Моделирование автоматов на основе сетей Петри

Задача 1. Моделирования системы автомат-продавец.

Автомат-продавец находится в состоянии ожидания до тех пор, пока не появится заказ, который он выполняет и посылает на доставку. Условиями для такой системы являются:

а) автомат-продавец ждет;

б) заказ прибыл и ждет;

в) автомат-продавец выполняет заказ;

г) заказ выполнен.

Событиями для этой системы являются:

1.Заказ поступил.

2. Автомат-продавец начинает выполнение заказа.

3. Автомат-продавец заканчивает выполнение заказа.

4. Заказ посылается на доставку.

Для перечисленных событий можно составить следующую таблицу их пред- и постусловий

Событие	**Предусловия**	**Постусловия**
1	нет	б
2	а, б	в
3	в	г, а
4	г	нет

Такое представление системы легко моделировать сетью Петри. В сети Петри условия моделируются позициями, события – переходами. При этом входы перехода являются предусловиями соответствующего события; выходы – постусловиями. Возникновение события моделируется запуском соответствующего перехода. Выполнение условия представляется фишкой в позиции, соответствующей этому условию. Запуск перехода удаляет фишки, представляющие выполнение предусловий и образует новые фишки, которые. представляют выполнение постусловий.

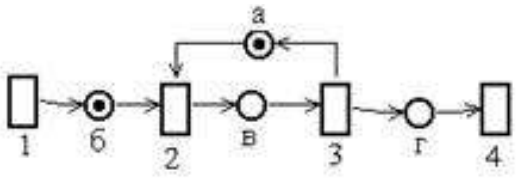

Рисунок 1. Сеть Петри, моделирующая систему автомат-продавец

На рисунке 1 предусловие выполняется для события 2.

Задача 2. Моделирование последовательной обработки запросов сервером базы данных. Сервер находится в состоянии ожидания до тех пор, пока от пользователя не поступит запрос, который он обрабатывает и отправляет результат такой обработки пользователю.

Условиями для такой системы являются:

а) сервер ждет;

б) запрос поступил и ждет;

в) сервер обрабатывает запрос;

г) запрос обработан.

Событиями для этой системы являются:

1.Запрос поступил.

2. Сервер начинает обработку запроса.

3. Сервер заканчивает обработку запроса.

4. Результат обработки отправляется.

Таблица пред- и постусловий для перечисленных событий совпадает с аналогичной таблицей для системы автомат-продавец из задачи 1. Сеть Петри, моделирующая последовательную обработку запросов сервером базы данных, совпадает с аналогичной сетью Петри для системы автомат продавец, показанной на рисунке 1.

Задача 3. Моделирование параллельной обработки запросов сервером базы данных. Сервер, как и в задаче 2, находится в состоянии ожидания до тех пор, пока от пользователя не поступит запрос, который он обрабатывает и отправляет результат пользователю. Отличие состоит в том, что он может обрабатывать одновременно два запроса с помощью двух своих процессорных элементов ПЭ1 и ПЭ2.

Условиями для такой системы являются:

а1) ПЭ1 ждет;

а2) ПЭ2 ждет;

б) запрос поступил и ждет;

в1) ПЭ1 обрабатывает запрос;

в2) ПЭ2 обрабатывает запрос;

г) запрос обработан.

Событиями для этой системы являются:

1.Запрос поступил.

2. ПЭ1 начинает обработку запроса.

3. ПЭ1 заканчивает обработку запроса.

4. ПЭ2 начинает обработку запроса.

5. ПЭ2 заканчивает обработку запроса.

6. Результат обработки отправляется.

Для перечисленных событий можно составить следующую таблицу их пред- и постусловий:

Событие	**Предусловия**	**Постусловия**
1	нет	б
2	а1, б	в1
3	в1	г, а1
4	а2, б	в2
5	в2	г, а2
6	г	нет

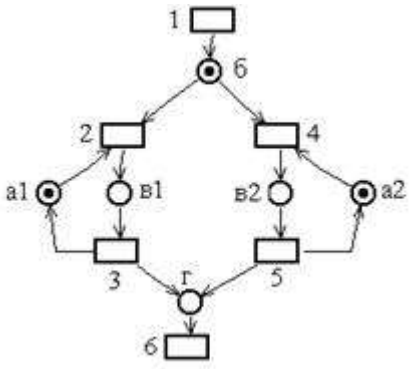

Рисунок 2. Сеть Петри, моделирующая систему задачи 3.

