информатика

Компьютерные инструменты в образовании, 2014
№ 5: 16–24
УДК: 519.682.2
http://ipo.spb.ru/journal

РЕАЛИЗАЦИЯ ТРАНСЛЯТОРА ДЛЯ ЯЗЫКА ПРОГРАММИРОВАНИЯ РЕФАЛ-5Е СО ВСТРОЕННЫМ ИНТЕРПРЕТАТОРОМ И ВОЗМОЖНОСТЬЮ ПОДКЛЮЧЕНИЯ БИБЛИОТЕК КОДА

Гошев Владимир Андреевич

Аннотация

Язык программирования рефал-5е расширяет существующий диалект языка программирования рефал, названный рефал-5, и позволяет решать современные важные задачи программирования эффективнее, чем другие диалекты языка рефал при использовании современной вычислительной техники. Среди особенностей языка программирования рефал-5е можно выделить такие, как: наличие встроенного интерпретатора, возможность подключения внешних динамически загружаемых библиотек кода, добавление возможности возврата функциями таких значений, как «Успех» и «Неуспех», возможность реализации функций высшего порядка и анонимных функций. Также транслятор языка программирования рефал-5е использует возможности современных многопроцессорных компьютеров, что позволяет увеличить скорость выполнения программ в несколько раз.

Ключевые слова: *рефал-5, рефал-5е, компиляция, трансляция, интерпретация, бэктрекинг.*

1. ВВЕДЕНИЕ

Язык рефал-5е — современный эффективный диалект языка программирования рефал, Он описан в работе [5]. Диалект рефал-5е основан на наиболее популярном диалекте языка рефал, называемом рефал-5.

Основные особенности языка программирования рефал-5е [5] — это наличие таких конструкций языка, как вызов из программы встроенного интерпретатора для исполнения динамически создаваемого кода, анонимные функции, функции высшего порядка и замыкания. Язык рефал-5е также содержит некоторые возможности, свойственные языкам рекурсивно-логического программирования: возможность возврата функцией таких значений, как «Неуспех» и «Успех», а также наличие элементов бэктрекинга. Также добавлено удобное взаимодействие с динамически загружаемыми библиотеками кода, написанными на других языках программирования.

Как было сказано выше, диалект рефал-5е основан на более раннем диалекте языка рефал — рефал-5. Язык рефал-5 и его синтаксис описан в работе В.Ф. Турчина [11].

2. ТРАНСЛЯТОР ЯЗЫКА, ОБЩАЯ СХЕМА РАБОТЫ

Транслятор языка рефал-5е написан на языке С и по своей структуре похож на трансляторы других диалектов языка программирования рефал тем, что так же состоит из двух исполняемых файлов:

2. refc — программа, которая компилирует исходный код программы на языке рефал-5е в язык сборки рефал-5е машины (Р-код).

3. refgo — рефал-5е-машина или, другими словами, интерпретатор языка сборки рефал-5е машины.

Кратко и в упрощенном виде алгоритм работы компилятора (refc) можно представить следующим образом:

1. Компилятор открывает переданный ему файл с кодом рефал-5е программы и передает его лексическому анализатору.

2. Лексический анализатор читает файл и, пропуская незначащие символы и комментарии, формирует термы, которые передаются синтаксическому анализатору.

3. Синтаксический анализатор последовательно анализирует каждый терм и, если необходимо, предыдущие термы и на основе этого анализа делает выводы о свойствах анализируемой конструкции.

4. Если синтаксический анализатор полностью получил некоторую конструкцию, он сохраняет ее в специальной таблице.

5. После того как весь файл успешно прочитан, компилятор переводит созданную таблицу в новый файл в формате языка сборки рефал-5е машины.

6. В случае если лексический или синтаксический анализатор не смог разобрать какую-то конструкцию, то сначала производится откат к предыдущим термам и проверяются другие возможные варианты, а если все они исчерпаны, то считается, что исходный код неверен, и компилятор выдает ошибку с описанием того, где она обнаружена, и предложениями её исправления, если это возможно (например, компилятор ожидал символ «;», но он не был найден).

