Всё, что вообще может быть сказано, должно быть сказано ясно, а о чём невозможно говорить, о том следует молчать.
Л. Витгенштейн

Лекция 5.
МЕТОДЫ СПЕЦИФИКАЦИИ СЕМАНТИКИ ФУНКЦИЙ

Основные подходы к спецификации семантики функций. Табличный подход, метод таблиц решений. Алгебраический подход: операционная, денотационная и аксиоматическая семантики. Языки спецификаций.

5.1. Основные подходы к спецификации семантики функций.

Для спецификации семантики функций используются следующие подходы: табличный, алгебраический и логический [5.1, стр.30-73], а также графический [5.2].

Табличный подход для определения функций хорошо известен ещё со средней школы. Он базируется на использовании таблиц. В программировании эти методы получили развитие в методе таблиц решений.

Алгебраический подход для определения функций базируется на использовании равенств. В этом случае для определения некоторого набора функций строится система равенств вида:

$$\begin{aligned} &L_1 = \underline{R}_1, \\ &\ldots \\ &L_n = R_n. \end{aligned} \qquad (5.1)$$

где L_i и R_i, $i = 1, \ldots n$, некоторые выражения, содержащие предопределенные операции, константы, переменные, от которых зависят определяемые функции (формальные параметры этих функций), и вхождения самих этих функций. Семантика определяемых функций извлекается в результате интерпретации этой системы равенств. Эта интерпретация может осуществляться по-разному (базироваться на разных системах правил), что порождает разные семантики. В настоящее время активно исследуются операционная, денотационная и аксиоматическая семантики.

Третий подход, логический, базируется на использовании предикатов – функций, у которых аргументами могут быть значения различных типов, а результатами являются логические значения (ИСТИНА и ЛОЖЬ). В этом случае набор функций может определяться с помощью системы предикатов. Заметим, что система равенств алгебраического подхода может быть задана с помощью следующей системы предикатов:

$$\begin{aligned} &\text{РАВНО}\ (L_1, R_1), \\ &\ldots \end{aligned} \qquad (5.2)$$

РАВНО (L_n, R_n),

где предикат РАВНО истинен, если равны значения первого и второго его аргументов. Это говорит о том, что логический подход располагает не меньшими возможностями для определения функций, однако он требует от разработчиков ПС умения пользоваться методами математической логики, что, к сожалению, не для всех разработчиков оказывается приемлемым. Более подробно этот подход мы рассматривать не будем.

Графический подход также известен еще со средней школы. Но в данном случае речь идет не о задании функции с помощью графика, хотя при данном уровне развития компьютерной техники ввод в компьютер таких графиков возможен и они могли бы использоваться (с относительно небольшой точностью) для задания функций. В данном случае речь идет о графическом задании различных схем, выражающих сложную функцию через другие функции, связанными с какими-либо компонентами заданной схемы. Графическая схема может определять ситуации, когда для вычисления представляемой ею функции должны применяться связанные с этой схемой более простые функции. Графическая схема может достаточно точно определять часть семантики функции. Примером такой схемы может быть схема переходов состояний конечного автомата, такая, что в каждом из этих состояний должна выполняться некоторая дополнительная функция, указанная в схеме.

5.2. Метод таблиц решений.

Метод таблиц решений базируется на использовании таблиц следующего вида (см. табл. 5.1).

Переменные/ условия	Ситуации (комбинации значений)				
x_1	$a[1, 1]$	$a[1, 2]$	…	$a[1, m]$	*
x_2	$a[2, 1]$	$a[2, 2]$	…	$a[2, m]$	*
…			…		
x_n	$a[n, 1]$	$a[n, 2]$	…	$a[n, m]$	*
s_1	$u[1, 1]$	$u[1, 2]$	…	$u[1, m]$	$u[1, m+1]$
s_2	$u[2, 1]$	$u[2, 2]$	…	$u[2, m]$	$u[1, m+1]$
…			…		
s_k	$u[k, 1]$	$u[k, 2]$	…	$u[k,m]$	$u[k, m+1]$
Действия	Комбинации выполняемых действий				

Табл. 5.1. Общая схема таблиц решений.

