Теория вычислительных процессов

1. Стандартные схемы программ. Базис класса стандартных схем программ. Графовая форма стандартной схемы. Линейная форма стандартной схемы. Интерпретация стандартных схем программ.

2. Свойства и виды стандартных схем программ. Эквивалентность, тотальность, пустота, свобода. Свободные интерпретации. Согласованные свободные интерпретации.

3. Трансляция схем программ. Схемы с процедурами.

4. Моделирование стандартных схем программ. Однолентоточные, многоленточные, двухголовочные автоматы. Построение схемы, моделирующей автомат.

5. Обогащенные и структурированные схемы. Классы обогащенных схем. Трансляция обогащенных схем. Структурированные схемы.

6. Рекурсивные схемы. Рекурсивное программирование. Определение рекурсивной схемы.

7. Теоретические модели вычислительных процессов. Взаимодействующие последовательные процессы.

8. Теоретические модели вычислительных процессов. Параллельные процессы.

9. Теоретические модели вычислительных процессов. Разделяемые ресурсы.

10. Теоретические модели вычислительных процессов. Программирование параллельных вычислений.

11. Теоретические модели вычислительных процессов. Модели параллельных вычислений

12. Сети Петри. Основные понятия и определения. Маркировка, правила выполнения.

13. Моделирование систем на основе сетей Петри.

14. Анализ сетей Петри на основе дерева достижимости.

15. Верификация программ. Правила верификации К. Хоара.

16. Верификация программ. Методы доказательства правильности программ. Использование утверждений в программах.

17. Семантическая теория программ. Операционная и декларативные семантики

18. Семантическая теория программ. Денотационная семантика.

1. Стандартные схемы программ. Базис класса стандартных схем программ. Графовая форма стандартной схемы. Линейная форма стандартной схемы. Интерпретация стандартных схем программ.

Стандартные схемы программ (ССП) характеризуются базисом и структурой схемы.

Базис класса фиксирует символы, из которых строятся схемы, указывает их роль (переменные, функциональные символы и др.), задает вид выражений и операторов схем.

Полный *базис B класса стандартных схем* состоит из 4-х непересекающихся, счетных множеств символов и множества операторов - слов, построенных из этих символов.

1. Стандартные схемы программ. Базис класса стандартных схем программ. Графовая форма стандартной схемы. Линейная форма стандартной схемы. Интерпретация стандартных схем программ.

Множества символов полного базиса:

1. $X = \{x, x_1, x_2..., y, y_1\ y_2..., z, z_1, z_2...\}$ - множество символов, называемых *переменными*;
2. $F = \{f^{(0)}, f^{(1)}, f^{(2)}..., g^{(0)}, g^{(1)}, g^{(2)}..., h^{(0)}, h^{(1)}, h^{(2)}...\}$ - множество *функциональных символов*; верхний символ задает *местность символа*; нульместные символы называют константами и обозначают начальными буквами латинского алфавита a, b, c...;
3. $P = \{p^{(0)}, p^{(1)}, p^{(2)}...; q^{(0)}, q^{(1)}, q^{(2)}...; \}$ - множество *предикатных символов*; $p^{(0)}, q^{(0)}$ - ; нульместные символы называют логическими константами;
4. {**start, stop, ...,:=** и т. д.} - множество специальных символов.

1. Стандартные схемы программ. Базис класса стандартных схем программ. Графовая форма стандартной схемы. Линейная форма стандартной схемы. Интерпретация стандартных схем программ.

Графовая форма стандартной схемы

Представим стандартную схему программ как размеченный граф, вершинам которого приписаны операторы из некоторого базиса B.

Стандартной схемой в базисе B называется конечный (размеченный ориентированный) граф без свободных дуг и с вершинами следующих пяти видов:

1. *Начальная вершина* (ровно одна) помечена начальным оператором. Из нее выходит ровно одна дуга. Нет дуг, ведущих к начальной вершине.
2. *Заключительная вершина* (может быть несколько). Помечена заключительным оператором. Из нее не выходит ни одной дуги.
3. *Вершина-преобразователь.* Помечена оператором присваивания. Из нее выходит ровно одна дуга.
4. *Вершина-распознаватель.* Помечена условным оператором (называемым условием данной вершины). Из нее выходит ровно две дуги, помеченные 1 (левая) и 0 (правая).
5. *Вершина-петля.* Помечена оператором петли. Из нее не выходит ни одной дуги.