Алгоритм работы интерпретатора рефал-5е в упрощенном виде можно представить следующим образом:

1. Рефал-5е машина читает переданный ей файл с программой на языке сборки рефал-5е машины и загружает необходимые данные в память.

2. Если программа использует другие модули, то рефал-5е машина читает файлы необходимых модулей и загружает в память необходимую информацию.

3. Если некоторый модуль используется для доступа к внешней библиотеке кода, то рефал-5е машина ищет и загружает эту библиотеку и ищет в ней необходимые функции.

4. В случае, если программу, один из модулей или библиотеку кода не удалось найти или загрузить, рефал-5е машина сообщает об ошибке, освобождает память и завершает свое выполнение.

5. Рефал-5е машина создает потоки выполнения в количестве, равном количеству центральных процессоров компьютера.

6. После того как программа и все необходимые модули и библиотеки загружены, происходит поиск входной точки программы (функции Main в главном файле программы), которая помещается в поле зрения рефал-5е машины. Затем начинается выполнение кода.

7. При вызове функции (которой, в частности, является точка входа в программу) и передаче (возможно, пустого) аргумента рефал-5е машина последовательно сравнивает каждый из образцов с переданным функции аргументом, и если какой-то из образцов подошел,

то проверяются условия его применения. Если условия также выполняются, то рефал-5е машина начинает выполнение действий, связанных с данным образцом и условиями. В противном случае производится возврат к одному из предыдущих условий, а при их отсутствии — к образцу, для увеличения на один терм последней е-переменной, для которой это возможно. Более подробно сопоставление с образцом будет описано в разделе описания работы рефал-5е машины.

8. Если рефал-5е машина встретила команду вызова функции и эта функция не создает никаких побочных действий, то она добавляется в очередь выполнения с указанием, куда должны быть помещены результаты выполнения функции. Далее начинается выполнение этой функции. При этом имеется возможность параллельного выполнения функций, что в ряде случаев позволяет значительно увеличить быстродействие программы.

9. Если рефал-5е машина встретила команду вызова встроенного интерпретатора, то она берет передаваемый код функции и осуществляет рекурсию, то есть передает его встроенному транслятору, который выдает скомпилированный код функции. Рефал-5е машина помещает его в таблицу данного модуля или программы и выполняет ее.

10. В случае, когда ни один из образцов не подошел или условия с учетом бэктрекинга не были выполнены, результатом выполнения данной функции является «Неуспех».

11. После того как рефал-5е программа выполнена, рефал-5е машина освобождает память, занятую программой и модулями, выгружает загруженные библиотеки и завершает свое выполнение.

3. ЯЗЫК СБОРКИ РЕФАЛ-5Е МАШИНЫ

Язык сборки рефал-5е машины был разработан таким образом, чтобы рефал-5е машина, которая его использует, работала более эффективно. Язык сборки рефал-5е машины представляет собой иерархическую древовидную структуру, на верхнем уровне которой находятся директивы компилятора (такие, как импорт модуля) и функции, определенные в программе или модуле. Каждая функция состоит из имени функции и упорядоченного списка триплетов, которые состоят из образца, списка условий и действия. Рассмотрим каждый из этих элементов подробнее.

В языке программирования рефал-5е образец состоит из таких элементов, как литералы, числа, макро-цифры, рефал-символы, e-, t- и s- переменные и структурные скобки. В языке сборки рефал-5е машины образец представлен как упорядоченный список, в котором данные элементы идут в том порядке, как это было указано разработчиком программы. Структурные скобки при этом представлены таким же упорядоченным списком элементов, в них содержащихся. Таким образом, описание образца является рекурсивным.

Каждое условие является парой: вызов функции и образец или пара образцов, такой парой они и записаны в языке сборки рефал-5е машины. Так как у каждого образца может быть несколько условий, то условия записаны в виде упорядоченного списка.