Верхняя часть этой таблицы определяет различные ситуации, в которых требуется выполнять некоторые действия (операции). Каждая

строка этой части задает ряд значений некоторой переменной или некоторого условия. Первый столбец этой части представляет собой список переменных или условий, от значений которых зависит выбор определяемых ситуаций. В каждом следующем столбце указывается комбинация значений этих переменных (условий), определяющая конкретную ситуацию. При этом последний столбец определяет ситуацию, отличную от предыдущих, т.е. для любых других комбинаций значений (будем обозначать их звёздочкой *), отличных от первых, определяется одна и та же, (m + 1)-ая, ситуация. Впрочем, в некоторых таблицах решений этот столбец может отсутствовать.

Нижняя часть таблицы решений определяет действия, которые требуется выполнить в той или иной ситуации, определяемой в верхней части таблицы решений. Она также состоит из нескольких (k) строк, каждая из которых связана с каким-либо одним конкретным действием, указанным в первом поле (столбце) этой строки. В остальных полях (столбцах) этой строки (т.е. для $u\,[i, j]$, $i = 1, \ldots m + 1$, $j = 1, \ldots k$) указывается, следует ли выполнять (при $u\,[i, j]$ = «+») это действие в данной ситуации или не следует (при $u\,[i, j]$ = «−»). Таким образом, первый столбец нижней части этой таблицы представляет собой список обозначений действий, которые могут выполняться в той или иной ситуации, определяемой этой таблицей. В каждом следующем столбце этой части указывается комбинация действий, которые следует выполнить в ситуации, определяемой в том же столбце верхней части таблицы решений. Для ряда таблиц решений эти действия могут выполняться в произвольном порядке, но для некоторых таблиц решений этот порядок может быть предопределён, например, в порядке следования строк в нижней части этой таблицы.

Рассмотрим в качестве примера описание работы светофора у пешеходной дорожки. Переключение светофора в нормальных ситуациях должно производиться через фиксированное для каждого цвета число единиц времени ($T_{кр}$ – для красного цвета, $T_{жёл}$ – для жёлтого, $T_{зел}$ – для зелёного). У светофора имеется счётчик таких единиц. При переключении светофора в счётчике устанавливается 0. Работа светофора усложняется необходимостью пропускать привилегированные машины (при их появлении на светофор поступает специальный сигнал) с минимальной задержкой, но при обеспечении безопасности пешеходов. Приведённая на рис. 5.2 таблица решений описывает работу такого светофора, управляющего порядком движения у него в каждую единицу времени. Звёздочка (*****) в этой таблице означает произвольное значение соответствующего условия.

Условия	Ситуации							
Состояние светофора	Кр	Кр	Кр	Жёл	Жёл	Зел	Зел	Зел
$T = T_{кр}$	Нет	Нет	Да	*	*	*	*	*
$T = T_{жёл}$	*	*	*	Нет	Да	*	*	*
$T > T_{зел}$	*	*	*	*	*	Нет	Да	Да
Появление привилегированной машины	Нет	Да	*	*	*	*	Нет	Да
Включить красный	–	–	–	–	–	–	+	–
Включить жёлтый	–	+	+	–	–	–	–	–
Включить зелёный	–	–	–	–	+	–	–	–
$T := 0$	–	+	+	–	+	–	+	–
$T := T + 1$	+	–	–	+	–	+	–	+
Освобождение пешеходной дорожки	–	–	–	+	–	–	–	–
Пропуск пешеходов	+	+	+	–	–	–	–	–
Пропуск машин	–	–	–	–	–	+	+	+
Действия	Комбинации выполняемых действий							

Рис. 5.2. Таблица решений «Светофор у пешеходной дорожки».

5.3. Операционная семантика.

