Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

Р А С Ч Ё Т Н О - П О Я С Н И Т Е Л Ь Н А Я З А П И С К А

к дипломной работе на тему:

Разработка системы имитационного моделирования в форме библиотеки языка Haskell

---

Студент _____ _____ **И. В. Миникс**
(Подпись, дата) (И.О.Фамилия)

Руководитель дипломной работы _____ _____ _____
(Подпись, дата) (И.О.Фамилия)

Консультант по исследовательской части __ _____ _____
(Подпись, дата) (И.О.Фамилия)

Консультант по конструкторско-технологической части __ _____ _____
(Подпись, дата) (И.О.Фамилия)

Консультант по организационно-экономической части __ _____ _____
(Подпись, дата) (И.О.Фамилия)

Консультант по охране труда и экологии ___ _____ _____
(Подпись, дата) (И.О.Фамилия)

Москва, 2013

Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой _____
(Индекс)

_____ _____
(И.О.Фамилия)

« ___ » _____ 20 __ г.

З А Д А Н И Е
на выполнение дипломной работы

Студент __Миникс Игорь Владимирович _____
(Фамилия, имя, отчество)

Разработка системы имитационного моделирования в форме библиотеки языка Haskell _____
(Тема дипломной работы)

_____

_____

Источник тематики (НИР кафедры, заказ организаций и т.п.) __НИР кафедры_____

_____

_____

_____

Тема дипломной работы утверждена распоряжением по факультету № ____
от « ___ » _____ 20__ г.

1. Исходные данные

Техническое задание, содержащее следующие требования: _____
__- разработать библиотеку языка Haskell, позволяющую осуществлять имитационное моделирование систем массового обслуживания; _____
__- разработать механизм трансляции описания систем из формата GPSS в формат разработанной библиотеки; _____
__- обеспечить возможность распространения моделей, разработанных при помощи библиотеки, в виде самостоятельных приложений._____

_____

2. Технико-экономическое обоснование

Существующие системы имитационного моделирования либо являются коммерческими, либо имеют серьезные функциональные ограничения (ограничено максимальное число используемых блоков, поддерживаются не все операционные системы и др.) _____

_____

_____

_____

_____

(обзор и анализ альтернативных решений; выбор вариантов для сравнения;
конкретные улучшаемые характеристики или параметры; возможный технико-экономический эффект и т.п.)

3. Научно-исследовательская часть

Сравнить характеристики системы массового обслуживания, полученные теоретически, с помощью разработанного ПО и с помощью одной из существующих реализаций GPSS. _____

_____

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

4. Проектно-конструкторская часть

Определить синтаксис описания систем массового обслуживания. Разработать структуры данных для хранения описания систем и методы и алгоритмы имитации и определения характеристик систем. _____

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

5. Технологическая часть

Осуществить выбор конкретных технологий и сторонних библиотек, позволяющих реализовать спроектированную библиотеку. Провести тестирование разработанного ПО на предмет корректности его работы и соответствия заданию._____

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

6. Организационно-экономическая часть

_____

_____

_____

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

7. Охрана труда и экология

_____

_____

_____

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

8. Оформление дипломной работы

8.1. Расчетно-пояснительная записка на ____ листах формата А4.

8.2. Перечень графического материала (плакаты, схемы, чертежи и т.п.) _____

_____

_____

Дата выдачи задания « ___ » _____ 20__ г.

В соответствии с учебным планом дипломную работу выполнить в полном объеме в срок до « ___ » _____ 20__ г.

Руководитель дипломной работы _____ _____
(Подпись, дата) (И.О.Фамилия)

Студент _____ _____
(Подпись, дата) (И.О.Фамилия)

Примечание:

1. Задание оформляется в двух экземплярах; один выдаётся студенту, второй хранится на кафедре.

Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой _____
(Индекс)

_____ _____
(И.О.Фамилия)

« ___ » _____ 20 __ г.