Задача 4. Автомат, вычисляющий чётность количества единиц во входном двоичном числе. Пусть Q={q_1,q_2}, S=D={0,1,R}, где R – признак конца числа (символ сброса). Автомат начинает работу в состоянии q_1. Выход копирует вход до тех пор, пока входным символом не окажется символ сброса R. Выходом для символа сброса будет 0 в случае нечетности и 1 – в случае чётности.

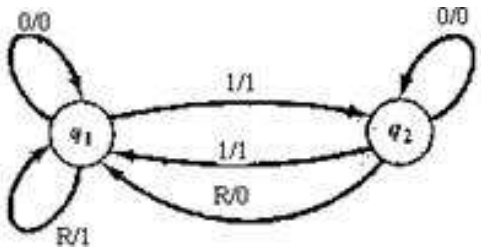

Рисунок 3. Конечный автомат из задачи 4.

Метод моделирования конечных автоматов.

Представим каждый входной и выходной символ, а также каждое состояние автомата позицией в сети Петри. Текущее состояние отмечается фишкой; все остальные позиции пусты. Теперь для определения переходов из состояния в состояние можно ввести переходы сети следующим образом. Для каждой пары (состояние, входной символ) мы определяем переход, входными позициями которого являются позиции, соответствующие состоянию и входному символу, а выходными позициями – позиции, соответствующие следующему состоянию и выходу.

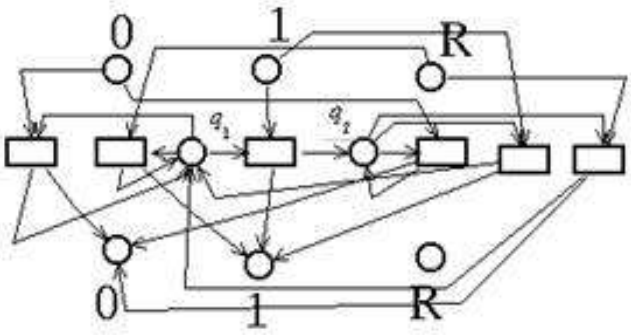

Рисунок 4. Сеть Петри, моделирующая автомат из задачи 4.

Замечание 1. Осуществлённое моделирование конечных автоматов сетями Петри показывает возможность применения сетей Петри для разработки аппаратного обеспечения ВС низкого уровня.

Замечание 2. Одно из основных преимуществ моделей автоматов в виде сетей Петри перед самими автоматами состоит в том, что они существенно упрощают композицию автоматов, как последовательную, так и параллельную.

Способность сетей Петри моделировать параллелизм и строить модель системы путём довольно простого объединения моделей подсистем делают их весьма полезным инструментом моделирования сложной аппаратуры вычислительных систем.

Конвейерные вычислительные системы и их моделирование сетями Петри.

В последнее время достаточно интенсивно развиваются высокопроизводительные средства параллельной вычислительной техники. К их числу, в частности, относятся вычислительные системы с *конвейерной обработкой*. Этот метод обработки подобен функционированию сборочного конвейера и особенно удобен для работы с векторами и массивами. Конвейер состоит из набора функциональных устройств (ФУ), которые реализуют различные операции, и могут работать одновременно. Когда операция k завершается, она передает свой результат операции (k + 1) и ожидает от операции (k – 1) нового задания. Если каждая операция занимает t единиц времени и всего n операций, то завершение конвейерной обработки одного операнда потребует $n*t$ единиц времени. Однако, если на конвейерную обработку продолжают поступать новые операнды, результаты могут выдаваться со скоростью один в каждые t единиц времени.

