МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

**ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ**

Н.Ю. Салмина

ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ

Учебное пособие

2015

Рассматриваются общие понятия имитационного моделирования, некоторые вопросы статистического моделирования. Проведено общее описание программных средств моделирования систем и языков моделирования.

Рассмотрен язык моделирования GPSS: синтаксис, особенности применения, возможности применения языка для исследования различных систем и объектов, статистический анализ результатов моделирования. Теоретический материал иллюстрируется многочисленными примерами.

Предназначено для студентов, обучающихся по направлению 080700 «Бизнес-информатика» и студентов родственных направлений.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Управление в современном мире становится все более трудным делом, поскольку организационная структура общества усложняется. Исследованию подвергаются все более и более сложные системы, в которых изменение одной из характеристик может легко привести к изменениям во всей системе или создать потребность в изменениях в других частях системы. Соответственно возникает необходимость в использовании все более сложных методов научных исследований.

Известны три общих направления в научных исследованиях: экспериментальное и теоретическое исследования и компьютерное моделирование.

Экспериментальные исследования проводятся с реальным объектом в форме натурного эксперимента. В результате получают экспериментальные данные о поведении и свойствах объекта исследования. Известны две стратегии проведения натурного эксперимента: активная и пассивная. В первом случае экспериментатор имеет возможность изменять внешние условия, определяющие состояние объекта, во втором — такой возможности нет. В этом случае говорят о наблюдении за изменениями состояния объекта — исследования (социальные, природные) соответствующих объектов длительны и трудоемки.

Теоретические исследования. Теория имеет дело не с реальным объектом исследования, а с его идеализированным представлением, допускающим применение адекватного математического аппарата для формального описания объекта. Здесь основой является некая совокупность фактов, полученная в результате наблюдений или натурного эксперимента. В результате теоретического осмысления фактов у исследователей возникает ряд гипотез, объясняющих поведение и свойства объекта. При последующей экспериментальной проверке одна из гипотез может принять форму научного закона. Однако этот традиционный путь формирования теоретического знания в случае сложных систем оказывается неприменимым, и роль законов выполняют математические методы. Разрабатывается математический аппарат теории, который обеспечивает возможность проведения исследования объекта аналитическими методами. Но и

эти методы в реальных ситуациях часто оказываются неприменимыми. Причины: недостаточная разработанность математического аппарата; чрезмерно большая размерность решаемой задачи; наличие большого числа случайных факторов. Возникает необходимость использования компьютерного моделирования.

Компьютерное моделирование. При использовании экспериментальных и теоретических методов исследования создается (накапливается) исходная информация об объекте исследования, а ***модель*** используется как средство интеграции и хранения знаний об объекте исследования. Так, теоретические исследования проводятся с целью построения модели, а натурный эксперимент проводится с целью проверки адекватности модели. Здесь модель играет прикладную роль, она является и результатом исследования и средством хранения знаний об объекте исследований.

Различают два вида ***компьютерных экспериментов***: вычислительный и имитационный. Модель при вычислительном эксперименте строится в виде уравнений (дифференциальных, алгебраических и пр.). При этом приходится использовать численные методы и комплексы эффективных вычислительных алгоритмов. В отличие от вычислительных экспериментов ***модели в имитационных экспериментах*** из-за отсутствия теории объекта ***описывают поведение*** реального объекта и ***реализуются в виде набора алгоритмов, отображающих ситуации, возникающие в состоянии моделируемого объекта и изменяющиеся по определенным сценариям.*** Результатами имитационного эксперимента являются не численные решения каких-либо уравнений модели, а реализации созданных с помощью компьютера процессов, имитирующих поведение реального объекта. Термин «реальный» будем использовать в смысле «существующий или способный принять одну из форм существования», т.е. системы, находящиеся только в стадии планирования или разработки, могут моделироваться так же, как и действующие системы.

В настоящее время, при исследовании сложных систем все чаще предпочтение отдается именно имитационному моделированию. Это обусловлено как стремительным развитием компьютерной техники, так и совершенствованием

программного обеспечения, к которому относятся и языки имитационного моделирования.

Цель имитационного моделирования состоит в воспроизведении поведения исследуемой системы на основе результатов анализа наиболее существенных взаимосвязей между ее элементами или другими словами — разработке симулятора (simulation modeling) исследуемой предметной области для проведения различных экспериментов.

Под **имитационной моделью** понимают логико-математическое описание объекта, которое может быть использовано для экспериментирования на компьютере в целях проектирования, анализа и оценки функционирования объекта.

Другими словами, **имитационная модель** — это компьютерная программа, которая описывает структуру и воспроизводит поведение реальной системы во времени. Имитационная модель позволяет получать подробную статистику о различных аспектах функционирования системы в зависимости от входных данных.

Имитационное моделирование позволяет имитировать поведение системы во времени. Причём плюсом является то, что временем в модели можно управлять: замедлять в случае с быстропротекающими процессами и ускорять для моделирования систем с медленной изменчивостью. Можно имитировать поведение тех объектов, реальные эксперименты с которыми дороги, невозможны или опасны.

Модель служит обычно средством, помогающим нам в объяснении, понимании или совершенствовании системы. В настоящее время моделирование становится не только эффективным методом научных исследований сложных объектов, но и мощным инструментом конструирования и проектирования сложных систем. Качество решений задач, получаемых с помощью имитационного моделирования, определяется степенью адекватности модели реальному объекту (т.е. степенью соответствия результатов моделирования результатам работы реального объекта). Поэтому необходимо помнить, что подобно всем мощным средствам, существенно зависящим от искусства их при-менения (имитационное моделирование и экспериментирование во многом остаются ин-

туитивными процессами), моделирование способно дать либо очень хорошие, либо очень плохие результаты; либо пролить свет на решение проблемы, либо ввести в заблуждение. Результат моделирования зависит от степени адекватности модели, правильности исходных предпосылок, умения исследователя правильно применять используемые методы, правильно интерпретировать результаты.

В настоящее время существует несколько больших классов моделей. Так как выбор класса зависит от целей исследования и свойств сложной системы, рассмотрим основные функции, выполняемые моделями сложных систем [2].

1. Объяснительная функция модели заключается в способности модели упорядочить нечеткие или противоречивые понятия: выявить взаимозависимости, временные соотношения; помочь интерпретировать данные натурного эксперимента. Уже сама попытка формализовать существующие явления и процессы помогает в понимании функционирования объекта;

2. Информационная функция заключается в возможности использования модели как средства для накопления и хранения знаний об объекте;

3. Обучающая функция модели может служить для обучения и тренажа лиц, которые должны уметь справляться с всевозможными случайностями до возникновения реальной критической ситуации (модели космических кораблей, различные тренажеры, деловые игры);

4. Предсказательная функция модели связана с возможностью прогнозировать с заданной точностью по некоторым данным натурных экспериментов поведение и свойства объекта (одна из наиболее важных);

5. Функция постановки и проведения экспериментов для объектов дает возможность характеризовать различные состояния объектов моделирования, таких, где экспериментирование на реальных системах невозможно или нецелесообразно (люди, природа, атомные реакторы).

Зачастую одна модель может выполнять одновременно несколько функций (использоваться для проведения экспериментов и для прогноза; проведения экспериментов и объяснения; для обучения и накопления знаний).

В общем случае модель может служить для достижения двух целей: ***описательной*** — для объяснения или лучшего понимания объекта; ***предписывающей***

— для предсказания и/или воспроизведения характеристик объекта, определяющих его поведение. Обычно предписывающие модели являются и описательными, но не наоборот. Предписывающие модели явно предпочтительнее по своим возможностям, но сложнее по реализации, поэтому их построение не всегда возможно.

Вероятно, в том что описательная модель не всегда полезна для целей планирования и проектирования и кроется одна из причин, по которой экономические модели, имеющие (или обнаруживающие) тенденцию к описательности, оказали небольшое воздействие на управление экономическими системами и мало применялись в качестве вспомогательного средства управления на высшем уровне, в то время как модели исследования операций (теория игр) по общему признанию оказали значительное воздействие на эти сферы. В моделях социальных систем наблюдается тенденция к описательности, объяснению явлений (слишком сложный объект моделирования); модели технических систем, предназначенные для разработки новых систем, носят и предписывающий, и описательный характер.

Модели вообще, и имитационные модели в частности, можно классифицировать различными способами. К сожалению, ни один из них не является удовлетворительным, хотя каждый служит определенной цели. Укажем некоторые **типовые группы моделей**, которые могут быть положены в основу системы классификации:

- статические и динамические;
- детерминированные и стохастические;
- дискретные и непрерывные.

Этапы моделирования

С развитием вычислительной техники наиболее эффективным методом исследования больших систем стало машинное моделирование. Рассмотрим этапы машинного моделирования реальных систем:

0) определение цели и задач моделирования;

1) определение системы — установление границ, ограничений и измерителей эффективности изучаемой системы;

2) формулирование модели — переход от реальной системы к некоторой логической схеме (абстрагирование);

3) подготовка данных — отбор данных, необходимых для построения модели, и представление их в соответствующей форме;

4) трансляция модели — описание модели на языке программирования или моделирования;

5) оценка адекватности — оценка степени уверенности в достаточной корректности выводов о реальной системе, полученных на основании обращения к модели;

6) планирование эксперимента — решение задачи получения необходимой информации о системе с помощью компьютерной модели, с учетом ограничения на ресурсы;

7) экспериментирование — процесс осуществления имитации с целью получения желаемых данных и анализа чувствительности;

8) интерпретация — построение выводов по данным, полученным путем имитации;

9) реализация — практическое использование модели и/или результатов моделирования.

Любое моделирование начинается с постановки задачи моделирования, то есть с ясного изложения целей моделирования. Цели обычно формируются в виде вопросов, на которые надо ответить, либо гипотез, которые надо проверить, либо воздействий, которые надо оценить.

Для определения ограничений задачи моделирования необходимо выявить характеристики системы, подлежащей изучению. Для этого сначала устанавливаются граничные условия, то есть то, что является или не является частью изучаемой системы. Далее определяются существенные параметры и переменные системы, которые характеризуют объект изучения и позволяют установить основные ограничения задачи моделирования.

На этапе формулирования модели разрабатывается моделирующий алгоритм, с помощью которого имитируются явления, составляющие исследуемый процесс. С этим этапом тесно связан этап подготовки данных, в процессе которого определяется, в каком виде будут представлены параметры и переменные системы – в виде констант, либо в виде генерируемых последовательностей случайных чисел.

На этапе трансляции модели возникает проблема ее описания на языке, приемлемом для использования компьютера. Здесь использование специальных языков моделирования вместо универсальных существенно экономит время программирования.

Процесс проверки адекватности модели включает в себя несколько шагов последовательных проверок. На первом шаге исследователь должен убедиться, что модель верна в первом приближении. На втором шаге оценки адекватности проверяются исходные предположения. Например, какие параметры и переменные модели можно считать существенными, и охвачены ли моделью все существенные параметры объекта. На третьем шаге проверяются преобразования информации от входа к выходу.

После проверки адекватности модели необходимо спланировать и провести эксперимент с моделью для получения желаемой информации. Анализ данных, полученных в результате эксперимента, позволяет делать выводы о возможностях дальнейшего использования самой модели или результатов моделирования.

Необходимо отметить, что моделирование может быть остановлено на любом этапе, если результат достигнут. Кроме того, возможны возвраты к предыдущим этапам. Так, если оценка адекватности модели отрицательна, то может быть возврат на любой из предыдущих этапов моделирования. Если при подготовке данных выясняется, что некоторые данные не могут быть получены или представлены в требуемом виде, то возможно повторное формулирование модели. После проведения эксперимента осуществляется анализ и интерпретация результатов. Если результаты удовлетворяют исследователя, то на этом процесс

моделирования завершается, в противном случае возможен возврат на любой предыдущий этап моделирования.

В предложенном учебном пособии рассмотрены основные общие моменты построения имитационных моделей, рассмотрены основные вопросы метода имитационного или статистического моделирования. Основное внимание в пособии уделено одному из наиболее распространенных языков моделирования GPSS.

1. ОРГАНИЗАЦИЯ СТАТИСТИЧЕСКОГО МОДЕЛИРОВАНИЯ СИСТЕМ

1.1. Общая характеристика метода

Статистическое моделирование представляет собой метод получения с помощью имитационных моделей статистических данных о процессах, происходящих в моделируемой системе. Как правило, такое моделирование используется для исследования стохастических систем. Для получения оценок характеристик моделируемой системы с учетом воздействия внешней среды статистические данные обрабатываются и классифицируются с использованием методов математической статистики.

В некоторых случаях, когда можно построить аналитическую модель случайного процесса (например, систему дифференциальных уравнений для вероятностей состояния или алгебраических уравнений для предельных состояний), статистическое моделирование не является обязательным. Если в системе протекают Марковские процессы, то построение аналитической модели, как правило, не вызывает затруднений. В тех случаях, когда построение аналитической модели является трудно осуществимым, применяется другой метод моделирования — метод статистических испытаний (метод Монте-Карло), который является одним из наиболее широко используемых.

Сущность метода Монте-Карло можно описать следующим образом: производится «розыгрыш» — моделирование случайного явления с помощью некоторой процедуры, дающей случайный результат. Проводя «розыгрыш» много раз, мы получаем статистический материал — множество реализаций случайного явления, который можно обработать обычными методами статистики. Часто этот прием оказывается проще, чем попытки построить аналитическую модель явления и исследовать зависимость между его параметрами на этой модели. Однако заметим, что он оправдан только в том случае, если его использование проще аналитического.

Сущность самого метода статистического моделирования (МСМ) сводится к построению для процесса функционирования исследуемой системы некоторого моделирующего алгоритма с использованием метода Монте-Карло. Данный алгоритм имитирует поведение и взаимодействие элементов системы с учетом случайных входных воздействий и воздействий внешней среды.

Рассматриваются две области применения МСМ:

1) для изучения стохастических систем — систем массового обслуживания (СМО), статистических систем РО и т.п.;

2) для решения детерминированных задач (например, вычисление интегралов, обработка больших массивов информации).

При решении детерминированных задач основная идея заключается в замене детерминированной задачи эквивалентной схемой некоторой стохастической системы, выходные характеристики которой совпадали бы с результатами решения детерминированной задачи. Естественно, при замене получается приближенное решение. Погрешность уменьшается с увеличением числа испытаний N.

В результате статистического моделирования получается серия частных значений искомых величин или функций, статистическая обработка которых позволяет получить сведения о поведении реального объекта или процесса в произвольные моменты времени. Допустим, необходимо вычислить математическое ожидание случайных величин x, подчиняющихся некоторому закону распределения $F(x)$. Для этого реализуют датчик случайных чисел, имеющий данное распределение, и определяют оценку математического ожидания:

$$M(x) = \frac{x_1 + x_2 + \ldots + x_N}{N}.$$

Если N достаточно велико, то полученные результаты моделирования приобретают статистическую устойчивость и с достаточной точностью могут быть приняты в качестве оценок искомых характеристик (теоретическая основа — предельная теорема Бернулли, Пуассона, закон больших чисел Чебышева).

В качестве оценки дисперсии используется эмпирическая или выборочная дисперсия, определяемая по формуле:

$$S^2 = \frac{1}{N-1}\left(\sum_{i=1}^{n} x_i^2 - \frac{1}{N}\left(\sum_{i-1}^{n} x_i\right)^2\right)$$

Пример 1.1 (моделирование стохастической системы)

С помощью МСМ необходимо найти оценки выходной характеристики стохастической системы S_k, для которой: $x = 1 - e^{-\lambda}$ — входное воздействие; $v = 1 - e^{-\varphi}$ — воздействие внешней среды, где φ и λ — случайные величины с известными функциями распределения. Выходная характеристика распределения описывается выражением $y = \sqrt{x^2 + v^2}$. В качестве оценки математического ожидания может выступать среднее арифметическое искомой величины: $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$, где y_i — случайное значение величины y.

Алгоритм решения:

1. Генерация λ_i, φ_i;

2. $x_i = 1 - e^{-\lambda_i}$, $v_i = 1 - e^{-\varphi_i}$;

3. $h_i = x_i^2 + v_i^2$;

4. $y_i = \sqrt{h_i}$.

Точность и достоверность результатов моделирования будет определяться числом реализаций N.

Для получения λ_i, φ_i используются датчики случайных чисел.

Пример 1.2 (решение детерминированной задачи)

Найти оценку площади фигуры, ограниченной осями координат, ординатой $\alpha = 1$ и кривой $r = f(\alpha)$; для определенности предполагается, что $0 \le f(\alpha) \le 1$ для всех α, $0 \le \alpha \le 1$.

Аналитическое решение сводится к вычислению интеграла:

$$S = \int_0^1 F(\alpha)\, d\alpha .$$

Искомая величина является площадью заштрихованной фигуры (рис. 1.1) и может быть получена с помощью МСМ: построить адекватную по выходным характеристикам стохастическую систему S_D, оценки характеристик которой будут совпадать с искомыми.

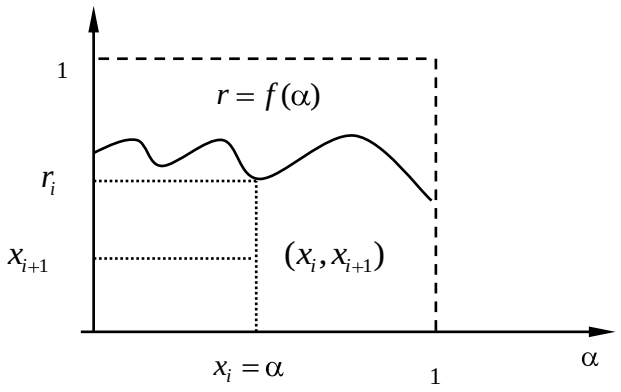


Рис. 1.1. Графическое представление решения

Алгоритм решения:

1. Генерация равномерно распределенных на интервале [0,1] случайных чисел x_i, x_{i+1};
2. Вычисление ординаты $r_i = f(x_i)$;
3. Проверка попадания точки (x_i, x_{i+1}) в площадь фигуры:

$$h_i = \begin{cases} 1, \text{если } x_{i+1} \le f(x_i), \\ 0 \text{ в противномслучае.} \end{cases}$$

4. Первые три шага повторяются N раз;
5. Оценка площади фигуры $\bar{S} = \frac{1}{N}\sum_{i=1}^{N} h_i$.

Таким образом, независимо от природы объекта исследования (статистической или детерминированной) подход для решения является общим. Отметим, что при этом не требуется запоминание всего множества генерируемых случайных чисел. При расчете оценок характеристик исследуемых процессов используются только накопленная сумма исходов и общее число реализаций. Это не-

маловажное обстоятельство вообще является характерным при реализации имитационных моделей с помощью МСМ.

Одна из основных задач, которую необходимо решить при построении любого моделирующего алгоритма — это генерация последовательностей случайных чисел (в примерах 1.1 и 1.2 — это $\lambda_i, r_i, x_i, x_{i+1}$). Далее мы рассмотрим вопросы, связанные с решением этой задачи.

1.2. Псевдослучайные числа и процедуры их машинной генерации

Успех и точность статистического моделирования зависит в основном от качества генерируемых последовательности случайных чисел. Кроме этого, возможность практического использования компьютерного моделирования во многом определяется наличием простых и экономичных способов формирования последовательностей случайных чисел. Рассмотрим особенности и возможности получения таких последовательностей.

Задача получения случайных чисел с помощью программной имитации обычно разбивается на две:

1) получают последовательность случайных чисел, имеющих равномерное распределение на интервале [0,1]. Может быть выбран и другой базовый процесс, однако при дискретном моделировании базовой является последовательность независимых, равномерно распределенных величин;
2) с помощью различных функциональных преобразований из базовой последовательности получают последовательность случайных чисел с произвольным законом распределения.

Компьютерное представление вещественных чисел всегда ограничено количеством разрядов k, следовательно, максимальное число не совпадающих между собой чисел равно 2^k. Поэтому в машине вместо непрерывной совокупности

равномерных случайных чисел можно использовать только дискретную последовательность, множество чисел в которой конечно. Такое распределение называется квазиравномерным. Вероятность каждого значения для квазиравномерного распределения равна $\frac{1}{2^k}$. При достаточно больших k математическое ожидание и дисперсия квазиравномерной случайной величины близки к математическому ожиданию и дисперсии равномерной случайной величины.

Для получения случайных чисел алгоритмическим путем с помощью специальных программ разработано большое количество методов. Так как с помощью компьютера невозможно получить идеальную последовательность случайных чисел хотя бы потому, что на ней можно набрать только конечное количество чисел, то последовательности чисел, генерируемых с помощью ЭВМ, называются псевдослучайными. На самом деле повторяемость или периодичность в последовательности псевдослучайных чисел наступает значительно раньше и обусловливается спецификой алгоритма их получения.

Наибольшее применение на практике для моделирования последовательностей псевдослучайных чисел находят алгоритмы вида $x_{i+1} = \Phi(x_i)$, называемые рекуррентными соотношениями первого порядка, для которых начальное число x_0 либо задается константой, либо генерируется случайным образом (обычно – от таймера).

Равномерное распределение является базовым при имитационном моделировании, наверно, именно поэтому в настоящее время в любой язык программирования и в любые специализированные среды включены датчики случайных чисел с равномерным законом распределения. При разработке имитационной модели можно всегда воспользоваться готовым датчиком, поэтому в данном пособии мы не будем рассматривать методы генерации равномерно распределенных случайных чисел. При желании читатель может изучить эти методы, обратившись к другим источникам.

1.3. Моделирование случайных воздействий

Перейдем к рассмотрению вопросов преобразования базовых последовательностей случайных чисел $\{x_i\}$ в последовательности $\{y_i\}$ для имитации воздействия на моделируемую систему. Эти задачи очень важны, т.к. на практике существенное количество операций (временных ресурсов) расходуется на действия со случайными числами. Таким образом, наличие эффективных методов, алгоритмов и программ формирования таких последовательностей во многом определяет возможность практического использования машинной имитации.
Простейший случайный объект — ***случайное событие*** A, наступающее с вероятностью p.

Здесь процедура моделирования состоит в следующем: генерируется случайная величина x_i, равномерно распределенная на интервале [0,1] и сравнивается с вероятностью p. Если условие $x_i \le p$ выполняется, то исходом испытания является наступление события A, в противном случае событие A не наступает.

Следующим рассмотрим ***группу событий***: известно, что может наступить одно из событий $A_1, A_2 \dots A_k$. Заданы вероятности наступления событий $p_1, p_2, \dots p_k$. Так как в любой момент времени какое-то из событий наступает обязательно, два события одновременно наступить не могут, то выполняется условие $\sum_{i=1}^{k} p_i = 1$.