1. Стандартные схемы программ. Базис класса стандартных схем программ. Графовая форма стандартной схемы. Линейная форма стандартной схемы. Интерпретация стандартных схем программ.

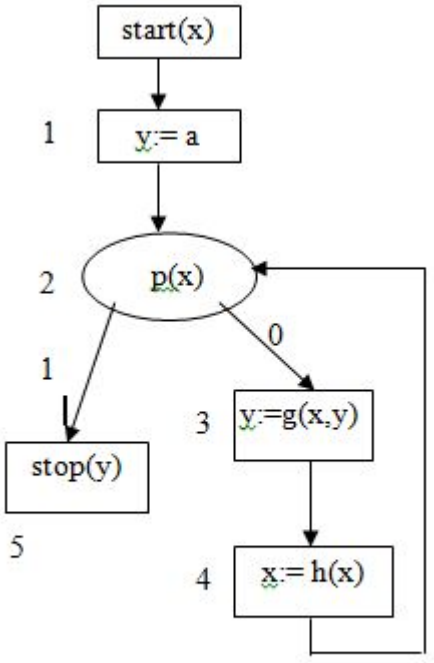
Из определения следует, что один и тот же оператор может помечать несколько вершин схемы.

Вершины именуются (метки вершины) целым неотрицательным числом (0, 1, 2...). Начальная вершина всегда помечается меткой 0.

Схема S называется правильной, если на каждой дуге заданы все переменные.

Пример правильной ССП S_1 в графовой форме приведен на рисунке 1.2, а.

Вершины изображены прямоугольниками, а вершина-распознаватель - овалом. Операторы записаны внутри вершины.



а

Рисунок 1.2

1. Стандартные схемы программ. Базис класса стандартных схем программ. Графовая форма стандартной схемы. Линейная форма стандартной схемы. Интерпретация стандартных схем программ.

Для использования линейной формы СПП множество специальных символов расширим дополнительными символами {**:, goto, if, then, else**}. СПП в линейной форме представляет собой последовательность *инструкций*, которая строится следующим образом:

1. если выходная дуга начальной вершины с оператором **start**(x_1,..., x_n) ведет к вершине с меткой L, то начальной вершине соответствует инструкция:
   0: **start(**x_1,..., x_n**) goto** L;
2. если вершина схемы S с меткой L - преобразователь с оператором присваивания x:=τ, выходная дуга которого ведет к вершине с меткой L1, то этому преобразователю соответствует инструкция:
   L: x: =τ **goto** L1;
3. если вершина с меткой L - заключительная вершина с оператором **stop**(τ_1,...τ_m), то ей соответствует инструкция
   L: **stop**(τ_1,..., τ_m);
4. если вершина с меткой L - распознаватель с условием p(τ_1,...τ_k), причем 1-дуга ведет к вершине с меткой L1, а 0-дуга - к вершине с меткой L0, то этому распознавателю соответствует инструкция
   L: **if** p(τ_1,...,τ_k) **then** L1 **else** L0;
5. если вершина с меткой L - петля, то ей соответствует инструкция
   L: **loop**.

Обычно используется сокращенная запись (опускание меток). Полная и сокращенная линейные формы ССП (рисунок 1.2, а) приведены ниже

```
0:  start(x) goto 1,
1:  y: = a goto 2,
2:  if p(x) then 5 else 3,
3:  y: = g(x,y) goto 4,
4:  x: = h(x) goto 2,
5:  stop(y).
```

```
    start(x),
    y: = a,
2:  if p(x) then 5 else 3,
3:      y: = g(x,y),
        x: = h (x) goto 2,
5:  stop(y).
```

1. Стандартные схемы программ. Базис класса стандартных схем программ. Графовая форма стандартной схемы. Линейная форма стандартной схемы. Интерпретация стандартных схем программ.

Интерпретация стандартных схем программ

ССП не является записью алгоритма, поэтому позволяет исследовать только структурные свойства программ, но не семантику вычислений. При построении «семантической» теории схем программ вводится понятие интерпретация ССП. Определим это понятие.