Задача 5. Конвейер для сложения двух чисел с плавающей точкой. Конвейер образуют следующие операции:

1. Выделить экспоненты обоих чисел.
2. Сравнить экспоненты.
3. Сдвинуть точку в числе с меньшей экспонентой для их уравнения.
4. Сложить дроби.
5. Нормализовать результат.

Этот конвейер позволяет выполнять до пяти сложений одновременно.

Конвейерная обработка может быть синхронной; т. е. время t, отпущенное на выполнение каждой операции, постоянно и является общим для всех ФУ. Каждые t единиц времени результат каждой операции перемещается по конвейеру, чтобы стать входом для следующей операции. Однако при синхронном подходе многие функциональные устройства могут простаивать довольно длительное время из-за того, что требующееся для них время обработки меньше t. В этом случае время t должно определяться самой медленной операцией конвейера, при этом необходимо учитывать, что время выполнения операции зависит от обрабатываемых данных.

Конвейерная обработка также может быть и асинхронной. В асинхронном конвейере процесс обработки может быть ускорен по сравнению с синхронным. Результат операции k асинхронного конвейера может быть послан операции (k+1), как

только операция *k* выполнена, а операция (*k*+1) свободна. Обычно асинхронный конвейер предполагает наличие регистров. Операция использует значение своего входного регистра для вычисления значения выходного регистра.

При этом ФУ возобновляет свою работу, когда его входной регистр заполнен новым значением, а само ФУ и его выходной регистр свободны. Передача содержимого выходного регистра *k*-того ФУ во входной регистр *k*+1-вого ФУ может произойти, если выходной регистр *k*-того ФУ заполнен, а входной регистр *k*+1-вого ФУ и само ФУ свободны.

Метод моделирования конвейерной обработки данных.

Для моделирования работы *k*-того ФУ конвейера необходимо выделить следующие условия:

- входной регистр *k*-того ФУ заполнен – ($а_k$);
- входной регистр *k*-того ФУ и само ФУ свободны – ($б_k$);
- выходной регистр *k*-того ФУ заполнен – ($в_k$);
- выходной регистр *k*-того ФУ и само ФУ свободны – ($г_k$);
- пересылка осуществляется от *k*-того ФУ к *k*+1-вому ФУ – ($д_k$);
- *k*-тое ФУ выполняет операцию – ($е_k$).

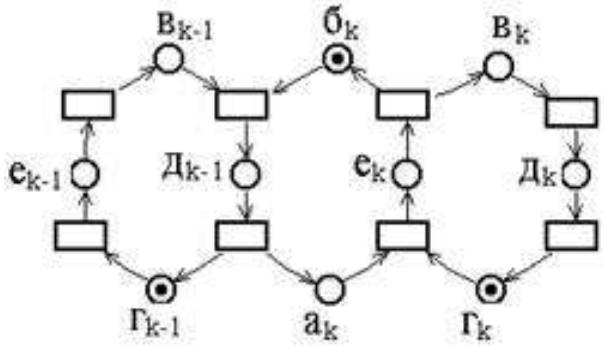

Рисунок 5. Сеть Петри, моделирующая работу *k*-того ФУ в асинхронном конвейере

Здесь внутренняя работа ФУ и пересылка данных промоделирована, как непримитивные события. Эта модель абстрагируется от внутренней работы ФУ и сосредотачивается на их правильном взаимодействии. Реализуемые с помощью ФУ операции также могут быть промоделированы сетями Петри.

Моделирование последовательных процессов.

Вырожденным случаем параллельной системы процессов является система с одним процессом. Сначала рассмотрим, как сетью Петри может быть представлен

отдельный процесс, а затем путем комбинации сетей Петри, представляющих несколько процессов, получим модель системы параллельных процессов. Отдельный процесс описывается программой. Эта программа может быть написана на одном из существующих языков программирования.

Пример 1. Последовательная программа на абстрактном языке программирования, вычисляющая Y! И произведение всех чётных чисел из отрезка [1, Y] для произвольного положительного целого Y.

```
begin
        read(Y);
        X1:=1;
        X2:=1;
        while Y>0 do
                begin
                        if mod(Y,2)=0
                                then begin
                                                X1:=X1*Y;
                                        end;
                        X2:=X2*Y;
                        Y:=Y-1;
                end;
        write(X1);
        write(X2);
end.
```

Программа представляет два различных аспекта процесса: вычисление и управление. Вычисление связано с текущими арифметическими и логическими операциями, вводом и выводом, обычными манипуляциями над участками памяти и их содержимым. Управление же связано не со значениями или выполняемыми вычислениями, а порядком действий в ходе такого выполнения.