Здесь процедура моделирования состоит в сравнении равномерно распределенной случайной величины x_i со значениями $l_r = \sum_{i=1}^{r} p_i$ для соответствующего A_r. Событие A_m наступает, если $l_{m-1} < x_i \le l_m$.

Пример 1.3

Самолет, производящий полет над территорией противника, после стрельбы по нему может оказаться в одном из следующих состояний:

A_1 — невредим;

A_2 — поврежден, продолжает полет;

A_3 — совершил вынужденную посадку;

A_4 — сбит.

Вероятности соответствующих состояний равны:

$$P(A_1) = 0{,}4;\ P(A_2) = 0{,}1;\ P(A_3) = 0{,}15;\ P(A_4) = 0{,}35.$$

Генератор событий моделируется следующим образом:

1) генерация равномерно распределенного на [0, 1] случайного числа x_i;

2) если $0 \le x_i \le 0{,}4$, то наступает событие A_1, иначе —

если $0{,}4 < x_i \le 0{,}5$, то наступает событие A_2, иначе —

если $0{,}5 < x_i \le 0{,}65$, то наступает событие A_3, иначе —

если $0{,}65 < x_i \le 1$, то наступает событие A_4.

Заметим, что для группы событий вероятность наступления отдельного события зависит от ширины рассматриваемого интервала, и не зависит от его расположения на отрезке [0,1]. Так, если в примере 1.3 рассматривать наступление события A_1 как попадание x_i в интервал [0.6, 1], то вероятность его наступления по-прежнему остается равной 0.4, и результаты моделирования не изменятся.

Если искомый вариант зависит от двух ***независимых событий*** A и B, то в результате возможны исходы $AB, \overline{A}B, A\overline{B}, \overline{A}\overline{B}$, с вероятностями $p_A p_B$, $(1-p_A)p_B$, $p_A(1-p_B)$, $(1-p_A)(1-p_B)$.

Для моделирования существуют два возможных варианта:

1) последовательная проверка условия $x_i \le p$ для каждого события в отдельности;

2) моделирование ситуации как группы событий $(A_1 = AB, A_2 = \bar{A}B\ldots)$.

Для первого варианта требуется сгенерировать два числа x_i и выполнить два сравнения; для второго варианта требуется одно число x_i, но больше сравнений. Первый вариант является более предпочтительным: удобство алгоритма, экономия количества операций, ячеек памяти и т.д.

Рассмотрим два ***зависимых события***: пусть события A и B могут наступить с вероятностями, соответственно, p_A и p_B. При этом наступление события B зависит от наступления события A с вероятностью $p(B/A)$. Один из вариантов моделирования:

1) генерируются два равномерно распределенных случайных числа x_1 и x_2;

2) x_1 используется для проверки наступления события A по условию $x_1 \le p_A$;

3) если событие A наступило, то наступление события B проверяется по условию $x_2 \le p(B/A)$;

4) если событие A не наступило, то вычисляется вероятность $p(B/\bar{A}) = (p_B - p_A \cdot p(B/A))/(1 - p_A)$ и проверяется выполнение условия $x_2 \le p(B/\bar{A})$.

Для формирования возможных значений ***случайных величин*** необходимо получить последовательность с заданным законом распределения.

Дискретная ***случайная величина*** η принимает значения $y_1 \le y_2 \le \ldots \le y_i \le \ldots$ с вероятностью $p_1, p_2, \ldots, p_i, \ldots$.

Интегральная функция распределения

$F_\eta(y) = p(\eta \le y) = \sum_{j=1}^{m} p_j$, если $y_m \le y \le y_{m+1}$,

$F_\eta(y) = 0$, если $y < y_1$.

Для получения η можно использовать метод обратной функции. Если ξ — равномерно распределенная случайная величина на интервале [0,1], то $\eta = F_\eta^{-1}(\xi)$.

Здесь алгоритм генерации аналогичен моделированию группы событий:

если $x_1 \le p_1$, то $\eta = y_1$, иначе —

если $x_2 \le p_1 + p_2$, то $\eta = y_2$, иначе —

… ,

где x_i — равномерно распределенное случайное число.

Непрерывная случайная величина задана интегральной функцией распределения

$$F_\eta(y) = p(\eta \le y) = \int_{-\infty}^{y} f_\eta(y)\, dy .$$

Можно воспользоваться как и для дискретной случайной величины методом обратной функции (другое название — ***метод нелинейных преобразований***)

$$\eta = F_\eta^{-1}(\xi).$$

Функция преобразует равномерно распределенную случайную величину ξ в случайную величину η с требуемой плотностью распределения $f_\eta(y)$. Другими словами, необходимо разрешить относительно y_i уравнение

$$\int_{-\infty}^{y_i} f_\eta(y) dy = x_i \ .$$

Пример 1.4

Необходимо сгенерировать последовательность чисел с экспоненциальным законом распределения

$$f_\eta(y) = \lambda e^{-\lambda y}, y > 0 \ \Rightarrow \ \lambda \int_0^{y_i} e^{-\lambda y} dy = x_i ,$$

где x_i — равномерно распределенное случайное число. Отсюда следует

$$1 - e^{-\lambda y_i} = x_i \Rightarrow y_i = -\ln(1 - x_i)/\lambda .$$

Учитывая, что случайная величина $\xi_1 = 1 - \xi$ также имеет равномерное распределение в $[0,1]$, можно записать $y_i = -\ln(x_i)/\lambda$.

Метод нелинейных преобразований является единственным точным методом генерации случайных величин с заданным законом распределения, хотя и имеет ограниченную сферу применения по следующим причинам:

1) для многих законов распределения интеграл не берется, т.е. приходится прибегать к численным методам решения, что увеличивает затраты машинного времени;

2) если интеграл берется, получаются формулы, содержащие логарифм, корень и т.д., что резко увеличивает затраты машинного времени за счет большого количества выполняемых операций.

По причине указанных недостатков метода нелинейных преобразований, на практике используются и другие способы моделирования непрерывных случайных величин (все они являются приближенными):

а) универсальные — для любых законов распределения;

б) неуниверсальные — для конкретных законов распределения.

Наиболее простым из ***универсальных способов*** является ***метод Неймана*** (режекции). Он заключается в генерации двух равномерно распределенных случайных чисел x_1 и x_2 и их преобразовании: $x_1' = a + (b-a)\cdot x_1$, $x_2' = W \cdot x_2$. Здесь a и b — левая и правая границы интервала задания $f_\eta(y)$, а $W = \max\limits_{[a,b]} f_\eta(y)$. Если $x_2' \le f_\eta(x_1')$, то x_1' — искомое значение случайной величины η.

Метод кусочной аппроксимации

Требуется получить последовательность $\{y_i\}$ с функцией плотности распределения $f_\eta(y)$, возможные значения которой лежат в интервале (a,b). Представим $f_\eta(y)$ в виде кусочно-постоянной функции — разобьем (a,b) на интервалы, на которых функцию плотности распределения будем аппроксимировать равномерным законом. Тогда случайная величина может быть получена по формуле $\eta = a_k + \eta_k^*$, где a_k — абсцисса левой границы k-го интервала, η_k^* — случай-

ная величина, значения которой располагаются равномерно внутри k-го интервала, т.е. считается равномерно распределенной.

Чтобы аппроксимировать $f_\eta(y)$ наиболее удобным для практических целей способом, целесообразно разбить (a,b) на m интервалов так, чтобы вероятность попадания η в любой интервал была постоянной (рис. 1.2), т.е. не зависела от k, тогда для вычисления a_k используют формулу $\int_{a_k}^{a_{k+1}} f_\eta(y)dy = \frac{1}{m}$ (**).

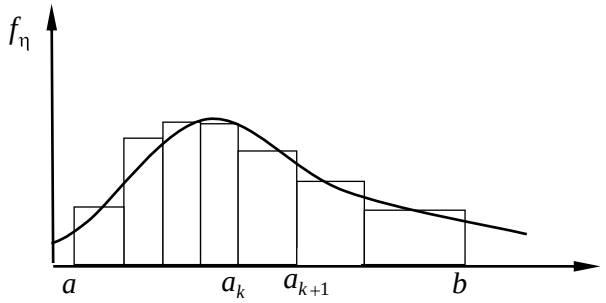


Рис. 1.2. Метод кусочной аппроксимации

Алгоритм генерации:

1) генерируется случайное число x_i на интервале [0,1];

2) с помощью этого числа случайным образом выбирается интервал $[a_k, a_{k+1}]$;

3) генерируется x_{i+1} и масштабируется с целью приведения к интервалу $[a_k, a_{k+1}]$, т.е. $(a_{k+1} - a_k) \cdot x_{i+1}$;

4) $y_j = a_k + (a_{k+1} - a_k) \cdot x_{i+1}$ — вычисляется y_j с требуемым законом распределения.

Рассмотрим подробнее второй шаг: целесообразно для этого построить таблицу, содержащую номера интервалов k, их границ и коэффициентов масштабирования, определенные из (**).

Сгенерировав x_i, мы можем определить номер интервала. Так как вероятность выбора интервалов одинакова и равна $\frac{1}{m}$, то используя метод генерации группы событий, получаем:

если $0 \le x_i \le \frac{1}{m}$, то выбираем 1-й интервал;

если $\frac{1}{m} < x_i \le \frac{2}{m}$, то выбираем 2-й интервал, и т.д.

Достоинство данного метода состоит в небольшом количестве операций, т.к. границы интервалов мы рассчитываем один раз до начала генерации, а сам алгоритм генерации содержит только операции сравнения, сложения и вычитания. Точность метода всегда можно повысить за счет увеличения количества интервалов m. На практике обычно выбирают m в пределах от 16 до 20.

Неуниверсальные способы строятся на основе предельных теорем теории вероятностей и предназначены для получения последовательностей случайных чисел с конкретным законом распределения. Рассмотрим некоторые из этих способов.

Нормальный закон распределения

Требуется получить нормальное распределение с математическим ожиданием a и среднеквадратичным отклонением σ.

Будем формировать y_i в виде сумм последовательностей $\{x_i\}$. По центральной предельной теореме: если независимые одинаково распределенные случайные величины имеют каждая математическое ожидание a_1 и среднеквадратичное отклонение σ_1, то сумма $\sum_{i=1}^{N} x_i$ асимптотически нормальна с математическим ожиданием $a = N \cdot a_1$ и среднеквадратическим отклонением $\sigma = \sigma_1\sqrt{N}$. Сумма имеет распределение близкое к нормальному уже при небольшом N (может быть $8 \div 12$ или даже $4 \div 5$ — в простейших случаях).

Генерируем нормально распределенную величину $V = \sum_{i=1}^{N} x_i$, где x_i — равномерно распределенная случайная величина на [0,1] с математическим ожиданием равным 0,5, поэтому математическое ожидание $m_V = N \cdot 0{,}5$.

$D_{x_i} = \frac{(\beta - \alpha)^2}{12} = \frac{1}{12}$ — для равномерно распределенных величин на [0,1], отсюда

$D_V = \sum_i D_{x_i} = \frac{N}{12}$, а $\sigma_V = \sqrt{D_V} = \sqrt{\frac{N}{12}}$.

$z = \dfrac{V - m_V}{\sigma_V} = \dfrac{V - N \cdot 0{,}5}{\sqrt{N/12}}$ — нормально распределенная величина с нулевым математическим ожиданием и единичной дисперсией.

$y = \sigma_y z + m_y$ — нормально распределенная величина с заданными σ_y и m_y.

Пуассоновское распределение

Случайная величина с пуассоновским распределением — это целочисленная величина, которая характеризует количество наступивших событий. Вероятность наступления m событий равна:

$$P(m, a) = \frac{a^m}{m!} e^{-a},$$

где a — математическое ожидание.

Для получения пуассоновских чисел с заданным математическим ожиданием можно использовать следующий метод. Образуется произведение равномерно распределенных последовательных случайных чисел x_i. Количество сомножителей k выбирается таким, чтобы удовлетворялось неравенство $\prod_{i=1}^{k} x_i < e^{-a}$. Число $k-1$ представляет собой случайную величину η, принадлежащую совокупности, распределенной по закону Пуассона с математическим ожиданием a. Если неравенству удовлетворяет первое из равномерно распределенных чисел, то $\eta = 0$.

В таблице 1.2 приведен ряд неуниверсальных методов для генерации случайных величин с указанными законами распределения, основанных на использовании нормально распределенных величин. Здесь z_i – нормально распределенная величина с математическим ожиданием, равным 0, и среднеквадратическим отклонением, равным 1, а ν – число степеней свободы. При этом для каждого закона указано, какими должны быть математическое ожидание и дисперсия случайной величины.

Таблица 1.2

Закон распределения	**Формула**	**M**	**Дисперсия**
Хи-квадрат	$u=\sum_{i=1}^{\nu} z_i^2$	ν	2ν
Стьюдента	$T=\frac{z}{\sqrt{u/\nu}}$ или $T=\frac{z}{\sqrt{\sum_{i=1}^{\nu} z_i^2/\nu}}$	0	$\frac{\nu}{\nu-2}$
F-распределение	$W=\frac{u_1/\nu_1}{u_2/\nu_2}$ или $W=\frac{\sum_{i=1}^{\nu_1} z_i^2/\nu_1}{\sum_{i=1}^{\nu_2} z_i^2/\nu_2}$	$\frac{\nu_2}{\nu_2-2}$	$\frac{2\nu_2^2(\nu_1+\nu_2-2)}{\nu_1(\nu_2-2)^2(\nu_2-4)}$

Рассмотрим пример построения имитационного алгоритма работы стохастической системы.

Пример 1.5

Начальник пожарной охраны обнаружил, что число пожаров за сутки следует распределению Пуассона со средним значением 4 пожара в сутки.
Изучив данные по прежним пожарам, он нашел, что в 70% случаев для тушения потребовалась только одна пожарная машина, а время, необходимое для ликвидации пожара, имеет нормальное распределение с m=2.5 часа и $\delta = 0.5$ часа. В остальных 30% случаев нужны были 2 пожарные машины, а время для ликвидации этих пожаров распределено нормально с m=3.5 часа и $\delta = 1$час.

Предполагая, что необходимые пожарные машины всегда находятся под рукой, необходимо определить, сколько часов в среднем они бывают нужны каждые сутки (построить алгоритм).

При построении алгоритма считаем, что генераторы случайных чисел x_i, равномерно распределенных на интервале [0,1], существуют.

Сначала построим алгоритм генератора случайных чисел для моделирования числа пожаров в сутки ***gen1***. Для генерации случайных чисел с Пуассоновским распределением воспользуемся не универсальным методом генерации:

1. p:=1; k:=0;
2. генерируем равномерно распределенное случайное число x_i;
3. p:=p* x_i; k:=k+1;
4. если $p > \exp(-4)$, то переход к п.2;
5. выход: k-1.

Теперь построим генератор нормально распределенных случайных чисел ***gen2*** для моделирования времени, необходимого для ликвидации пожара с входными параметрами m и δ. Для этого также воспользуемся не универсальным методом генерации:

1. генерируем 12 равномерно распределенных случайных чисел x_i;
2. суммируем $y = \sum_{i=1}^{12} x_i$;
3. вычисляем $N := \delta \cdot (y - 6) + m$;
4. выход: N.

Для определения оценки среднего времени промоделируем работу системы в течение 100 дней. Общий алгоритм работы системы будет выглядеть следующим образом:

1. $N = 0$ {счетчик дней};
2. $T = 0$ {общее время тушения пожаров};
3. $N = N + 1$;
4. $K = gen1$ {определяем количество пожаров сегодня};

5. $i = 0$;

6. $i = i + 1$;

7. генерируем x_i {для определения типа пожара};

8. если $x_i \le 0.7$, то $t = gen2(2.5,0.5)$ {требовалась 1 машина}

иначе $t = gen2(3.5,1) \cdot 2$ {требовались 2 машины};

9. $T = T + t$ {добавляем сгенерированное время тушения пожара};

10. если $t < k$, перейти к п. 6 {цикл по количеству пожаров сегодня};

11. если $N < 100$, перейти к п. 3 {моделируем в течение 100 лней};

12. $T = \frac{T}{100}$ {оценка среднего времени тушения пожаров в день}

1.4. Идентификация закона распределения

Еще одна проблема, возникающая при исследовании и моделировании стохастических систем, — как проверить совместимость экспериментальных данных с некоторым теоретическим распределением. Иначе говоря, возникает вопрос: соответствует ли частота наблюдаемых выборочных значений той частоте, с которой они должны бы появляться при некотором вероятностном распределении, отвечающем определенному теоретическому закону. Если частота близка к теоретической, то в дальнейшем можно строить модель событий на основе теоретического распределения, используя любой метод генерации случайных величин из рассмотренных выше.

Если собрать достаточное количество данных натурного эксперимента, то можно построить и проанализировать гистограмму. Визуальное сравнение гистограммы и функции плотности распределения позволяет лишь предположить, к какому теоретическому распределению нужно «подогнать» экспериментальное. После того как аналитик подобрал одно или несколько теоретических распределений, близких к экспериментальным данным, ему

следует определить параметры распределения, с тем, чтобы подвергнуть их проверке по статистическим критериям. Если предполагаемое распределение является функцией двух параметров, последние обычно удается оценить на основе выборочного среднего и выборочной дисперсии.

Когда экспериментальные данные разбиты на группы (гистограмма), среднее и дисперсию можно вычислить по формулам:

$$\overline{X} = \frac{\sum_{i=1}^{k} M_i F_i}{n}, \quad S^2 = \frac{\sum_{i=1}^{k} M_i^2 F_i - n\overline{X}^2}{n-1},$$

где n — полный объем выборки, $n = \sum_{i=1}^{k} F_i$;

k — число групп (интервалов выборки);

M_i — средняя точка i-го интервала или (для дискретных данных) значение i-й группы;

F_i — частота появления i-го интервала.

После получения оценок параметров распределения проверяются гипотезы принадлежности экспериментальных данных тому или иному закону распределения. Для проверки используют критерии согласия, два из которых мы рассмотрим на примере.

Оценка по критерию согласия «хи-квадрат»

Для статистической оценки гипотезы о том, что совокупность эмпирических или выборочных данных незначительно отличается от той, которую можно ожидать при некотором теоретическом распределении, можно использовать критерий χ^2, который определяется формулой:

$$\chi^2 = \sum^{k} \frac{(f_0 - f_e)^2}{f_e},$$

где f_0 — наблюдаемая частота для каждого интервала;

f_e — ожидаемая частота для каждого интервала;

$\sum$ — предсказанная теоретическим распределением сумма по всем группам или интервалам.

Если $\chi^2 = 0$, то наблюдаемые и теоретические значения частот точно совпадают. Чем больше χ^2, тем больше расхождение между наблюдаемыми и ожидаемыми значениями. При $\chi^2 > 0$ статистика сравнивается с табличным значением (табулирована для различных чисел степеней свободы и различных уровней доверительной вероятности $1-\alpha$).

Высказывается нулевая гипотеза H_0 о том, что между наблюдаемым и ожидаемым распределениями нет значительных расхождений. Если $\chi^2_{\text{расч}} > \chi^2_{\text{табл}}$, то гипотеза H_0 отвергается.

Ограничения при использовании критерия:

1) необходимо использовать абсолютные значения частот (не относительные);

2) значения наблюдений для каждого интервала должны быть больше или равны 5;

3) число степеней свободы $\nu = k - 1 - m$, где k — число интервалов или групп, m — число параметров, определяемых опытным путем для вычисления ожидаемых значений частот.

Пример 1.6

Пусть мы проверяем, является ли частота телефонных запросов Пуассоновским распределением.

Для определения расчетного значения χ^2 построим таблицу:

Число запросов	Число одноча-совых интер-валов с чис-лом запросов f_0	Относи-тельная частота	Теоретиче-ская веро-ятность $P(n)$	Ожидае-мая часто-та $f_e = P(n)x$	$\frac{(f_0 - f_e)}{f_e}$
0	315	0,619	0,571	291	1,98
1	142	0,279	0,319	162	2,47
2	40	0,078	0,089	45	0,56
3	9	0,018	0,017	9	
4	2 } 12	0,004	0,003	1 } 11	} 0.09
5	1	0,002	0,001	1	
Сумма	509	1,000	1,000	509	5,1

Здесь $\bar{x} = 0{,}5147$, $s^2 = 0{,}6007$. Так как для Пуассоновского закона должно выполняться $\bar{x} = s^2$, то можно уточнить оценку параметра: $\lambda = \frac{\bar{x} + s^2}{2} = 0{,}5577$.

Теоретическая вероятность наступления n событий определяется по формуле $P\{x = n\} = \frac{\lambda^n e^{-\lambda}}{n!}$.

Для доверительной вероятности $0{,}95$ и числа степеней свободы $4-1-1=2$ имеем $\chi^2_{\text{табл}} = 5{,}99$. Значит $\chi^2_{\text{расч}} < \chi^2_{\text{табл}}$ и, следовательно, гипотеза принимается.

Критерий Колмогорова-Смирнова

Еще один используемый критерий, который применяется, когда проверяемое распределение непрерывно и известны среднее и дисперсия совокупности, это критерий Колмогорова-Смирнова. Здесь проверка осуществляется путем задания интегральной функции, следующей из

теоретического распределения, и ее сравнения с интегральной функцией распределения эмпирических данных.

Сравнение основывается на выбранной группе, в которой экспериментальное распределение имеет наибольшее абсолютное отклонение от теоретического. Эта абсолютная разность сопоставляется с критическим значением $D_{крит} = \frac{\lambda}{\sqrt{N}}$, где λ – табличное значение функции Колмогорова при заданной доверительной вероятности.

Пример 1.6

Возьмем данные из примера 1.5 и рассчитаем абсолютную разность по всем интервалам.

Расчет абсолютных разностей между экспериментальным и теоретическим распределением

	I	II	III	IV	V	VI
Число запросов	f_0	Наблюдаемая вероятность	Теоретическая $P(n)$	Интегральная вероятность II	Интегральная вероятность III	Абсолютная разность IV-V
0	315	0,619	0,571	0,619	0,571	0,048
1	142	0,279	0,319	0,838	0,890	0,008
2	40	0,078	0,089	0,976	0,979	0,003
3	9	0,018	0,017	0,994	0,996	0,002
4	2	0,004	0,003	0,998	0,999	0,001
5	1	0,002	0,001	1,000	1,000	0

При $\alpha = 0{,}05$ и $n = 509$ $D_{\text{крит}} = \frac{1{,}36}{\sqrt{509}} = 0{,}0603$. Максимальная абсолютная разность $0{,}048$ меньше $D_{\text{крит}}$ — гипотеза принимается.