К А Л Е Н Д А Р Н Ы Й П Л А Н
выполнения дипломной работы

Студент _Миникс Игорь Владимирович _____
(Фамилия, имя, отчество)

_Разработка системы имитационного моделирования в форме библиотеки языка Haskell_____
(Тема дипломной работы)

_____

_____

№ п/п	Наименование этапов дипломной работы	Выполнение этапов		Примечание
		Срок	Объем, %	
1.	Разработка структур данных и выбор методов и алгоритмов.	17.02.2012	15%	
2.	Определение синтаксиса описания систем.	24.02.2012	20%	
3.	Написание программной части.	31.03.2012	55%	
4.	Тестирование и отладка.	7.04.2012	60%	
5.	Исследовательская часть.	14.04.2012	65%	
6.	Подготовка расчетно-пояснительной записки.	30.04.2012	80%	
7.	Оформление организационно-экономической и экологической части.	12.05.2012	85%	
8.	Оформление графической части.	19.05.2012	90%	
9.	Подготовка к защите.	26.05.2012	100%	

Руководитель дипломной работы _____ _____
(Подпись, дата) (И.О.Фамилия)

Студент _____ _____
(Подпись, дата) (И.О.Фамилия)

Содержание

Введение

Зародившаяся в начале прошлого века с целью упорядочить работу телефонных станций, теория массового обслуживания нашла применения в моделировании самых разнообразных систем, таких как системы связи, обработки информации, снабжения, производства и др.

Несмотря на имеющиеся достижения в области математического исследования характеристик систем массового обслуживания, наиболее универсальным подходом по прежнему остается имитационное моделирование.

Язык имитационного моделирования GPSS создан специально для моделирования систем массового обслуживания и на данный момен является доминирующим в этой области. Однако, существующие версии систем имитационного моделирования на основе языка GPSS либо слишком дороги, либо ограничены в возможностях и не позволяют провести все необходимые исследования.[1] Помимо этого, на данный момент затруднено интегрирование моделей, разработанных при помощи GPSS в другие программные средства (напимер, в целях оптимизации параметров исследуемой системы).

Целью данной работы является создание системы имитационного моделирования, основанной на принципах и синтаксисе GPSS, однако позволяющей разрабатывать модели как часть более крупной программы.

В качестве языка разработки был выбран Haskell. Haskell является динамично развивающимся функциональным языком проограммирования, который получает все больше сторонников во всем мире, в том числе и в России. [2]. Для Haskell характерны строгая статическая типизация, модульность, строгое разделение функций на чистые и не чистые, ленивые вычисления, функции высших порядков и др.[3] Помимо этого использование языка Haskell позволит производить описание систем при помощи синтаксиса схожего с синтаксисом GPSS, при этом разработанные модели будут являться объектами первого класса, что позволит, например, передать модель как параметр в функцию оптимизации.

Для достижения поставленной цели необходимо решить следующие задачи:

— изучить принципы функционирования и синтаксис описания моделей в GPSS;

— разработать синтаксис описания моделей схожий с синтаксисом GPSS, но при этом позволяющий составлять модели в виде функций языка Haskell;

— выбрать подмножество блоков GPSS, которые следует реализовать в системе;

— реализовать алгоритмы описания моделей и имитационного моделирования;

— разработать и реализовать транслятор моделей GPSS в формат разработанной системы моделиования;

— провести тестирование разработанного программного обеспечения;

— провести моделирование некоторой эталонной системы массового обслуживания в разработанной системе, GPSS и аналитически и убедиться в совпадении полученных результатов.

1 Аналитический раздел

В данном разделе проводится обзор принципов функционирования и синтаксиса системы GPSS, а также производтся выбор блоков, которые следует реализовать в разрабатываемой системе моделирования.

1.1 Краткий обзор GPSS

GPSS стал одним из первых языков моделирования, облегчающих процесс написания имитационных программ. Он был создан в виде конечного продукта Джефри Гордоном в фирме IBM в 1962 г.[4] В свое время он входил в десятку лучших языков программирования и по сей день широко используется для решения практических задач.

Основой имитационных алгоритмов GPSS является дискретно-событийный подход — моделирование системы в дискретные моменты времени, когда происходят события, отражающие последовательность изменения состояний системы во времени.[4]

1.2 Объекты языка GPSS

Основными Объектами языка GPSS являются транзакты и блоки, которые отображают соответственно динамические и статические объекты моделируемой системы.

