ЯЗЫК ДЛЯ ПРОГРАММИРОВАНИЯ НЕПРЕРЫВНЫХ ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОВ

В статье определяются основные концепции языка программирования, предназначенного для реализации непрерывных процессов, взаимодействующих между собой в реальном времени, и тем самым не описываемых классической машиной Тьюринга. Приводится описание синтаксиса и семантики языка, а также пример программы для решени конкретной задачи. Намечаются пути дальнейшего развития языка и обсуждаются проблемы, которые могут возникнуть при его реализации.

1. Введение

В настоящее время большую актуальность приобрели работы по проектированию и созданию вычислительных сетей (в частности, нейронных), в которых каждый элемент решает конкретную задачу (например, оптимизацию некоторой целевой функции), причем обмен данными между элементами происходит в режиме реального времени (т.е. практически непрерывно). Результат вычисления накапливается в некоторой области памяти как суммарный итог процесса. Работа сети прекращается при выполнении некоторого условия. Иногда при определенных условиях происходит перестройка системы с изменением ее конфигурации и функций отдельных устройств. При этом сохраняется часть накопленных данных. Теоретически работа такой системы, будучи непрерывной, не описывается классической машиной Тьюринга [1].

Существующие языки программирования, представлющие собой в той или иной мере реализацию классической машины Тьюринга, малопригодны для описания функционирования подобных систем. Эти языки, как правило, описывают дискретные процессы, состоящие из последовательности отдельных операций, а встроенные в них средства параллельного программирования предназначены для обмена сообщениями в определенные моменты времени. Не предусмотрены в этих языках и средства для работы в реальном времени, такие, например, как описание зависимости процесса от времени и предыстории самого процесса.

В настоящей статье делается попытка описания языка, свободного от перечисленных недостатков. Мы стремились определить только минимум

базовых концепций языка, которых, тем не менее, было бы достаточно дл простого описания данных систем. Чтобы сосредоточиться на описании базовых концепций и не отвлекаться на определение конструкций, общих для всех языков программирования (таких, как арифметические выражения, условные предложения, циклы и вообще средства для описания обычных дискретных последовательных процессов), мы предусмотрели возможность включать в программу фрагменты на стандартном языке программирования. В будущем, вероятно, предстоит разработать синтаксис и семантику таких конструкций.

Следует отметить, что реализация предлагаемого языка отнюдь не будет тривиальной задачей. Как выходящий за рамки классической машины Тьюринга, он, вообще говоря, не предназначен для реализации на обычных цифровых вычислительных устройствах (хотя его моделирование в той или иной степени на таких устройствах, разумеется, возможно), а только на аналоговых, нейро-, биокомпьютерах и т.п. Тем не менее мы старались исключить из языка особенности, чрезмерно усложняющие реализацию (например, необходимость сохранять информацию о всей предыстории процесса). Предполагается, что при реализации будут широко использоваться средства оптимизации кода и памяти. По этой причине, если будет принято решение об использовании внутри программы фрагментов на стандартном языке программирования, потребуется существенно переработать его транслятор и систему поддержки (хотя наш язык определен так, что фрагменты на стандартном языке программирования могут транслироваться и исполняться совершенно автономно). Заметим, что структурные единицы языка могут транслироваться независимо друг от друга.

Статья построена следующим образом. Сначала дается неформальное описание базовых конструкций языка. Далее следует строгое описание их синтаксиса и семантики. Мы не стремились к излишнему формализму описания в духе грамматик ван Вейнгаардена [2], нашей целью было дать описание, достаточное для реализаторов языка и исключающее двусмысленности. Многие вещи, впрочем, оставлены на усмотрение реализации. После этого приводится пример программы, решающей конкретную оптимизационную задачу. В заключение обсуждаются возможности дальнейшего развития языка путем добавления в него новых конструкций.

2. Базовые конструкции языка

2.1. Модули

Модуль — это основная составляющая программы, представляющая один из параллельных процессов. Модули бывают двух видов: собственно модули и модули-процедуры. Модуль-процедура — это процедура, написанная на стандартном языке программирования.