Сети Петри удачно представляют структуру управления программ. Сети Петри предназначены для моделирования упорядочения действий и потока информации, а не для действительного вычисления самих значений. Модель системы в виде сети Петри абстрагируется от характера данных и их преобразований.

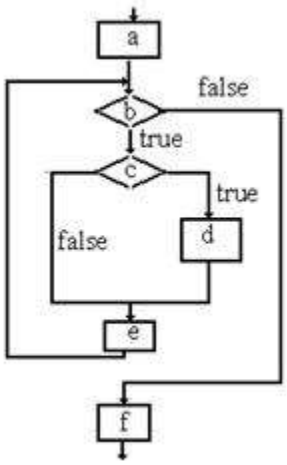

Рисунок 6. Блок-схема примера 1.

Блок-схема программы состоит из узлов двух типов (принятия решения, обозначаемых ромбами, и вычисления, обозначаемых прямоугольниками) и дуг между ними. В сети Петри действия моделируются переходами, а в блок-схеме действия представляются узлами. Таким образом, в сети Петри, моделирующей блок-схему, узлы блок-схемы представляются переходами сети Петри,, а дуги блок-схемы – позициями сети Петри. Каждая дуга блок-схемы соответствует точно одной позиции в сети Петри. Узлы блок-схемы представляются по-разному в зависимости от типа узла: вычисления или принятия решения.

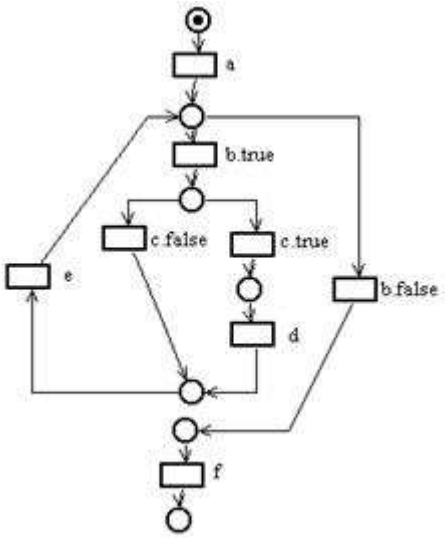

Рисунок 7. Сеть Петри, представляющая блок-схему программы из примера 1.

Фишка в сети Петри представляет счетчик команд блок-схемы. Фишка, находящаяся в позиции, означает, что счетчик команд установлен на готовность выполнения следующей команды. Каждая позиция имеет единственный выходной переход, за исключением позиции, которая предшествует принятию решения; такие

позиции имеют по два выходных перехода, соответствующих истинному и ложному значению предиката.

Переходы, соответствующие вычислительным узлам блок-схемы, имеют один вход и один выход. Переходы, представляющие действие по принятию решения находятся в конфликте.

Моделирование параллельных систем процессов.

Параллельная система может строиться несколькими способами. Один из способов состоит в простом объединении процессов, без взаимодействия во время их одновременного выполнения. Так, например, если система строится этим способом из двух процессов, каждый из которых может быть представлен сетью Петри, то сеть Петри моделирующая одновременное выполнение двух процессов, является простым объединением сетей Петри для каждого из двух процессов. Начальная маркировка составной сети Петри имеет две фишки, по одной в каждой сети, представляя первоначальный счетчик команд процесса.

Такой способ введения параллелизма имеет низкое практическое значение. Далее будем рассматривать параллельные системы процессов, допускающие взаимодействие процессов во время их параллельного выполнения. Существуют различные виды взаимодействия (синхронизации) процессов, в том числе: взаимодействие посредством общей памяти; - посредством передачи сообщения различных видов. Таким образом, для моделирования сетями Петри параллельных систем процессов, помимо последовательных процессов, необходимо уметь моделировать различные механизмы взаимодействия (синхронизации) процессов. Далее покажем, как сети Петри могут моделировать различные механизмы синхронизации процессов, на основе решения с помощью сетей Петри ряда задач, ставших классическими в области синхронизации.