Транзакты — динамические элементы GPSS-модели. В реальной системе транзактам могут соответствовать такие элементы как заявка, покупатель автомобиль и др. Состояни транзакта в процессе моделирования хараактеризуется следующими атрибутами:

— параметры — набор значений связанных с транзактом. Каждый транзакт может иметь произвольное число параметров. Каждый параметр иметт уникальный номер, по которому на него можно сослаться;

— приоритет — определяет порядок продвижения транзактов при конкурировании за общий ресурс;

— текущий блок — номер блока, в котором транзакт находится в данный момент;

— следующий блок — номер блока, в который транзакт попыытается войти;

— время появления блока — момент времени в который транзакт был создан;

— состояние — состояние, показывающее в каких списках транзакт находится в даннный момент. Транзакт может находиться в одном из следующих состояний:

а) активен — транзакт находится в списке текущих событий и имеет наивысший приоритет;

б) приостановлен — транзакт находится в списке будущих событий либо в списке текущих событий, но с меньшим приоритетом;

в) пассивен — транзакт находится в списке прерываний, списке синхронизации, списке блокировок или списке пользователя;

г) завершен — транзакт уничтожен и болше не участвует в модели.

Диаграмма состояний транзакта показана на Рисунке 1.1.

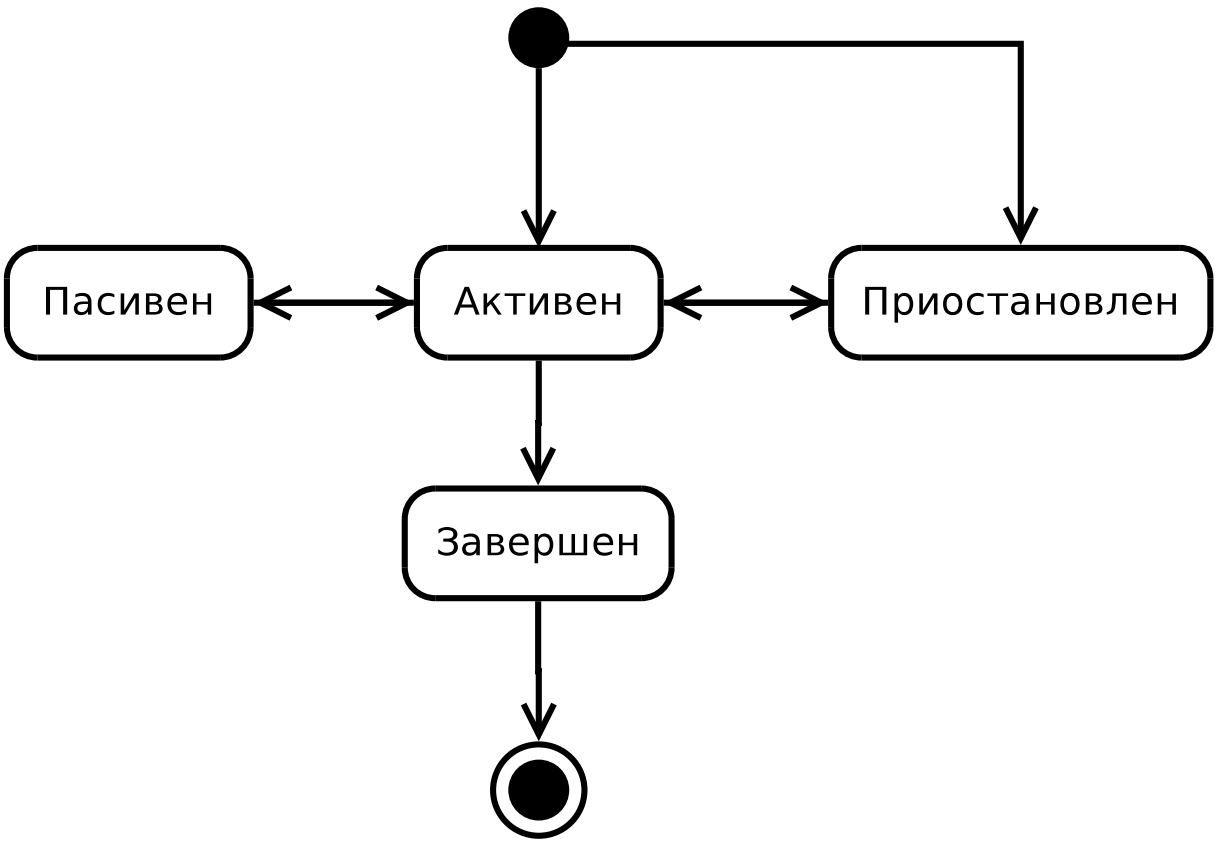

Рисунок 1.1 — Состояния транзакта

Блоки — статические элементы GPSS-модели. Модель в GPSS может быть представленна как диаграмма блоков, т.е. ориентированный граф, узлами которого являются блоки, а дугам — направления движения транзактов. с каждым блоком связано некоторое действие, изменяющее состояние прочих элементов модели. Транзакты проходят блоки один за другим, до тех пор пока не достигнут блока TERMINATE. В ряде случаев транзакт может быть остановлен в одном из блоков до наступления некоторого события.