Один из модулей (по имени *main*) является главным. Он запускается сразу после начала работы программы, а его остановка означает конец ее работы. Модуль *main* может запускать другие модули, те, в свою очередь — следующие и т.д. Рекурсивный запуск модулей (кроме модулей-процедур) недопустим, но можно запустить сразу много экземпляров одного и того же модуля (подробнее об этом ниже, в разделе "Множественный запуск модуля").

Все модули работают параллельно. Это значит, что модуль запускаетс сразу после начала работы запустившего его процесса (о процессах см. следующий раздел) и останавливается автоматически после остановки этого процесса. Даже если он по какой-то причине остановился раньше, его данные сохраняются неизменными вплоть до остановки процесса.

Единственный способ обмены данными между модулями - через их параметры. Параметры делятся на три категории: постоянные, неизменяемые и изменяемые. Постоянные параметры не изменяются ни в данном модуле, ни в каком-либо другом. Неизменяемые параметры не могут изменяться ни в данном модуле, ни в любом модуле, прямо или косвенно запущенном из него, но могут изменяться (и предполагается, что изменяются) в каком-либо ином модуле. Изменяемые параметры могут меняться в данном модуле либо в каком-либо модуле, прямо или косвенно запущенном из него. Заметим, что во избежание конфликтов люба переменная может меняться только в одном модуле из работающих одновременно. Таким образом, если параметр принят как постоянный, он может быть передан дальше только как постоянный. Если он принят как неизменяемый, он может быть передан только как неизменяемый. Если же он принят как изменяемый, он может быть передан как изменяемый либо как неизменяемый, причем первый вариант допустим, только если

он не меняется в данном процессе и только для одного модуля из всех, запускаемых в этом процессе.

Все параметры в собственно модули передаются по ссылке. В модули-процедуры изменяемые параметры передаются по ссылке, постоянные — по значению. Неизменяемые параметры в модули-процедуры передаваться не могут, так как большинство языков программирования не имеют возможности простым образом запретить изменение параметров, передаваемых по ссылке.

Модуль *main* принимает параметры так же, как и любой другой. Подготовка входных и обработка выходных параметров остаются на усмотрение реализации. Вероятно, для этого понадобится писать некоторый драйвер на стандартном языке программирования.

Подчеркнем, что все переменные там, где они меняются, меняются непрерывно. Непрерывно осуществляется и взаимодействие между модулями.

2.2. Процессы

Каждый модуль состоит из нескольких процессов (может быть, одного). Процессы запускаются последовательно: прекращение одного процесса означает либо запуск следующего, либо прекращение работы модуля. Первым запускается процесс с именем *main_process*, который должен присутствовать в каждом модуле. Далее процессы могут запускаться в любом порядке, в том числе и по нескольку раз. Процесс *main_process* также может быть запущен повторно. Как уже говорилось, по окончании процесса прекращают работу все модули, которые были из него запущены. При запуске нового процесса ему может быть передана часть данных из предыдущего. Параметры модуля доступны во всех его процессах.

2.3. Переменные

Переменные определяются внутри процесса и локальны в нем. Для каждой переменной задаются закон ее изменения и начальное значение, причем задание последнего обязательно. Закон изменения переменной может быть задан либо в том процессе, в котором она определена, либо в некотором модуле, запущенном (прямо или косвенно) из этого процесса.

В последнем случае она должна быть передана в этот модуль как изменяемый параметр. Закон изменения переменной может быть задан и в нескольких модулях, но только при условии, что они не запускаются одновременно. Если закон изменения переменной не задан нигде, то она является константой. Начальное значение переменной должно вычисляться статически.

Если в процессе, в котором определена переменная, задан закон ее изменения, то она может быть передана в запускаемые из него модули только как неизменяемый параметр. Если же закон изменения переменной не задан, то в один (и только один!) из модулей, запускаемых из данного процесса, она может быть передана как изменяемый параметр; в остальные модули она передается как неизменяемый параметр. В противном случае переменная становится константой и может передаваться либо как постоянный, либо как неизменяемый параметр. Правила дальнейшей передачи переменной в качестве параметра описаны выше в разделе "Модули".

Начальное значение переменной может быть передано из предыдущего (динамически) процесса; в этом случае начальное значение, указанное в описании переменной, игнорируется.