Задача 6. О взаимном исключении.

Пусть несколько процессов разделяют общую переменную, запись, файл или другой элемент данных. Для обновления разделяемого элемента данных процесс должен сначала считать старое значение, затем вычислить новое и, наконец, записать его на то же место. Если два процесса P1 и P2 в одно и то же время пытаются выполнить такую последовательность действий, то могут возникнуть трудности. Возможна следующая последовательность:

1. Процесс P1 считывает значение из разделяемого объекта;

2. Процесс P2 считывает значение x из разделяемого объекта;

3. Процесс P1 вычисляет новое значение x'=f(x);

4. Процесс P2 вычисляет новое значение x"=g(x);

5. Процесс P1 записывает x' в разделяемый объект;

6. Процесс P2 записывает x" в разделяемый объект, уничтожая значение x

Результат вычисления процесса P1 потерян.

Теперь значением разделяемого объекта является g(x);в то время как им должно быть либо g(f(x)), либо f(g(x)). Очевидно, что это должно быть так, например, в случае, когда g(x) – снять со счета x 1000$, f(x) – поместить на счет x 1000$, а процессы P1 и P2 – банковские операции.

Для исключения подобных проблем используется метод взаимного исключения, основанный на понятии критическая секция. Критическая секция – это участок кода процесса, на котором он осуществляет доступ к разделяемому объекту данных. Прежде, чем выполнить свою критическую секцию, процесс ждёт, пока другой процесс не закончит выполнение собственной критической секции (если такое выполнение имеет место). Затем он входит в критическую секцию и блокирует доступ для любого другого процесса к своей критической секции. После выполнения процессом критической секции деблокируется доступ для других процессов к разделяемому объекту данных.

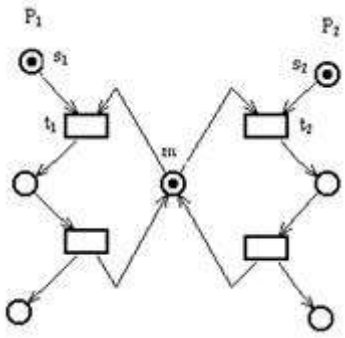

Рисунок 8. Сеть Петри моделирует механизм взаимного исключения для двух процессов P_1 и P_2.

Данная сеть легко обобщается на произвольное число процессов.

Позиция m представляет условие – критическая секция свободна, разрешающее вход в критическую секцию. Попытка процесса $P_1(P_2)$ войти в критическую секцию осуществляется после помещения фишки в его позицию s_1 (s_2). Такая попытка может увенчаться успехом, если в позиции m содержится фишка. Если оба процесса пытаются

войти в критическую секцию одновременно, то переходы t_1 и t_2 вступят в конфликт, и только один из них сможет запуститься. Запуск t_1 запретит запуск перехода t_2, вынуждая процесс P_2 ждать, пока процесс P_1 выйдет из своей критической секции, и возвратит фишку обратно в позицию m.

Задача 7. О производителе/потребителе.

В задаче о производителе/потребителе также присутствует разделяемый объект – буфер, посредством которого реализуется взаимодействие через асинхронную передачу сообщений. Процесс-производитель создает сообщения, которые помещаются в буфер. Потребитель ждет, пока сообщение не будет помещено в буфер, извлекает его оттуда и использует. Такое взаимодействие может быть промоделировано следующей ниже сетью Петри.

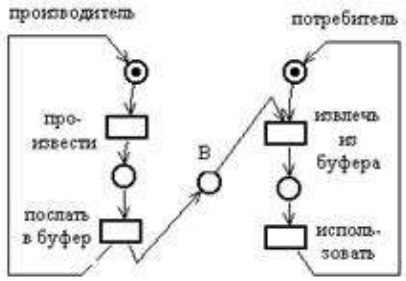

Рисунок 9. Сеть Петри о производителе/потребителе

Позиция B представляет буфер, каждая фишка соответствует сообщению, которое произведено, но еще не использовано.