Пусть в некотором базисе B определен класс ССП. Интерпретацией базиса B в области интерпретации D называется функция I, которая сопоставляет:

1. каждой переменной x из базиса B - некоторый элемент d = I(x) из области интерпретации D;
2. каждой константе a из базиса B - некоторый элемент d = I(a) из области интерпретации D;
3. каждому функциональному символу $f^{(n)}$ - всюду определенную функцию $F^{(n)}=I(f^{(n)})$;
4. каждой логической константе $p^{(0)}$ - один символ множества { 0,1 };
5. каждому предикатному символу $p^{(n)}$ - всюду определенный предикат $P^{(n)} = I(p^{(n)})$.

Пара (S,I) называется *интерпретированной стандартной схемой* (ИСС), или *стандартной программой* (СП).

2. Свойства и виды стандартных схем программ. Эквивалентность, тотальность, пустота, свобода. Свободные интерпретации. Согласованные свободные интерпретации.

ССП S в базисе B тотальна (пуста), если для любой интерпретации I базиса B программа (S, I) останавливается (зацикливается).

Стандартные схемы S1 и S2 в базисе B функционально эквивалентны (S1 ~ S2), если либо обе зацикливаются, либо обе останавливаются с одинаковым результатом.

Примеры тотальных, пустых и эквивалентных схем S2, S3, S4, S5 приведены на рисунке 1.3