2.4. Версии

По умолчанию в качестве текущего значения переменной берется ее усреднение по промежутку времени $(t-\epsilon, t)$, где t - текущее значение времени. При необходимости переменную можно определить как *версию* другой переменной, представляющую собой либо ее усреднение по промежутку $(t-a, t-b)$, где $a > 0,\ b \geq 0,\ a > b$, либо значение в момент $t-c$ $(c > 0)$. Значения a, b и c должны вычисляться статически. Задание переменной в качестве версии приравнивается к заданию закона ее изменения; начальное значение в этом случае задавать не нужно. Следует помнить, что значения переменной в отрицательные моменты времени считаютс равными ее начальному значению; если не учитывать это правило, то можно получить неожидаемые результаты. Единица измерения времени и синхронизация по времени с фрагментами на стандартном языке программирования остаются на усмотрение реализации.

2.5. Константы и функции

Константы и функции определяются вне какого-либо модуля и глобальны во всей программе. Значение константы должно вычисляться статически; функции — это просто функции на стандартном языке программирования. Константы и функции могут передаваться как постоянные параметры.

2.6. Массивы

Переменные, константы и функции могут объединяться в массивы. Многомерные массивы не предусмотрены, однако возможны массивы массивов любого уровня вложенности. Правила обращения с массивами те же, что и со скалярными переменными, константами и функциями.

Нижняя граница массива всегда равна 1; верхняя граница определяется начальным значением. Для задания начального значения предусмотрена *запись массива,* которая может содержать как выражения для отдельных переменных, так и циклические предложения для присвоения значений нескольким элементам массива. Нижняя граница цикла всегда равна 1, независимо от того, для присвоения начального значения элементам с какими номерами он предназначен. Верхняя граница цикла должна определяться статически. Все элементы массива массивов должны иметь одинаковую длину.

2.7. Множественный запуск модуля

Один и тот же модуль может быть запущен одновременно в нескольких экземплярах, вероятно, с разными параметрами. В этом случае в качестве параметров ему должны быть переданы массивы соответствующих типов, причем, разумеется, они должны иметь одно и то же число элементов, которое и определяет количество запускаемых экземпляров модуля.

3. Формальное описание языка

В этой главе мы не будем повторять семантические правила, изложенные в предыдущей главе, будем только при необходимости их уточнять. В основном же мы сосредоточимся на описании синтаксиса. Мы также не будем раскрывать общепринятые понятия, такие, как "идентификатор".

При описании грамматики будут применяться следующие обозначения:
< ... > — понятие.
(...) — группа понятий, трактуемая как отдельное понятие.
<понятие> <понятие> — понятия должны следовать друг за другом.
<понятие> | <понятие> — должно присутствовать одно из двух понятий.
<понятие>? — необязательное понятие.
<понятие>* — понятие может повторяться сколько угодно раз или отсутствовать.
<понятие>+ — понятие может повторяться сколько угодно раз, но не менее одного.

Если скобка (или) должна представлять саму себя, то она заключается в апострофы.

3.1. Выражения

<выражение> — это выражение на стандартном языке программирования. Под стандартным языком программирования далее везде понимается Алгол 68 [2]-[4], однако реализация может предусмотреть любой другой стандартный язык программирования, соответственно изменив семантику выражений, или разработать свой собственный.

Если не оговорено иное, то выражение представляет собой основу в сильной позиции, выдающую **real.** Используемые в выражениях скалярные внешние переменные и константы имеют вид **real,** массивы переменных и констант — одномерный массив из **real** соответствующего уровня вложенности, функции и их массивы — в соответствии с их описанием (см. ниже "Описания функций и массивов функций"). Идентификаторы модулей и процессов в выражениях встречаться не могут.

<константное выражение> — это выражение, в котором в качестве внешних переменных могут встречаться только константы, постоянные параметры модуля, переменные, для которых не задан закон изменения и которые во все модули передаются как постоянные либо неизменяемые параметры, переменные цикла и функции.

<статическое выражение> — это выражение, не содержащее внешних переменных. Заметим, что идентификаторы из стандартного и библиотечного вступлений (такие, как *sin*) внешними переменными не считаются.