В операционной семантике алгебраического подхода к описанию семантики функций рассматривается следующий частный случай системы равенств (5.1):

$$\begin{aligned} f_1 (x_1, x_2, \ldots, x_k) &= E_1, \\ &\ldots \\ f_n (x_1, x_2, \ldots, x_k) &= E_n, \end{aligned} \qquad (5.3)$$

где в левых частях этих равенств явно указаны определяемые функции, каждая из которых зависит (для простоты) от одних и тех же параметров

$x_1, x_2, \ldots, x_k$,

а правые части этих равенств представляют собой выражения, содержащие, вообще говоря, вхождения этих функций, т.е. определяемые функции могут быть *рекурсивными*. Поэтому определяемая функция может иметь

дополнительные вхождения в левые части этой системы равенств с постоянными значениями некоторых параметров. Таким образом, каждый параметр x_j вхождения определяемой функции f_i в левую часть равенств (5.3) будем понимать либо как переменную y_j, либо как константу c_{ij}, причём совокупность переменных

$y_1, y_2, \ldots, y_k$

представляет входные данные, для которых требуется вычислять определяемые функции.

Операционная семантика интерпретирует эти равенства как систему подстановок. Под подстановкой

$$E \Big|_{T}^{s}$$

выражения (терма) T в выражение E вместо символа s будем понимать *переписывание* выражения E с заменой каждого вхождения в него символа s на выражение T. Каждое равенство

$f_i (x_1, x_2, \ldots , x_k) = E_i$

задает в параметрической форме множество правил подстановок вида

$$f_i (T_1, T_2, \ldots , T_k) \rightarrow \Big| \begin{matrix} x_1, x_2, \ldots , x_k \\ E_i \\ T_1, T_2, \ldots , T_k \end{matrix} ,$$

где T_1, T_2, ... , T_k – конкретные аргументы (значения или определяющие их выражения) данной функции. Каждое такое правило допускает замену в каком-либо выражении вхождения его левой части на её правую часть.

Интерпретация системы равенств (5.3) для получения значений определяемых функций в рамках операционной семантики производится следующим образом. Пусть задан набор входных данных (аргументов)

$d_1, d_2, \ldots , d_k$.

На первом шаге осуществляется подстановка этих данных в левые и правые части равенств с выполнением там, где это возможно, предопределённых операций и с *переписыванием* получаемых в результате этого равенств. В результате этого будет сформирована исходная *преобразуемая* система равенств.

На каждом следующем шаге по *текущей преобразуемой* системы равенств производится формирование новой *преобразуемой* системы равенств (которая становится *текущей*) путем *переписывания* этих равенств со следующими преобразованиями.

(1) Если правая часть очередного равенства является каким-либо значением, то это равенство не изменяется (это значение и является значением функции, указанной в левой части этого равенства).

(2) В противном случае правая часть является выражением, содержащим вхождения каких-либо определяемых функций с теми или иными наборами аргументов. В этом случае для каждого вхожде-

ния функции в эту правую часть (с конкретным набором аргументов) просматриваются левые части преобразуемых равенств и выполняются следующие действия.

(2.1) Если для этого вхождения в *текущей преобразуемой* системе равенств находится совпадающая с ним левая часть некоторого равенства, то проверяется правая часть этого равенства

(2.1.1) и в случае, если она является уже вычисленным значением, производится подстановка этого значения вместо указанного вхождения определяемой функции,

(2.1.2) если же эта правая часть не является вычисленным значением, то указанное вхождение *переписывается* в неизменном виде.

(2.2) В том же случае, если для указанного вхождения в *текущей преобразуемой* системе равенств не находится совпадающей с ним левой части никакого равенства, то к новой *преобразуемой* системе равенств дописывается *новое* равенство. Это равенство получается из исходного равенства для определяемой функции

fi(y1, y2, …, yk),

вхождение которой в данный момент исследуется, путем подстановки аргументов этой функции из исследуемого вхождения вместо параметров y1, y2, …, yk этой функции (с выполнением предопределённых операций там, где это возможно).

Эти шаги будут осуществляться до тех пор, пока все определяемые функции не будут иметь вычисленные значения.

В качестве примера операционной семантики рассмотрим определение функции $F(n) = n!$. Она определяется следующей системой равенств:

$F(0) = 1$,

$F(n) = F(n-1) \times n$.