Действия могут быть представлены одним из трех вариантов:

1. В виде подфункции, в таком случае, действия записываются в виде пары -аргумент и упорядоченный список триплетов, состоящих из образца, списка условий и действия.

2. В виде строки, указывающей последовательность действий.

3. В виде нескольких строк с последовательностями действий. В таком случае все они должны быть заключены в фигурные скобки. Рефал-5е машина выполнит первую строку, и если при выполнении указанных действий будет получено значение «Неуспех», то интерпретатор перейдет к выполнению следующего варианта действий. Если все строки действий вернули «Неуспех», то и сама функция возвращает «Неуспех».

Строка действий представлена в языке сборки рефал-5е машины в виде упорядоченного списка выполняемых действий, при этом вызов функции записывается как пара- имя функции и аргумент функции.

4. ОПИСАНИЕ РАБОТЫ КОМПИЛЯТОРА ЯЗЫКА РЕФАЛ-5Е В ЯЗЫК СБОРКИ РЕФАЛ-5Е МАШИНЫ

Как было сказано ранее, компилятор читает переданный файл, используя лексический анализатор. Лексический анализатор работает следующим образом:

1. Лексический анализатор читает очередной символ.

2. Если не удалось прочитать очередной символ, так как достигнут конец файла, то анализатор сообщает об этом.

3. Если указано, что необходимо пропускать пробельные символы, то прочитанный символ проверяется, является ли он пробельным, если проверка успешна, то происходит возврат к пункту 1.

4. Если прочитанный символ «/» (косая черта) и указано, что необходимо пропускать комментарии, то анализатор смотрит на следующий символ. Затем:

4.1. Если следующий символ «/» (косая черта), то обнаружен однострочный комментарий, и анализатор пропускает все символы до перевода строки, и происходит возврат к пункту 1.

4.2. Если следующий символ «*» (звездочка), то обнаружен блочный комментарий, и пропускаются все символы, пока не встретится подстрока «*/» (звездочка и косая черта).

4.3. В противном случае алгоритм продолжает работу.

5. Полученный символ и информация о текущей позиции возвращаются вызывающей функции.

Синтаксический анализатор является LL(*)-анализатором, то есть нисходящим синтаксическим анализатором для подмножества контекстно-свободных грамматик. Он анализирует входной поток слева направо и строит левый вывод грамматики. При компиляции файла синтаксический анализатор работает по следующему алгоритму:

1. Читается первый символ, без учета пробельных символов и комментариев. Затем проверяется совпадение этого символа с символом «$». При совпадении символов анализатор пытается прочитать и обработать директиву компилятора (переход к пункту 2) В противном случае считается, что анализируется объявление функции (то есть осуществляется переход к пункту 3).

2. Директива компилятора состоит из символа «$», имени директивы, необязательного аргумента (его наличие зависит от имени директивы) и признака окончания операции, то есть символа «;». Анализ директивы происходит по следующему алгоритму:

2.1. Первый символ должен быть «$», далее синтаксический анализатор читает и запоминает все символы до пробельного символа или символа «;». Так получается имя директивы, последний прочитанный символ возвращается обратно в поток ввода.

2.2. Имя директивы сверяется со списком поддерживаемых директив и проверяется, необходим ли для данной директивы аргумент. Если данное имя не является одним из поддерживаемых типов директив, то считается, что сейчас происходит анализ не директивы (и происходит переход к пункту 2.5). Если у данной директивы должен быть аргумент, то происходит переход к пункту 2.3, иначе — к пункту 2.4.

2.3. Анализатор читает следующий символ без учета пробельных символов и комментариев, если данная директива должна содержать аргумент, то этот символ должен быть «“», если это не так, то анализатор переходит к пункту 2.5, в противном случае анализатор читает и запоминает все символы до того, как встретит символ «“», который является признаком конца строки литералов. Так анализатор получает аргумент директивы.