3.2. Программа

<программа> ::= <описание>* <описание модуля main> <описание>*
<описание> ::= <описание константы> | <описание массива констант>
| <описание функции> | <описание массива функций> |
<описание модуля> | <фрагмент на стандартном языке программирования>

Порядок описаний не имеет значения. Первым в любом случае запускается модуль *main,* далее — в соответствии с правилами, изложенными в предыдущей главе.

Семантика фрагментов на стандартном языке программирования остается на усмотрение реализации. В частности, они могут потребоваться для связи программы с внешним миром (хотя бы для операций ввода-вывода) или дл передачи параметров в модуль *main,* если таковые имеются.

3.3. Комментарии

Чтобы чрезмерно не усложнять синтаксис, мы не стали включать в него понятия комментария. Комментарий — это любой текст, заключенный в символы ## или **comment comment.** Комментарии могут встречаться в любом месте программы, кроме как внутри ключевых слов и идентификаторов.

Примеры.
это комментарий#
comment *это комментарий, содержащий внутри себя символ* #
comment

3.4. Описания констант и массивов констант

<описание константы> ::= **const** <идентификатор> =
<статическое выражение>;
<описание массива констант> ::= **array**+ <описание константы>

Выражение не должно содержать внешних идентификаторов. В случае массива оно должно выдавать одномерный массив из **real** уровня вложенности, равного количеству слов **array.**

Примеры.
const $pi = 3.1416$;
array const $porridge = (1.2, 65e-4, sin2)$;

3.5. Описания функций и массивов функций

<описание функции> ::= <заголовок функции> =
<статическое выражение>;
<заголовок функции> ::= **function** <идентификатор>
('(' <параметр> (, <параметр>)* ')')?
<описание массива функций> ::= **array**+ <описание функции>;
<параметр> ::= **array***

В описании функции выражение должно выдавать процедуру, выдающую **real.** Число параметров процедуры должно быть равно числу параметров функции; вид параметров должен быть **real,** если на соответствующем месте в списке параметров функции пусто, либо одномерный массив из **real** уровня вложенности, равного числу слов **array** на соответствующем месте в списке параметров функции.

В описании массива функций выражение должно выдавать одномерный массив из процедур соответствующего вида уровня вложенности, равного числу слов **array** в начале описания.

Пример.

function$minge$(**array**,) =
функция выдает наименьший элемент массива, больший данного, а если таких нет, то наибольшее целое, доступное в системе
#
([ ] **real**r, **real**x) **real** :
(**real**res := $maxreal$;
for$i$ **to upb**r **do**
realri = $r[i]$;
ifri > x **and**ri < res **then**res := ri **fi**
od;
res);

3.6. Описания модулей

<описание модуля> ::= **module** <идентификатор>
((<параметры собственно модуля>? <тело собственно модуля>) |
(<параметры модуля-процедуры>? = <выражение>));
<параметры собственно модуля> ::= '(' <параметр собственно модуля>

(,<параметр собственно модуля>)* ')'
<параметры модуля-процедуры> ::= '(' <параметр модуля-процедуры>
(,<параметр модуля процедуры>)* ')'
<параметр собственно модуля> ::= <параметр модуля-процедуры> |
(**value**? **array*** <идентификатор>)
<параметр модуля-процедуры> ::= (**array*** <заголовок функции>) |
((**const** | **var**) **array*** <идентификатор>)

const обозначает постоянный параметр, **value** — неизменяемый, **var** — изменяемый.

Пример.
module*mod*1(**function***f*(**array**,), **const***c*, **value array array***vl*,
var array*vr*)
тело модуля

3.7. Тело модуля

<тело модуля> ::= '(' <описание процесса>*
<описание процесса *main_process*> <описание процесса>* ')'
<описание процесса> ::= **process** <идентификатор> '(' (<фраза> ;)*
<завершение процесса> (; <фраза>)* ')'
<фраза> ::= <описание переменной> | <описание версии> |
<запуск модуля> | <завершение процесса>
<завершение процесса> ::= <завершение модуля> | <запуск процесса>

Порядок описаний процессов не имеет значения. Первым в любом случае запускается процесс *main_process*, далее — по очереди в соответствии с операторами завершения процесса, имеющимися в теле модуля (см. ниже "Завершение процесса").