Помимо транзактов и блоков в GPSS используются следующие объекты: устройства, многоканальные устройства (хранилища, памяти), ключи, очереди, списки пользователя и др.

1.3 Управления процесом моделирования в GPSS

В системе GPSS итерпретатор поддерживает сложные структуры организации списков (см. Рисунок 1.2).[4] Два основных из них — список текущих событий (СТС) и список будущих событий (СБС).

В СТС входят все события запланированные на текущий момент модельного времени. Интерпретатор в первую очередь просматривает этот список и перемещает по модели те транзакты, для которых выполнены все условия. Если таких транзактов в списке не оказалось интерпретатор обращается к СБС. Он переносит все события, запланированные на ближайший момент времени и вновь возвращается к просмотру СТС. Перенос также осуществляется в случае совпадения текущего момента времени с моментом наступления ближайшего события из СБС.

В целях эффективной организации просмотра транзактов, движение которых заблокировано (например, из-за занятости некоторого ресурса), используются следующие вспомогательные списки:

— списки блокировок — списки транзактов, которые ожидают освобождения некоторого ресурса;

— список прерываний — содержит транзакты, прерванные во время обслуживания. Используется для организации обслуживания одноканальных устройств с абсолютным приоритетом;

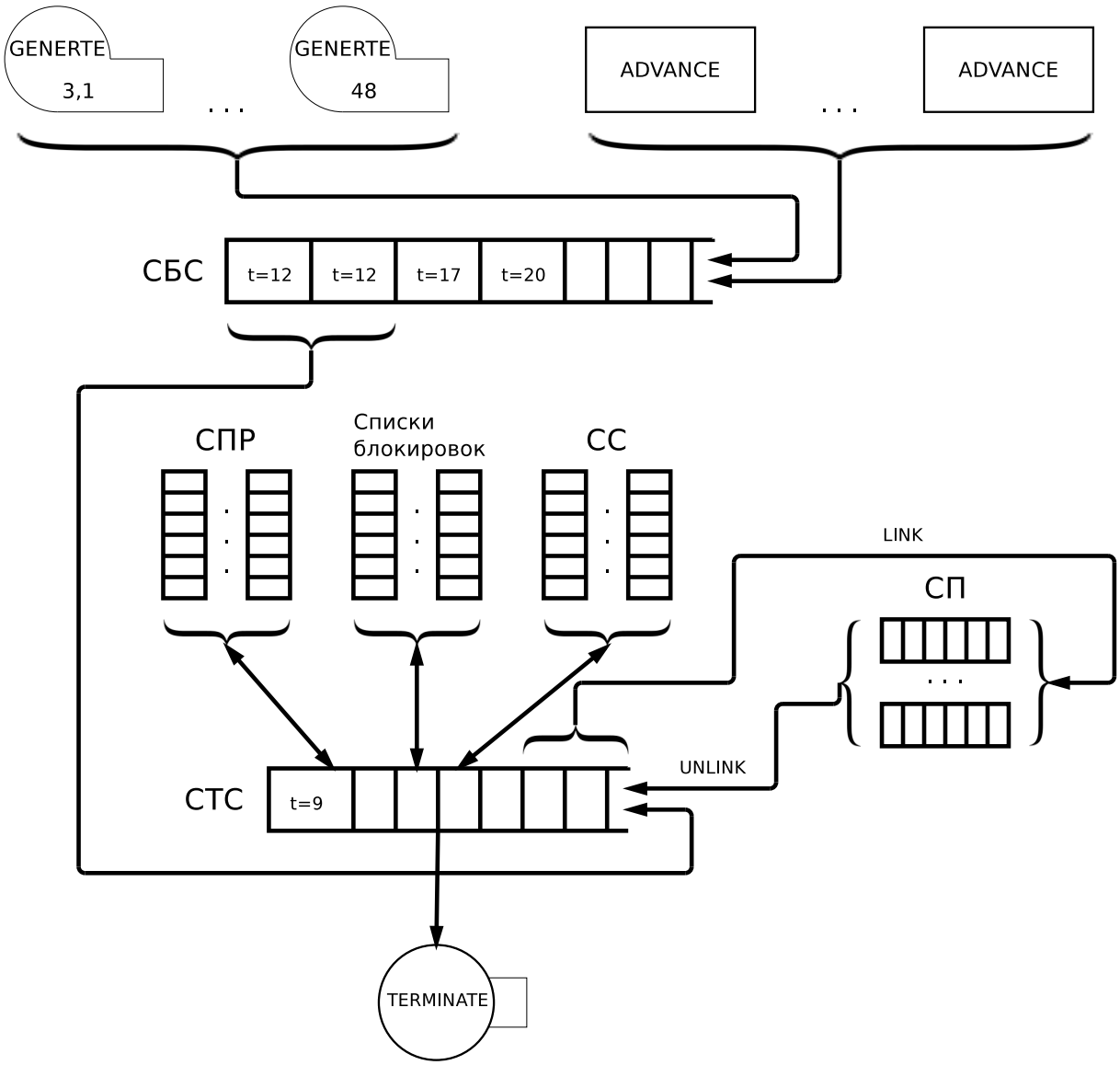

Рисунок 1.2 — Списки GPSS

— списки синхронизации — содержат транзакты одного семейства (созданные блоком SPLIT), которые ожидают синхронизации в блоках (MATCH, ASSEMBLE или GATHER);

— списки пользователя — содержат транзакты, выведенные пользователем из СТС с помощью блока LINK. Транзакты могут быть возвращены в СТСс помощью блока UNLINK.

1.4 Выбор подмножества реализуемых блоков