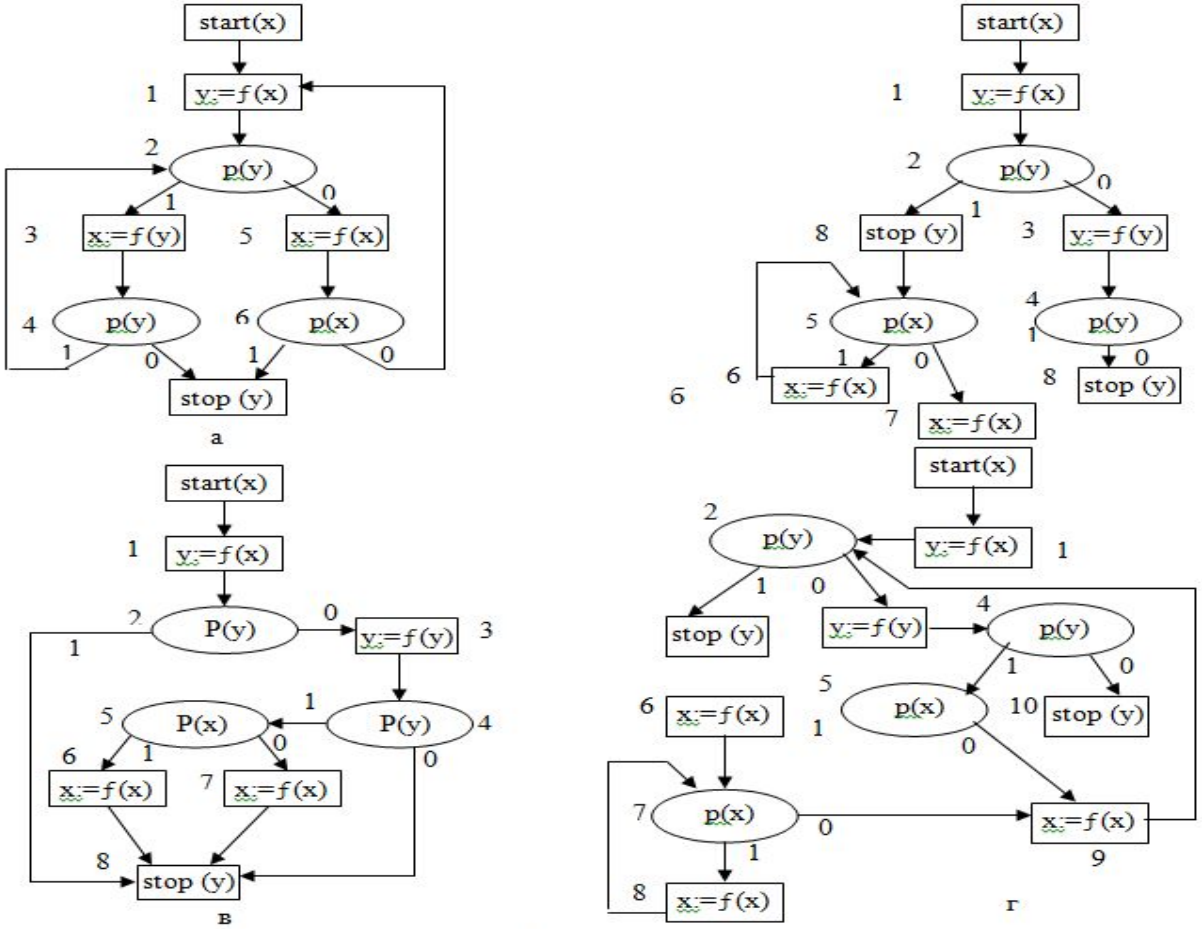


Рисунок 1.3

2. Свойства и виды стандартных схем программ. Эквивалентность, тотальность, пустота, свобода. Свободные интерпретации. Согласованные свободные интерпретации.

Отношение функциональной эквивалентности, а также свойства тотальности и пустоты стандартных схем, определены с использованием понятия множества всех возможных интерпретаций базиса. Таким образом, для установления этих свойств схем необходимо это сделать для все интерпретаций. Такой путь крайне не перспективен, так как класс всех возможных интерпретаций базиса крайне широк.

Однако, существует подкласс интерпретаций, называемых *свободными*, позволяющий установить справедливость свойств стандартны схем.

2. Свойства и виды стандартных схем программ. Эквивалентность, тотальность, пустота, свобода. Свободные интерпретации. Согласованные свободные интерпретации.

Множество всех интерпретаций очень велико и поэтому вводится класс свободных интерпретаций (СИ), который образует ядро класса всех интерпретаций в том смысле, что справедливость высказываний о семантических свойствах ССП достаточно продемонстрировать для программ, получаемых только с помощью СИ.

Будем говорить, что интерпретация I и свободная интерпретация I_h того же базиса $\mathcal{B}$ *согласованы*, если для любого логического выражения π справедливо $I_h(\pi) = I(\pi)$.

Лемма 1

Для каждой интерпретации I базиса $\mathcal{B}$ существует согласованная с ней свободная интерпретация этого базиса.

4. Трансляция схем программ. Схемы с процедурами.

Программы для ЭВМ, будь то программы, записанные на операторном языке, или программы на рекурсивном языке, универсальны в том смысле, что любую вычислимую функцию можно запрограммировать и найти ее значения для заданных значений аргументов. При этом не требуется богатого набора программных примитивов и базовых операций: достаточно тех средств, которые моделируются стандартными схемами. Это значит, что различные классы программ не имеет смысла сравнивать способности реализовать различные алгоритмы,— все они оказываются универсальными. В то же время программисты знают, что одни программные примитивы являются «более выразительными», чем другие, что запись алгоритмов с привлечением рекурсии короче, чем итерационное представление, но вычисления по такой программе могут потребовать больше времени, и т. д.

При переходе к схемам программ возникает возможность поставить и исследовать проблему выражения одних наборов примитивов через другие в более чистом виде. Задачи такого типа образуют сравнительную схематологию, основу которой составляют теоремы о возможности или невозможности преобразования схем из одного класса в схемы другого. При этом наряду с основной задачей — выяснением соотношений между различными средствами программирования — решается и другая, внутренняя задача схематологии. Действительно, если мы умеем трансформировать один класс схем в другой, то сможем переносить результаты, полученные для некоторого класса схем, на другие классы.

4. Трансляция схем программ. Схемы с процедурами.

Схемы с процедурами строятся в объединенном базисе классов стандартных и рекурсивных схем. Она состоит из двух частей - главной схемы и множества схем процедур. Главная схема - это стандартная схема, в которой имеются операторы присваивания специального вида x:= F(n)(y1,y2,…yn), называемые операторами вызова процедур.

Схема процедуры состоит из заголовка и тела процедуры, разделенных символом равенства. Заголовок имеет тот же вид, что и левая часть рекурсивных уравнений. Тело процедуры - это стандартная схема того же вида, что и главная схема. Заключительный оператор тела процедуры всегда одноместен (stop(x)). Ниже приведен пример схемы с процедурами.