Порядок фраз имеет некоторое значение только для операторов завершения процесса (подробнее см. там же).

3.8. Описание переменной

<описание переменной> ::= <описание новой переменной> |
<переопределение параметра модуля>
<описание новой переменной> ::= <описание скалярной переменной> |
<описание массива>

<описание скалярной переменной> ::= <идентификатор>
<закон изменения переменной>? **init** <константное выражение>
<описание массива переменных> ::= **array**+ <идентификатор>
<закон изменения массива>? **init** <константный вектор>

После **init** следует начальное значение переменной. Константный вектор отличается от обычного тем, что все выражения в нем должны быть константными (далее это специально не оговаривается).

<переопределение параметра модуля> ::=
<переопределение скалярного параметра> | <переопределение массива>
<переопределение скалярного параметра> ::= <идентификатор> <закон изменения переменной>
<переопределение массива> ::= **array** <идентификатор>
<закон изменения массива>

Цель переопределения параметра — задать закон его изменения, поэтому переопределять можно только изменяемые параметры.
<закон изменения переменной> ::= = <выражение>
<закон изменения массива> ::= = <вектор>

Переменная меняется непрерывно в соответствии с изменением переменных, входящих в закон ее изменения (которые могут изменяться в том числе и в модулях-процедурах) и/или в зависимости от времени. Дл обозначения времени служит системная переменная t. Рекурсивные законы изменения и законы изменения со взаимными ссылками недопустимы; в силу этого в закон изменения параметра не должны входить неизменяемые параметры того же модуля. Однако в законе изменения переменной допустимы прямые или косвенные ссылки на версии этой же переменной, если они целиком обращены в прошлое, т.е. верхняя граница промежутка, по которому происходит усреднение значений переменной, меньше текущего значения времени. Отсюда понятна необходимость указывать начальное значение, даже если указан закон изменения переменной.

<вектор> ::= <выражение> | <цикл> | <запись массива>

Вектор должен иметь значение одномерного массива из **real** соответствующего уровня вложенности и такое же значение должно выдавать выражение.

<цикл> ::= **for** <идентификатор> **to** <константное выражение> **do** <вектор> **od**

Выражение (верхняя граница цикла) должно выдавать **union (int, real).** Если значение выражения есть **real,** то оно округляется при помощи операции **round.**

Вектор (тело цикла) должен иметь значение одномерного массива из **real** на единицу меньшей вложенности (либо **real,** если исходная вложенность была равна 1).

Переменная цикла локальна в теле цикла, кроме верхних границ внутренних циклов (иначе было бы возможно определять непрямоугольные массивы).

<запись массива> ::= [<вектор> (,<вектор>)+]

Если вектор является выражением или записью массива, то он должен иметь значение одномерного массива из **real** на единицу меньшей вложенности (либо **real,** если исходная вложенность была равна 1; в этом случае допустимо только выражение), если циклом — той же вложенности.

Примеры.

a **init**0

$b = x * t$ **init**7.6

array array$r$ **init**

for$i$ **to**10 **do**

$[i**2,$ **for**j **to**5 **do**$i * j$ **od,**1000]

od

3.9. Описание версии

<описание версии> ::= **version** <идентификатор>

'(' <идентификатор переменной>, <статическое выражение>

(,<статическое выражение>)? ')'

Если присутствует только одно выражение со значением a $(a > 0)$, то берется значение переменной в момент $t - a$. Если присутствуют два выражения со значениями соответственно b и c $(b > c \geq 0)$, то берется среднее значение переменной по промежутку $(t - b,\ t - c)$.

Примеры.

version$lastx(x, 1)$

version$oldy(y, 20, 10)$

3.10. Запуск модуля

<запуск модуля> ::= <одиночный запуск модуля> |
<множественный запуск модуля>
<одиночный запуск модуля> ::= **call** <идентификатор модуля>
'(' <выражение> (,<выражение>)* ')'
<множественный запуск модуля> ::= **call array**+ <идентификатор модуля>
'(' <вектор> (,<вектор>)* ')'

Выражения (вектора) в скобках представляют собой параметры, передаваемые в модуль. Количество и типы параметров должны соответствовать количеству и типам параметров в описании модуля. При множественном запуске модуля вместо параметров должны передаваться их массивы уровня вложенности, равного количеству слов **array,** содержащие одинаковое количество элементов на всех уровнях вложенности. Количество запускаемых экземпляров модул определяется количеством скалярных элементов в каждом из массивов. Все выражения, в том числе и входящие в состав векторов, должны быть идентификаторами либо вырезками, первичным которых является идентификатор, а всеми индексами — константные выражения. Рекурсивный запуск модулей недопустим.