2.4. Анализатор читает следующий символ без учета пробельных символов и комментариев и проверяет, является ли этот символ признаком конца операции — «;». Если не является, то происходит переход к пункту 2.5. Иначе данная проанализированная директива

сохраняется в структуре (для последующего сохранения в файл). Затем происходит переход к пункту 1.

2.5. Если анализ директивы не удался, то анализатор восстанавливает состояние (какое оно было до начала анализа директивы) и считает, что необходимо проанализировать функцию. Поэтому происходит переход к пункту 3.

3. Напомним, что функция состоит из имени функции, символа «{», то есть признака начала тела функции, некоторого количества предложений, оканчивающихся символом «;», признака окончания тела функции, то есть символа «}» (соответствующей вложенности) и символа «;» — признака окончания операции. Поэтому при анализе функции анализатор работает по следующему алгоритму:

3.1. Анализатор читает и сохраняет все символы до достижения пробельного символа или символа «{», последний прочитанный символ возвращается в очередь. Так формируется имя функции.

3.2. Анализатор читает следующий символ без учета пробельных символов и комментариев, этот символ должен быть «{». В противном случае осуществляется переход к пункту 3.6.

3.3. Анализатор читает следующий символ без учета пробельных символов и комментариев. Если этот символ «}», то описание функции закончилось, и происходит переход к пункту 3.5. В противном случае встречено начало следующего предложения, и происходит переход к пункту 4.

3.4. После успешного чтения предложения анализатор читает очередной символ без учета пробельных символов и комментариев. Если этот символ совпадает с «;», то происходит переход к пункту 3.2. В противном случае осуществляется переход к пункту 3.6.

3.5. Анализатор читает очередной символ без учета пробельных символов и комментариев, этот символ должен быть «;», в противном случае осуществляется переход к пункту 3.6. Если символ ожидаемый, то информация о проанализированной функции записывается в структуре для последующего хранения в файле. Затем происходит переход к пункту 1.

3.6. В случае ошибки анализатор сообщает об этом пользователю и прекращает дальнейший анализ файла.

4. Напомним, что предложение состоит из образца, возможно, одного или нескольких условий, знака «=» или аргумента подфункции, функциональных термов (возможно, заключенных в фигурные скобки, если присутствует несколько предложений или подфункция). Анализ предложения происходит следующим образом:

4.1. Анализатор читает и сохраняет все символы до достижения символа «,» или «=». Так анализатор формирует образец.

4.2. Если очередной символ «,», то происходит анализ условия или аргумента подфункции. Если далее встречен символ «<», то встречено начало условия, и выполняется переход к пункту 4.3, в противном случае анализируется аргумент: анализатор читает и сохраняет все символы до достижения символа «:», если следующий символ «{», то анализатор рекурсивно вызывает себя для анализа подфункции. В противном случае анализатор читает и сохраняет все символы до нахождения символа «,» или «=». Так анализатор получает всю необходимую информацию для сохранения в структуре условия для данного предложения. Далее происходит переход к пункту 4.2.

4.3. Условие состоит из символа «,», вызова функции: символа «<», имени и аргумента функции, символа «>». Далее идет символ «:», и происходит переход к пункту 4.4. Если следующий символ «=», то далее идет описание действий, происходит переход к пункту 4.5.

4.4. Анализатор читает очередной символ без учета пробельных символов и комментариев, этот символ должен быть «<», в противном случае осуществляется переход к пункту 4.6. Если это символ «<», то анализатор читает и сохраняет все символы до нахождения

символа «<>». Далее анализатор читает очередной символ без учета пробельных символов и комментариев. Этот символ должен быть «:», в противном случае осуществляется переход к пункту 4.6. После этого анализатор читает и сохраняет все символы до нахождения символа «,» или «=». Так анализатор получает всю необходимую информацию для сохранения в структуре условия для данного предложения. Далее происходит переход к пункту 4.2.