В другом варианте этой задачи несколько производителей порождают сообщения, помещаемые в общий буфер, для нескольких потребителей. Моделирующая сеть Петри для этого варианта совпадает с сетью для предыдущего варианта задачи, за исключением того, что для представления s производителей и t потребителей используются s фишек в начальной позиции процесса-производителя и t фишек в начальной позиции процесса-потребителя.

Ещё в одном варианте задачи о производителе/потребителе используется буфер ограниченного размера, имеющий n ячеек для временного хранения сообщений. Здесь возможна ситуация, когда производитель будет вынужден ждать, если потребитель работает медленно и буфер заполнен. Для этого варианта задачи моделирующая сеть Петри выглядит следующим образом:

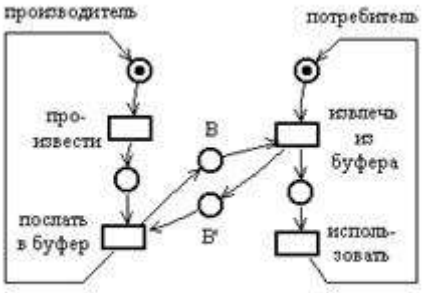

Рисунок 10. Сеть Петри о производителе/потребителе. Вариант 2

Ограниченному буферу сопоставляются две позиции: *B* представляет количество сообщений, которые произведены, но еще не использованы (число заполненных ячеек), и *B*' – количество пустых ячеек в буфере.

Первоначально *B*' имеет *n* фишек, а *B* фишек не имеет. Если буфер заполнен, то *B*' фишек не имеет, а *B* имеет *n* фишек. Если теперь производитель попытается поместить еще одно сообщение в буфер, то он будет остановлен, так как в *B*' нет фишек.

Задача 8. Об обедающих мудрецах.

Задача об обедающих мудрецах была предложена Дейкстрой и связана с пятью мудрецами, которые попеременно то думали, то ели. Мудрецы сидят за круглым столом, на котором много блюд китайской кухни. Между соседями лежит одна палочка для еды. Для приема китайской пищи необходимо две палочки. Поэтому каждый мудрец должен сначала взять палочку слева и палочку справа, а затем приступать к еде. Возможна ситуация, в которой каждый мудрец возьмёт палочку слева, а затем будет ждать, когда освободится палочка с правой стороны. Так они будут ждать, пока не умрут от голода. Тем самым, это состояние системы обедающие мудрецы является *тупиковым*.

Проблема тупика в этой системе может быть решена путём следующей модификации её правил поведения: Пусть мудрец при переходе из состояния размышления в состояние приёма пищи захватывает, не по очереди, а одновременно обе палочки (слева и справа), если они свободны. Следующая ниже сеть Петри моделирует такую модифицированную систему обедающих мудрецов, свободную от тупиков.