При малых выборках критерий χ^2 не применим, здесь предпочтительнее критерий Колмогорова-Смирнова. Напротив, если объем выборки велик, предпочтителен критерий χ^2 (при $n \geq 100$ данный критерий является достаточно мощным). Чем больше число интервалов, тем точнее выводы. Для критерия Колмогорова-Смирнова можно относить каждое наблюдение к отдельной группе — тогда он работает эффективнее.

Вопросы для самопроверки

1. Какая последовательность называется базовой при статистическом моделировании?
2. Почему при машинном моделировании генерируемые последовательности чисел называются псевдослучайными?
3. Каким образом моделируются случайные события?
4. В чем преимущества и недостатки различных способов генерации последовательностей псевдослучайных чисел?
5. Какие методы генерации случайных величин с заданным законом распределения Вы знаете?
6. Каковы их недостатки?

2. ЯЗЫК МОДЕЛИРОВАНИЯ СИСТЕМ GPSS

2.1. Программные средства имитационного моделирования

Успех или неудача любого исследования системы на программно реализуемой модели прежде всего зависит от правильности моделирующего алгоритма, а также от эффективности программы. Поэтому при машинной реализации модели большое значение имеет вопрос правильного выбора языка моделирования.

Каждый язык моделирования должен отражать определенную структуру понятий для описания широкого класса явлений. При выборе языка необходимо учитывать большую роль уровня его проблемной ориентации. Высокий уровень проблемной ориентации языка моделирования значительно упрощает программирование моделей и анализ результатов экспериментов.

Определим основные моменты, характеризующие качество языка моделирования:

- удобство описания процесса функционирования системы;
- удобство ввода исходных данных моделирования;
- удобство варьирования структуры, алгоритмов и параметров модели;
- реализуемость статистического моделирования;
- эффективность анализа и вывода результатов моделирования;
- простота отладки и контроля работы моделирующей программы;
- доступность восприятия и использования языка.

Языки общего назначения (ЯОН) или алгоритмические языки (такие как Паскаль, Си и пр.) предназначены для формального представления и решения задач любого типа. В отличии от ЯОН, любой язык моделирования представляет собой процедурно-ориентированный язык, обладающий специфическими чертами. Основные языки моделирования разрабатывались в качестве программного обеспечения имитационного подхода к изучению процесса функционирования определенного класса систем.

Целесообразность использования языков имитационного моделирования вытекает из трех причин:

1) удобство программирования модели системы;

2) концептуальная направленность языков на класс систем, необходимая на этапе построения модели системы и выбора общего направления исследований в планируемом машинном эксперименте. Практика моделирования систем показывает, что именно использование языков имитационного моделирования во многом определило успех имитации как метода экспериментального исследования сложных реальных объектов;

3) языки моделирования позволяют описывать системы в терминах, разработанных на базе основных понятий имитации, что образует связь между отдельными разработками в единых способах описания.

При создании системы моделирования необходимо решить вопрос о синхронизации процессов в модели: для одновременно протекающих событий должна быть реализована псевдопараллельная организация имитируемых процессов. Это является основной задачей монитора моделирования, который выполняет следующие функции: управление процессами (согласование системного и машинного времени) и управление ресурсами (выбор и распределение в модели средств моделируемой системы).

При разработке моделей систем возникает целый ряд специфических трудностей. В связи с этим, в языках имитационного моделирования должен быть предусмотрен набор таких программных средств и понятий, которые не встречаются в универсальных языках программирования. К таким понятиям относятся:

1) **Совмещение.** Параллельно протекающие в реальных системах процессы представляются с помощью последовательно работающих компьютеров, следовательно, возникает необходимость во введении понятия системного времени.

2) **Размер.** Большие по объему модели требуют динамического распределения памяти, когда компоненты модели появляются в оперативной памяти или покидают ее в зависимости от текущего состояния. Отсюда возникает необходимость обеспечения блочной конструкции модели.

3) **Изменения.** Пространственная конфигурация динамических систем претерпевает изменения во времени, поэтому в языках имитационного моделирования предусматривают обработку списков, отражающих изменения состояний процесса функционирования системы.

4) **Взаимосвязанность.** При моделировании процесса функционирования системы необходимо учитывать сложные условия свершения событий, поэтому в языки моделирования включают логические возможности и понятия теории множеств.

5) **Стохастичность.** Для моделирования случайных событий и процессов в язык моделирования включают процедуры генерации случайных чисел, на основе которых можно получать стохастические воздействия на систему.

6) **Анализ.** Для анализа результатов моделирования необходимы статистические характеристики процесса функционирования модели, поэтому в языках моделирования предусмотрены различные способы статистической обработки и анализа результатов.

В настоящее время существует несколько десятков развитых языков имитационного моделирования. Язык GPSS, рассматриваемый в данном учебном пособии, является одним из наиболее распространенных языков моделирования, используемых как для исследований систем, так и для учебных целей в различных университетах мира.

2.2. Общие сведения о языке GPSS

Язык моделирования GPSS (General Purpose System Simulation) разработан фирмой IBM в США и с 1962 года входил в стандартное математическое обеспечение машин серии IBM 360/370. Язык GPSS получил наиболее широкое распространение по сравнению с другими языками моделирования. Он включен в учебные курсы ВУЗов по моделированию систем у нас в стране и изучается в аналогичных курсах во многих колледжах и университетах США и других стран. В данном учебном пособии рассматривается одна из версий языка GPSS.

Язык GPSS ориентирован на решение задач статистического моделирования на ЭВМ процессов с дискретными событиями. Такими процессами описывается, прежде всего, функционирование систем массового обслуживания произвольной структуры и сложности: систем обработки данных, систем транспорта и связи, технологических процессов, предприятий торговли, а также функционирование вычислительных систем и разного рода автоматизированных систем.

Язык основан на схеме *транзактов* (сообщений). Под транзактом понимается формальный объект, который "путешествует" по системе (перемещается от блока к блоку), встречая на пути всевозможные задержки, вызванные занятостью тех или иных единиц оборудования. Транзакты имеют прямую аналогию с заявками в системах массового обслуживания. В качестве транзакта может выступать программа обработки информации, телефонный вызов, покупатель в магазине, отказ системы при исследовании надежности и т.д. Каждый транзакт обладает совокупностью параметров, которые называются атрибутами транзакта. В процессе имитации атрибуты могут меняться в соответствии с логикой работы исследуемой системы.

Язык GPSS — язык интерпретируемого типа, он связан с пошаговым выполнением операторов, называемых блоками. Совокупности блоков описывают функционирование самой моделируемой системы либо содержат информацию о порядке моделирования (о продвижении транзактов). Каждое продвижение транзакта (сообщения) является событием в модели. Комплекс программ, планирующий выполнение событий, реализующий функционирование блоков моделей, регистрирующий статистическую информацию о прохождении транзактов, называется *симулятором* [4]. Симулятор регистрирует время наступления каждого из известных на данный момент событий и выполняет их с нарастающей временной последовательностью. Симулятор обеспечивает отсчет модельного времени в принятых единицах, называемых *абсолютным условным временем*. С каждым сообщением связано *относительное условное время*, отсчет которого начинается при входе сообщения в моделируемую систему и заканчивается при выходе сообщения из системы. *Основными функциями* управляющих операторов/блоков языка являются:

1) создание и уничтожение транзактов,

2) изменение их атрибутов,

3) задержка транзактов,

4) изменение маршрутов транзактов в системе.

Основные группы объектов языка:

1) объекты, имитирующие единицы оборудования системы (устройство, память и логические переключатели);

2) статистические объекты (очередь, таблица),

3) вычислительные объекты (ячейка, арифметическая и логические переменные),

4) списки,

5) прочие объекты.

Дадим описание некоторых объектов.

Устройство имитирует единицу оборудования, которое может одновременно обрабатывать только один транзакт. Устройство аналогично обслуживающему прибору в системах массового обслуживания. Оно служит для моделирования таких средств обработки элементов потоков, как станки, рабочие, каналы связи и т.п. На устройствах можно реализовать самые различные дисциплины обслуживания транзактов, включающие учет требуемого времени обслуживания, значения приоритетов, возможности прерывания и т.д.

Память (накопитель) имитирует единицу оборудования, в которой может обрабатываться (храниться) несколько транзактов одновременно. Память позволяет легко моделировать средства обработки с ограниченной емкостью (стоянки автотранспорта, портовые причалы, складские помещения, конвейеры и т.п.).

Очередь - объект, связанный со сбором статистики о задержках, возникающих на пути прохождения транзакта. Чаще всего очередь помещают перед устройством либо памятью. Следует учитывать, что естественно образующиеся в процессе моделирования очереди транзактов обрабатываются симулятором автоматически, а описываемый объект языка служит лишь для обеспечения вывода на печать соответствующих статистических данных.

Таблица обеспечивает накопление в процессе моделирования статистики о каком-либо заданном случайном параметре модели. По окончании прогона модели эта статистика автоматически обрабатывается и выводится на печать, в частности, в виде таблицы относительных частот попадания значений случайного параметра (аргумента таблицы) в указанные частотные интервалы. Печатаются также среднее значение и среднее квадратичное отклонение аргумента.

Ячейки используются для записи, накопления и хранения численных значений различных входных и выходных параметров моделируемой системы. Эти значения могут быть использованы для организации счетчиков числа проходящих транзактов, для вывода значений варьируемых параметров модели, для временного хранения значений *стандартных числовых атрибутов* (СЧА). Значения ячеек всегда выводятся на печать.

Арифметическая переменная позволяет выполнить заданную последовательность арифметических операций над любыми СЧА модели для вычисления значения зависимого от них параметра.

Любая программа на GPSS связана с созданием транзактов, проведением их через последовательность блоков и уничтожением транзактов. При этом создание или генерация транзактов основывается на знании закономерностей информационных потоков, циркулирующих в моделируемой системе, а путь прохождения транзакта через блоки определяется спецификой работы оборудования исследуемой системы.

Вложить в рамки формальной схемы GPSS конкретное смысловое содержание, определяемое исследуемой системой — задача непростая: для этого необходимо знать как формализмы языка, так и логику работы моделируемой системы. Тем не менее, программирование на GPSS существенно облегчает пользователю процесс моделирования, сокращая и время чистого программирования (по сравнению с универсальными алгоритмическими языками), и время отладки программы.

2.3. Синтаксис языка

Алфавит. Алфавит языка GPSS состоит из латинских букв от *A* до *Z*, цифр от 0 до 9 и следующих специальных символов: $, #, *, +, -, /, (,), ', точка, запятая, пробел.

Идентификаторы. Они состоят из алфавитно-цифровых символов, причем первый символ должен быть буквой. Идентификаторы используются для формирования имен объектов и блоков. При формировании собственного имени необходимо помнить, что оно не должно совпадать с ключевым словом языка, поэтому рекомендуется использовать имена с количеством букв в начале имени не менее трех. Исключение составляют ячейки и атрибуты транзактов, которые могут обозначаться не только идентификаторами, но и просто числами.

Блоки/операторы. Каждый блок языка записывается в отдельной строке и имеет следующую структуру:

[метка] операция [операнды] [комментарии]. Каждое поле отделяется друг от друга пробелами, обязательным является только поле операции, остальные поля могут отсутствовать.

Метка является именем-идентификатором блока. Поле операндов может содержать от 1 до 7 подполей: *A, B, C, D, E, F, G*, содержимое которых отделяется друг от друга запятой. Для пропуска одного из подполей поля операндов ставится просто запятая, например: *A,,C*.

Комментарии, кроме поля комментариев, могут быть заданы отдельной строкой: любая строка, начинающаяся с символа "*" или ";", тоже будет комментарием. Поле комментария должно начинаться с символа ";".

Стандартные числовые атрибуты (СЧА). В процессе моделирования язык GPSS автоматически регистрирует и корректирует определенную информацию различных объектов, используемых в модели. Доступ к этой информации осуществляется с помощью СЧА, которые однозначно определяют статус объектов модели. СЧА меняются в процессе имитации, изменить их может как симулятор, так и пользователь. Для указания конкретного объекта, по которому необходимо получить требуемую информацию, за именем СЧА должно следовать

числовое или символьное имя этого объекта. Если используется символьное имя, то между СЧА и именем объекта ставится знак $.

В таблице 2.1 приведены некоторые СЧА основных объектов языка. Здесь каждый СЧА обозначается либо <имя СЧА> i, либо

<имя СЧА>$ <имя объекта>,

где i обозначает номер объекта.

Таблица 2.1

Объект	***СЧА***	***Назначение***
Блок	N$<имя блока>	Число транзактов, вошедших в блок с указанным именем
	W$<имя блока>	Число транзактов, находящихся в указанном блоке
Генераторы случайных чисел	Rni	Случайное число в диапазоне 0-999. При использовании СЧА в качестве аргумента функции представляются действительными числами в диапазоне 0.- 0.999999
Транзакт	Pi	Значение i-го параметра
	PR	Значение приоритета
Память	S$<имя памяти>	Текущее содержимое памяти
	R$<имя памяти>	Свободный объем памяти
	SA$<имя памяти >	Среднее число занятых единиц памяти
	SC$<имя памяти >	Число транзактов, вошедших в память с начала моделирования
	SE$<имя памяти >	Память пуста? Если да – 1, нет - 0
	SF$<имя памяти >	Память полна? Если да – 1, нет - 0
	SM$<имя памяти >	Максимальное число занятых единиц памяти

Продолжение таблицы 2.1

Объект	***СЧА***	***Назначение***
	ST$<имя памяти >	Среднее время нахождения транзакта в памяти
Очередь	Q$<имя очереди> QA$<имя очереди> QC$<имя очереди> QM$<имя очереди> QX$<имя очереди> QZ$<имя очереди> QT$<имя очереди>	Текущая длина очереди Средняя длина очереди Число транзактов, вошедших в очередь с начала моделирования Максимальная длина очереди Среднее время нахождения транзакта в очереди без нулевых входов Количество нулевых входов Среднее время нахождения транзакта в очереди
Устройство	F$<имя устройства> FC$<имя устройства> FT$<имя устройства>	Состояние устройства: занято – 1, свободно – 0 Число транзактов, вошедших в устройство с начала моделирования Среднее время занятия транзактом устройства
Переменные	V$<имя переменной>	Значение арифметической переменной
Ячейки	X$<имя ячейки> или Xi	Значение ячейки
Функции	FN$<имя функции>	Значение функции

Мнемокоды. В некоторых блоках языка требуется указывать состояние объектов, для этого используются следующие коды:

Состояние объекта	Мнемокод
Память: пуста	SE
не пуста	SNE
заполнена	SF
не заполнена	SNF
Устройство: свободно	NU
занято	U
Логический переключатель:	
включен	LS
выключен	LR

2.4. Блоки языка GPSS

2.4.1. Создание и уничтожение транзактов

Генерирование транзактов — GENERATE. Этот блок генерирует поток сообщений - транзактов, поступающих в систему. Программа составляется с учетом того, что в этот блок не могут входить транзакты. В простых программах это обычно первый блок, временные интервалы между поступающими в систему транзактами определяются содержимым поля операндов. Подполя:

A — среднее время между поступлениями транзактов в систему (по умолчанию равно 1);

B — модификатор времени;

C — начальная задержка (время появления первого транзакта);

D — общее число транзактов, которое должно быть сгенерировано этим блоком (по умолчанию — неограниченное число транзактов);

E — приоритет транзакта, может принимать значения от 0 до 127. Приоритет возрастает в соответствии с номером (по умолчанию равен 0).

В поле B может быть модификатор двух типов: модификатор-интервал и модификатор-функция. Если задан модификатор-интервал (просто число), то для каждого временного интервала поступления транзактов длительность определяется как значение случайной величины, равномерно распределенной на интервале $[A-B, A+B]$.

Значение параметров A и B могут задаваться как константами, так и любым СЧА, за исключением СЧА параметра транзакта (эта величина в момент генерации транзакта еще не определена).

Например, блок GENERATE 10,5 будет генерировать транзакты через интервалы времени, длительность каждого из которых выбирается случайно в пределах от 5 до 15. Каждое из этих значений будет выбираться с одинаковой вероятностью. Таким образом, блок генерирует случайный поток транзактов, в котором время между транзактами равномерно распределено в диапазоне $A \pm B$ и имеет среднее значение A.

При использовании модификатора-функции интервал времени между транзактами определяется произведением содержимого полей A и B. Функция определяется специальными блоками языка, которые будут рассмотрены чуть позже.

В программе может быть несколько блоков GENERATE. Все эти блоки работают параллельно и начинают генерировать транзакты одновременно с момента начала моделирования.

Необходимо помнить, что смысл единицы времени в языке GPSS (секунда, минута, час, день и т.д.) закладывает пользователь, поэтому при написании программы необходимо все операнды, связанные со временем, привести к единому масштабу.

Примечания:

- время не может быть отрицательной величиной;

- в обязательном порядке должно быть задано либо поле A, либо поле D.

Блок уничтожения транзактов — TERMINATE. Обычно для простых программ это последний блок программы. Транзакты, попадающие в этот блок,

уничтожаются и больше не участвуют в процессе моделирования. Никаких других действий этот блок не выполняет, если единственный возможный операнд *A* в блоке не задан. Если же операнд *A* задан, то его значение вычитается из содержимого блока транзактов. Операнд *A* может принимать только положительное целочисленное значение.

Первоначальная величина счетчика устанавливается специальным управляющим блоком START и пишется в поле *A* этого блока. Когда в результате входа очередного транзакта в блок TERMINATE значение счетчика становится нулевым или отрицательным, симулятор прекращает моделирование и передает управление программе вывода, которая распечатывает накопленные симулятором данные о модели.

Например:

```
TERMINATE  1
START  100
```

через программу модели пропускается 100 транзактов.

В программе должен быть хотя бы один блок TERMINATE с заданным операндом *A*.

Если в программе несколько блоков TERMINATE, то обычно операнд *A* задается только в одном блоке; чаще всего — в блоке, относящемся к имитатору интервала времени моделирования (таймеру).

```
GENERATE  480
TERMINATE  1
START  1
```

Таймер взаимодействует только с блоком START и никак не связан с содержательной стороной остальных фрагментов модели. Таймер служит для задания времени моделирования.

2.4.2. Задержка транзактов в блоках

Блок ADVANCE предназначен для задержки транзактов на определенные интервалы модельного времени.

Обязательный операнд A задает время задержки транзакта в блоке ADVANCE. Необязательный операнд B является модификатором-функцией или модификатором-интервалом. Значение операнда B используется здесь для модификации значения операнда A также, как и в блоке GENERATE.

Любой транзакт входит в блок ADVANCE беспрепятственно. В нем транзакт задерживается на период модельного времени, величина которого определяется операндами A и B. После этого транзакт направляется к следующему блоку.

Например, в блоке

```
ADVANCE  10
```

транзакт будет задержан на 10 единиц модельного времени.

В блоке

```
ADVANCE  10,P1
```

транзакт будет задерживаться на случайное время, выбранное из диапазона $10 \pm$ значение первого параметра транзакта (следует помнить, что значение первого параметра при этом не должно превышать 10, т.к. время не может быть отрицательным).

Рассмотрим суммарную задержку в блоках

```
ADVANCE  10,10
ADVANCE  10,10
ADVANCE  10,10
ADVANCE  10,10
ADVANCE  10,10
ADVANCE  10,10
```

Задержка в каждом из них имеет равномерное распределение вероятностей на интервале (0,20). Следовательно, ее среднее значение составляет $M = 20 \cdot (1/2) = 10$; дисперсия $D = 20 \cdot (1/2)$. Поэтому сумма шести таких задержек имеет среднее значение $6 \cdot M = 60$ и среднее квадратическое отклонение $\sqrt{6 * D} \approx 14$. По центральной предельной теореме теории вероятностей заключа-

ем, что закон распределения суммарной задержки приблизительно нормальный. Поэтому ни в коем случае нельзя заменять эти пять блоков на один

ADVANCE 50,50 ,

т.к. этот блок будет определять задержку как равномерно распределенную величину.

2.4.3. Работа с устройствами

Блок SEIZE - занять устройство. При входе транзакта в блок SEIZE выполняется операция занятия устройства, имя которого задается операндом *A* блока SEIZE. Занятие устройства транзактом выполняется следующим образом. Когда транзакт направляется из какого-нибудь блока в блок SEIZE, симулятор проверяет, свободно ли соответствующее устройство. Если оно не свободно, транзакт не может войти в этот блок. Он остается в предыдущем блоке до тех пор, пока устройство не освободится. Если же устройство свободно, то транзакт передвигается в блок SEIZE, занимает устройство и в тот же момент времени направляется к следующему за SEIZE блоку.

Блок RELEASE - освободить устройство. При входе транзакта в блок RELEASE происходит освобождение устройства, имя которого задается операндом *A*.

При составлении моделей пользователь должен соблюдать правило: освободить устройство может только тот транзакт, который его занимает. Если транзакт попытается освободить устройство, занятое другим транзактом, симулятор прервет выполнение модели и выдаст сообщение об ошибке.

В момент освобождения устройства должен быть решен вопрос о том, какой из задержанных транзактов (перед блоком SEIZE) имеет право первым занять устройство. Этот вопрос решается следующим образом: когда транзакты задерживаются перед блоком SEIZE, они регистрируются симулятором в списке, где упорядочиваются по приоритетам: любой транзакт с более высоким приоритетом ставится впереди транзакта, имеющего более низкий приоритет. Если у двух транзактов одинаковые приоритеты, то они упорядочиваются меж-

ду собой по времени прихода: впереди ставится транзакт, который раньше обратился к устройству. В момент освобождения устройства его занимает тот из задержанных транзактов, который находится в списке первым. Транзакт может занимать любое число устройств. Освобождать занятые устройства транзакт может в любом порядке.

Пример 2.1

Посетители приходят в кассу кинотеатра через 20 ± 10 с, знакомятся в течение 15 ± 15 с обстановкой и занимают очередь. Каждый посетитель приобретает у кассира билеты в течение 20 ± 5 с в зависимости от числа билетов. Построить модель работы кассы кинотеатра в течение четырех часов.

```
GENERATE   20,10    ;приход посетителей
ADVANCE    15,15    ;знакомство с обстановкой
SEIZE      KASS     ;обращение к кассиру
ADVANCE    20,5     ;покупка билета
RELEASE    KASS     ;освобождение кассира
TERMINATE           ;уход посетителя
GENERATE   1440     ;таймер
TERMINATE  1
START      1
```

В результате выполнения модели на печать автоматически выводится информация о наличии транзактов в каждом блоке на момент завершения моделирования, а также информация обо всех устройствах, к которым производилось обращение в модели. Формат выводимых данных приведен в приложении 1.