4.5. Анализатор читает и сохраняет все символы до достижения символа «;», последний символ возвращается в очередь, так анализатор получает список действий для данного предложения. После чего собранное и проанализированное предложение передается вызывающему.

4.6. В случае ошибки анализатор сообщает об этом пользователю и прекращает дальнейший анализ файла.

После того как анализатор проанализировал весь файл и не нашел ошибок, структуры преобразуются в выходной файл в формате языка сборки рефал-5е машины. Код компилятор языка рефал-5е в язык сборки рефал-5е машины содержит 2587 строк кода на языке С.

5. ОПИСАНИЕ РАБОТЫ РЕФАЛ-5Е МАШИНЫ

Рефал-5е машина, или же интерпретатор языка сборки рефал-5е, работает по следующему алгоритму:

1. Интерпретатор загружает необходимый файл в память, каждая функция также добавляется по своему имени в хэш-таблицу для быстрого и удобного поиска.

2. Для каждой директивы импорта модуля рефал-5е машина проверяет, загружен ли этот модуль. Если такой модуль еще не загружен, то он загружается (происходит переход к пункту 1 с указанием необходимого файла).

3. Если указано, что необходимо загрузить некоторую функцию из внешней библиотеки кода, то такая библиотека загружается, и производится поиск необходимой функции.

4. После того как все модули и библиотеки кода загружены, рефал-5е машина ищет функцию «Main» в главном файле программы, если она не найдена, то интерпретатор сообщает об ошибке и прекращает работу.

5. Если рефал-5е машина обнаруживает, что она запущена на многопроцессорном компьютере, то она создает дополнительные потоки выполнения, по одному на каждый процессор.

6. Рефал-5е машина переходит к выполнению функции «Main» с пустым аргументом.

7. При выполнении функции интерпретатор последовательно сопоставляет переданный аргумент с образцами предложений, если сопоставление прошло успешно, то происходит связывание переменных и проверяются все условия. Сопоставление с образцом и проверка условий происходит по следующему алгоритму:

7.1. Образец посимвольно сопоставляется с аргументом.

7.2. Если встретилась свободная, (то есть не связанная) *s*- или *t*-переменная, то она связывается с очередным символом или термом соответственно, при этом *s*-переменная вмещает в себя ровно один литерал, рефал-символ или макроцифру, а *t*-переменная – ровно один терм, то есть литерал, рефал-символ, макроцифру или структурные скобки с их содержимым. Если связывание не получилось, то идет возврат к последней *e*-переменной и пункту 7.4.

7.3. Если встретилась связанная переменная любого типа, то проверяется, что в аргументе на данном месте содержатся термы, соответствующие значению переменной.

7.4. Если встретилась свободная e-переменная, то она связывается с пустой строкой. Если рефал-5е машина по каким-либо причинам осуществила возврат к этой e-переменной, то число термов в ней увеличивается на единицу, если это возможно, и сопоставление с образцом продолжается, начиная с текущей позиции. Если число термов увеличить невоз-

можно, то рефал-5е машина переходит к предыдущей *e*-переменной. Если предыдущей *e*-переменной нет, то сопоставление с образцом считается неудачным.

7.5. Проверка условий производится последовательно и происходит по следующему алгоритму:

7.5.1. Если в качестве левой части условия стоит функция, то она выполняется с передачей ей необходимых аргументов. Если в качестве левой части условия стоит образец, то в нем раскрываются все переменные.

7.5.2. Результат вызова функции или образец сопоставляется с образцом в правой части условия по тому же алгоритму, что и сопоставление аргумента функции с образцом, то есть происходит переход к пункту 7.1.

7.5.3. Если сопоставление результата функции или образца с образцом из правой части условия оказалось неудачным, то происходит переход к предыдущему условию. Если предыдущего условия нет, то происходит переход к последней свободной e-переменной и переход к пункту 7.4.

8. Если все условия выполнены, то рефал-5е машина переходит к исполнению действий, связанных с вычислением функционального терма.