Главная схема	Множество схем процедур
(**start**(x), 1: z:=x, 2: u:=a, 3: x:=F(x, z, u), 4: u:=b, 5: z:=F(z, x, u) 6: **stop**(z))	F(y, v, w) = **start**, 1: **if** p(y) **then** 2 **else** 4, 2: y:=h(y), 3: v:=G(v, w) **goto** 1, 4: **if** q(w) **then** 5 **else** 6, 5: y:= v, 6: **stop**(y)) G(t, r) = (**start**, 1: **if** q(r) **then** 2 **else** 3, 2: t := f(t), 3: **stop**(t);

5. Обогащенные и структурированные схемы. Классы обогащенных схем. Трансляция обогащенных схем. Структурированные схемы.

Выделяют следующие классы обогащенных схем: класс счетчиковых схем, класс магазинных схем, класс схем с массивами.

Классы счетчиковых и магазинных схем образован добавлением в базис ССП счетного множества счетчиков и магазинов с их интерпретированными операторами.

Счетчик — интерпретированная переменная, у которой областью значений является множество Nat; начальное значение счетчика равно 0.

5. Обогащенные и структурированные схемы. Классы обогащенных схем. Трансляция обогащенных схем. Структурированные схемы.

Диаграмма на рис. 1.12. дает полную информацию о возможности трансляции одного класса схем в другой, классы имеют следующие обозначения:

Y — стандартные схемы; Y(M) — магазинные схемы;

Y(R) — рекурсивные схемы; Y(A) — схемы с массивами;

Y(c) — счетчиковые схемы; Y(P) — схемы с процедурами.

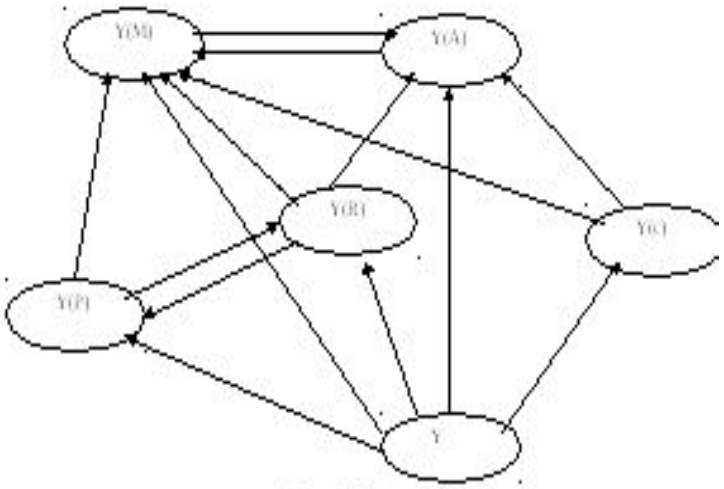


Рис. 1.12

Диаграмма показывает, что классы Y(M) и Y(A) являются универсальными в том смысле, что схемы всех других классов транслируемы в них. В то же время, в класс Y не транслируются схемы ни одного другого класса. Следует отметить, что класс Y(c) достигает полной мощности при количестве счетчиков не менее 2, т.е. класс Y(c) с одним счетчиком равномощен классу Y.

5. Обогащенные и структурированные схемы. Классы обогащенных схем. Трансляция обогащенных схем. Структурированные схемы.

Возрастающая сложность программ привлекает все большее внимание к проблемам технологии программирования. Технологические соображения заставили, в первую очередь, пересмотреть принципы организации программ, их структуру.

Дейкстра первым высказался против неупорядоченного использования переходов на метки, которое может привести и фактически приводит к переусложнению структуры программы, затрудняющему ее понимание и декомпозицию на более простые фрагменты. Реализуя концепцию так называемого структурированного программирования, он предложил, отказаться от использования переходов и ограничиться более дисциплинирующими средствами управления вычислениями, такими, как условные операторы и операторы цикла.

6. Рекурсивные схемы. Рекурсивное программирование. Определение рекурсивной схемы.

Рекурсивное определение позволяет связать искомое значение функции для заданных аргументов с известными значениями той же функции при некоторых других аргументах.