Для вычисления значения $F(3)$ осуществляются следующие шаги.

1-й шаг:

$F(0) = 1, F(3) = F(2) \times 3$.

2-й шаг:

$F(0) = 1, F(3) = F(2) \times 3, F(2) = F(1) \times 2$.

3-й шаг:

$F(0) = 1, F(3) = F(2) \times 3, F(2) = F(1) \times 2, F(1) = F(0) \times 1$.

4-й шаг:

$F(0) = 1, F(3) = F(2) \times 3, F(2) = F(1) \times 2, F(1) = 1$.

5-й шаг:

$F(0) = 1, F(3) = F(2) \times 3, F(2) = 2, F(1) = 1$.

6-й шаг:

$F(0) = 1, F(3) = 3, F(2) = 2, F(1) = 1$.

Значение $F(3)$ на 6-ом шаге получено.

5.4. Денотационная семантика.

В денотационной семантике алгебраического подхода рассматривается также система равенств вида (5.3), которая интерпретируется как система функциональных уравнений, а определяемые функции являются некоторым решением этой системы. В классической математике изучению функциональных уравнений (в частности, интегральных уравнений) уделяется большое внимание и связано с построением достаточно глубокого математического аппарата. Применительно к программированию этими вопросами серьёзно занимался Д. Скотт [5.3].

Основные идеи денотационной семантики проиллюстрируем на более простом случае, когда система равенств (5.3) является системой языковых уравнений:

$$\begin{aligned} X_1 &= phi\,[1,\,1] \cup phi\,[1,\,2] \cup \ldots \cup phi\,[1,\,k_1],\\ X_2 &= phi\,[2,\,1] \cup phi\,[2,\,2] \cup \ldots \cup phi\,[2,\,k_2],\\ &\ldots\\ X_n &= phi\,[n,\,1] \cup phi\,[n,\,2] \cup \ldots \cup phi\,[n,\,k_n], \end{aligned} \qquad (5.4)$$

причём i-ое уравнение при $k_i = 0$ имеет вид

$X_i = \varnothing$.

Формальный язык − это множество цепочек в некотором алфавите. Такую систему можно рассматривать как одну из интерпретаций набора правил некоторой грамматики, представленную в форме Бэкуса-Наура (каждое из приведенных уравнений является аналогом некоторой такой формулы). Пусть фиксирован некоторый алфавит $A = \{a_1, a_2, \ldots, a_m\}$ терминальных символов грамматики, из которых строятся цепочки, образующие используемые в системе (5.4) языки. Символы $X_1, X_2, \ldots, X_n$ являются метапеременными грамматики, здесь будут рассматриваться как переменные, значениями которых являются языки (множества значений этих метапеременных). Символы $phi\,[i, j]$, $i = 1, \ldots, n$, $j = 1, \ldots, k_j$, обозначают цепочки в объединённом алфавите терминальных символов и метапеременных:

$phi\,[i, j] \in (A \cup \{X_1, X_2, \ldots, X_n\})^*$.

Цепочка $phi\,[i, j]$ рассматривается как некоторое выражение, определяющее значение, являющееся языком. Такое выражение определяется следующим образом. Если значения $X_1, X_2, \ldots, X_n$ заданы, то цепочка

$phi = Z_1 Z_2 \ldots Z_k$, $Z_i \in (A \cup \{X_1, X_2, \ldots, X_n\})$ для $i = 1, \ldots, k$,

обозначает *сцепление* множеств $Z_1, Z_2, \ldots, Z_k$, причём вхождение в эту цепочку символа a_j представляет множество из одного элемента $\{a_j\}$. Это означает, что *phi* определяет множество цепочек

$\{p_1 p_2 \ldots p_k \mid p_j \in Z_j, j = 1, \ldots, k\}$,

причём цепочка $p_1 p_2 \ldots p_k$ представляет собой последовательность записанных друг за другом цепочек $p_1 p_2 \ldots p_k$ (результат выполнения операции

конкатенации цепочек). Таким образом, каждая правая часть уравнений системы (5.4) представляет собой объединение множеств цепочек.