2.4.4. Сбор статистических данных с помощью очередей

Некоторые виды статистических данных накапливаются симулятором автоматически. Другие виды данных могут быть получены с помощью специаль-

ных блоков. При необходимости сбора данных по задержке транзактов перед блоками занятия устройства или памяти используются блоки QUEUE и DEPART.

Блок QUEUE - поставить в очередь. При входе транзакта в этот блок он ставится в очередь, имя которой задается операндом *A*. В начальный момент времени, когда очередь пуста, ее длина равна нулю. В момент входа транзакта в блок QUEUE ее длина увеличивается на величину, указанную в поле *B*. Если операнд *B* пуст, то длина очереди увеличивается на единицу.

Блок DEPART - вывести из очереди. При входе транзакта в блок DEPART длина очереди, имя которой задается операндом *A*, уменьшается на величину, указанную в операнде *B*. При использовании пустого поля *B* в блоках QUEUE и DEPART длина очереди в каждый момент времени соответствует текущему числу транзактов в этой очереди. Транзакты могут проходить любое число блоков QUEUE и DEPART с произвольными значениями полей *A* и *B*, чередующихся в любом порядке.

Необходимо помнить, что данные блоки не влияют на реальное образование очередей транзактов, а служат только для сбора статистических данных. Поэтому пользователь должен следить за правильным расположением этих блоков, чтобы не получать отрицательные длины образуемых очередей. Симулятор только подсчитывает статистику по очередям и не считает за ошибку отрицательные длины очередей.

Пример 2.2

Изменим модель, построенную в примере 2.1 таким образом, чтобы получить информацию об очереди, образующейся перед кассой.

```
GENERATE    20,10
ADVANCE     15,15
QUEUE       OCH     ; включение в очередь
SEIZE       KASS    ; обращение к кассиру
DEPART      OCH     ; выход из очереди
ADVANCE     20,5
```

```
RELEASE     KASS
TERMINATE
GENERATE    1440    ; таймер
TERMINATE   1
START       1
```

В этой модели момент включения каждого транзакта в очередь OCH совпадает с моментом его обращения к блоку SEIZE , т.к. блок QUEUE выполняется в модельном времени мгновенно. Каждый транзакт находится в очереди до тех пор, пока не займет устройство KASS. Момент занятия устройства совпадает с моментом выхода транзакта из очереди. В данном случае очередь OCH имеет естественную интерпретацию как очередь посетителей к кассиру, а длина очереди интерпретируется как число посетителей в очереди.

При наличии в модели очередей симулятор выдает статистику по очередям. Формат выводимых данных приведен в приложении 1.

2.4.5. Функции

При использовании в блоках GENERATE и ADVANCE поля B в качестве модификатора функции, саму функцию необходимо описать специальным блоком языка FUNCTION.

В поле метки данного блока стоит имя функции (поле метки в данном случае является обязательным). В операнде A блока FUNCTION указывается аргумент функции, а в операнде B — тип функций и количество пар аргументов и значений.

Аргумент функции задается с помощью СЧА. Чаще всего в качестве аргумента используются датчики случайных чисел RN1, RN2, RN200.

Существуют следующие типы функций: C, D, E, L, M. Функции типа C — непрерывны (аргумент X и значение Y функции задаются типами integer и real), типа D — дискретны (Y кроме integer и real может принимать значение «имя»). Например, C 12 означает, что функция непрерывна и для ее описания

будет использоваться 12 пар аргументов-функций. Тип E – дискретная функция со значением Y, заданным с помощью СЧА. Типы L и M – функции со списком, здесь для L значение функции интерпретируется как номер элемента списка, для M аргумент функции является элементом списка.

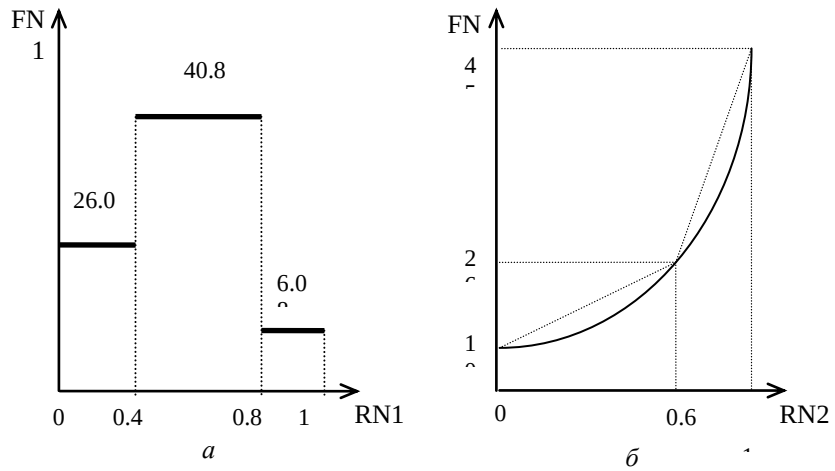
При описании любой из функций с помощью языка GPSS происходит интерполяция. Для дискретных функций — это кусочно-постоянная интерполяция, для непрерывных — линейная интерполяция. Координаты функции, задаваемые парами, являются узлами интерполяции.

За блоком описания функции FUNCTION всегда следует блок задания функции, в котором задаются координаты и значения функции. Каждая пара чисел координата-значение отделяется друг от друга слэджем, пробелы недопустимы. В паре аргумент отделяется от значения функции запятой.

Необходимо помнить, что аргумент функции может принимать только неотрицательные значения. Поле комментариев в данном блоке не используется.

Например, функция, график которой показан на рис. 2.1, *а*), описывается на языке GPSS следующим образом:

```
FUNC1  FUNCTION  RN1, D3
.4,26.0/.8,40.8/1,6.0
```



а *б*

Рис. 2.1. Графики дискретной (*а*) и непрерывной (*б*) функций

Непрерывная функция, показанная на рис. 2.1, *б*):

```
FUNC2  FUNCTION  RN2, C3
0,10/.6,26/1,45
```

Блоки описания и задания функции располагаются в начале программы, до первого блока GENERATE. Координаты точек функции записываются как числа с фиксированной точкой либо именем (для значения функции типа D).

В языке существует 3 датчика равномерно распределенных случайных чисел, которые обозначаются RN1, RN2, RN200. Эти датчики выдают равновероятные целочисленные значения из диапазона 0 - 999. Если датчик используется в качестве аргумента функции, то он выдает вещественные числа в диапазоне 0 - 1.

Для генерации случайных величин, распределенных по экспоненциальному закону, можно использовать встроенное вероятностное распределение, которое описывается выражением

```
EXPONENTIAL(A,B,C)
```

и при обращении к нему заключается в скобки.

Здесь поле A определяет номер датчика случайных чисел, поле B – сдвиг среднего и поле C – сжатие функции. В результате значение среднего определяется суммой полей B и C, а значение дисперсии равно квадрату поля C.

Пример 2.3

В аэропорту производится регистрация пассажиров перед посадкой в самолет. На регистрацию подходят отдельные пассажиры через каждые 20±10 с, либо туристические группы через каждые 60±20 сек. При этом туристические группы обслуживаются вне очереди. Время обслуживания подчинено экспоненциальному закону и равно в среднем для отдельных пассажиров — 15 сек, для туристических групп — 25 сек. Промоделировать работу отдела регистрации, изучив статистику по очереди за 2 ч.

```
EXP  FUNCTION  RN1,C6
0,0/.1,.1/.2,.2/.5,.69/.8,1.6/1,8
GENERATE  20,10              ; приход отдельных пассажиров
QUEUE     LIN                ; включение в очередь
```

```
SEIZE          REG
DEPART         LIN              ; выход из очереди
ADVANCE    15,FN$EXP            ; регистрация пассажира
RELEASE        REG
TERMINATE                       ; уход пассажира
GENERATE   60,20,,,1            ; приход туристической группы
QUEUE          LIN
SEIZE          REG
DEPART         LIN
ADVANCE    25,FN$EXP            ; регистрация группы
RELEASE        REG
TERMINATE                       ; уход группы
; таймер
GENERATE   720
TERMINATE 1
START          1
```

Здесь отдельные пассажиры и туристические группы встают в одну и ту же очередь и обслуживаются одним регистратором. Внеочередность обслуживания групп в модели обеспечивается заданием приоритета для транзактов, имитирующих туристические группы.

В программе три самостоятельных сегмента, каждый из которых начинается блоком GENERATE и заканчивается блоком TERMINATE. Они могут быть поставлены в программе в любом порядке. При этом процесс моделирования останется неизменным: все блоки GENERATE работают параллельно.

В примере экспоненциальная функция распределения описана первыми двумя строками программы. Можно пользоваться встроенной функцией, тогда описание функции можно опустить, а первый блок ADVANCE, например, будет выглядеть следующим образом:

```
ADVANCE    (EXPONENTIAL(1,0,15))    ;регистрация пассажира
```

2.4.6. Изменение маршрутов сообщений

Блок TRANSFER позволяет осуществлять безусловные, статистические и условные переходы. Тип перехода определяется в операнде *A*, направление перехода - в операндах *B*, *C* и *D*.

В режиме *безусловного перехода* операнд *A* в блоке пуст. Все транзакты переходят к блоку, указанному в поле *B*. Например:

```
TRANSFER ,NEXT
```

Если блок, к которому направляется транзакт, в текущий момент системного времени не может его принять (например, блок SEIZE), то транзакт остается в блоке TRANSFER и повторяет попытку перехода при каждом пересчете системного времени симулятором.

Если в поле *A* блока TRANSFER записана десятичная дробь, начинающаяся точкой, то блок работает в режиме *статистического перехода*. Здесь десятичная дробь определяет вероятность перехода транзакта к блоку, имя которого указывается в поле *C*. При этом поле *B* пустое. С вероятностью (1–<*A*>) транзакт переходит к блоку, следующему за блоком TRANSFER.

Если оба блока заняты, то транзакт остается в блоке TRANSFER и повторяет попытку перехода к выбранному ранее блоку при каждом изменении системного времени.

С помощью этого блока можно промоделировать, например, выбор покупателями в магазине одного из двух отделов, если известно, что половина покупателей направляется в 1-й отдел, а вторая половина - во 2-й отдел:

```
TRANSFER        .5,,OTD2
OTD1  SEIZE     PROD1
        .
        .
OTD2  SEIZE     PROD2
```

При моделировании статистических переходов необходимо обращать внимание на следующий момент: если требуется разбить поток заявок более, чем на два, необходимо внимательно рассматривать вероятности статистических

переходов в модели. В этом случае, если вы используете несколько блоков статистического перехода TRANSFER, для всех блоков, кроме первого, необходимо пересчитывать вероятности переходов с учетом того, что часть заявок уже была перенаправлена к другому блоку.

Например, известно, что поток покупателей разбивается на три потока (три отдела в магазине) следующим образом: 50% покупателей направляется в 1-й отдел, 20% - во второй и 30% - в третий. Тогда перенаправление транзактов будет происходить следующим образом:

```
TRANSFER    .5,,OTD1
TRANSFER    .4,,OTD2   ; 20% от оставшихся 50% - это 40%
TRANSFER    ,OTD3      ; все остальные – в 3-й отдел
```

Ситуация значительно усложняется, если проценты распределения заявок равны, например 17, 47 и 36 (теперь надо будет рассчитать долю 47 от оставшихся 100-17=83, а это 0.566265…). Чтобы упростить моделирование такой ситуации, можно построить дискретную функцию типа D, значением которой будут имена блоков, к которым должны направляться потоки заявок. Такая функция строится по принципу моделирования группы событий (см. главу 1).

Для распределения заявок на 17, 47 и 36 процентов можно построить следующую функцию:

```
PEREX FUNCTION RN1,D3
.17,OTD1/.64,OTD2/1,OTD3
```

а затем обращаться к функции в блоке безусловного перехода:

```
TRANSFER ,FN$PEREX
```

Условный переход. Режим условного перехода определяется мнемокодом, заданным в поле *A*. Рассмотрим различные режимы.

Если в поле *A* определено значение BOTH, то транзакт первоначально направляется к блоку, имя которого определено в поле *B*. Если переход невозможен (например, занято устройство), то делается попытка перейти к блоку, чье имя определено в поле *C*. Если оба блока заняты, то транзакт остается в блоке

TRANSFER и повторяет попытку перехода при каждом изменении системного времени.

Если в поле A определено значение ALL, то поля B и C содержат имена блоков, поле D содержит целое число. Транзакт последовательно пытается войти в блоки, отстоящие друг от друга на расстояние D, начиная с блока B и заканчивая блоком C до первой успешной попытки. Если ни один из блоков не может принять транзакт, то он остается в блоке TRANSFER и повторяет попытку перехода при каждом изменении системного времени. Здесь значение поля D должно задаваться таким образом, чтобы выполнялось условие $№C = №B + M \cdot D$, где M – любое целое число. Если поле D не задано, то транзакт пытается последовательно войти в каждый блок между B и C.

Если в поле A определено значение PICK, то поля B и C содержат имена блоков, а транзакт направляется в любой блок между блоками B и C, выбранный случайным образом.

Блок GATE позволяет изменять путь транзакта в зависимости от состояния моделируемого оборудования. Блок имеет следующую структуру:

GATE O A,B

В поле O задается проверяемое состояние оборудования в виде мнемокода. В поле A задается имя проверяемой единицы оборудования, в поле B – имя блока, к которому направляется транзакт, если проверяемое условие ложно.

Данный блок может работать в двух режимах: в режиме отказа и в режиме условного перехода.

Режим отказа: транзакт задерживается в блоке GATE до тех пор, пока не выполнится условие состояния проверяемого объекта. Как только это произойдет, транзакт направляется к следующему за GATE блоку. В этом режиме поле B опускается.

Режим условного перехода: если проверяемый объект не находится в требуемом состоянии, транзакт направляется к блоку, указанному в поле B. В противном случае транзакт направляется к следующему за GATE блоку. Например, в блоке

GATE SF STR

транзакт будет задержан до тех пор, пока память с именем STR не будет полной.

Блок TEST изменяет маршрут транзакта в зависимости от выполнения разнообразных логических условий, определенных на множестве СЧА. Блок имеет следующую структуру:

TEST O A,B,C

В поле *O* указывается мнемоника отношения: “L” – ”<”, “LE” – “≤”, “E” – “=”, “NE” – “≠”, “G” – “>”, “GE” – “≥”.

В полях *A* и *B* указываются левое и правое значения условия, соответственно. В поле *C* указывается имя блока, к которому направляется транзакт, если проверяемое условие ложно. Если проверяемое условие истинно, то транзакт переходит к следующему за TEST блоку.

Например, при входе транзакта в оператор

TEST G Q$OCH,5,OTD1

проверяется длина очереди OCH. Если длина очереди больше пяти, то транзакт направляется к следующему за TEST блоку, иначе транзакт переходит к блоку с именем OTD1.

Пример 2.4

В магазине находится два отдела: продовольственный и промтоварный. Около 30-ти процентов приходящих в магазин покупателей направляются в промтоварный отдел, остальные — в продовольственный. Причем, если очередь в промтоварном отделе больше двух человек, а в продовольственном — больше пяти, то покупатели уходят из магазина, не дожидаясь обслуживания. Время прихода и обслуживания покупателей распределено экспоненциально. Среднее значение времени прихода равно соответственно 20 сек, времени обслуживания в продовольственном отделе – 30 сек и в промтоварном - 35 сек.

Модель, имитирующая работу магазина за 8 ч:

```
GENERATE  (EXPONENTIAL(1,0,20))        ;приход покупателей
TRANSFER   .3,,PROM                    ; выбор покупателем отдела
 ; работа продовольственного отдела
```

```
PROD  TEST  LE  Q$LIN1,5,BYBY    ;если очередь больше 5-ти
                                  ;чел. — уход  покупателя
    QUEUE  LIN1                   ;поставить в очередь в
                                  ;продовольственный отдел
    SEIZE  PROD1                  ;занять продавца
    DEPART  LIN1                  ;покинуть очередь в
                                  ;продовольственный отдел
    ADVANCE  (EXPONENTIAL(1,0,30))  ;обслуживание покупателя
    RELEASE  PROD1                ;освободить продавца
    TERMINATE                     ;уход покупателя
      ; работа промтоварного отдела
    PROM  TEST  LE  Q$LIN1,2,BYBY
    QUEUE  LIN2
    SEIZE  PROD2
    DEPART  LIN2
     ADVANCE  (EXPONENTIAL(1,0,35))
    RELEASE  PROD2
    BYBY  TERMINATE
    ; таймер
    GENERATE  2880
    TERMINATE  1
     START  1
```

2.4.7. Работа с памятью

Память — особый объект языка, который призван имитировать разного рода накопители, используемые в исследуемых системах, в которых может одновременно находиться несколько транзактов. Для каждой применяемой памяти пользователь должен указать ее емкость - объём памяти, определяющий максимальное количество транзактов, которые могут одновременно находиться

в ней. Для указания емкости используется *оператор описания памяти* STORAGE. Как любой оператор описания языка этот блок помещается до первого блока GENERATE. Поле метки содержит имя памяти, а операнд *A* указывает емкость памяти. Например, для описания памяти емкостью 10 единиц используется блок

```
STR  STORAGE  10
```

Блок ENTER - занять память. В поле *A* блока указывается имя памяти, в которую помещается транзакт, в поле *B* — число единиц памяти, занимаемых транзактом при входе. Когда транзакт входит в блок ENTER, определяется число свободных единиц памяти. Если значение операнда *B* не превышает числа свободных единиц памяти, то число занятых единиц увеличивается на значение операнда *B*. В этом случае транзакт входит в блок ENTER без задержки. Если же значение операнда *B* превышает число свободных единиц памяти, то транзакт задерживается перед входом в блок ENTER. Задержанные при обращении к памяти транзакты упорядочиваются по приоритету.

Если поле *B* в блоке ENTER пустое, то число занимаемых единиц памяти принимается равным единице.

Пусть транзакт "*x*" задержан перед входом в блок ENTER. Если для транзакта "*y*", приходящего после "*x*", свободной емкости памяти достаточно, то "*y*" войдет в блок без задержки.

Примечание: если вы используете блок ENTER, память должна быть обязательно описана ранее командой STORAGE.

Блок LEAVE - освободить память. В поле *A* блока указывается имя освобождаемой памяти, в поле *B* — число освобождаемых единиц. В случае пустого поля *B* число освобождаемых единиц памяти принимается равным единице. При входе транзакта в блок LEAVE количество занятых единиц памяти, указанной в поле *A*, уменьшается на значение операнда *B*. Перед входом в блок транзакты не задерживаются. Транзакт не должен освобождать большее число единиц памяти, чем их всего занято. Если же транзакт пытается это сделать, то

симулятор выдает на печать сообщение об ошибке и прекращает выполнение модели.

В тот момент модельного времени, когда транзакт освобождает память, симулятор просматривает список задержанных у памяти транзактов, если они есть. Для каждого очередного транзакта проверяется, может ли он теперь быть обслужен памятью. Если такая возможность есть, то симулятор перемещает этот транзакт в блок ENTER, и в результате число занятых единиц памяти соответствующим образом увеличивается.

Транзакт не обязан освобождать такое же число единиц памяти, какое занимал. Он может также освобождать память, которую не занимал. Транзакт имеет право занимать и освобождать любое количество памятей, при этом операции занятия и освобождения могут чередоваться в произвольном порядке.

Пример 2.5

Автомобили подъезжают к бензозаправочной станции в среднем каждые 4±2 мин. На станции есть две бензоколонки, каждая из которых используется в среднем 5±1 мин. Автостоянка при станции рассчитана на 4 автомобиля. Если подъехавший автомобиль застает обе бензоколонки занятыми, то он встает в очередь на автостоянку. Если же все места и на автостоянке заняты, то автомобиль проезжает мимо. Промоделировать работу станции за 12 часов.

```
STO  STORAGE  4                ;места под автостоянку
COL  STORAGE  2                ;бензоколонки
; описание работы бензоколонки
GENERATE  4,2                  ;приезд автомобиля
GATE  SNF  STO,BYBY            ;если места заняты - проезжает
ENTER  STO                     ;занять место на автостоянке
ENTER  COL                     ;занять бензоколонку
LEAVE  STO                     ;освободить автостоянку
ADVANCE  5,1                   ;заправиться
LEAVE  COL                     ;освободить бензоколонку
BYBY  TERMINATE                ;покинуть станцию
```

```
;таймер
GENERATE  720
TERMINATE  1
START  1
```

2.4.8. Вычислительные объекты языка

Арифметические переменные. Для того чтобы использовать в программе переменную, необходимо сначала ее описать оператором описания VARIABLE либо FVARIABLE. В поле метки оператора записывается имя переменной, в операнде *A* - арифметическое выражение, составляемое из СЧА, знаков арифметических операций и круглых скобок. Используются следующие арифметические операции: + , - , # (умножение) , /, @ (взять остаток от деления), \ (целое от деления). Приоритет операций стандартный. Деление на ноль не считается ошибкой, и результатом такого деления является ноль. Остаток от деления на ноль также считается равным нулю.

При использовании переменной в программе указывается СЧА переменной: V$<имя переменной>, например,

ADVANCE V$VAR1 — задержать транзакт на время, заданное переменной VAR1.

Примечание: переменная является единственным объектом языка, по которому по окончанию моделирования в отчете не выдается никакой информации. Поэтому, если вам необходимо по окончанию моделирования проанализировать значение переменной, то можно присвоить ее значение ячейке, значение которой выводится в результирующем отчете.

Ячейки служат для хранения некоторых постоянных и/или изменяющихся значений данных программы. В отличие от большинства объектов языка ячейка может обозначаться как именем, так и числом. Для работы с ячейками используется блок SAVEVALUE. В поле *A* этого блока указывается номер/имя ячейки, сохраняющей значение, и вид изменения этого значения ("+" — накопление,

"–" — уменьшение). В поле *B* содержится либо СЧА, либо число, которое добавляется либо вычитается, либо заменяет содержимое ячейки.

Например, оператор

SAVEVALUE 10+,1

означает, что при поступлении транзакта в блок, к содержимому 10-й ячейки прибавляется единица. Или оператор

SAVEVALUE FRT,V$VAR1

означает, что при поступлении транзакта в блок, в ячейку с именем FRT записывается значение переменной VAR1.