Примером рекурсивно определяемой функции является факториальная функция FACT: Nat → Nat:

FACT(x) = 1, если x = 0, FACT(x) = x ´ FACT(x — 1), если x > 0.

Рекурсивная схема (РС) определяется в некотором базисе. Полный базис РС, как и базис ССП, включает четыре счетных множества символов: переменные, функциональные символы, предикатные символы, специальные символы.

6. Рекурсивные схемы. Рекурсивное программирование. Определение рекурсивной схемы.

В базисе РС нет множества операторов, вместо него – множество логических выражений и множество термов.

Терм — выражение формального языка (системы) специального вида.

Простые термы определяются так же, как термы–выражения в СПП. Среди простых термов выделим *базовые термы*, которые не содержат определяемых функциональных символов, а также *вызовы-термы* вида $F^{(n)}(t_1,t_2,\ldots t_n)$, где $t_1,t_2,\ldots t_n$ - простые термы, $F^{(n)}$ - определяемый функциональный символ.

6. Рекурсивные схемы. Рекурсивное программирование. Определение рекурсивной схемы.

Терм - это *простой терм*, или *условный терм*, т.е. слово Σ **if** π **then** τ_1 **else** τ_2,

где π - логическое выражениτ_1, τ_2 - простые термы, называемые *левой* и соответственно *правой альтернативой*.

Рекурсивной схемой называется гτра (, M τ где - терм, называемый главным термом схемы (или ее входом). M - такое множество рекурсивных уравнений, что все определяемые функциональные символы в левых частях уравнений различны и всякий определяемый символ, встречающийся в правой части некоторого уравнения или в главном терме схемы, входит в левую часть некоторого уравнения. Другими словами, в РС имеется определение всякой вызываемой в ней функции, причем ровно одно.

7. Теоретические модели вычислительных процессов. Взаимодействующие последовательные процессы.

Вычислительный процесс ВП - последовательность сменяющих друга состояний некоторой информационной среды.

Модель ВП – описание последовательной смены состояний определенной среды. Описывается в виде теоретической математической схемы (алгоритм) или компьютерной программы.

Взаимодействующие последовательные процессы (англ. *communicating sequential processes*, **CSP**) — формальный язык для описания моделей взаимодействия в параллельных системах. Относится к математическим теориям параллелизма, известных как исчисление процессов (или алгебра процессов), основанных на передаче сообщений по каналам. Оказал влияние на разработку языка Go.

7. Теоретические модели вычислительных процессов. Взаимодействующие последовательные процессы.

Подходящей сферой применения теоретического программирования служит **спецификация, разработка и реализация вычислительных систем**, которые работают непрерывно и взаимодействуют со своим окружением. На основе модели взаимодействующих последовательных процессов (ВзПП) эти системы можно разбить на параллельно работающие подсистемы, взаимодействующие друг с другом, а также со своим общим окружением.

Такой подход обладает целым рядом преимуществ. Он позволяет избежать многих традиционных для параллельного программирования проблем, таких, например, как **взаимное исключение, прерывания** и т. д.

Кроме того, он включает в себя модели структурного программирования, такие, например, как мониторы, классы, модули, пакеты, критические секции и подпрограммы.

Неформально процесс можно представить себе как группу ячеек памяти, содержимое которых меняется по определенным правилам. В вычислительной машине эти правила описываются программой, которую выполняет центральный процессор.

8. Теоретические модели вычислительных процессов. Параллельные процессы.

Практически любая более или менее сложная система имеет в своем составе компоненты, работающие одновременно, или, как принято говорить на языке техники, параллельно.

Асинхронный параллельный процесс - такой процесс, состояние которого не зависит от состояния другого параллельного процесса (ПП).

Синхронный ПП - такой процесс, состояние которого зависит от состояния взаимодействующих с ним ПП.

Подчиненный ПП - создается и управляется другим процессом (более высокого уровня).

Независимый ПП - не является подчиненным ни для одного из процессов.

Способ организации параллельных процессов в системе зависит от физической сущности этой системы.

8. Теоретические модели вычислительных процессов. Параллельные процессы.

Остановимся несколько подробнее на особенностях реализации параллельных процессов в вычислительных системах (ВС). Это обусловлено следующей причиной.