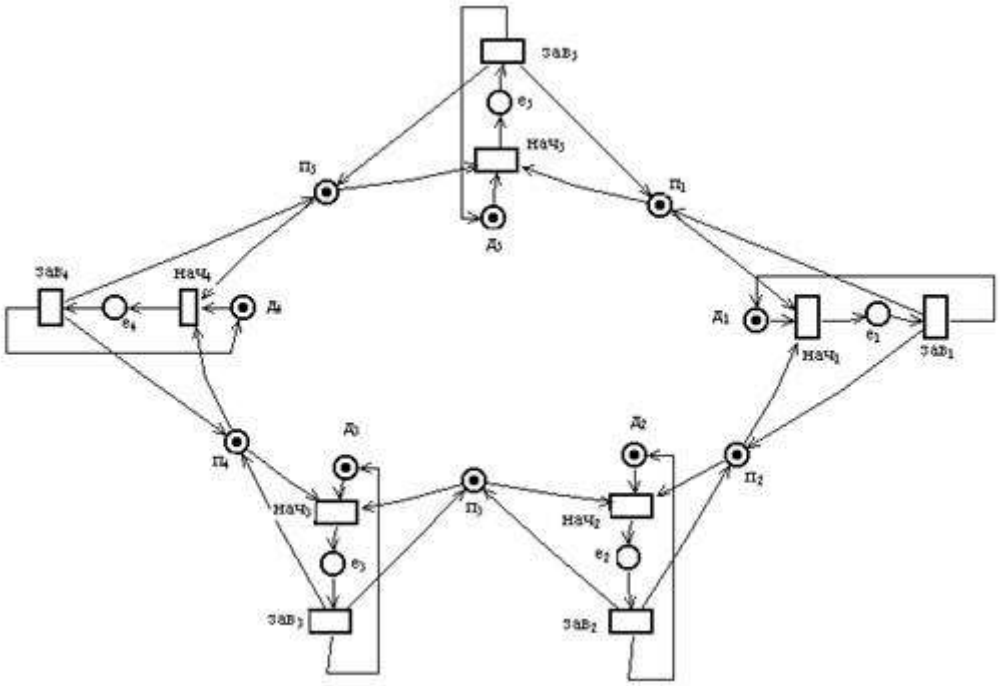

Рисунок 11. Сеть Петри задача о мудрецах

В этой сети Петри позиция $п_i$, где i ϵ {1,2,3,4,5}, представляет условие i-тая палочка свободна. В начальной маркировке каждая из этих позиций имеет фишку. Каждому мудрецу i ϵ {1,2,3,4,5} соответствует две позиции: позиция $д_i$ – представляющая условие i-тый мудрец думает; и позиция e_i. – представляющая условие i-тый мудрец ест. В начальной маркировке каждая позиция $д_i$ содержит фишку, а каждая позиция $\mathbf{e_i}$ пуста.

Каждому мудрецу i ϵ {1,2,3,4,5} также соответствует два перехода: переход $нач_i$ – представляющий событие начало приёма пищи i-тым мудрецом; и переход $зав_i$ – представляющий событие завершение приёма пищи i-тым мудрецом.

Замечание 3. В системе обедающие мудрецы: мудрецы могут интерпретироваться, как последовательные процессы, а палочки, как разделяемые ресурсы, посредством которых эти процессы взаимодействуют.

Задача 9. О чтении/записи.

Имеются процессы двух типов: процессы чтения и процессы записи. Все процессы совместно используют общий элемент данных. Процессы чтения не изменяют элемент данных в отличие от процессов записи. Таким образом, процессы записи должны взаимно исключать все другие процессы чтения и записи. При этом несколько процессов чтения могут иметь доступ к разделяемым данным одновременно. Задача

состоит в определении структуры управления, которая не приведет к тупику и не допустит нарушения механизма взаимного исключения.

Пусть имеются *s* процессов чтения и *t* процессов записи, а количество одновременно работающих процессов чтения ограничено величиной *n*. Моделирующая сеть Петри в этом случае имеет вид:

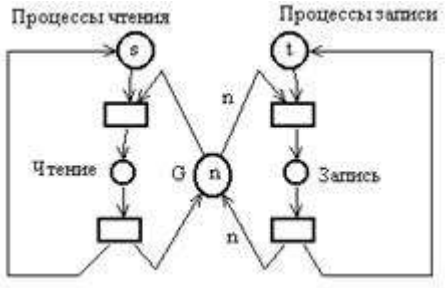

Рисунок 12. Сеть Петри задача о чтении/записи

В этой сети позиция Чтение представляет количество одновременно работающих процессов чтения. При инициализации каждого процесса чтения в эту позицию добавляется фишка, а по окончании процесса чтения фишка удаляется из неё. С помощью позиции *G* реализуется ограничение на количество одновременно работающих процессов чтения, а также – разрешение на запуск процесса записи (в начальный момент она содержит *n* фишек). Инициализация процесса записи возможна, если в позиции *G* находится *n* фишек (т.е. ни один из процессов не работает). В результате запуска такого процесса из позиции *G* все *n* фишек удаляются, тем самым блокируется возможность запуска для любого другого процесса. По окончании работы процесса записи *n* фишек возвращаются в позицию *G*.