Решением системы (5.4) является набор языков

$(L_1, L_2, \ldots, L_n)$,

если все уравнения системы (5.4) превращаются в тождество при $X_1 = L_1$, $X_2 = L_2, \ldots, X_n = L_n$.

Рассмотрим в качестве примера частный случай системы (5.4), состоящий из одного уравнения

$X = a\,X \cup b\,X \cup X \cup c$

с алфавитом $A=\{a, b, c\}$. Решением этого уравнения является язык

$L = \{\ phi\ c \mid phi \in \{a, b\}^*\}$.

Система (5.4) может иметь несколько решений. Так в рассмотренном примере помимо L решениями являются также

$L_1 = L \cup \{\ phi\ a \mid phi \in \{a, b\}^*\}$

и

$L_2 = L \cup \{\ phi\ b \mid phi \in \{a, b\}^*\}$.

В соответствии с денотационной семантикой в качестве *определяемого* решения системы (5.4) принимается наименьшее решение. Решение

$(L_1, L_2, \ldots, L_n)$

системы (5.4) называется *наименьшим*, если для любого другого решения

$(L_1', L_2', \ldots, L_n')$

выполняются соотношения

$L_1 \subseteq L_1', L2 \subseteq L_2', \ldots, L_n \subseteq L_n'$.

Так в рассмотренном примере наименьшим (а значит, определяемым денотационной семантикой) является решение L.

В качестве метода решения систем уравнений (5.3) и (5.4) можно использовать метод последовательных приближений. Сущность этого метода для системы (5.4) заключается в следующем. Обозначим правые части уравнений системы (5.4) операторами

$T_i\,(X_1, X_2, \ldots, X_n)$.

Тогда система (5.4) примет вид

$$\begin{aligned} X_1 &= T_1(X_1, X_2, \ldots, X_n), \\ X_2 &= T_2\,(X_1, X_2, \ldots, X_n), \\ &\ldots \\ X_n &= T_n\,(X_1, X_2, \ldots, X_n). \end{aligned} \qquad (5.5)$$

В качестве начального приближения решения этой системы принимается набор языков

$(\ L_1[0], \ \ldots, L_n[0] = (\ \varnothing, \varnothing, \ldots, \varnothing\ )$.

Каждое следующее приближение будет определяться по формуле:

$$(\ L_1[i], \ldots, L_n[i]\ ) = (T_1\,(L_1[i-1], \ldots, L_n[i-1]),$$
$$\ldots\ldots\ldots\ldots$$
$$T_n\,(\ L_1[i-1], \ldots, L_n[i-1])\ ).$$

Так как операции объединения и сцепления множеств являются монотонными функциями относительно отношения порядка $\subseteq$, то этот процесс сходится к решению

$(L_1, \ldots, L_n)$

системы (5.5), т.е.

$(L_1, \ldots, L_n) = (T_1(L_1, \ldots, L_n), \ldots, T_n(L_1, \ldots, L_n))$

и это решение является наименьшим. Это решение называют еще *наименьшей неподвижной точкой* системы операторов $T_1, T_2, \ldots, T_n$.

В рассмотренном примере этот процесс даёт следующую последовательность приближений:

$L[0] = \varnothing$,

$L[1] = \{c\}$, $L[2] = \{c, ac, bc\}$,

$L[3] = \{c, ac, bc, aac, abc, bac, bbc\}$,

. . . . . .

Этот процесс сходится к указанному выше наименьшему решению L.

С помощью денотационной семантики можно определять более широкий класс грамматики по сравнению с формой Бэкуса-Наура. Так в форме Бэкуса-Наура не определены правила вида

$X ::= X$

тогда как уравнение вида

$X = X$

имеет вполне корректную интерпретацию в денотационной семантике.

5.5. Аксиоматическая семантика.

В аксиоматической семантике алгебраического подхода система (5.1) интерпретируется как набор аксиом в рамках некоторой формальной логической системы, в которой есть правила вывода и/или интерпретации определяемых объектов.