При необходимости обращения к ячейке указывается СЧА ячейки: X$<имя ячейки> (если ячейка задана именем) или XN (если ячейка задана номером N).

Чаще всего на базе ячеек организуются разного рода счетчики. Перед началом имитации содержимое всех используемых в программе ячеек устанавливается в 0. Если же требуется задать значение какой-либо из ячеек до начала моделирования, то для этого используется оператор INITIAL, в поле *A* которого задается СЧА ячейки, в поле *B* – присваиваемое значение. Например,
INITIAL X$UCH1,10 — присвоить ячейке с именем UCH1 значение 10. Этот оператор должен помещаться до первого блока GENERATE.

Необходимо помнить, что в поле *B* ячейки не могут стоять арифметические выражения. При необходимости используйте в этом поле СЧА переменной, которая описывает требуемое выражение.

Матрицы служат для хранения некоторых постоянных и/или изменяющихся значений данных программы в виде массивов. Для того чтобы использовать в программе матрицу, необходимо сначала ее описать оператором описания MATRIX. В поле метки оператора записывается имя матрицы. Поле *A* в операторе не используется, поля *B* и *C* содержат числовые значения, определяющие количество строк и количество столбцов в матрице, соответственно. Начальные значения всех элементов матрицы равны нулю. Если необходимо присвоить всем элементам матрицы одинаковые значения, отличные от нуля, используется

оператор INITIAL, в поле *A* которого задается имя матрицы, в поле *B* – присваиваемое значение.

Для изменения значения отдельного элемента матрицы используется *блок MSAVEVALUE.* В поле *A* блока указывается имя матрицы, после которого может быть указан «+» или «–». Поля *B* и *C* служат для выбора конкретного элемента матрицы и содержат номер строки и номер столбца, соответственно. В поле *D* указывается значение, которое должно быть добавлено, вычтено или присвоено элементу матрицы. Если после имени матрицы не стоит никакого знака, то значение *D* присваивается элементу. Если после имени матрицы стоит знак «+» или «–», то указанное значение добавляется или вычитается из текущего значения элемента, соответственно.

При необходимости обращения к элементу матрицы указывается СЧА элемента: MX$<имя матрицы>(I,J) (если матрица задана именем) или XN(I,J) (если матрица задана номером N). Здесь I означает номер строки, J – номер столбца.

Пример 2.6.

Модель работы двухтактного буферного запоминающего устройства.

Идея двухтактного буфера: совместить во времени процессы сбора и записи информации в буферное запоминающее устройство (БЗУ) ограниченного объема и перезаписи информации в долговременное запоминающее устройство (ДЗУ), объем которой неограничен.

Техническое осуществление — с использованием канала прямого доступа и программного канала. Рассмотрим упрощенную модель системы двухтактного буферирования. Экспериментальная информация в процессе сбора записывается словами в буфер. При заполнении приемного БЗУ до определенного уровня, генерируется запрос к управляющему устройству (УУ) на передачу информации в ДЗУ (включается система прерываний). Между моментом возникновения запроса и переключением ключей (рис. 2.2) проходит определенное время, по истечении которого приемное БЗУ становится передающим (например, подключается канал прямого доступа и информация пишется на диск), а для сбора

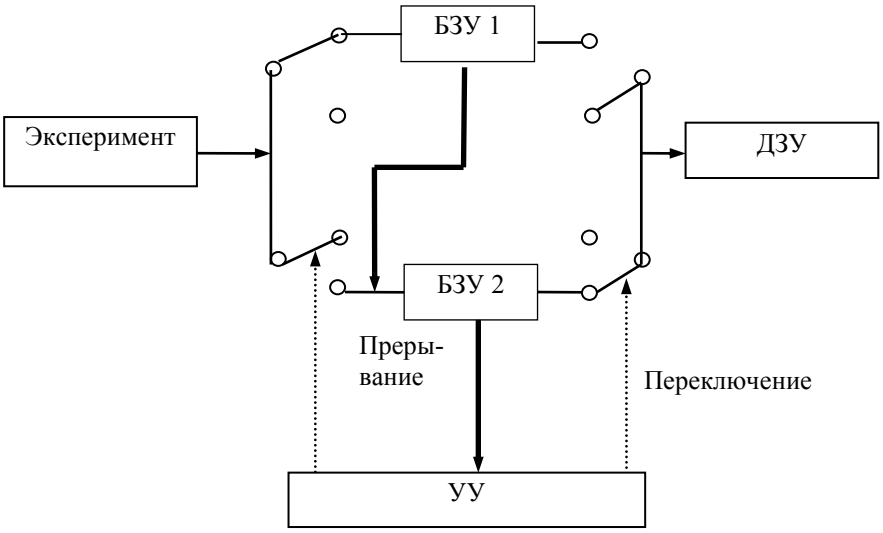


Рис. 2.2. Схема двухтактного БЗУ

информации в оперативной памяти выделяется новая область, которая становится приемным БЗУ.

При исследовании такой системы на модели могут ставиться различные вопросы: каков должен быть уровень заполнения БЗУ, при котором в системе генерируется запрос на переключение ключей; как связан этот уровень с интенсивностью потока экспериментальных данных и вероятностью переполнения приемного БЗУ за время между запросом и переключением ключей и т.п.

Рассмотрим упрощенное описание модели на языке GPSS. Допустим, объем БЗУ — 1024 слова. Разрядность слова — 8 единиц. Тогда память БЗУ1/БЗУ2:

```
VAR2  VARIABLE  1024#8
STR1  STORAGE  V$VAR2
```

Допустим, запрос на прерывание возникает при заполнении БЗУ до 1020 слов, тогда уровень заполнения:

```
VAR3  VARIABLE  1020#8
```

Еще одна переменная — описание среднего времени задержки между запросом и переключением БЗУ:

```
VAR4  VARIABLE  10/4
```

Пусть время между приходом слов в БЗУ распределено экспоненциально и равно в среднем 10 мин., тогда функционирование этой системы будет выглядеть следующим образом:

```
GENERATE  (EXPONENTIAL(1,0,10))
GATE  SNF  STR1,OTKAZ          ;если память заполнена — отказ
ENTER  STR1,8
TEST  E  S$STR1,V$VAR3,ENDM
ADVANCE  (EXPONENTIAL(1,0,V$VAR4))  ;переключение БЗУ
LEAVE  STR1,S$STR1
TERMINATE  1
OTKAZ   SAVEVALUE  1+,1        счетчик слов, потерянных из-за
                               переполнения БЗУ
ENDM  TERMINATE
```

Здесь транзакт попадает в блок ADVANCE только тогда, когда БЗУ заполнена до необходимого уровня. Блок LEAVE полностью очищает память STR1, что равносильно переключению на новое БЗУ.

Пример 2.7.

Производство деталей включает длительный процесс сборки, заканчивающийся коротким периодом обжига в печи. Поскольку содержание печи обходится дорого, несколько сборщиков используют одну печь, в которой одновременно можно обжигать только одну деталь. Сборщик не может начать новую сборку, пока не вытащит из печи предыдущую деталь.

Время сборки детали равно 30±5 минут. Время обжига равно 8±2 минуты. Зарплата сборщика составляет 60 рублей в час, стоимость эксплуатации печи – 3200 рублей за 8 часов. Цена материала, идущего на изготовление одного изделия равна 100 рублей, стоимость готового изделия – 380 рублей. Тогда прибыль от изготовления детали составит 380-100=280 рублей.

Необходимо определить оптимальное количество сборщиков исходя из максимизации прибыли за неделю (возьмем 8-часовой рабочий день без выходных 8*7=56 часов).

Для удобства исследования число сборщиков в модели зададим переменной SBOR. Меняя значение переменной, мы сможем подобрать требуемое число рабочих.

Затраты на содержание печи и зарплату сборщиков опишем в переменной ZATR. Доходы от производства будем подсчитывать с помощью ячейки IZD. А результирующую прибыль определим в переменной PRIB.

В качестве транзактов в модели будем рассматривать самих сборщиков, подразумевая, что в каждый момент времени один сборщик производит одну деталь. Устройство PECH вводим для обозначения печи.

```
        SBOR   VARIABLE   N    ; при прогоне модели вместо N
                                ; ставим конкретное целое число
        ZATR   VARIABLE   3200#7+V$SBOR#60#56
        PRIB   VARIABLE   X$IZD-V$ZATR

        GENERATE   ,,,V$SBOR
PROD    ADVANCE   30,5  ; процесс сборки изделия
        SEIZE   PECH
        ADVANCE   8,2
        RELEASE   PECH
        SAVEVALUE   IZD+,280
        TRANSFER   ,PROD

        GENERATE   (56#60)
        SAVEVALUE   REZULT,V$PRIB
        TERMINATE   1
        START   1
```

При прогоне модели для разного количества работников получим следующие результаты:

N	Прибыль	Комментарий
1	-10360	1 работник – убыточное предприятие
2	840	
3	12320	
4	22400	
5	30240	Максимальная прибыль – оптимальное количество работников
6	29960	

2.4.9. Приоритеты

Каждый транзакт может иметь свой приоритет — от 0 до 127. Чем больше номер, тем больше приоритет. Предпочтение в системе отдается транзактам с большим приоритетом, ранее поступившим.

Для изменения приоритета транзакта в процессе его путешествия по системе используется блок PRIORITY. Поле *A* этого блока определяет значение присваиваемого приоритета. Например, при прохождении через блок PRIORITY 3 транзакту будет присвоен приоритет 3.

2.4.10. Изменение параметров транзакта

Каждый транзакт может иметь до 100 параметров (атрибутов). *Значения параметрам присваиваются* с помощью *блока ASSIGN*. В поле *A* этого блока указывается номер/имя параметра и вид его изменения, в поле *B* определяется записываемое в параметр значение, в поле *C* задается при необходимости модификатор значения *B* в виде имени функции, значение которой умножается на *B*.

Приписывая к номеру параметра в поле *A* символ + или –, можно обеспечить не запись значения поля *B* в параметр, а добавление или вычитание этого значения из значения параметра. В поле *B* значение может быть задано как целым числом, так и СЧА. Например:

```
ASSIGN  1,10                ;занести 10 в P1
ASSIGN  2+,V$VAR1,EXP       ;добавить в  P2  значение
                            ;V$VAR1*FN$EXP
ASSIGN  TRE–,S$STR   ;вычесть из P$TRE значение текущего со-
                     ;держимого памяти
```

Используя блок ASSIGN, можно организовывать циклы в программе. Например, если необходимо прогнать транзакт 10 раз через блок ADVANCE, это можно осуществить следующим образом:

```
ASSIGN  1,10                ;занести 10 в P1 транзакта
PROD  ADVANCE  52
ASSIGN  1–,1                ;вычесть 1 из P1
TEST  E  P1,0,PROD          ;продолжать цикл пока счетчик не
                            ;обнулится
```

Пример 2.8.

В магазине электротоваров работают два консультанта. Посетители заходят в магазин в среднем каждые 2 минуты. Покупатели осматривают товар в среднем в течение 10 минут, после чего примерно 70% из них обращаются к консультанту за помощью (время на консультацию составляет в среднем 5 минут). После осмотра товара и получения консультации примерно 40% посетителей уходят без покупки. Остальные покупатели с выбранным товаром направляются к кассе. Кассир обслуживает клиентов в среднем 3 минуты.

Стоимость покупки распределена равномерно на интервале от 500 до 20000 рублей. Известно, что примерно у 3 процентов покупателей есть карточка со скидкой в 10%, а у 12 процентов покупателей есть карточки со скидкой в 5%.

Необходимо оценить прибыль магазина за 10-часовой рабочий день.

Наличие карточки со скидкой у покупателя мы опишем функцией SKIDKA, и будем записывать ее значение в 1-й атрибут транзакта.

Стоимость производимой покупки опишем с помощью функции POKUP, прибыль магазина опишем переменной SUM.

Консультанты в модели будут представлены памятью CONS, кассир – устройством KAS.

Модель работы магазина:

```
CONS    STORAGE   2
POKUP   FUNCTION RN1,C2
0,500/1,20000
SKIDKA   FUNCTION   RN1,D3
.03,0.10/.15,0.05/1,0
SUM   VARIABLE   FN$POKUP#(1-P1)

    GENERATE   (EXPONENTIAL(1,0,2))
    ASSIGN   1,FN$SKIDKA
    ADVANCE   (EXPONENTIAL(1,0,10))
    TRANSFER   .3,,MET1
    ENTER   CONS
    ADVANCE   (EXPONENTIAL(1,0,5))
    LEAVE CONS
MET1   TRANSFER   .4,,UXOD
    SEIZE   KAS
    ADVANCE   (EXPONENTIAL(1,0,3))
    SAVEVALUE   PRIB+,V$SUM
    REKEASE   KAS
UXOD   TERMINATE

    GENERATE   600
    TERMINATE   1
    START   1
```

2.4.11. Косвенная адресация

Косвенная адресация используется в сочетании с любым СЧА, кроме текущего времени C1, случайного числа RNi и времени пребывания транзакта в системе M1.

Структура записи косвенной адресации СЧА*<целое число/имя>, где <целое число/имя> – это номер/имя параметра, в котором указан номер или имя соответствующего СЧА, например:

SEIZE FN*1.

Здесь при поступлении транзакта определяется сначала имя функции (по первому параметру транзакта), а затем – значение этой функции. По этому значению и определяется устройство, занимаемое транзактом. Получается, что разные транзакты, попадая в данный блок, могут занимать различные устройства в зависимости от значения собственных параметров и значения выбранных функций.

В блоке GENERATE косвенная адресация допускается только в поле A.

2.4.12. Списки

Списки определяют внутреннюю организацию GPSS. В языке существует 5 типов списков. Следующие четыре типа ведет симулятор (пользователь не имеет к ним доступа):

– списки текущих событий: содержат транзакты, у которых время очередной передвижки в программе модели меньше системного времени; активные транзакты находятся в состоянии задержки, например, по причине занятости устройства;

– списки будущих событий: содержат транзакты, у которых время очередной передвижки больше системного времени (например, все транзакты в блоках ADVANCE);

– списки прерываний: содержат транзакты, обслуживание которых на определенных устройствах было прервано другими транзактами;

– списки синхронизируемых и задержанных транзактов.

Пятый тип списков – это списки пользователя, которые можно создавать и использовать в процессе моделирования. Управление списками осуществляется с помощью двух следующих блоков.

Блок LINK выполняет включение транзакта в список пользователя. Поле *A* блока содержит номер или имя списка, куда включается транзакт. Транзакт, попадая в блок LINK, помещается в список, определенный полем *A*, и временно исключается из процесса (становится пассивным).

Дисциплина постановки в список определяется полем *B*, которое может принимать одно из следующих значений:

FIFO (первым пришел, первым вышел) – вновь поступивший транзакт помещается в конец списка;

LIFO (последним пришел, первым вышел) – вновь пришедший транзакт помещается в начало списка;

В любом другом случае значение поля *B* вычисляется, и транзакт помещается в список в соответствии с вычисленным значением (за транзактом, у которого соответствующее значение больше и перед транзактом, у которого соответствующее значение меньше). Например, транзакты могут быть упорядочены в соответствии со значениями их параметров. Если используется СЧА PR, то транзакты упорядочиваются в списке по приоритетам.

Блок UNLINK осуществляет извлечение транзакта из списка. Поле *A* блока содержит номер или имя списка, из которого извлекается транзакт. Поле *B* содержит имя блока, к которому направляется извлеченный транзакт. В поле *C* указывается количество транзактов, извлекаемых из списка (по умолчанию извлекаются все транзакты).

При попадании некоторого транзакта в блок UNLINK пассивный транзакт, находящийся в списке пользователя, извлекается оттуда и направляется к блоку, имя которого указано в поле *B*, основной же транзакт, вызвавший это извлечение, продолжает свое обычное движение по программе.

Со списками связаны следующие СЧА:

CA$< имя списка > – среднее число транзактов в списке;

CC$< имя списка > – общее число транзактов, прошедших через указанный список;

CH$< имя списка > – текущее число транзактов в списке;

CM$< имя списка > – максимальное число транзактов в списке;

CT$< имя списка > – среднее время, проведенное транзактом в списке.

2.4.13. Статистические таблицы

Объект типа таблицы представляет собой эквивалент понятия «гистограмма». Гистограммы применяются для статистического анализа такой случайной величины, функция распределения которой неизвестна, но зато имеется достаточно большое число независимых реализаций этой величины.

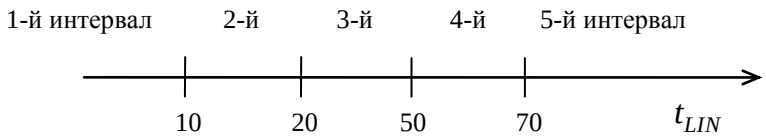
Для того, что таблицы можно было использовать в модели, они должны быть описаны.

Для описания таблицы используется *блок TABLE*. В поле метки этого блока задается имя таблицы, в поле *A* — аргумент таблицы в виде СЧА. Здесь аргументом таблицы является исследуемая случайная величина. В поле *B* указывается верхняя граница первого частотного интервала, в поле *C* — ширина интервалов, а в поле *D* — их число, включающее оба полубесконечных интервала.

Исключением может быть время, проводимое транзактом в очереди. Если необходимо исследовать это время, то используется *блок QTABLE*, в поле *A* которого указывается имя очереди, время нахождения в которой нас интересует. Остальные поля данного блока описываются аналогично блоку TABLE. Например, если нас интересует гистограмма времени, проводимого одним транзактом в очереди LIN, то мы можем описать таблицу следующим блоком:

```
TBL  QTABLE  LIN,10,20,5
```

Графически гистограмма должна располагаться следующим образом:



Если таблица описана, то транзакты могут *фиксировать* в ней *информацию* с помощью *блока TABULATE*. В поле *A* этого блока указывается имя таблицы, в которой накапливается информация. При входе транзакта в блок TABULATE вычисляется значение аргумента указанной таблицы и определяется, в какой из интервалов таблицы это значение попадает. После этого счетчик соответствующей интервальной частоты увеличивается на 1.

Примечание: если используется блок QTABLE, то блок TABULATE не нужен, вычисление искомой характеристики в данном случае происходит автоматически.

В результате моделирования на печать по каждой таблице выдается информация в виде, показанном в приложении 2. Для каждой таблицы автоматически осуществляются оценки среднего и среднеквадратического отклонений.

Пример 2.9

Модель вычислительной системы с несколькими абонентскими пунктами (АП)

Пусть ЭВМ обслуживают 5 АП. Программа управления каналом связи опрашивает АП в соответствии со списком опроса. Если у опрашиваемого АП имеется сообщение для передачи, оно посылается в ЭВМ. После завершения передачи данных канал освобождается, и как только выходное сообщение от ЭВМ готово, оно занимает канал связи для передачи, т.е. выходного сообщения (от ЭВМ). Выходное сообщение имеет приоритет перед входным (от АП).

Исходные данные:

1) опрос циклический;

2) время, затрачиваемое на опрос одного АП - 100 мс;

3) интервалы времени между опросами - 10 мс;

4) передача информации происходит со скоростью 300 сотен символов/сек;

5) сообщение может содержать от 6 до 60 сотен символов;

6) интервалы времени между возникающими сообщениями от АП распределены экспоненциально со средним значением 500 мс;

7) время обработки сообщения от ЭВМ - 500 мс.

Определим вычислительные объекты:

- функция OPR имитирует список опроса: аргумент — параметр P1, при условии, что в него будет записываться номер опрашиваемого АП;
- функция NAP определяет случайным образом номер АП, на котором возникло сообщение для ЭВМ;
- функция SMV определяет число символов во входном сообщении;
- переменная TZAN определяет время занятия канала связи, равное длине сообщения, разделенное на скорость передачи. (Время переведено в мс):

$$30000\frac{\text{символов}}{\text{с}} = \frac{30000}{1000}\cdot\frac{\text{символов}}{\text{мс}}$$

Программа:

```
OPR  FUNCTION  P1,D5
1,2/2,3/3,4/4,5/5,1
NAP   FUNCTION  RN1,D5
.2, 1/.4, 2/.6, 3/.8, 4/1, 5
SMV  FUNCTION  RN2,C2
.0,6/1.0,61
TZAN  VARIABLE  FN$SMV*10/3
GENERATE  (EXPONENTIAL(1,0,500))  ;возникновение
 ;сообщения от АП
ASSIGN  1,FN$NAP                    ;определение номера АП
LINK  P1,FIFO
; опрос абонентских пунктов
GENERATE  ,,,1                      ;циклический опрос от ЭВМ
ASSIGN  1,1
POLL  ASSIGN  1,FN$OPR              ;определяется очередной АП
SEIZE  CAN                          ;занять канал под опрос
```

```
ADVANCE  100
TEST  NE  CH*1,0,PROD              ;если нет сообщения — про-
                                   ;должать опрос
UNLINK  P1,XMIT,1                  ;передача сообщения в ЭВМ
PROD  RELEASE  CAN                 ;освободить канал от опроса
ADVANCE  10                        ;задержка между опросами
TRANSFER  ,POLL                    ;продолжение опроса
; передача и обработка сообщения
XMIT  SEIZE  CAN                    ;передача к ЭВМ
ADVANCE  V$TZAN
RELEASE  CAN
ADVANCE  500                       ;обработка сообщения
PRIORITY  1
SEIZE  CAN                         ;передача от ЭВМ
ADVANCE  V$TZAN
RELEASE  CAN
TERMINATE
; таймер
GENERATE  10000
TERMINATE  1
START  1
```

В программе при возникновении сообщения от АП в список с соответствующим номером заносится единица. Блок TEST проверяет, пуст ли список, соответствующий опрашиваемому АП. Если да – то продолжается опрос, если нет – то блок UNLINK выделяет из списка сообщение и передает управление на блок XMIT. Транзакт, вызвавший извлечение из списка сообщения, по-прежнему направляется к блоку PROD и продолжает цикл опроса. Таким образом в модели опроса единственный транзакт циркулирует «по кругу», имитируя циклический опрос АП.

2.4.14. Логические переключатели

Логические переключатели могут находится в двух положениях: «включен» и «выключен». Перед началом выполнения программы все переключатели устанавливаются в положение «выключен».

Для работы с логическими переключателями используется *блок LOGIC*. При поступлении транзакта в блок состояние логического переключателя, номер или имя которого указан в поле *A*, меняется в соответствии с мнемоникой:

LOGIC R 1 – логический переключатель с номером 1 устанавливается в состояние «выключен»;

LOGIC S 1 – логический переключатель с номером 1 устанавливается в состояние «включен»;

LOGIC I 1 – состояние логического переключателя с номером 1 инвертируется.

