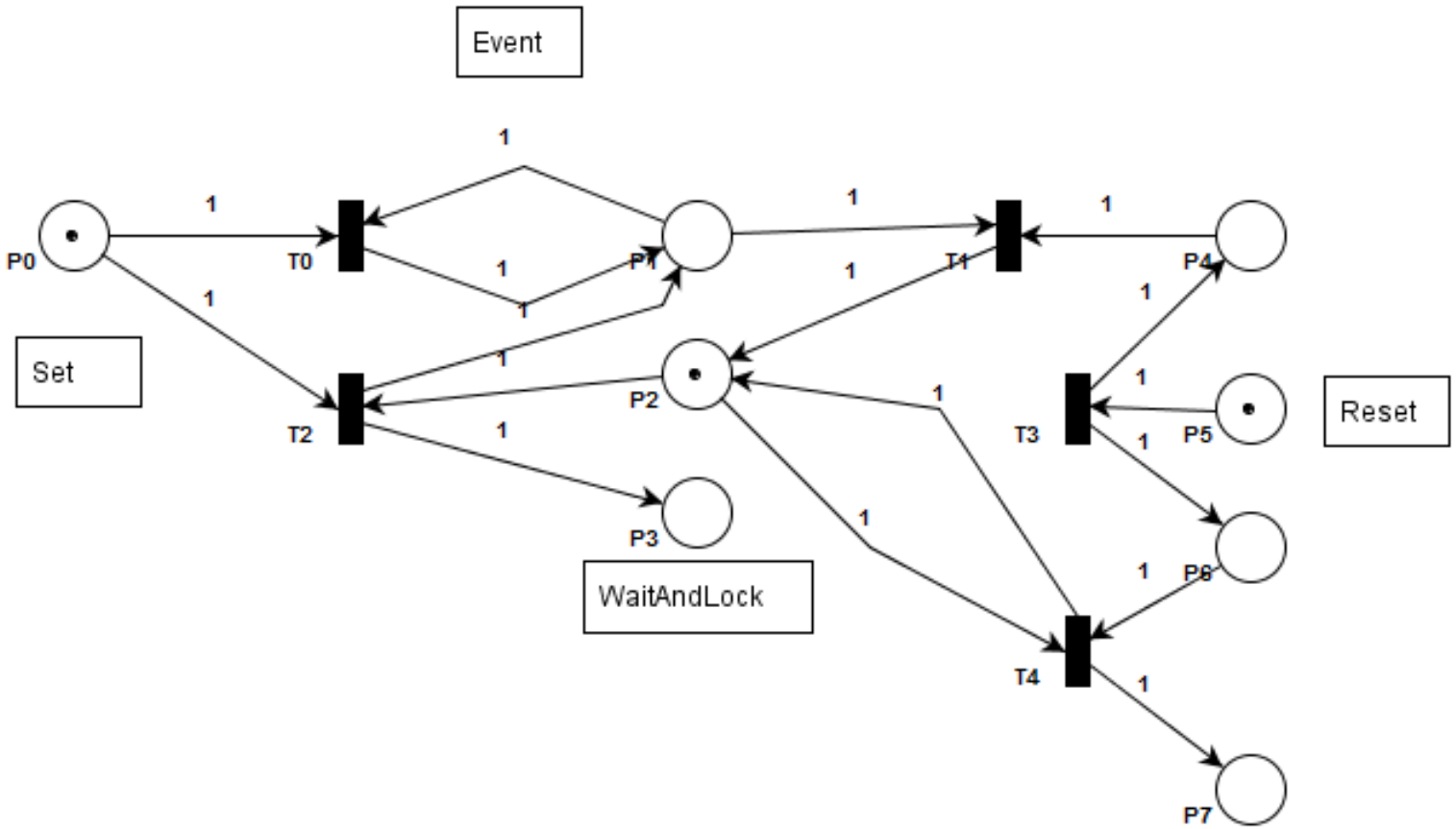
Формализация примитивов синхронизации

- `Mutex`
  - `lock`
  - `unlock`
- `Event`
  - `set`
  - `reset`
  - `wait`

Формализация примитива Mutex



Формализация примитива Event



О формализации примитивов

- Примитив `Mutex` удобно отображается на машину сетей Петри, решаются задачи на тупики.
- Примитив `Event` есть комбинация сигнальных переменных и блокировок.
- Исследование задачи на проблемы, связанные с `Event`, находит решение задач на тупики и потери сигналов.