9. Если действие не является вызовом функции, то рассматриваемый рефал-символ, литерал, число или содержимое переменной добавляется в результату функции.

10. Если действие является вызовом функции, то вызов данной функции добавляется в очередь вызовов с указанием, куда необходимо записать ее результат.

11. Потоки выполнения функций проверяют очередь вызова функций, и если она не пуста, то первый свободный поток начинает выполнение этой функции (пункт 7).

12. После выполнения функции ее результат возвращается вызвавшей ее функции.

13. Если не удалось сопоставить аргумент функции ни с одним из образцов, то функция возвращает значение «Неуспех».

14. После того как функция «Main» выполнена, рефал-5е машина завершает свою работу.

Код рефал-5е машины содержит 4159 строк кода на языке С.

6. ПРИМЕРЫ ПРОГРАММ НА ЯЗЫКЕ РЕФАЛ-5Е

В заключение приведем несколько простых примеров программ на языке рефал-5е, показывающих его мощь и лаконичность.

Первый пример – программа для проверки строки на палиндром, то есть читается ли строка слева направо так же, как и справа налево:

```
$import SystemIO; /*Подключаем стандартный модуль ввода-вывода.*/
Main { /* Главная функция и точка входа в программу.*/
     = { <Palindrom <SystemIO::ReadLn> > <SystemIO::PrintLn 'Да'> ;
          /*Читаем строку с устройства стандартного ввода и передаем
ее в функцию Palindrom, если она завершилась успехом, то выводим «Да».*/
       <SystemIO::PrintLn 'Нет'> ;}
          /*В противном случае, если функция вернула «Неуспех», выводим
слово «Нет».*/
}

Palindrom { /*Функция, проверяющая, является ли переданная ей строка
палиндромом.*/
    t.1 e.2 t.1 = <Palindrom e.2>; /* Если первый и последний символы
одинаковы, то отбрасываем их и рекурсивно вызываем себя.*/
    t.1 = ; /* Если остался один символ, то строка — палиндром, возвращаем
пустое значение, что эквивалентно значению «Успех».*/
     = ; /* Пустая строка также является палиндромом.*/
```

```
/* В противном случае (если ни один из образцов не подошел) функция
вернет значение «Неуспех».*/
 }
```

Более сложный пример — программа, которая проверяет, является ли переданная ей строка правильной записью рефал-5е функции и возвращает ли эта функция ожидаемый результат:

```
$import SystemIO; /*Подключаем стандартный модуль ввода-вывода.*/
$import Cmp; /*Подключаем стандартный модуль сравнения значений.*/
$import Eval; /*Подключаем модуль доступа к встроенному интерпретатору.*/

Main { /*Главная функция и точка входа в программу.*/
     = <SystemIO::PrintLn <Check (<SystemIO::ReadLn>) (1 2) (3) > >;
/*Запрашиваем у пользователя строку и передаем ее в функцию Check, также
передаем аргумент для проверяемой функции и ожидаемый результат. По
завершении функции выводим на экран результат её выполнения.*/
};

Check { /*Функция принимает код функции, аргумент для этой функции и
ожидаемый результат и проверяет, возвращает ли функция ожидаемый результат
при данном аргументе.*/
 (e.Code) (e.Args) (e.Result) = {
    <Cmp::Eq (<Eval::Run (e.Code) e.Args >) (e.Result)> 'Ok'; /*Функция
Eval::Run передает встроенному интерпретатору код функции и аргумент для
нее. Если данный код удается успешно скомпилировать в функцию, то
возвращается результат выполнения функции с переданным аргументом. Если
компиляция не удалась, то возвращается значение «Неуспех». Далее
возвращенное значение сравнивается с ожидаемым результатом при помощи
функции Cmp:Eq. Если сравнение проходит успешно, то функция возвращает
строку «Ok».*/
    'Fail'; /* В противном случае возвращается строка «Fail».*/
     };
 };
```