В современной версии языка GPSS (входящей в пакет GPSS World) поддерживается 53 различных блока.[5] В рамках данной работы не представляется возможным реализовать аналоги каждого из них. Поэтому сле-

дует выделить некоторое подмножество блоков, которое с одной не будет слишком обширным, а с другой — позволит решать практические или по крайней мере учебные задачи.

В качетсве примера рассмотрим задачу из курса Модели оценки качества аппаратно программных комлексов:

В вычислительной системе, содержащей N процессоров и M каналов обмена данными, постоянно находятся K задач. Разработать модель, оценивающую производительность системы с учетом отказов и восстановлений процессоров и каналов. Имеется не более L ремонтных бригад, которые ремонтируют отказывающие устройства с бесприоритетной дисциплиной. Интенсивность отказов, восстановлений, средние времена обработки сообщения и среднее время обдумывания также известны.

Как и подаляющее большинство других задач, данная задача, безусловно, не может быть решена без использования блоков GENERATE, TERMINATE и ADVANCE. Так как моделируемая система является замкнутой, при описании модели не обойтись без блока TRANSFER.

К содалению, не представляется возможным реализовать процессоры и каналы как многоканальные устройства, т.к. многоканальные устройства в GPSS не поддерживают абсолютные приоритеты и не позволяют смоделировать выход из строя отдельных каналов устройства. Однако, требуемую систему можно описать при помощи множества одноканальных устройств и блока TRANSFER в режиме ALL. Таким образом, также понадобятся блоки SEIZE и RELEASE. Для моделирования отказов устройств можно воспользоваться блоками SAVAIL и SUNAVAIL либо блоками PREEMPT и RETURN.

Наконец, доступные ремонтные бригады можно смоделировать с помощью многоканального устройства. Соответственно, понадобятся блоки ENTER и LEAVE.