Разработка и использование любой имитационной модели предполагает ее программную реализацию и исследование с применением ВС. Поэтому для реализации моделей, имитирующих параллельные процессы, в некоторых случаях применимы механизмы, используемые при выполнении параллельных вычислений. Вместе с тем реализация параллельных процессов в ВС имеет свои особенности:

- на уровне задач вычислительные процессы могут быть истинно параллельными только в многопроцессорных ВС или вычислительных сетях;
- многие ПП используют одни и те же ресурсы, поэтому даже асинхронные ПП в пределах одной ВС вынуждены согласовывать свои действия при обращении к общим ресурсам;
- в ВС дополнительно используется еще два вида ПП: родительский и дочерний ПП; особенность их состоит в том, что процесс-родитель не может быть завершен, пока не завершатся все его дочерние процессы.

8. Теоретические модели вычислительных процессов. Параллельные процессы.

В силу перечисленных особенностей для организации взаимодействия параллельных процессов в ВС используются три основных подхода:

- на основе «взаимного исключения»;
- на основе синхронизации посредством сигналов;
- на основе обмена информацией (сообщениями).

«Взаимное исключение» предполагает запрет доступа к общим ресурсам (общим данным) для всех ПП, кроме одного, на время его работы с этими ресурсами (данными).

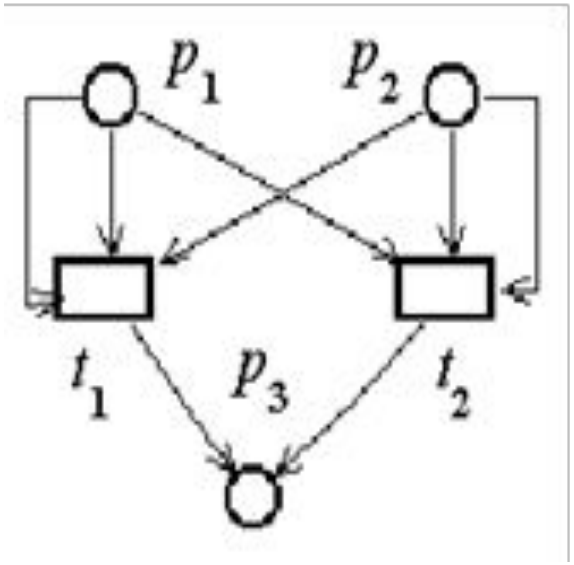
Синхронизация подразумевает обмен сигналами между двумя или более процессами по установленному протоколу. Такой «сигнал» рассматривается как некоторое событие, вызывающее у получившего его процесса соответствующие действия.

Часто возникает необходимость передавать от одного ПП другому более под робную информацию, чем просто «сигнал-событие». В этом случае процессы согласуют свою работу на основе обмена сообщениями.

Перечисленные механизмы реализуются в ВС на двух уровнях - системном и прикладном.

12. Сети Петри. Основные понятия и определения. Маркировка, правила выполнения.

- **Сети Петри** — математический аппарат для моделирования динамических дискретных систем. Впервые описаны Карлом Петри в 1962 году.



- Сеть Петри представляет собой **двудольный ориентированный мультиграф**, состоящий из вершин двух типов — позиций и переходов, соединённых между собой дугами. Вершины одного типа не могут быть соединены непосредственно. В позициях могут размещаться метки (маркеры), способные перемещаться по сети.

- Событием называют срабатывание перехода, при котором метки из входных позиций этого перехода перемещаются в выходные позиции. События происходят мгновенно либо разновременно, при выполнении некоторых условий.

12. Сети Петри. Основные понятия и определения. Маркировка, правила выполнения.

Сеть Петри есть **мультиграф**, так как он допускает существование кратных дуг от одной вершины графа к другой. Так как дуги являются направленными, то это **ориентированный мультиграф**. Вершины графа можно разделить на два множества (позиции и переходы) таким образом, что каждая дуга будет направлена от элемента одного множества (позиций или переходов) к элементу другого множества (переходов или позиций); следовательно, такой граф является **двудольным ориентированным мультиграфом.**

Сети Петри разрабатывались для моделирования систем с параллельными взаимодействующими компонентами.

12. Сети Петри. Основные понятия и определения. Маркировка, правила выполнения.