Примеры.
call$module1(x, y[5])$
call array$module2$ # *запускаем 10 штук* #
$(r,$ # *r - это массив длиной 10* #
for$i$ **to**10 **do**$z[i]$ **od**$)$

3.11. Завершение процесса

<завершение процесса> ::= <завершение модуля> |
<запуск нового процесса>
<завершение модуля> ::= **stop** '(' <выражение> ')'
<запуск нового процесса> ::= **switch** <идентификатор процесса>
'(' <выражение> ')' (<идентификатор> = <вектор>
(,<идентификатор> = <вектор>)*)?

Выражение в скобках должно выдавать **bool** и представляет собой условие завершения процесса. Процесс завершается, как только становится истинным какое-либо из условий, указанных в операторах завершения

процесса. Если несколько условий становятся истинными одновременно, то выполняется оператор, первый по тексту.

В конце оператора запуска нового процесса могут указываться начальные значения, которые следует присвоить некоторым из переменных, определенных в запускаемом процессе. Начальные значения, указанные в описаниях таких переменных, в этом случае игнорируются. Присвоение начального значения одной и той же переменной дважды недопустимо.

Примеры.

stop$(a - b < eps)$

switch$newprocess(x > y)a = 1/z$

3.12. Контекстные условия на идентификаторы

В языке имеется пять уровней вложенности идентификаторов ("слоев" в терминологии Алгола 68), причем два последних сами могут содержать сколь угодно много уровней вложенности:

1. Идентификаторы констант, функций, модулей, системной переменной *t,* других системных переменных, если они предусмотрены реализацией, идентификаторы, входящие в стандартное и библиотечное вступления.
2. Идентификаторы параметров модулей и процессов внутри модуля.
3. Идентификаторы переменных внутри процесса.
4. Идентификаторы переменных цикла.
5. Внутренние идентификаторы выражений.

Одинаковые идентификаторы на одном уровне вложенности недопустимы. Одинаковые идентификаторы на разных уровнях вложенности допускаютс (в этом случае внутренний идентификатор перекрывает внешний), но их применение нежелательно из-за возможности ошибок.

4. Пример программы

Пусть у нас имеется модуль *minimize,* минимизирующий по y заданную функцию $f(y, s)$, где вектор параметров s меняется в реальном времени, при заданном начальном приближении y_0. Пусть заданы функции $g_1(y, s), \ldots, g_n(y, s)$. Наша задача состоит в минимизации неизвестной функции $G(y, s)$, для которой известны значения минимумов $e(w_1), \ldots, e(w_m)$

на некоторой выборке значений постоянных параметров $w_1, \ \ldots, \ w_m$. Мы будем искать функцию g_i, минимумы которой на данной выборке наилучшим образом приближают минимумы функции G, иначе говоря, число $i \in \{1, \ \ldots, \ n\}$, минимизирующее функционал

$$F(i) = \sum_{j=1}^{m} |\min_y g_i(y, w_i) - e(w_i)|.$$

Для этого мы запустим nm экземпляров модуля *minimize,* причем экземпляр с номером (i, j) будет минимизировать функцию $g_i(y, w_j)$. Обозначим r_{ij} текущее приближение минимума каждой из этих функций. Запустим еще один экземпляр модуля *minimize,* минимизирующий функцию G_{ctrl} с параметрами размерности n :

$$G_{ctrl}(y, s) = \sum_{i=1}^{n} |y^{(i)}|s^{(i)} + K|1 + \sum_{i=1}^{n} |y^{(i)}||,$$

где

$$s^{(i)} = \sum_{j=1}^{m} |r_{ij} - e(w_i)|.$$

При достаточно большом $K(K \gg \max_i g_i(y_0, s_i))$ минимум этой функции по y равен $\min_i s^{(i)}$, т.е. при $t \to \infty$ он стремится к $\min_i F(i)$, причем это значение достигается при $y^{(k_{min})} = 1, \ y^{(i)} = 0, \ i \neq k_{min}$. Обозначим y_t текущее приближение вектора оптимизируемых параметров данной функции.