Приблизительная модель системы показана в Листинге 1.1

Листинг 1.1 — Приблизительная модель системы

```
1 ;Доступное число ремонтных бригад
2 REPAIRERS STORAGE 5
3
```

```
4  ;Общее время моделирования
5  GENERATE  ,,,1
6  ADVANCE 1000
7  TERMINATE 1
8
9  ;Основная часть модели
10 GENERATE  ,,,10
11
12 ;фаза обдумывания
13 LUSER ADVANCE 4,1
14       TRANSFER ALL,LCPU1,LCPUN,4
15
16 ;Первая фаза обработки
17 LCPU1  SEIZE CPU1
18       ADVANCE 2,1
19       RELEASE CPU1
20       TRANSFER ,LPHASE2
21
22 ...
23
24 LCPUN  SEIZE CPUN
25       ADVANCE 2,1
26       RELEASE CPUN
27       TRANSFER ,LPHASE2
28
29 LPHASE2 TRANSFER ALL,LCHAN1,LCHANM,4
30
31 ;Вторая фаза обработки
32 LCHAN1  SEIZE CHAN1
33         ADVANCE 2,1
34         RELEASE CHAN1
35         TRANSFER ,LUSER
36
37 ...
38
39 LCHANM SEIZE CHANM
40       ADVANCE 2,1
41       RELEASE CHANM
42       TRANSFER ,LUSER
43
44 ;Моделирование отказов и восстановлений.
45            GENERATE  ,,,1,10
46 CPUBROKE1 ADVANCE 20,4
47            PREEMPT CPU1
48            ENTER REPAIRERS
49            ADVANCE 10,4
```

```
50              LEAVE  REPAIRERS
51              RETURN  CPU1
52              TRANSFER  ,CPUBROKE1
53
54  ...
55
56              GENERATE  ,,,1,10
57  CPUBROKEN   ADVANCE  20,4
58              PREEMPT  CPUN
59              ENTER  REPAIRERS
60              ADVANCE  10,4
61              LEAVE  REPAIRERS
62              RETURN  CPUN
63              TRANSFER  ,CPUBROKEN
64
65               GENERATE  ,,,1,10
66  CHANBROKE1  ADVANCE  20,4
67              PREEMPT  CHAN1
68              ENTER  REPAIRERS
69              ADVANCE  10,4
70              LEAVE  REPAIRERS
71              RETURN  CHAN1
72              TRANSFER  ,CHANBROKE1
73
74  ...
75
76              GENERATE  ,,,1,10
77  CHANBROKEN  ADVANCE  20,4
78              PREEMPT  CHANM
79              ENTER  REPAIRERS
80              ADVANCE  10,4
81              LEAVE  REPAIRERS
82              RETURN  CHANM
83              TRANSFER  ,CHANBROKEM
```

Таким образом, разрабатываемая система имитационного моделирования должна поддерживать аналоги по крайней мере следующих блоков: ADVANCE, ENTER, GENERATE, LEAVE, PREEMPT, RELEASE, RETURN, SEIZE, TERMINATE и TRANSFER.

1.5 Описание выбранных блоков

Ниже представлено описание выбранных блоков в соответствии со справочным руководством GPSS World.[5]

ADVANCE A,B

Блок ADVANCE осуществляет задержку продвижения транзактов на заданный промежуток времени.

A — Среднее время задержки. Не обязательный параметр. Значение по умолчанию — 0.

B — Максимально допустимое отклонение времени задержки либо функция-модификатор.

ENTER A,B

При входе в блок ENTER транзакт либо занимает заданное колличество каналов указанного многоканального устройства либо блокируется до его освобождения.

A — Имя или номер многоканального устройства. Обязательный параметр.

B — Число требуемых каналов. Не обязательный параметр. Значение по умолчанию — 1.

GENERATE A,B,C,D,E

Блок GENERATE предназначен для создания новых транзактов.

A — Среднее время между генерацией последовательных заявок. Не обязательный параметр.

B — Максимальное допустимое отклонение времени генерации либо функция-модификатор. Не обязательный параметр.

C — Задержка до начала генерации первого транзакта. Не обязательный параметр.

D — Ограничение на максимальное допустимое число созданных транзактов. Не обязательный параметр. Пол умолчанию ограничение отсутствует.

E — Уровень приоритета создаваемых заявок. Не обязательный параметр. Значение по умолчанию — 0.

LEAVE A,B

При входе в блок LEAVE транзакт освобождаает заданное число каналов указанного многоканального устройства.

A — Имя или номер многоканального устройства. Обязательный параметр.

B — Число требуемых каналов. Не обязательный параметр. Значение по умолчанию — 1.

PREEMPT A,B,C,D,E

RELEASE A

Блок RELEASE освобождает одноканальное устройство.

A — Имя или номер одноканального устройства. Обязательный параметр.

RETURN A

Блок RELEASE освобождает одноканальное устройство.

A — Имя или номер одноканального устройства. Обязательный параметр.