- Процесс функционирования сети Петри может быть наглядно представлен графом достижимых маркировок. Состояние сети однозначно определяется её маркировкой — распределением фишек по позициям. Вершинами графа являются допустимые маркировки сети Петри, дуги помечены символом срабатывающего перехода. Дуга строится для каждого возбуждённого перехода. Построение прекращается, когда мы получаем маркировки, в которых не возбуждён ни один переход, либо маркировки, содержащиеся в графе. Отметим, что граф достижимых маркировок представляет собой автомат.

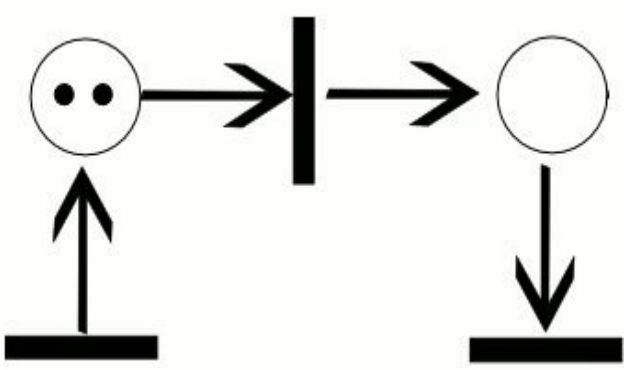
Некоторые виды сетей Петри:

- ***Временна́я сеть Петри*** — переходы обладают весом, определяющим продолжительность срабатывания (задержку).
- ***Стохастическая сеть Петри*** — задержки являются случайными величинами.
- ***Функциональная сеть Петри*** — задержки определяются как функции некоторых аргументов, например, количества меток в каких-либо позициях, состояния некоторых переходов.
- ***Цветная сеть Петри*** — метки могут быть различных типов, обозначаемых цветами, тип метки может быть использован как аргумент в функциональных сетях.
- ***Ингибиторная сеть Петри*** — возможны ингибиторные дуги, запрещающие срабатывания перехода, если во входной позиции, связанной с переходом ингибиторной дугой, находится метка.
- ***Иерархическая сеть*** — содержит не мгновенные переходы, в которые вложены другие, возможно, также иерархические, сети. Срабатывание такого перехода характеризует выполнение полного жизненного цикла вложенной сети.
- ***WF-сети***

12. Сети Петри. Основные понятия и определения. Маркировка, правила выполнения.

Основными свойствами сети Петри являются:

- ✔ ограниченность — число меток в любой позиции сети не может превысить некоторого значения K;
- ✔ безопасность — частный случай ограниченности, K=1;
- ✔ сохраняемость — постоянство загрузки ресурсов,
- ✔ достижимость — возможность перехода сети из одного заданного состояния (характеризуемого распределением меток) в другое; живость — возможность срабатывания любого перехода при функционировании моделируемого объекта.



В основе исследования перечисленных свойств лежит анализ достижимости. Методы анализа свойств сетей Петри основаны на использовании графов достижимых (покрывающих) маркировок, решении уравнения состояний сети и вычислении линейных инвариантов позиций и переходов. Применяются также вспомогательные методы редукции, позволяющие уменьшить размер сети Петри с сохранением её свойств, и декомпозиции[2], разделяющие исходную сеть на подсети.

17. Семантическая теория программ. Операционная и декларативные семантики

Семантика сопоставляет значение с программой.

Операционная семантика позволяет процедурно описывать значение программы.

Операционное значение логической программы *P* –- это множество основных целей, являющихся примерами вопросов, которые программа *P* решает с помощью абстрактного интерпретатора.

Декларативная семантика логических программ основана на стандартной теоретико-модельной семантике логики первого порядка. Для ее описания потребуется некоторая новая терминология.

18. Семантическая теория программ. Денотационная семантика.

Денотационная семантика устанавливает значения программам, основываясь на объединении программы с функцией, определенной в области вычисления программы. Значение программы определяется как наименьшая неподвижная точка функции, если такая точка существует. Областью вычислений логических программ являются интерпретации.

Все различные определения семантики в действительности описывают один и тот же объект. Показано, что операционная, денотационная и декларативная семантики совпадают. Это позволяет нам определить значение логической программы как минимальную модель программы.

Спасибо за внимание!