Отображение исполнителя на язык сетей Петри

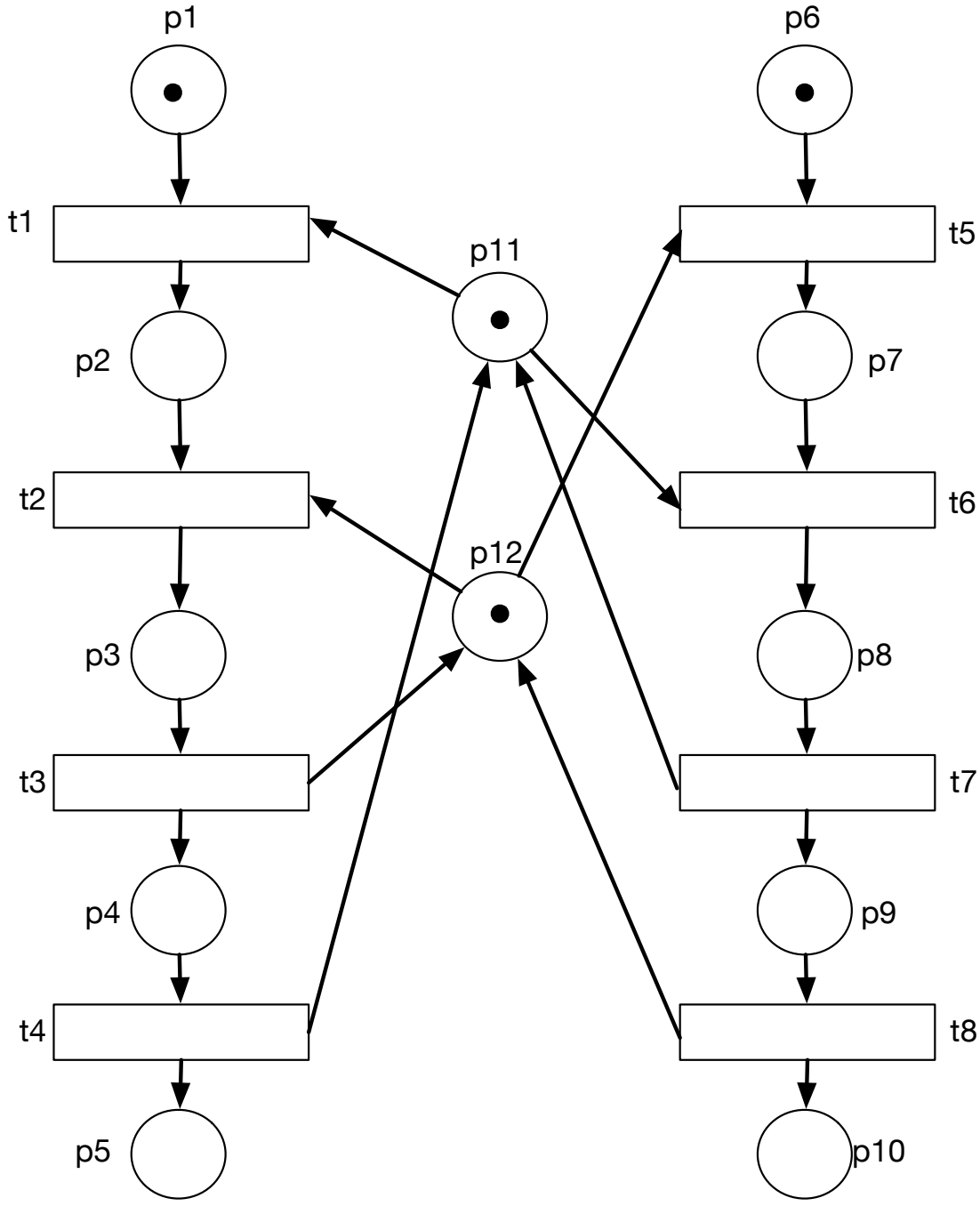
- Если будет иметься отображение языка программирования на модель, то задача решена.
- В HP группа встраивает в язык Си транслятор в модель языка сетей Петри. Этим она занимается 12 лет.
- Если не ставить такую глобальную задачу, то можно создать простой язык описания, транслирующийся в сети Петри.