Состояние логического переключателя может быть проверено в любой части модели с помощью блока GATE или с помощью СЧА LS$<имя логического переключателя>, который принимает значение 1, если логический переключатель «включен», и 0 – в противном случае.

Пример 2.10

Паспортный стол работает с 9 до 18 часов с часовым перерывом на обед с 13 до 14 часов. Посетители приходят в среднем каждые 5 минут, причем все сначала направляются к начальнику паспортного стола, который работает с каждым посетителем в среднем 4 минуты. После начальника примерно 5% посетителей покидают отделение (получен отказ либо вопрос решен), а остальные направляются в отдел прописки, в котором работают три паспортистки. Время приема посетителя в отделе прописки равно в среднем 12 минутам (с каждым посетителем). Время прихода и время обслуживания в системе распределено экспоненциально.

Те посетители, кто стоял в очереди и не успел обслужиться до обеда, обслуживаются после перерыва в первую очередь. Будем считать, что во время

обеденного перерыва никто не приходит. Примерно за полчаса до окончания рабочего дня просят не занимать очередь к начальнику паспортного стола и подошедшие в это время посетители не обслуживаются.

Проверить, успеют ли все посетители, стоящие в очереди, обслужиться до конца рабочего дня. Протабулировать время нахождения посетителей в очередях к начальнику паспортного стола и в отдел прописки.

В переменной RAZN подсчитывается количество посетителей, которые встали в очередь, но не успели обслужиться до конца рабочего дня.

```
PROP   STORAGE   3
NAB1   TABLE   OCH_NACH,10,10,10
TAB2   QTABLE   OCH_PROP,10,10,10
RAZN   VARIABLE   N$VXOD-N$UXOD
;работа начальника паспортного стола
GENERATE   (EXPONENTIAL(1,0,5))
GATE   LR   TIME,BYE              ;если рабочий день закончился - уход
VXOD   QUEUE   OCH_NACH
GATE   LR   OBED ;ожидание окончания обеда
SEIZE   NACH
DEPART   OCH_NACH
ADVANCE   (EXPONENTIAL(1,0,4))
RELEASE   NACH
TRANSFER   .05,,UXOD
; работа отдела прописки
QUEUE   OCH_PROP
GATE   LR   OBED ; ожидание окончания обеда
ENTER   PROP
DEPART   OCH_PROP
ADVANCE   (EXPONENTIAL(1,0,12))
LEAVE   PROP
UXOD   TERMINATE
BYE   TERMINATE
```

```
; таймер
GENERATE   240,,,1                ; начало рабочего дня
LOGIC   S   OBED   ; начало обеда
ADVANCE   60
LOGIC   R   OBED  ; окончание обеда
ADVANCE   210
LOGIC   S   TIME    ; за 30 минут до конца рабочего дня
ADVANCE   30
SAVEVALUE   NO_OBSL,V$RAZN              ; подсчет посетителей,
                    ; которые встали в очередь, но не успели
                    ; обслужиться до конца рабочего дня
TERMINATE   1
START   1
```

2.4.15. Синхронизация транзактов

Любые элементы в системах прямо или опосредованно связаны, взаимодействуют. Зависимость между процессами, протекающими в разных частях системы, нередко выражается в форме синхронизации, то есть в форме взаимного согласования этих процессов по времени.

Блок SPLIT предназначен для *моделирования одновременного начала нескольких процессов*. В момент входа транзакта в блок SPLIT создается несколько копий этого транзакта. Число копий задается в поле *A*. Все копии переходят в блок, определенный в поле *B*. Исходный (порождающий) транзакт переходит к блоку, следующему за SPLIT. Если поле *C* блока SPLIT пустое, то все копии идентичны породившему их транзакту. Например, при входе транзакта в блок

```
SPLIT  4,NEXT
```

порождается четыре транзакта, идентичных вошедшему, и передается в блок, в поле метки которого записано NEXT. Породивший их транзакт передается в блок, записанный после блока SPLIT. Всего из этого блока SPLIT выходит пять транзактов.

Если поле C непустое, то его значение интерпретируется как номер или имя параметра транзакта. Пусть N — значение этого параметра в момент входа транзакта в блок SPLIT. Тогда в момент выхода из SPLIT данный параметр у исходного транзакта будет иметь значение $N+1$, а у копий транзактов соответственно $N+2, N+3, \ldots, N+K$, где K — общее число вышедших из блока SPLIT транзактов. Например, если транзакт, имеющий нуль в десятом параметре, войдет в блок

SPLIT 2,ABCD,10 ,

то параметр P10 у этого транзакта приобретет значение 1, а у копий — соответственно 2 и 3.

Транзакты — копии могут двигаться в модели независимо друг от друга. Копии могут проходить блоки SPLIT и порождать новые копии.

Множество, состоящее из исходного транзакта и всех его копий, называется семейством транзактов. Копия члена семейства является членом того же семейства. Любой транзакт — член только одного семейства.

Блок ASSEMBLE — одновременное завершение нескольких процессов. Блок собирает заданное в поле A число транзактов одного семейства и превращает их в один транзакт. Первый из транзактов какого-либо семейства, вошедший в блок, задерживается до тех пор, пока в этом блоке не накопится заданное число транзактов того же семейства. После этого первый транзакт выходит из блока ASSEMBLE, а остальные транзакты этого семейства уничтожаются.

В одном блоке ASSEMBLE могут одновременно проходить сборку транзакты, принадлежащие к разным семействам. Например, если в блок

ASSEMBLE 4

поступают транзакты разных семейств, то транзакты каждого семейства собираются по четыре и каждая четверка превращается в один транзакт.

Блок GATHER работает аналогично блоку ASSEMBLE с тем отличием, что транзакты, попав в блок GATHER, не уничтожаются, а только задерживаются и после того, как в блоке накапливается заданное число транзактов, они *все* переходят к следующему блоку.

Блок MATCH предназначен для *синхронизации процессов.* В программе всегда должно быть два поименованных блока MATCH. В поле *A* блока указывается имя парного блока MATCH. Когда транзакт попадает в блок, определяется второй блок MATCH и проверяется, находится ли в нем транзакт этого же семейства. Если да, то транзакты выходят из обоих блоков одновременно. Если в парном блоке нет транзакта, то в первом блоке транзакт задерживается до тех пор, пока во второй блок не поступит транзакт этого же семейства. Таким образом, использование блоков MATCH позволяет синхронизировать передвижение транзактов по модели.

Пример 2.11

Промоделировать сборку изделий рабочими A, B и C. Изделия в разобранном виде поступают каждые 300±100 мин. Каждое из них разделяется между рабочими A и B, которые параллельно готовят свою часть изделия к сборке. Подготовка состоит из двух фаз, причем после первой фазы производится сверка с одновременным участием обоих рабочих, а затем A и B независимо выполняют вторую фазу работы. На первой фазе рабочий A тратит на работу 100±20 мин, рабочий B – 80±20 мин; на второй фазе рабочий A тратит 50±5 минут, рабочий B – 80±20 минут. После окончания работы рабочими A и B рабочий C выполняет сборку изделия за 50±5 мин, причем он может начинать сборку только тогда, когда оба первых рабочих закончат свою работу.

Модель сборки изделий:

```
GENERATE  300,100                ;поступление изделий
SPLIT  1,MANB                    ;разделение изделий
SEIZE  RABA                      ;занять рабочего A
ADVANCE  100,20                  ; 1-я фаза
FAZ1A  MATCH  FAZ1B      ;ждать, если B не закончил 1-ю фазу
ADVANCE  50,5                    ; 2-я фаза
RELEASE  RABA
TRANSFER  ,MANC
MANB  SEIZE  RABB                ;занять рабочего B
```

```
ADVANCE  80,20
FAZ1B  MATCH  FAZ1A      ;ждать, если А не закончил 1-ю фазу
ADVANCE  80,20
RELEASE  RABB
MANC  ASSEMBLE  2                ;ждать обе части изделия
SEIZE  RABC                      ;занять рабочего С
ADVANCE  50, 5
RELEASE  RABC
TERMINATE  1                     ;завершение сборки
START 1000
```

2.4.16. Прерывание работы устройства

Блок PREEMPT — захватить устройство. Транзакт, попадающий в блок PREEMPT, захватывает устройство, имя которого указано в поле *A* блока. Если при захвате устройства оно свободно, то транзакт просто занимает устройство, в этом случае блок PREEMPT работает аналогично блоку SEIZE. Если при входе транзакта в блок PREEMPT устройство занято другим транзактом, то в этом случае транзакт входит в блок PREEMPT, а устройство прерывает обслуживание занимающего его транзакта и переключается на обслуживание транзакта, вошедшего в блок PREEMPT. При этом из состояния «занято» устройство переходит в состояние «захвачено». Когда транзакт, захватывающий устройство, освободит его, устройство возобновит прерванное обслуживание другого транзакта и перейдет в состояние «занято».

Если прерываемый транзакт в момент прерывания находится в блоке ADVANCE, то, начиная с момента прерывания, отсчет времени пребывания транзакта в этом блоке прекращается до тех пор, пока не будет восстановлено обслуживание транзакта. Таким образом, в момент восстановления прерванного обслуживания транзакта время, оставшееся этому транзакту до выхода из блока ADVANCE, такое же, каким оно считалось непосредственно в момент преры-

вания. Такое прерывание обслуживания называется прерыванием с последующим дообслуживанием.

Все транзакты, задержанные при обращении к устройству, упорядочиваются по приоритету. Кроме поля *A*, в блоке PREEMPT могут быть заданы операнды *B*, *C*, *D* и *E*. Операнд *B* записывается в виде обозначения PR, задающего приоритетный режим работы блока. В этом режиме транзакт захватывает устройство, если оно свободно или обслуживает менее приоритетный транзакт. Прерывание обслуживания менее приоритетного транзакта происходит с последующим дообслуживанием.

Для определения последующего движения прерванных транзактов используются другие операнды. В поле *C* может быть указана метка какого-либо блока, на который будет передан прерванный транзакт. При этом прерванный транзакт продолжает претендовать на данное устройство. В поле *D* блока может быть задан номер параметра транзакта. Тогда, если прерванный транзакт находится в блоке ADVANCE, то вычисляется остаток времени обслуживания (время дообслуживания), и полученное значение помещается в параметр, заданный в поле *D*. Прерванный транзакт при этом будет послан в блок, указанный в поле *C*. Прерванный транзакт продолжает претендовать на данное устройство. Если в поле *E* блока записано обозначение RE, то прерванный транзакт больше не будет претендовать на данное устройство.

Необходимо помнить, что если поле *E* не задано, а поле *C* указано, то прерванный транзакт не может быть уничтожен до тех пор, пока он явно не освободит устройство (он должен пройти либо блок RELEASE либо блок RETURN). Чтобы не забывать явно освободить устройство, обычно поля *C* и *E* применяют одновременно.

Блок RETURN —освободить устройство. Этот блок используется в паре с блоком PREEMPT. Если транзакт захватил устройство посредством блока PREEMPT, то освободить его он может только в блоке RETURN. Имя освобождаемого устройства задается в поле *A* блока.

Пример 2.12

Детали поступают в цех обработки в среднем каждые 5±2 минуты. Мастер обрабатывает детали в среднем 4±1 минуту. Каждые 30±5 минут приходит срочный заказ на обработку деталей второго типа, для обработки которых мастеру требуется еже 10±3 минуты. Детали второго типа имеют безусловный приоритет. Если в момент прихода детали второго типа мастер обрабатывает деталь первого типа, то он прерывает свою работу, текущую деталь передает для доделки своему ученику, а сам принимается за обработку вновь поступившей детали второго типа. Ученику требуется в два раза больше времени на обработку (или доработку) детали. Если у мастера скапливается очередь из деталей больше двух, то вновь приходящие детали первого типа также передаются на обработку ученику. Детали второго типа обрабатываются только мастером. Если в момент прерывания обработки детали первого типа мастеру не хватило меньше 30 секунд, то считается, что деталь обработана.

Промоделировать работу мастера и ученика в течение 4-х часов. Проверить, не будет ли скапливаться очередь у ученика. Определить загрузку мастера и ученика.

Для моделирования процесса обработки деталей учеником введем переменную VAR (чтобы увеличить время обработки в два раза).

Функция OBSLU используется для определения времени обработки деталей первого типа. Данная функция описывает равномерный закон распределения на интервале от 3 до 5.

```
VAR  VARIABLE  P1#2
OBSLU  FUNCTION  RN1,C2
0,3/1,5
; работа мастера с деталями первого типа
GENERATE  5,2              ; поступление деталей
ASSIGN  1,FN$OBSLU     ; определение времени обработки
TEST  LE  Q$MAS,2,UCH ; если скопилась очередь – деталь
                                ; направляется к ученику
QUEUE  MAS
```

```
SEIZE  MAST
DEPART  MAS
ADVANCE  P1
RELEASE  MAST
TERMINATE
; работа мастера с деталями второго типа
GENERATE  30,5,,,1
QUEUE  MAS
PREEMPT  MAST,PR,UCH,1,RE          ;прерванная деталь 1-го типа
                                    ; направляется к ученику
DEPART  MAS
ADVANCE  10,3
RETURN  MAST
TERMINATE
; работа ученика
UCH  TEST  G  P1,0.5,UXOD      ; если время дообработки меньше
                                ;30 секунд, то  деталь не обрабатывается
SEIZE  UCHEN
ADVANCE  V$VAR
RELEASE  UCHEN
UXOD  TERMINATE
; таймер
GENERATE  240
TERMINATE  1
START  1
```

2.4.17. Организация циклов

Для организации циклов используется *блок LOOP*. Поле *A* этого блока содержит имя или номер параметра, который выполняет функцию счетчика циклов. Каждый раз при поступлении транзакта в блок LOOP из указанного пара-

метра вычитается единица, и полученная разность снова записывается в данный параметр. Как только значение параметра становится равным нулю, транзакт направляется в блок, следующий за блоком LOOP. Если значение параметра остается положительным, то транзакт направляется к блоку, указанному в поле *B*.

Например, если необходимо, чтобы через блок ADVANCE все транзакты проходили по 10 раз, то этот процесс можно промоделировать следующим образом.

```
ASSIGN  1,10
POVT ADVANCE  34,12
LOOP  1,POVT
```

2.4.18. Работа с группами

В языке рассматривается два типа групп: Группы Транзактов (Transaction Groups) и Числовые Группы (Numeric Groups). С группами связаны следующие СЧА:

GN$<имя группы> - количество элементов в Числовой Группе;

GT$<имя группы> - количество элементов в Группе Транзактов.

Рассмотрим блоки для работы с указанными группами.

Блок JOIN добавляет элемент в группу. В поле *A* блока указывается имя группы. Если поле *B* не указано, то рассматривается Группа Транзактов и блок добавляет транзакт в группу *A*. Если используется поле *B*, то оно имеет числовое значение, которое добавляется в качестве элемента в Числовую Группу *A*. Если транзакт или число уже является элементом группы, то никаких действий не производится.

Блок ALTER предназначен для изменения приоритета или значения параметра у элементов Группы Транзактов. Блок имеет следующую структуру:

```
ALTER  O  A,B,C,D,E,F,G
```

Поле *O* может быть пустым или содержит условный оператор со следующими возможными значениями: “L” – ”<”, “LE” – “≤”, “E” – “=”, “NE” – “≠”, “G” – “>”, “GE” – “≥”, “MAX”, “MIN”.

Поле *A* блока содержит имя Группы Транзактов, которая будет тестироваться. Поле *B* содержит максимальное число тестируемых транзактов (по умолчанию просматривается вся Группа). В поле *C* указывается имя или номер параметра транзакта, который должен быть изменен. Если значение этого поля равно PR, то изменяться будет приоритет транзакта. Поле *D* содержит значение, которое должно быть присвоено параметру, указанному в поле *C*. Поле *E* имеет значение PR или номер/имя параметра транзакта, который проверяется на условие. Поле *F* содержит значение, с которым сравнивается параметр из поля *E*. Поле *G* содержит имя блока. Поля *E*, *F* и *G* используются только совместно с полем *O*.

Если поле *O* пропущено, то *B* транзактам из группы *A* в параметр *C* записывается значение *D*. Например:

ALTER Spis,5,First,3.5

Здесь для пяти первых транзактов из группы Spis в параметр с именем First запишется значение 3.5.

Если поле *O* задано, то в параметр *C* записывается значение *D* только в том случае, если между *E* и *F* выполняется заданное условие отношения. Из Группы Транзактов выбираются последовательно элементы до тех пор, пока не найдется *B* транзактов, для которых выполнится заданное условие. Транзакт, вошедший в блок ALTER, переходит к блоку *G*, если не находится *B* транзактов, для которых выполняется условие тестирования. Например:

ALTER NE Bin,10,Price,49.95,Part,99.95,Out

Здесь, когда активный транзакт попадает в блок ALTER, в Группе Транзактов с именем Bin ищутся транзакты, у которых значение параметра Part не равно 99.95. Для первых десяти найденных транзактов значение параметра Price устанавливается равным 49.95. Если в группе не находится десяти транзактов с указанным условием, то активный транзакт напрвляется к блоку Out. В противном случае активный транзакт переходит к следующему блоку.

Если в поле *O* используются значения MAX или MIN, то поле *F* не используется, а значение параметра, указанного в поле *E*, должно быть равно максимальному (минимальному) значению этого параметра для всей группы.

Блок EXAMINE используется для проверки элементов группы. В поле A блока указывается имя группы. Если поле B не указано, то рассматривается Группа Транзактов, если используется поле B, то оно задается числом и рассматривается Числовая Группа. Если поле B не указано, то проверяется, является ли активный транзакт элементом группы A. Если является, то он направляется к следующему за EXAMINE блоку, если нет – то транзакт направляется к блоку, имя которого указано в поле C. Если задано поле B, то проверяется, является ли заданное в этом поле число элементом группы. Если является, то активный транзакт направляется к следующему за EXAMINE блоку, если нет – то транзакт направляется к блоку, имя которого указано в поле C.

Блок REMOVE исключает элементы группы по заданному условию. Блок имеет следующую структуру:

REMOVE O A,B,C,D,E,F,G

Поле O может быть пустым или содержит условный оператор со следующими возможными значениями: “L” – ”<”, “LE” – “≤”, “E” – “=”, “NE” – “≠”, “G” – “>”, “GE” – “≥”, “MAX”, “MIN”.

В поле A блока указывается имя группы. Тип группы определяется полем C. Если поле C не задано, то рассматривается Группа Транзактов. Если поле C задано, то рассматривается Числовая группа, а в поле C указывается число, которое должно быть исключено из группы. Если указанное число найдено в Числовой группе, определенной полем A, то оно исключается из группы, а активный транзакт направляется к следующему за REMOVE блоку. Если число, указанное в поле C не является элементом группы и используется поле F, то активный транзакт направляется к блоку, имя которого указано в F. Для Числовых Групп используются только поля A, C и F.

Если рассматривается Группа Транзактов и не определены поля B, D и E (режим самоисключения), то из группы исключается активный транзакт (тот транзакт, который вошел в блок REMOVE). простейший пример:

REMOVE Self

Транзакт, вошедший в блок REMOVE, исключается из Группы Транзактов с именем Self, если он является членом этой группы.

В режиме тестирования группы оператор условия может быть задан или нет. Если определен оператор условия O, то он задает отношение между атрибутом транзакта или его приоритетом (поле D) и значением, которое определено в поле E. Из Группы Транзактов исключаются все (по умолчанию) или B штук транзактов, для которых выполняется заданное условие. Если оператор условия не определен, но заданы поля D и E, ищутся и исключаются те транзакты, у которых значение атрибута поля D равно значению E. Если в операторе условия заданы значения MAX или MIN, то исключаются транзакты с максимальным или минимальным значением атрибута D.

В поле B указывается максимальное количество элементов, исключаемых из группы (по умолчанию – все). Если поле D не задано, то просто B штук транзактов исключается из группы.

В поле F указывается имя блока, куда направляется активный транзакт, если

- в режиме самоисключения активный транзакт не является элементом группы;
- в режиме тестирования группы не исключен ни один транзакт;
- в режиме тестирования группы количество исключенных транзактов не достигло B.

Если поле F не определено, то активный транзакт всегда переходит к следующему блоку.

Например:

REMOVE G 3,10,,20,11.4,Jump

Здесь в Группе Транзактов 3 ищутся транзакты, у которых значение 20-го параметра превышает 11.4. Первые 10 транзактов, для которых выполняется указанное условие, исключаются из группы. Если 10 транзактов не найдено, то активный транзакт направляется к блоку Jump. В противном случае, транзакт переходит к следующему блоку.

Блок SCAN предназначен для поиска информации в Группе Транзактов для присвоения ее активному транзакту. Блок имеет следующую структуру:

SCAN O A,B,C,D,E,F

Поле *O* может быть пустым или содержит условный оператор со следующими возможными значениями: “L” – ”<”, “LE” – “≤”, “E” – “=”, “NE” – “≠”, “G” – “>”, “GE” – “≥”, “MAX”, “MIN”.

В поле *A* блока указывается имя группы. В поле *B* содержится имя тестируемого параметра или PR (если необходимо тестировать приоритет транзакта). В поле *C* содержится значение, которое сравнивается с *B* по заданному условию. В поле *D* указывается имя параметра транзакта группы, чье значение будет присваиваться активному транзакту. Поле *E* содержит имя параметра активного транзакта, в который должно быть записано значение параметра *D*. В поле *F* указывается имя блока, к которому направляется активный транзакт, если по заданному условию в группе не найдено ни одного транзакта.

Блок SCAN находит первый транзакт в Группе, который удовлетворяет заданным условиям, и значение его атрибута, определенного в поле *D*, присваивает атрибуту активного транзакта, определенного в поле *E*. В блоке обязательными являются поля *D* и *E*. Если не указаны поля *B*, *C* и условный оператор (никакого тестирования не производится), то из Группы выбирается первый же транзакт.

Если не используется условный оператор, но используются поля *B* и *C*, то в группе ищется транзакт, у которого значение атрибута *B* равно *C*.

Например:

SCAN E Lot,Part,127,Price,Sum,Phone

В данном примере, как только активный транзакт поступает в блок SCAN, в Группе Транзактов Lot ищется транзакт со значением атрибута Part равным 127. Если такой транзакт находится, значение его атрибута Price, присваивается атрибуту Sum активного транзакта. Если в группе не найдено транзакта с заданным условием, то активный транзакт направляется к блоку Phone.

2.4.19. Системное время

СЧА, связанные с системным временем, используются для проверки временных соотношений пребывания транзактов в системе.