В ячейке *res* будет помещаться значение

$$\sum_{i=1}^{n} i y_t^{(i)}.$$

При $t \to \infty$ это значение сойдется к номеру искомой функции.

Ниже приведена программа, реализующая указанный алгоритм. Желающие могут для сравнения попытаться составить аналогичную программу на стандартном языке программирования.

const$shift = 0.1$;

```
consteps = 0.0001;
array functiong(, ) =
#g #;
functionctrl(array, array) =
(ref[ ] realy, s) real:
(realres := 0, sum := 0;
fori to upby do
res+ := absy[i] * s[i];
sum+ := absy[i]
od;
res+ := s[upbs] * abs(1 − sum));
moduleminimize
(functionf(array, array), var arrayy0,
value arrays, varres)
# #;
module main(const arraye, bf varres)
(processmain
(array array y0 init# #;
array array r      init fori to upbe do
forj to upbg do0 od
od;
k = (realmaxy0 := y0[1][1];
fori to upbe do
forj to upbg do
realy0ij = y0[i][j];
ify0ij > maxy0 thenmaxy0 := y0ij fi
od
od; maxy0 * 2) init0;
arraysumg =
[fori to upbg do
(realrsum := 0;
forj to upbe do
rsum := abs(r[j][i] − e[i])
od;
rsum) od, k]
```

$\#k\#$
init for$i$ **to upb**$g+1$ **do** 0 **od**;
array $yctrl$ **init for**i **to upb**g **do**0.5 **od**;
$resctrl$ **init**0;
$res :=$ (**real**$ressum := 0$;
for$i$ **to upb**g **do**$ressum+ := i * yctrl[i]$ **od**;
$ressum$) **init**1;
version$lastres(res, shift)$;
call array array$minimize$
(**for**i **to upb**e **do**g **od**, $y0$,
for$i$ **to upb**e **do**
for$j$ **to upb**g **do**$e[i]$ **od**
od,r);
call$minimize(ctrl, yctrl, sumg, resctrl)$;
stop(**abs**$(res - lastres) < eps$ **and**$t > shift)))$

5. Заключение

Выше были перечислены только базовые концепции языка, необходимые для его корректного определения. Возможно дальнейшее расширение языка путем включения в него дополнительных конструкций (различные типы данных, в т.ч. структурные, присвоение начальных значений переменным по умолчанию, блокировка изменения переменных при выполнении определенного условия, дополнительные средства синхронизации между модулями и т.д.).

Особо следует остановиться на возможных средствах обмена информацией с внешним миром. С точки зрения концепции языка наиболее подходящим было бы вынесение средств обмена в отдельный модуль (модули), возможно, с применением дополнительных средств синхронизации, однако допустимы и другие решения.

В статье не было предложено никакого названия для языка. По нашему мнению, этот вопрос подлежит дальнейшему обсуждению; в качестве рабочего варианта при подготовке статьи применялось название Алгол 2000.

СПИСОК ЛИТЕРАТУРЫ

1. A.Turing. *On computable numbers, with an application to the Einsteindungsproblem.* Proc. London Math. Soc., vol. 42, pp. 230–265, 1936.

2. *Пересмотренное сообщение об Алголе 68.* М., Мир, 1979.

3. Г.Ф.Дейкало, Б.А.Новиков, А.П.Рухлин, А.Н.Терехов. *Новые средства программирования для ЕС ЭВМ: Транслятор с языка Алгол 68 и диалоговая система JEC.* М., Финансы и статистика, 1984.

4. А.Г.Владимирович, И.С.Золотухина, Ю.И.Карпов, Н.Б.Скачков, Г.С.Цейтин. *Отладочный транслятор-интерпретатор программ на Алголе 68.* Программирование, No. 2, 1990, с.21–27.