Модель взаимодействия параллельных потоков

```
shared mutex m1,m2;

shared thread firstThread {
  forever {
    m1.wait; @work1;
    m2.wait; @work2;
    m2.release;
    m1.release;
  }
}

shared thread secondThread {
  forever {
    m2.wait; @work1;
    m1.wait; @work2;
    m1.release;
    m2.release;
  }
}
```


p1
p6
t1
t5
p11
p2
p7
t2
t6
p12
p3
p8
t3
t7
p4
p9
t4
t8
p5
p10

Борьба с тупиками

Если тупики возникают как результат использования только `Mutex`-подобных блокировок, то имеется два основных способа:

- Метод иерархии ресурсов.
- Метод супервизора.

Метод иерархии ресурсов

- Каждый объект синхронизации имеет свой приоритет.
- Если потоку требуется в процессе управления захватить два или более объекта синхронизации, то захвату более приоритетного объекта синхронизации должен предшествовать захват менее приоритетного объекта синхронизации.
- В ядре ОС приоритеты обычно определяются адресами объектов.

```
shared thread secondThread {
  forever {
    m1.wait; @work1;
    m2.wait; @work2;
    m2.release;
    m1.release;
  }
}
```

Метод супервизора

- Оба потока используют одну и ту же пару объектов *m1* и *m2*.
- Для исполнения каждого из потоков требуются оба ресурса одновременно.
- Если бы операция выделения ресурсов была бы атомарной, то проблема решена.
- Либо должны быть захвачены оба ресурса, либо требуется ожидать освобождения обоих ресурсов сразу.

Метод супервизора

- Задача решается созданием объекта *m3*, супервизорного над *m1* и *m2*.

```
shared thread firstThread {
  forever {
    m3.wait;
    m1.wait; @work1;
    m2.wait; @work2;
    m2.release;
    m1.release;
    m3.release;
  }
}
```

Недостатки методов

Иерархии ресурсов.

- Иногда невозможно заранее спланировать условия синхронизации.

Супервизорного.

- Увеличивается гранулярность. Падает масштабируемость.

Тупики при использовании сигналов

- Потеря сигнала плохо диагностируется.
- Можно использовать модель примитива `Event`

Пример модели с Event (Apache bug #42031)

```
listener_thread(...) {
  ...
  apr_thread_mutex_lock(timeout_mutex);
  ...
  rv = apr_thread_mutex_lock(queue_info->idlers_mutex);
  ...
  rv = apr_thread_cond_wait(queue_info->wait_for_idler,
    queue_info->idlers_mutex); /**/
  ...
  rv = apr_thread_mutex_unlock(queue_info->idlers_mutex);
  ...
  apr_thread_mutex_unlock(timeout_mutex);
  ...
}
```

Пример модели с Event (Apache bug #42031)

```
worker_thread(...) {
  ...
  apr_thread_mutex_lock(timeout_mutex); /**/
  ...
  apr_thread_mutex_unlock(timeout_mutex);
  ...
  rv = apr_thread_mutex_lock(queue_info->idlers_mutex);
  ...
  rv = apr_thread_cond_signal(queue_info->wait_for_idler);
  ...
  rv = apr_thread_mutex_unlock(queue_info->idlers_mutex);
  ...
}
```

```
shared mutex timeout_mutex, idlers_mutex;
shared event wait_for_mutex;
thread listener {
  forever {
    timeout_mutex.wait; @work1;
    idlers_mutex.wait;  @work2;
    wait_for_mutex.wait;
    wait_for_mutex.reset;
    idlers_mutex.release; @work5;
    timeout_mutex.release;
  }
}
thread worker {
  forever {
     timeout_mutex.wait; @work1;
     timeout_mutex.release; @work2;
     idlers_mutex.wait; @work3;
     wait_for_mutex.set; @work4;
     idlers_mutex.release;
  }
}
```

Спасибо за внимание.

Следующая тема — формальное описание программ.