В СЧА C1 и AC1 хранится текущее значение системного времени. СЧА C1 содержит значение относительного системного времени (с момента последнего блока RESET). СЧА AC1 содержит значение абсолютного системного времени (с момента последнего блока CLEAR). Данные СЧА доступны пользователю в любой точке программы.

Каждый транзакт при генерации снабжается отметкой времени. Время пребывания транзакта в модели содержится в СЧА M1 или MP и отсчитывается от момента рождения:

M1 = AC1 - <дата рождения>.

СЧА M1 возвращает время пребывания транзакта в модели, СЧА MPi или MP$<имя параметра> возвращает значение, равное абсолютному системному времени минус значение соответствующего параметра транзакта.

«Дату рождения», зафиксированную блоком GENERATE, можно изменить в любом месте программы, используя *блок MARK* с пустым полем *A*. Блок MARK с пустым полем *A* изменяет «дату рождения» на текущее системное время. Если в блоке MARK используется поле *A*, то в этом поле содержится номер или имя параметра транзакта. В этом случае транзакт сохраняет «дату рождения», а в указанном параметре записывается текущее значение AC1. Работа блока

```
MARK  10
```

эквивалентна работе блока

```
ASSIGN  10,C1
```

Например:

```
. . . . .
MARK
. . . . .
MARK  10
```

. . . . .

TEST E C1,50,BGN1

SR1 TEST GE M1,MP10,BGN2

Здесь первый блок TEST проверяет условие, связанное с текущим значением системного времени (C1 = 50 ?). Второй блок TEST сравнивает M1 (время от попадания транзакта в блок MARK до попадания его в блок SR1) с MP10 (время от попадания транзакта в блок MARK 10 до попадания его в блок SR1).

Пример 2.13

В специализированный магазин промышленных товаров покупатели приходят в среднем каждые 15 минут. В магазине работает единственный продавец, который обслуживает покупателей в среднем 8 минут. Время прихода и время обслуживания подчиняется экспоненциальному закону. Известно, что примерно 20% покупателей в течение месяца приходят в магазин повторно.

В магазине действует накопительная система: если в течение месяца покупатель набирает товара на сумму, превышающую 800 рублей (по чекам), то он получает накопительную карту, по которой ему предоставляются скидки на покупки. Причем процент скидок зависит от общей накопленной на карте суммы. Кроме того, в конце месяца, покупателю, набравшему товара на максимальную сумму, в качестве премии начисляется на карту дополнительная сумма, равная сумме на карте, что позволяет ему в следующем месяце сразу получать максимальную скидку на стоимость товара.

Промоделировать работу в течение месяца и определить размер премии «лучшего» покупателя.

Для моделирования суммы покупок определим случайную функцию CONT, считая, что стоимость покупки может варьироваться в размере от 5 до 500 рублей.

Модель работы магазина будет выглядеть следующем образом:

```
CONT  FUNCTION  RN1,C2
0,5/1,500
; приход посетителей в магазин
GENERATE  (EXPONENTIAL(1,0,15)),,,,1
VXOD  ASSIGN  PRICE+,FN$CONT      ; общая сумма покупок
                                  ; посетителя магазина
QUEUE  OCH
SEIZE  PROD
DEPART  OCH
ADVANCE  (EXPONENTIAL(1,0,8))
TEST  G  P$PRICE,800,NEXT         ; если общая сумма превышает 800
JOIN  CART              ; посетителю выдается карта
RELEASE  PROD
TRANSFER  .2,POVT
TEST  G  C1,14400                 ; задерживаем транзакты группы, чтобы в
                ; конце месяца определить лучшего покупателя
TERMINATE
NEXT  RELEASE  PROD
TRANSFER  .2,,POVT
TERMINATE
; перед повторным посещением ставим задержку
POVT   ADVANCE  (EXPONENTIAL(1,0,600))
TRANSFER  ,VXOD
; определение лучшего покупателя в конце месяца
GENERATE  14400
SCAN  MAX  CART,PRICE,,PRICE,SUM
ASSIGN  SUM+,P$SUM
SAVEVALUE  20,P$SUM
TERMINATE  1
START  1
```

2.4.20. Работа с потоками данных

Ряд блоков языка предназначен для создания и работы с потоками данных. Потоки данных можно использовать для работы с текстовыми файлами, создаваемыми на диске, или для хранения данных в памяти. Язык GPSS позволяет работать одновременно с несколькими потоками данных (в дальнейшем будем обозначать их DS), каждому из которых для идентификации присваивается числовой номер – целое число. Рассмотрим эти блоки.

Блок OPEN создает поток данных. Поле *A* этого блока содержит имя файла в виде текстовой строки, с которым будет идентифицироваться создаваемый DS или пустую строку, если DS создается в памяти (только на время работы модели). Если файла с указанным именем на диске не существует, то он создается. В поле *B* указывается числовой идентификатор DS. Числовой идентификатор может быть задан в виде любого целого положительного числа и используется в дальнейшем для работы с DS. Поле *C* (не обязательно) содержит имя блока, куда направляется транзакт, если не удается создать DS (код ошибки создания DS не равен нулю). Если открывается существующий файл, то указатель устанавливается на начало файла.

Блок CLOSE уничтожает поток данных (закрывает файл) и возвращает код ошибки. В поле *A* (не обязательно) блока указывается имя/номер атрибута транзакта, в который записывается код ошибки. Заметим, что проанализировать код ошибки работы с потоком данных можно только после использования блока CLOSE с полем *A*. Поле *B* содержит числовой идентификатор DS. Поле *C* содержит имя блока, к которому направляется транзакт, если код ошибки не равен нулю.

Блок READ читает текстовую строку из DS с текущей позиции. После чтения указатель перемещается на следующую строку. В поле *A* блока указывается имя/номер атрибута транзакта, в который записывается прочитанная строка. Поле *B* содержит числовой идентификатор DS из которого читается информация. Поле *C* содержит имя блока, к которому направляется активный транзакт, если произошла ошибка чтения или дошли до конца файла.

Блок WRITE помещает текстовую строку в текущую позицию DS. Поле *A* содержит текстовую строку, которая записывается в поток данных. Может содержать число, строку, любой СЧА. Если в поле указано не строковое значение (число или СЧА), то сначала это значение преобразуется в строку, а затем помещается в DS. Поле *B* содержит числовой идентификатор DS. Поле *C* содержит имя блока, к которому направляется активный транзакт, если произошла ошибка записи. Поле *D* указывает режим записи и может принимать одно из двух значений: ON – режим вставки, OFF – режим замены. По умолчанию работает режим вставки.

Блок SEEK устанавливает новую текущую позицию (перемещает указатель) для DS. Поле *A* блока содержит номер новой текущей позиции (номер строки, на которую должен быть перемещен указатель). Поле *B* содержит числовой идентификатор DS. Заметим, что если номер новой текущей позиции, указанный в поле *A* превышает количество строк в потоке данных, то указатель просто устанавливается на конец DS и никакой ошибки не происходит.

Рассмотрим возможные коды ошибок, которые могут возникнуть при работе с потоками данных, и их интерпретацию:

0 – нет ошибки;

10 – ошибка OPEN (слишком длинное имя файла – более 200 символов);

11 – ошибка OPEN (ошибка чтения внешнего файла – не смогли загрузить в память);

12 – ошибка OPEN (не хватило памяти для файла);

21 – ошибка READ (не хватило памяти);

22 – ошибка READ (поток не открыт);

31 – ошибка WRITE (не хватило памяти);

32 – ошибка WRITE (поток не открыт);

41 – ошибка CLOSE (не смогли записать файл на диск);

43 – ошибка CLOSE (поток не открыт);

51 – ошибка SEEK (поток не открыт).

Пример 2.14

Посетители заходят в приемную в среднем каждые 6 минут, где с ними работает секретарь в среднем 3 минуты. Время прихода и время обслуживания подчиняется экспоненциальному закону распределения.

Необходимо промоделировать работу приемной в течение 8-часового рабочего дня. Время нахождения посетителей в очереди запишем в файл на диск для дальнейшего анализа и исследований.

Время нахождения посетителей в очереди будем определять по времени пребывания транзакта в модели M1 (с момента прихода в систему – постановка в очередь, по момент выхода из очереди). Модель работы системы:

```
; создание файла и потока данных
GENERATE  ,,,1
OPEN  ("MY.TXT"),1,ERROR
TERMINATE
; приход и обслуживание посетителей
GENERATE  (EXPONENTIAL(1,0,6))
QUEUE  OCH
SEIZE  CAN
DEPART  OCH
WRITE  M1,1,ERROR ; запись в файл времени
                  ; пребывания транзакта в очереди
ADVANCE  (EXPONENTIAL(1,0,3))
RELEASE  CAN
TERMINATE
; таймер
GENERATE  480
ERROR  CLOSE  OSH,1
SAVEVALUE  1,P$OSH                ; определение ошибки работы с DS
TERMINATE  1
START  1
```

2.4.21. Управляющие блоки

Блок START воспринимается как команда симулятору начать выполнение прочитанной части модели. В этом блоке в поле *A* задается начальное значение счетчика транзактов. Здесь также может быть использовано поле *B* в значении NP, что означает — не выводить статистику по окончании моделирования. Если задан блок

START 1,NP ,

то подавляется вывод стандартного отчета: всей информации об устройствах, памятях, очередях, таблицах и ячейках.

Содержимое счетчика транзактов уменьшается при входе транзактов в блок TERMINATE. Когда значение счетчика становится равным нулю или отрицательным, производится выдача статистики и заканчивается процесс моделирования.

Блок RESET предназначается для стирания в заданный момент времени статистики о предыстории процесса. Достигнутое состояние объектов при этом сохраняется.

Применение блока RESET позволяет уменьшить затраты машинного времени на сбор статистики о стационарном (в смысле вероятностных характеристик) процессе в тех случаях, когда предшествующий ему переходный процесс вносит заметные искажения в накапливаемую статистику.

Обычно блок RESET помещается в модели после блока START, а после RESET располагается следующий блок START. После прочтения блока RESET засылается нулевое содержимое в счетчики числа входов в блоки, коэффициенты использования устройств и памятей, а также обнуляются все накопленные статистики. При этом сохраняются текущие состояния и значения устройств, памятей, очередей, ячеек и датчиков случайных чисел.

Блок CLEAR переводит всю модель — всю статистику и все объекты — в исходное состояние. Исключением является лишь датчик случайных чисел — он не возвращается к начальному значению. Применение блока CLEAR позволяет осуществить независимые реализации моделируемого случайного процесса.

Пример 2.15

Моделирование работы заправочной станции.

Заправочная станция открыта с 7 часов до 19 часов. Машины, поступившие после 19 часов, не обслуживаются. Тем не менее все машины, попавшие в очередь до 19 часов, должны быть обслужены. Машины останавливаются на обслуживание лишь в том случае, если число ожидающих обслуживания автомашин меньше или равно числу обслуживаемых машин (т.е. не более одной машины в очереди на колонку). Провести моделирование для 1, 2 и 3 колонок в течение дня с целью определения такого их числа, при котором достигается максимальная прибыль.

С одной обслуженной машины получают доход 10 рублей. Расходы на содержание одной колонки составляют 700 рублей. Предусмотреть в модели, что в случае возникновения временного узла между событиями завершения обслуживания машины и прибытием другой машины, завершение обслуживания было бы обработано в первую очередь: это эквивалентно предположению о том, что водитель прибывшей машины видит возможность завершения обслуживания, т.е. может предпочесть остаться. Если есть временной узел между событиями поступления автомашины на станцию и закрытием заправочной станции в конце дня, прибывшая машина должна попасть до закрытия.

Пусть прибытие машин описывается простейшим потоком со средним временем между приходом машин равным 1 минуте. Время обслуживания равно 1±0.5 минут.

```
STR  STORAGE  1            ; работает 1 колонка
PRB  VARIABLE  N$DONE-(R$STR+S$STR)#70
GENERATE  (EXPONENTIAL(1,0,1)),,,,1    ; приход автомашины
GATE  LR  LOCK             ; если рабочий день еще не закончен,
   ;то машина поступает на обслуживание
TEST  LE  Q$LIN,S$STR,BYE
BEG  QUEUE  LIN
ENTER  STR
DEPART  LIN
```

```
PRIORITY  2
ADVANCE  1,0.5
DONE  LEAVE  STR
BYE  TERMINATE
GENERATE  720
LOGIC  S  LOCK              ; рабочий день закончился
TEST  E  N$BEG,N$DONE       ; если все машины обслужены
SAVEVALUE  1,V$PRB   ; подсчет прибыли
TERMINATE  1
START  1
CLEAR
STR  STORAGE  2             ; работает 2 колонки
START  1
CLEAR
STR  STORAGE  3             ; работает 3 колонки
START  1
```

Переменная PRB используется для расчета прибыли заправочной станции. Временные узлы между событиями обрабатываются через приоритеты.

Примечание: в результате прогона модели будут получены три стандартных отчета – по одному на каждый блок START. Каждый отчет размещается в отдельном файле выдачи.

2.5. Внутренняя организация GPSS

Система GPSS в целом как программный продукт состоит из ряда модулей, из которых только модуль управления (симулятор) находится постоянно в ОЗУ и осуществляет процесс имитации. Динамика функционирования симулятора основана фактически на схеме событий, при этом событием считается любое изменение состояния моделируемой системы. Основной функцией симулятора является поддержание правильного хода часов системного времени и выяснение возможностей продвижения транзактов в программе модели. Симулятор

оперирует с рядом информационных структур, основными из которых являются: список будущих событий (FEC), список текущих событий (CEC), список прерываний, список задержанных транзактов и другие списки.

Работа симулятора разделяется на три основные фазы:

1) изменение значения системного времени;

2) просмотр списка текущих событий;

3) движение сообщений.

Фаза «Изменение значения системного времени» (рис. 2.3) выполняется симулятором всегда, когда на текущий момент системного времени ни одно из активных сообщений, находящихся в CEC, не может быть продвинуто в программе модели и, кроме того, состояние системы не может быть изменено.

Выбирая первое сообщение, симулятор присваивает системному времени STIME время очередной передвижки этого сообщения TEV(H) в программе модели и перемещает его в CEC. Подобная процедура осуществляется для всех событий в FEC, время наступления которых равно TEV(H), т.е. текущему значению системного времени. При этом после просмотра в FEC останутся события, время наступления которых больше STIME, т.е. события, наступающие в будущем.

Фаза «Просмотр списка текущих событий» (рис. 2.4). Установив флаг изменения состояния системы в ноль, симулятор в зависимости от значения индикатора просмотра сообщения (транзакта) — 0 или 1 — решает вопрос: передать сообщение на третью фазу или нет. На *фазу «Движение сообщений»* (рис.2.5) передаются только активные сообщения, индикатор просмотра которых равен нулю. Пассивные сообщения находятся в состоянии задержки, например, по причине занятости имитируемого оборудования. Такие сообщения не попадут на третью фазу до тех пор, пока соответствующее оборудование не будет освобождено, т.е. пока не изменится состояние системы.

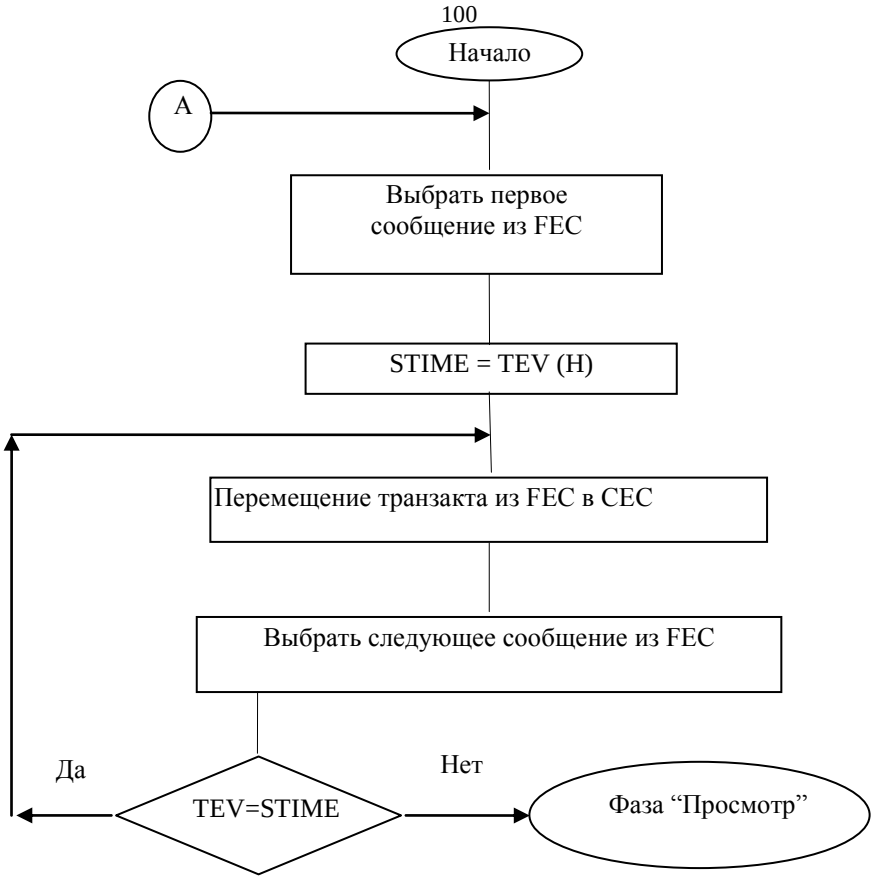


Рис. 2.3. Фаза «Изменение значения системного времени»

На *фазе «Движение сообщений»* активные сообщения симулятор пытается продвинуть как можно дальше по программе модели. Если при этой передвижке меняется состояние системы, все пассивные сообщения, находящиеся в CEC и задержанные по той или иной причине, получают статус активных. Их индикаторы просмотра устанавливаются в “0”. Если же при передвижке сообщений явно задана задержка, то сообщение перемещается в FEC. Таким образом, на третьей фазе происходит передвижка активных сообщений, изменение состояния системы, пересмотр индикаторов сообщений и планирование будущих событий (перемещение в FEC).

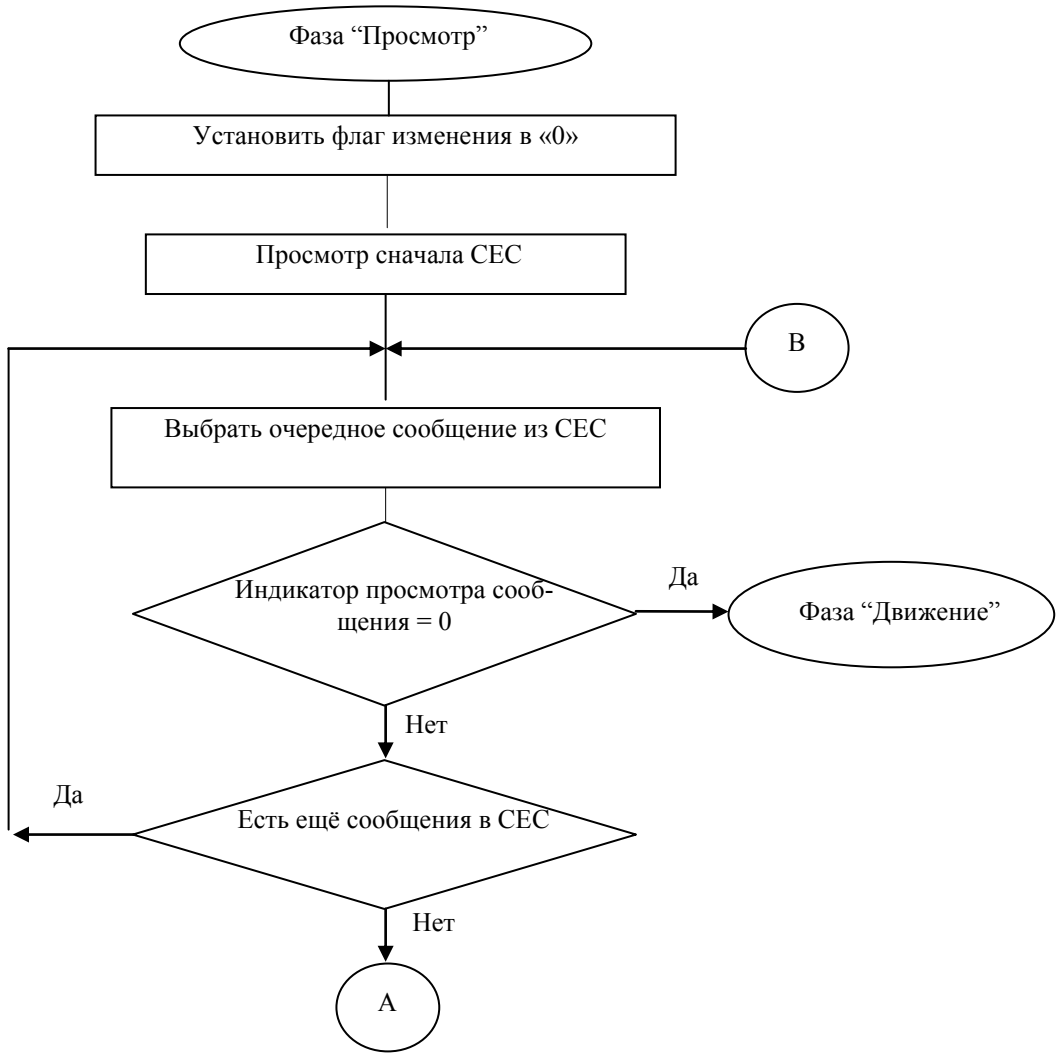


Рис. 2.4. Фаза “Просмотр списка текущих событий”