SEIZE A

При входе в блок SEIZE транзакт занимает указанное одноканальное устройство либо блокируется до его освобождения.

A — Имя или номер одноканального устройства. Обязательный параметр.

TERMINATE A

Блок TERMINATE завершает поступивший в него транзакт. И опционально уменьшает счетчик завершенных транзаков. Когда счетчик достигает нуля имитация останавливается.

A — Значение, на которое следует уменьшить счетчик завершенных транзактов. Не обязательный параметр. Значение по умолчанию — 0.

1.6 Выводы

Был проведен анализ устройства системы GPSS и осуществлен выбор подмножества блоков, необходимых для моделирования не сложных систем массового обслуживания.

2 Конструкторский раздел

В данном разделе проводится выбор синтаксиса описания моделей в разрабатываемой системе, а также описываются алгоритмы и структуры данных, используемые при формировании моделей и непосредственно при моделировании.

2.1

Синтаксис разрабатываемой системы должен быть, на сколько это возможно, схож с синтаксисом системы GPSS.

Программа на языке GPSS представляет из себя последовательность операторов, каждый из которы описывает тот или иной элемент модели (функцию, блок, устройство и др.). Этот подход естественен для императивных языков программирования, в которых программа является последовательностью комманд, меняющих состояние программы. Однако Haskell относится к категории функциональных языков, программы на которых лписываются как функции, значение которых вычисляется. При этом нет фиксированной, заданной программистом, последовательности операций, которые должны быть выполнены для достижения результата.

Тем не менее, в языке Haskell предусмотрен механизм, позволяющий описать конкретную последовательность вычислений — монады. В сочетании с так называемой do-нотацией, этот механизм позволит проводить описание моделей на Haskel, используя синтаксис схожий с GPSS.

2.2 Монады

Понятие монады в языке Haskell основано на теории категорий. В рамках данной теории монадо может быть определена (не вполне строго) как моноид в категории эндофункторов. Однако для практического использования этого понятия в рамках языка Haskell можно обойтись менее формальным определением.

В соответствии с [3] монада — это контейнейрный тип данных (то есть такой, который содержит в себе значения других типов), представляющий собой экземпляр класса Monad определенного в модуле Prelude.

Под классом в Haskell, понимается не тип данных, как в объектно-оринтированных языках, а набор методов (функций), которые применимы для работы с теми или иными типами данных, для которых объявлены экземпляры заданных классов. Наиболее близким аналогом классам в Haskell являются интерфейсы в таких языках как Java или C#. Более точно их следует называть классами типов, но т.к. в данной работе используется исключительно функциональная парадигма, в дальнейшем для краткости они будут называться просто классами.

Значения монад можно воспринимать, как значения м неккоторым дополнительным контекстом. В случае монады Maybe значения обладают дополнительным контекстом того, что вычисления могли закончиться неуспешно. Монада IO добавляет контекст, указывающий что получение значений связано с действиями ввода/вывода и потому не является детерминированным и может иметь побочные эффекты. В случае списков (которые также являются монадой) контекстом является то, что значение может являться множественным или отсутствовать.

Класс Monad определен в модуле Prelude следующим образом:

Листинг 2.1 — Класс Monad

```
1 class Monad m where
2     return :: a -> m a
3     (>>=)  :: m a -> (a -> m b) -> m b
4     (>>)   :: m a -> m b -> m b
5     fail   :: String -> m a
6
7     p >> q = p >>= \_ -> q
8     fail s = error s
```

Функция **return**[1] преобразует переданное ей значение типа a в монадическое значение типа m a. Другими словами она помещает значение в некоторый контекст по умолчанию, в зависимости от выбранной монады. Для списка это будет списко из одного элемента, для монады IO — дей-

[1]Следует отметить, что название return никак нельзя назвать удачным, так как оно неизбежно вызывает ассоциации с одноименным оператором из многих императивных языков программирования, на которые она не похожа ничем кроме названия. Данная функция не завершает выполнение функции, а лишь оборачивает переданное значение в монаду.

ствие ввода вывода, всегда возвращающее заданное значение и не имеющее побочных эффектов и т.д.

Функция $>>=$ определяет операцию связывания. Она принимает монадическое значение и передает его функции, которая принимает обычное значение и возвращает монадическое. При этом сохраняется накопленный контекст и к нему добавляется новый, полученный в результате выполнения функции.