Для интерпретации системы (5.1) вводится понятие аксиоматического описания (S, E) – логически связанной пары понятий: S – сигнатура используемых в системе (5.1) символов функций $f_1, f_2, \ldots, f_m$ и символов констант (нульместных функциональных символов) $c_1, c_2, \ldots, c_l$, а E – набор аксиом, представленный системой (5.1). Предполагается, что каждая переменная x_i, $i = 1, \ldots, k$, и каждая константа c_i, $i = 1, \ldots, l$, используемая в E, принадлежит к какому-либо из типов данных $t_1, t_2, \ldots, t_r$, а каждый символ f_i, $i = 1, \ldots, m$, представляет функцию, типа

$t_{i1} * t_{i2} * \ldots * t_{ik} \rightarrow t_{i0}$.

Такое аксиоматическое описание получит конкретную интерпретацию, если будут заданы конкретные типы данных $t_i = t_{i'}$, $i = 1, \ldots, r$, и конкретные значения констант $c_i = c_{i'}$, $i = 1, \ldots, l$. В таком случае говорят, что задана одна конкретная интерпретация $\boldsymbol{A}$ символов сигнатуры S, называемая *алгебраической системой*

$\boldsymbol{A} = (t_{1'}, \ldots, t_{r'}, f_{1'}, \ldots, f_{m'}, c_{1'}, \ldots, c_{l'})$,

где $f_{i'}$, $i = 1, \ldots, m$, конкретная функция, представляющая символ f_i. Таким образом, аксиоматическое описание (S, E) определяет класс алгебраических систем (частный случай: одну алгебраическую систему), удовлетворяющих системе аксиом E, т.е. превращающих в тождества равенства системы Е после подстановки в них $f_{i'}$, $i = 1, \ldots, m$, и $c_{i'}$, $i = 1, \ldots, l$, вместо f_i и c_i соответственно.

В программировании в качестве алгебраической системы можно рассматривать, например, тип данных, при этом определяемые функции представляют операции, применимые к данным этого типа. Так К. Хоор построил аксиоматическое определение набора типов данных [5.4], которые потом Н. Вирт использовал при создании языка Паскаль.

В качестве примера рассмотрим систему равенств

УДАЛИТЬ (*ДОБАВИТЬ* (*m*, *d*)) = *m*,

ВЕРХ (*ДОБАВИТЬ*(*m*, *d*)) = *d*,

УДАЛИТЬ (*ПУСТ*) = *ПУСТ*,

ВЕРХ (*ПУСТ*) = *ДНО*,

где *УДАЛИТЬ*, *ДОБАВИТЬ*, *ВЕРХ* – символы функций, а *ПУСТ* и *ДНО* – символы констант, образующие сигнатуру этой системы. Пусть D, $D1$ и M – некоторые типы данных, такие, что $m \in M$, $d \in D$, *ПУСТ* $\in M$, *ДНО* $\in D_1$, а функциональные символы представляют функции следующих типов:

УДАЛИТЬ: $M \to M$,

ДОБАВИТЬ: $M * D \to M$,

ВЕРХ: $M \to D_1$.

Данная сигнатура вместе с указанной системой равенств, рассматриваемой как набор аксиом, образует некоторое аксиоматическое описание.

С помощью этого аксиоматического описания определим абстрактный тип данных, называемый *магазином*. Для этого зададим следующую интерпретацию символов её сигнатуры: пусть D – множество значений, которые могут быть элементами магазина, $D_1 = D \cup \{$ *ДНО* $\}$, а M – множество состояний магазина, $M = \{d_1, d_2, \ldots, d_n \mid d_i \in D, i = 1, \ldots, n, n \geq 0\}$, *ПУСТ* = {}, *ДНО* – особое значение (зависящее от реализации магазина), не принадлежащее D. Тогда указанный набор аксиом определяет свойства магазина.

С аксиоматической семантикой связана логика равенств (эквациональная логика), изучаемая в курсе «Математическая логика». Эта логика содержит правила вывода из заданного набора аксиом других формул.