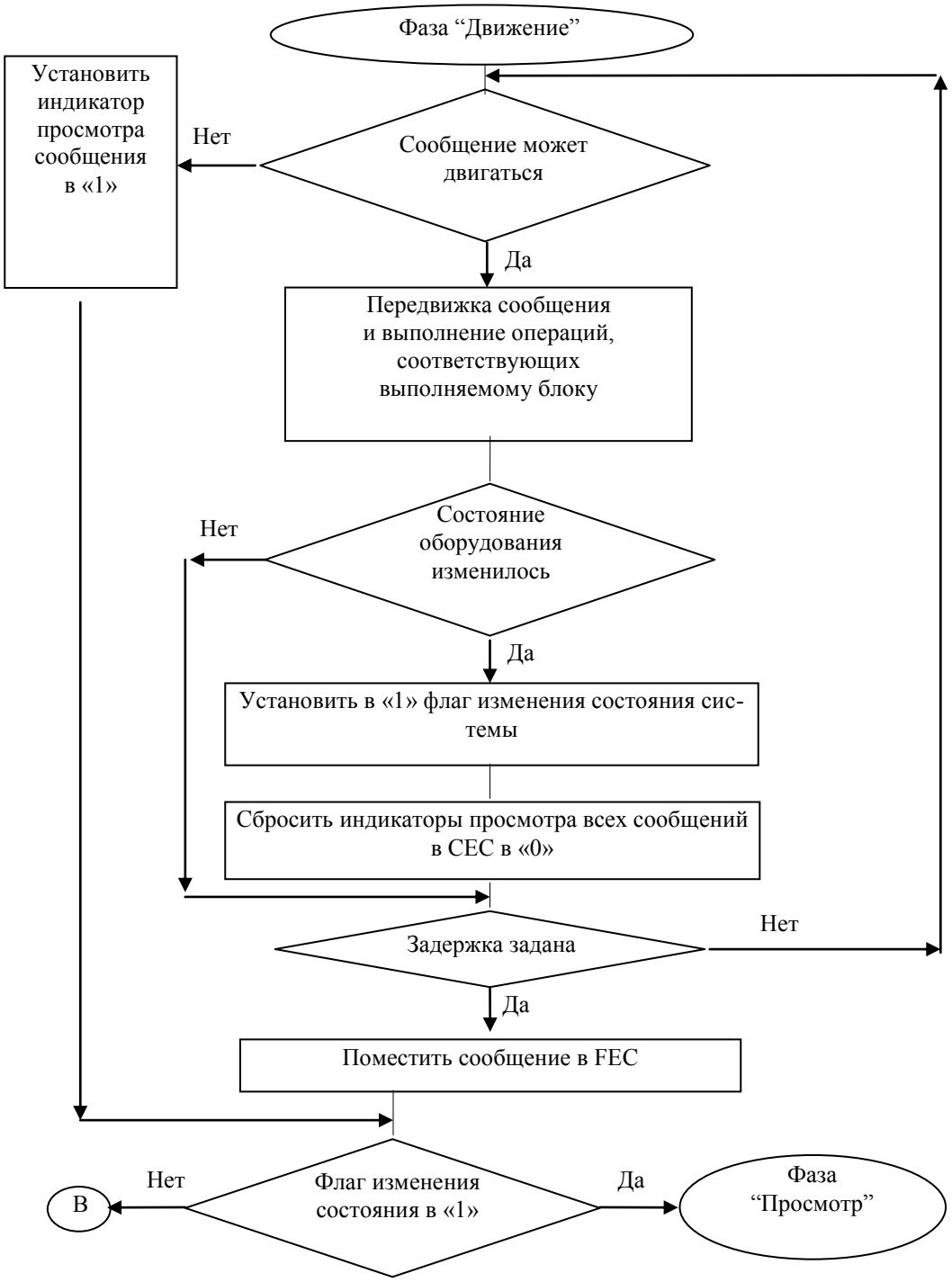


Рис. 2.5. Фаза “Движение сообщений”

Вопросы для самопроверки

1. Что такое транзакт?
2. Чем отличается работа устройства и работа памяти?
3. Перед какими блоками образуются физические очереди транзактов?
4. Для чего служат блоки работы с очередями?
5. С помощью каких объектов языка можно организовать счетчики транзактов?
6. На что влияет приоритет транзакта?
7. Как можно организовать обслуживание транзактов по приоритету?
8. С помощью каких блоков осуществляется синхронизация транзактов?

Заключение

В данном учебном пособии рассмотрены основные понятия имитационного моделирования. Сегодня понятия «имитационное моделирование» и «компьютерное моделирование» практически не различимы просто в силу того, что сам процесс построения имитационных моделей связан с построением алгоритмов и, соответственно, написания программ. Именно поэтому в данном учебном пособии большое место занимает рассмотрение одного из наиболее распространенных языков моделирования GPSS.

При описании языка автор попытался как можно полнее раскрыть его возможности и особенности на многочисленных примерах. Тем не менее, данный язык имеет намного больше возможностей и средств для построения имитационных моделей, чем это было представлено в учебном пособии. Задачей автора было показать подход построения имитационных моделей средствами, отличными от алгоритмических языков, а также преимущества и удобства такого подхода.

Автор надеется, что данный в пособии материал позволит читателю применять свои знания на практике, а при необходимости, в дальнейшем изучить недостающие разделы самостоятельно.

ЛИТЕРАТУРА

1. Полищук Ю.М. Имитационно-лингвистическое моделирование систем с природными компонентами. — Новосибирск: Наука, 1992.

2. Шеннон Р. Имитационные моделирование систем — искусство или наука. —М.: 1981.

3. Нургужин М.Р., Яворский В.В. Компьютерное моделирование систем. – Караганда: Изд-во КарГТУ, 2006. – 200 с.

4. Советов Б.Я., Яковлев С.А. Моделирование систем.— М.: Высшая школа, 1998.

5. Разработка САПР. Кн. 9. Имитационное моделирование: Практ. пособие / В.Н. Черненький: Под ред. А.В. петрова. — М.: Высш. шк., 1990.

6. Шрайбер Г. Дж. Моделирование на GPSS. — М.: Высш. шк., 1980.

7. Салмина Н.Ю. Моделирование систем. Учебное пособие. – Томск: ТУСУР, 2003. – 197 с.

Глоссарий

Арифметическая переменная – вычислительный объект языка GPSS, позволяющий выполнить заданную последовательность арифметических операций над любыми стандартными числовыми атрибутами модели для вычисления значения зависимого от них параметра.

Атрибут транзакта – параметр транзакта, характеризующий какие-то свойства транзакта. В процессе имитации значения параметра могут быть изменены. Каждый транзакт обладает совокупностью до 100 таких параметров.

Имитационная модель – логико-математическое описание объекта, которое может быть использовано для экспериментирования на компьютере в целях проектирования, анализа и оценки функционирования объекта.

Логический переключатель – объект языка моделирования GPSS, имитирующий единицу оборудования, которая может находится только в двух положениях: «включен» и «выключен». Перед началом выполнения программы все переключатели устанавливаются в положение «выключен».

Метод нелинейных преобразований – метод генерации случайных величин с заданным законом распределения, основанный на получении обратной функции, преобразующей равномерно распределенную случайную величину в искомую. Является универсальным, точным.

Метод кусочной аппроксимации – метод генерации случайных величин с заданным законом распределения, основанный на аппроксимации заданной функции плотности распределения равномерным законом. Является универсальным, приближенным.

Матрица – вычислительный объект языка GPSS, который служит для хранения некоторых постоянных и/или изменяющихся значений данных программы в виде массивов.

Модель – объект-заместитель, который в определенных условиях может заменять объект-оригинал, обеспечивая изучение некоторых свойств и характеристик оригинала.

Очередь – статистический объект языка моделирования GPSS, связанный со сбором статистики о задержках, возникающих на пути прохождения транзакта.

Память (накопитель) – объект языка моделирования GPSS, имитирующий единицу оборудования, в которой может обрабатываться (храниться) несколько транзактов одновременно. Память позволяет легко моделировать средства обработки с ограниченной емкостью (стоянки автотранспорта, портовые причалы, складские помещения, конвейеры и т.п.).

Семейство транзактов – множество транзактов, состоящее из исходного транзакта и всех его копий. Копия члена семейства является членом того же семейства. Любой транзакт — член только одного семейства.

Стандартный числовой атрибут (СЧА) – атрибут объекта языка GPSS, однозначно определяющий его статус. СЧА меняется в процессе имитации, изменить его значение может как симулятор, так и пользователь. Для указания конкретного объекта, по которому необходимо получить требуемую информацию, за именем СЧА должно следовать числовое или символьное имя этого объекта.

Симулятор – комплекс программ, планирующий выполнение событий, реализующий функционирование блоков моделей, регистрирующий статистическую информацию о прохождении транзактов. Основной функцией симулятора является поддержание правильного хода часов системного времени и выяснение возможностей продвижения транзактов в программе модели.

Таблица – статистический объект языка моделирования GPSS, обеспечивающий накопление в процессе моделирования статистики о каком-либо заданном случайном параметре модели. По окончании прогона модели эта статистика автоматически обрабатывается и выводится на печать, в частности, в виде

таблицы относительных частот попадания значений случайного параметра (аргумента таблицы) в указанные частотные интервалы.

Транзакт – формальный объект, который "путешествует" по системе, встречая на пути всевозможные задержки, вызванные занятостью тех или иных единиц оборудования. В качестве транзакта может выступать программа обработки информации, телефонный вызов, покупатель в магазине, отказ системы при исследовании надежности и т.д.

Устройство – объект языка моделирования GPSS, имитирующее единицу оборудования, которое может одновременно обрабатывать только один транзакт. Оно служит для моделирования таких средств обработки элементов потоков, как станки, рабочие, каналы связи и т.п.

Язык GPSS – язык имитационного моделирования, ориентированный на решение задач статистического моделирования на ЭВМ процессов с дискретными событиями.

Язык моделирования представляет собой процедурно-ориентированный язык, обладающий специфическими чертами. Основные языки моделирования разрабатывались в качестве программного обеспечения имитационного подхода к изучению процесса функционирования определенного класса систем.

Ячейка – вычислительный объект языка GPSS, используемый для записи, накопления и хранения численных значений различных входных и выходных параметров моделируемой системы. Эти значения могут быть использованы для организации счетчиков числа проходящих транзактов, для вывода значений варьируемых параметров модели, для временного хранения значений стандартных числовых атрибутов.

Приложение

Отчет

Приведем пример стандартного отчета для задачи, рассмотренной в примере 2.10.

GPSS World Simulation Report - pasport.3.1

Friday, January 21, 2005 13:51:41

START TIME	END TIME	BLOCKS	FACILITIES	STORAGES
0.000	540.000	26	1	1

NAME	VALUE
BYE	17.000
NACH	10008.000
NO_OBSL	10009.000
OBED	10007.000
OCH_NACH	10002.000
OCH_PROP	10004.000
PROP	10000.000
RAZN	10005.000
TAB1	10001.000
TAB2	10003.000
TIME	10006.000
UXOD	16.000
VXOD	3.000

LABEL	LOC	BLOCK	TYPE	ENTRY COUNT	CURRENT COUNT	RETRY
	1	GENERATE		88	0	0
	2	GATE		88	0	0
VXOD	3	QUEUE		82	0	0
	4	GATE		82	0	0
	5	SEIZE		82	0	0

	6	DEPART	82	0	0
	7	ADVANCE	82	0	0
	8	RELEASE	82	0	0
	9	TRANSFER	82	0	0
	10	QUEUE	78	0	0
	11	GATE	78	0	0
	12	ENTER	78	0	0
	13	DEPART	78	0	0
	14	ADVANCE	78	0	0
	15	LEAVE	78	0	0
UXOD	16	TERMINATE	82	0	0
BYE	17	TERMINATE	6	0	0
	18	GENERATE	1	0	0
	19	LOGIC	1	0	0
	20	ADVANCE	1	0	0
	21	LOGIC	1	0	0
	22	ADVANCE	1	0	0
	23	LOGIC	1	0	0
	24	ADVANCE	1	0	0
	25	SAVEVALUE	1	0	0
	26	TERMINATE	1	0	0

FACILITY	ENTRIES	UTIL.	AVE. TIME	AVAIL.	OWNER	PEND	INTER	RETRY	DELAY
NACH	82	0.542	3.570	1	0	0	0	0	0

QUEUE	MAX	CONT.	ENTRY	ENTRY(0)	AVE.CONT.	AVE.TIME	AVE.(-0)	RE-TRY
OCH_NACH	13	0	82	31	2.367	15.589	25.064	0
OCH_PROP	3	0	78	53	0.431	2.982	9.305	0

STORAGE	CAP.	REM.	MIN.	MAX.	ENTRIES	AVL.	AVE.C.	UTIL.	RETRY	DE-LAY
PROP	3	3	0	3	78	1	1.643	0.548	0	0

TABLE	MEAN	STD.DEV.	RANGE	RETRY	FREQUENCY	CUM.%

TAB1	15.589	21.814			0		
			_	-	10.000	52	63.41
			10.000	-	20.000	5	69.51
			20.000	-	30.000	9	80.49
			30.000	-	40.000	1	81.71
			40.000	-	50.000	4	86.59
			50.000	-	60.000	3	90.24
			60.000	-	70.000	7	98.78
			70.000	-	80.000	1	100.00
TAB2	2.982	9.252			0		
			_	-	10.000	72	92.31
			10.000	-	20.000	2	94.87
			20.000	-	30.000	1	96.15
			30.000	-	40.000	1	97.44
			40.000	-	50.000	1	98.72
			50.000	-	60.000	1	100.00

LOGICSWITCH	VALUE	RETRY
TIME	1	0
OBED	0	0

SAVEVALUE	RETRY	VALUE
NO_OBSL	0	0

FEC XN	PRI	BDT	ASSEM	CURRENT	NEXT	PARAMETER	VALUE
90	0	540.784	90	0	1		

Ниже приведена смысловая интерпретация выдаваемых в отчете результатов.

Заголовок.

GPSS World Simulation Report - pasport.3.1

Friday, January 21, 2005 13:51:41

В заголовок включена информация об имени файла, из которого получен отчет, а также информация о времени и дате прогона модели.

Общая информация.

START TIME	END TIME	BLOCKS	FACILITIES	STORAGES
0.000	540.000	26	1	1

- START TIME. Абсолютное системное время на начало рассматриваемого периода. START TIME устанавливается равным абсолютному системному времени, определенному командами RESET или CLEAR.
- END TIME. Абсолютное системное время на момент окончания моделирования.
- BLOCKS. Количество блоков в программе, исключая блоки описания.
- FACILITIES. Количество объектов «устройство» в программе.
- STORAGES. Количество объектов «память» в программе.

Имена.

NAME	VALUE
BYE	17.000
NACH	10008.000
NO_OBSL	10009.000
OBED	10007.000
OCH_NACH	10002.000
OCH_PROP	10004.000
PROP	10000.000
RAZN	10005.000
TAB1	10001.000
TAB2	10003.000
TIME	10006.000
UXOD	16.000
VXOD	3.000

- NAME. Определенные пользователем имена, используемые в программе.
- VALUE. Числовое значение, присвоенное имени. Система присваивает значения именам, начиная с 10000. Исключение составляют имена блоков, им присваивается числовое значение в соответствии с порядковым номером в программе.

Блоки.

LABEL	LOC	BLOCK TYPE	ENTRY COUNT	CURRENT COUNT	RETRY
	1	GENERATE	88	0	0
	2	GATE	88	0	0
VXOD	3	QUEUE	82	0	0
	4	GATE	82	0	0
	5	SEIZE	82	0	0
	6	DEPART	82	0	0
	7	ADVANCE	82	0	0
	8	RELEASE	82	0	0
	9	TRANSFER	82	0	0
	10	QUEUE	78	0	0
	11	GATE	78	0	0
	12	ENTER	78	0	0
	13	DEPART	78	0	0
	14	ADVANCE	78	0	0
	15	LEAVE	78	0	0
UXOD	16	TERMINATE	82	0	0
BYE	17	TERMINATE	6	0	0
	18	GENERATE	1	0	0
	19	LOGIC	1	0	0
	20	ADVANCE	1	0	0
	21	LOGIC	1	0	0
	22	ADVANCE	1	0	0
	23	LOGIC	1	0	0
	24	ADVANCE	1	0	0
	25	SAVEVALUE	1	0	0
	26	TERMINATE	1	0	0

- LABEL. Имя блока, которое ему присвоено.
- LOC. Порядковый номер блока в программе.
- BLOCK TYPE. Имя блока-оператора в GPSS.
- ENTRY COUNT. Количество транзактов, вошедших в данный блок с момента последнего RESET или CLEAR, или с момента начала моделирования.

- CURRENT COUNT. Количество транзактов, находящихся в блоке на момент окончания моделирования.
- RETRY. Количество транзактов, ожидающих выполнения специфических условий, зависящих от состояния объекта данного блока.

Устройства.

FACILITY	ENTRIES	UTIL.	AVE. TIME	AVAIL.	OWNER	PEND	INTER	RETRY	DELAY
NACH	82	0.542	3.570	1	0	0	0	0	0

- FACILITY. Имя или номер объекта «устройство».
- ENTRIES. Количество раз, которое устройство было занято, с момента последнего RESET или CLEAR, или с момента последнего запуска модели.
- UTIL. Средняя загрузка устройства за последний измеряемый период времени (доля системного времени, которое устройство было занято, от общего времени моделирования). Измеряемый период времени отсчитывается от начала моделирования или с момента последнего использования команды RESET или CLEAR.
- AVE. TIME. Среднее время нахождения одного транзакта в устройстве.
- AVAIL. Состояние доступности устройства на конец моделирования. 1 означает, что устройство доступно, 0 – не доступно.
- OWNER. Номер транзакта, который занимает устройство. 0 означает, что устройство свободно.
- PEND. Количество транзактов, ожидающих в очереди, чтобы занять устройство через блок PREEMPT.
- INTER. Количество транзактов, претендующих на устройство после прерывания.
- RETRY. Количество транзактов, ожидающих выполнения специфических условий, зависящих от состояния данного устройства.
- DELAY. Количество транзактов, ожидающих в очереди, чтобы занять устройство (включает транзакты, которые пытаются занять устройства через блоки SEIZE и PREEMPT).

Очереди.

QUEUE	MAX	CONT.	ENTRY	ENTRY(0)	AVE.CONT.	AVE.TIME	AVE.(-0)	RETRY
OCH_NACH	13	0	82	31	2.367	15.589	25.064	0
OCH_PROP	3	0	78	53	0.431	2.982	9.305	0

- QUEUE. Имя объекта «очередь».
- MAX. Максимальная длина очереди в течение рассматриваемого периода моделирования. Рассматриваемый период считается с момента начала моделирования или с момента последнего оператора RESET или CLEAR.
- CONT. Длина очереди на момент окончания моделирования.
- ENTRY. Общее количество входов за рассматриваемый период.
- ENTRY(0). Количество «нулевых» входов. Общее количество транзактов, находящихся в очереди 0 единиц врмени.
- AVE.CONT. Средняя длина очереди за рассматриваемый период.
- AVE.TIME. Среднее время нахождения одного транзакта в очереди.
- AVE.(-0). Среднее время нахождения одного транзакта в очереди за исключением «нулевых» входов.
- RETRY. Количество транзактов, ожидающих выполнения специфических условий, зависящих от состояния данной очереди.

Память.

STORAGE	CAP.	REM.	MIN.	MAX.	ENTRIES	AVL.	AVE.C.	UTIL.	RETRY	DELAY
PROP	3	3	0	3	78	1	1.643	0.548	0	0

- STORAGE. Имя объекта «память».
- CAP. Емкость памяти, определенная блоком STORAGE.
- REM. Количество свободных ячеек памяти на момент окончания моделирования.
- MIN. Минимальное количество занятых ячеек памяти в течение рассматриваемого периода.

- MAX. максимальное количество занятых ячеек памяти в течение рассматриваемого периода.
- ENTRIES. Общее количество входов за рассматриваемый период.
- AVL. Состояние доступности памяти на конец моделирования. 1 означает, что память доступна, 0 – не доступна.
- AVE.C. Среднее количество занятых ячеек памяти в течение рассматриваемого периода.
- UTIL. Средняя загрузка памяти за последний измеряемый период времени (доля системного времени, которое память была занята, от общего времени моделирования).
- RETRY. Количество транзактов, ожидающих выполнения специфических условий, зависящих от состояния данной памяти.
- DELAY. Количество транзактов, ожидающих в очереди, чтобы занять память через блок ENTER.

Таблицы.

TABLE	MEAN	STD.DEV.	RANGE	RETRY	FREQUENCY	CUM.%
TAB1	15.589	21.814		0		
			_ - 10.000		52	63.41
			10.000 - 20.000		5	69.51
			20.000 - 30.000		9	80.49
			30.000 - 40.000		1	81.71
			40.000 - 50.000		4	86.59
			50.000 - 60.000		3	90.24
			60.000 - 70.000		7	98.78
			70.000 - 80.000		1	100.00
TAB2	2.982	9.252		0		
			_ - 10.000		72	92.31
			10.000 - 20.000		2	94.87
			20.000 - 30.000		1	96.15
			30.000 - 40.000		1	97.44
			40.000 - 50.000		1	98.72
			50.000 - 60.000		1	100.00

- TABLE. Имя объекта «таблица».
- MEAN. Среднее арифметическое табулируемой величины.
- STD.DEV. Выборочное стандартное отклонение табулируемой величины, рассчитанное по формуле $s.d. = \sqrt{\frac{\sum x^2 - 1/N (\sum x)^2}{N-1}}$.
- RANGE. Границы интервалов, по которым рассчитывается частота попадания табулируемой величины. Интервалы, частота попадания в которые равна 0, не выводятся.
- RETRY. Количество транзактов, ожидающих выполнения специфических условий, зависящих от состояния данной таблицы.
- FREQUENCY. Частота попадания в интервал.
- CUM.% Интегральная частота попадания, выраженная в процентах.

Логические переключатели.

LOGICSWITCH	VALUE	RETRY
TIME	1	0
OBED	0	0

- LOGICSWITCH. Имя или номер логического переключателя.
- VALUE. Значение логического переключателя на момент окончания моделирования. 1 означает «включен» или «истина», 0 означает «выключен» или «ложь».
- RETRY. Количество транзактов, ожидающих выполнения специфических условий, зависящих от состояния данного логического переключателя.
- Ячейки.
- SAVEVALUE RETRY VALUE
- NO_OBSL 0 0
- SAVEVALUE. Имя или номер ячейки.
- VALUE. Значение ячейки на момент окончания моделирования.
- RETRY. Количество транзактов, ожидающих выполнения специфических условий, зависящих от состояния данной ячейки.

Список будущих событий.

FEC XN	PRI	BDT	ASSEM	CURRENT	NEXT	PARAMETER	VALUE
90	0	540.784	90	0	1		

- XN. Номер транзакта в списке будущих событий.
- PRI. Текущий приоритет транзакта.
- BDT. Время в абсолютном системном измерении, когда транзакт должен покинуть список будущих событий.
- ASSEM. Номер транзакта в общем списке транзактов.
- CURRENT. Номер блока, в котором находится транзакт на момент создания отчета.
- NEXT. Номер блока, куда будет направлен транзакт после выхода из списка будущих событий.
- PARAMETER. Номера или имена параметров транзакта.
- VALUE. Значение параметра.