Функция $>>$ также предназначена для связывания и используется в тех случаях, когда переданное монадическое значение не представляет интереса, а значение имеет только переданный с ним контекст вычислений. Для этой функции в классе определена реализация по умолчанию, по этому в большенстве случаев при определении экземпляра класса Monad в явном виде ее не реализуют.

Функция **fail** никогда не вызывается программистом явным образом и предназначена для обработки неуспешного окончания вычислений при сопоставлении с образцом в do-нотации, что позволяет избежать аварийного завершения программы и вернуть неудачу в контексте текущец монады.

2.3 Нотация do

Так как монады находят крайне широкое применение в программах на языке Haskell (в первую очередь, без использования монады IO невозможно осуществить ввод/вывод), в синтаксис языка было добавлено специальное ключевое слово **do**, призванное упростить написание монадических функций, сделать их более читаемыми и избавить от излишнего «синтаксического мусора».

Если в коде программы встречается контрукция с ключевым словом **do**, то транслятор выполняет следующие преобразования[1]:

```
1. do {e} → e
```

[1]В приведенных преобразованиях используются управляющие символы **;**, **{** и **}**, хотя в реальных программах на языке Haskell их можно встретить довольно редко. Это связано с тем, что в Haskell используется так называемый «двумерный синтаксис»: при правильной расстановке отступов, транслятор самостоятельно расставляет точки с запятой и фигурные скобки и в большинтве случаев нет смысла заграмождать ими исходный код. Тем не менее в случае необходимости их можно расставить и явным образом.

```
2. do {e; es} → e >> do {es}
3. do {let decls; es} → let decls in do {es}
4. do {p <- e; es} → let ok p = do {es}
                         ok _ = fail "..."
                     in e >>= ok
```

При помощи нотации **do** приведенный ниже фрагмент кода

```
foo :: Maybe String
foo = Just 3 >>= (\x -> Just "!" >>= (\y -> Just (show x ++ y)))}
```

может быть записан в следующей более читаемой форме:

```
foo :: Maybe String
foo = do x <- Just 3
         y <- Just "!"
         return (show x ++ y)
```

2.4 Монада State

Часто в процессе вычислений возникает необходимость хранить и изменять некоторое состояние, в завсимости от которого результат вычислений может меняться. Haskell является чистым функциональным языком программирования функции должны быть детерменированы и не иметь побочных эффектов, поэтому текущее состояние обычно передается в функции как еще один параметр, а возвращает функция пару из собственно результата и обновленного состояния.

Для того, чтобы упростить написание функций оперирующих некоторым соостоянием в Haskell была введена монада **State**. Она определена в модуле **Control.Monad.State** следующим образом:

```
newtype State s a = State {runState :: s -> (a, s)}

instance Monad (State s) where
    return x = State $ \s -> (x,s)
```

```
(State h) >>= f = State $ \s -> let (a, newState) = h s
                                    (State g) = f a
                                in g newState
```

Функция **return** создает вычисление с состоянием, которое всегда возвращает один и тот же результат и оставляет переданное в него состояние без изменений. Функция $>>=$ «склеивает» два вычисления с состоянием так, что конечное состояние первого становится начальным для второго, а результат и конечное состояние второго вычисления становятся также результатом и конечным состоянием итогового, составного вычисления.

Помимо этого для работы с монадой **State** используются две вспомогательные функции **put** и **get**. Функция **put** является вычислением, которое устанавливает состояние в заданное значение не зависимо от его предыдущего значения и не возвращает никакого результата (точнее возвращает кортеж нулевой длины ()). Функция **get** возвращает текущее состояние и оставляет его без изменений.

2.5 Описание модели как вычислене с состоянием

Список использованных источников

1. *Квитка М. Е. Сёмкин Ю. Ю., Томила С. О.* Разработка свободного аналога языка GPSS. — 2008.

2. *Р.В., Душкин*. Справочник по языку Haskell. / Душкин Р.В. — М.: ДМК Пресс, 2008.

3. *Р.В., Душкин*. Функциональное программирование на языке Haskell. / Душкин Р.В. — М.: ДМК Пресс, 2007.

4. *Томашевский В., Жданова Е.* Имитационное моделирование в среде GPSS. / Жданова Е. Томашевский В. — М.: Бестселлер, 2003.

5. GPSS Wrorld Reference Manual. — 2009.