5.6. Языки спецификаций.

Как уже отмечалось, функциональная спецификация представляет собой математически точное, но, как правило, не формальное описание поведения ПС. Однако, формализованное представление функциональной спе-

цификации имеет ряд достоинств, главным из которых является возможность применять некоторые виды автоматизированного контроля функциональной спецификации.

Под *языком спецификаций* понимается формальный язык, предназначенный для спецификации функций. В нём используется ряд средств, позволяющих фиксировать синтаксис и выражать семантику описываемых функций. Различие между языками программирования и языками спецификации может быть весьма условным: если язык спецификаций имеет реализацию на компьютере, позволяющую как-то выполнять представленные на нём спецификации (например, с помощью интерпретатора), то такой язык является и языком программирования, может быть, и не позволяющий создавать эффективные программы. Однако, для языка спецификаций важно не эффективность выполнения спецификации на компьютере, а её выразительность. Язык спецификации, не являющийся языком программирования, также может быть полезен в процессе разработки ПС (для автоматизации контроля, тестирования и т.п.).

Язык спецификации может базироваться на каком-либо из рассмотренных методов описания семантики функций, а также поддерживать спецификацию функций для какой-либо конкретной предметной области.

Упражнения к Лекции 5.

5.1. Функции

function F (*x, y*: *integer*): *integer;*
function G (*x, y*: *integer*): *integer;*
function R (*x, y*: *integer*): *integer;*

определены с помощью операционной семантики равенствами:

$R(x, y) = x*(y - 1)$,
$F(x, y) = R(x + 1, y) - R(x, y - 1)$,
$G(x, y) = F(x, R(x, y))$.

Найти значения G (3, 3).

5.2. Функции

function F (*n*: *integer*)*: integer;*
function G (*n*: *integer*)*: integer;*

определены с помощью операционной семантики равенствами:

$F(0) = 1$,
$G(0) = 2$,
$F(n) = G(n - 1)$,
$G(n) = F(n - 1) + G(n - 1)$.

Найти значения F(3) *и G*(3).

5.3. Формальные языки E и T определены над алфавитом

{*'a', '*', '&', '<', '>'*}

с помощью денотационной семантики равенствами

$E = T \cup$ *'*'* $T \cup E$ *'&'* T,

T= 'a' ∪ 'a' ∪ '<' E '>'*

Какие из следующих строк

*'*a&*a*&a*' ,*

*'*a&<a&a*>',*

'<*a*&a>&<*a*>*'*

принадлежат языку E и какие из них не принадлежат языку E.

5.4. Тип R определён с помощью следующей аксиоматической семантики.

Описания:

type R= record P1, P2, P3: CHAR end;

function READ(S: R): CHAR; {READ: R → CHAR}

function SHIFT(S: R): R; {SHIFT: R → R}

*function ADD(S: R, C: CHAR): R; {ADD: R * CHAR → R}*

function REMOVE(S: R): R; {REMOVE: R → R}

var X, Y, Z: CHAR;

U: R;

Аксиомы:

SHIFT(ADD(ADD(ADD(U, X), Y), Z)) =

ADD(ADD(ADD(U, Y), Z), X);

REMOVE(U) = SHIFT(ADD(U, '#'));

READ(SHIFT(ADD(U, X))) = X;

Найти значение:

READ(SHIFT(SHIFT(REMOVE(ADD(ADD(U, 'a'), 'b'))))) =

Литература к Лекции 5.

[5.1] В.Н. Агафонов. Спецификация программ: понятийные средства и их организация. – Новосибирск: Наука (Сибирское отделение), 1987.

[5.2] Ian Sommerville. Software Engineering. – Addison-Wesley Publishing Company, 1992.

[5.3] Д. Скотт. Теория решеток, типы данных и семантика / Данные в языках программирования. – М.: Мир, 1982. – С. 25-53.

[5.4] К. Хоор. О структурной организации данных / У. Дал, Э. Дейкстра, К. Хоор. Структурное программирование. – М.: Мир, 1975. – С. 98-197.