В качестве примера программы, выполнение которой значительно ускоряется на многопроцессорной системе, можно привести следующую программу:

```
$import SystemIO; /*Подключаем стандартный модуль ввода-вывода.*/
$import Math; /*Подключаем стандартный модуль математических операций.*/
Main { /*Главная функция и точка входа в программу.*/
    = <SystemIO::PrintLn <Test ((1 2) (3 4))((5 6) (7 8))> <Test (1 2) (3
4)> >; /*Вызываем функцию Test.*/
    }

Test {
    (e.1) (e.2) = <Test <Test e.1> <Test e.2> >;
/*Рекурсивно вызываем себя.*/
    s.1 s.2 = <Math::Plus s.1 s.2>;
/* Возвращаем сумму элементов.*/
    s.1 = s.1;
}
```

На компьютере с процессором Intel(R) Core(TM) i5-2410M CPU с четырьмя логическими ядрами, 16 гигабайтами оперативной памяти и операционной системой Gentoo Linux с ядром версии 3.15.0 удалось добиться ускорения выполнения программы при помощи выполнения в несколько потоков в 3 раза по сравнению с однопоточным выполнением.

Литература

1. *Бабаев И.О., Герасимов М.А., Косовский Н.К., Соловьев И.П.* Интеллектуальное программирование. Турбо Пролог и Рефал-5 на персональных компьютерах. Л.: Издательство ЛГУ, 1992.
2. *Гошев В.А.* Схема универсальной функции для языка рефал-5 / Материалы всероссийской научной конференции по проблемам информатики. 23–26 апр. 2013 г. СПб.: Издательство ВВМ, 2013.
3. Г*ошев В.А., Косовский Н.К.* Разработка и реализация транслятора основных конструкций языка Рефал-5 в Рефал-Плюс / Сборник тезисов конференции «Технологии Microsoft в теории и практике программирования». СПб.: Изд-во Политехн. Ун-та, 2012.
4. *Гошев В.А., Косовский Н.К.* Методика трансляции Пролога в рефал+ / https://sunx.me/uploads/Prolog-To-Refal-plus.pdf (дата обращения: 24.06.2014).
5. *Гошев В.А., Косовский Н.К.* Проект языка программирования рефал-5е с удобными расширениями препроцессором // Компьютерные инструменты в образовании, 2014. № 1. С. 3–13.
6. *Гурин Р.Ф, Романенко С.А.* Язык программирования Рефал Плюс. М.: ИНТЕРЕХ, 1991.
7. *Климов А.В, Романенко С.А.* Система программирования Рефал-2 для ЕС ЭВМ. Описание библиотеки функций. М.: ИПМ им. М.В. Келдыша ФН СССР, 1986, препринт N 200.
8. *Климов А.В.* Программирование на языке Рефал. 2004 / http://refal.net/~arklimov/refal6/manual.htm (дата побращения: 24.06.2014).
9. Сайт, посвященный развитию языка Рефал / http://wiki.botik.ru/Refaldevel/WebHome (дата обращения: 24.06.2014).
10. *Скоробогатов С.Ю. , Чеповский А.М.* Язык Refal с функциями высшего порядка // Информационные технологии, 2006. № 9.
11. *Turchin V.F.* Refal-5. Programming Guide and Reference Manual. New England Publishing Co., Holyoke, 1989.

IMPLEMENTATION OF REFAL-5E PROGRAMMING LANGUAGE TRANSLATOR WITH EMBEDDED INTERPRETER AND SUPPORT OF EXTERNAL CODE LIBRARIES

Goshev V. A.

Abstract

Refal-5e programming language extends the existing refal programming language dialect called refal-5 and allows to solve important modern programming tasks more efficiently than other refal dialects. The most interesting features of the refal-5e programming language are: a built-in interpreter; the ability to use external dynamic link libraries of code; the ability of functions to return values such as «success» and «failure»; the possibility of implementing high-order functions and anonymous functions. Also refal-5e programming language translator uses possibilities of modern multiprocessor computers that can speed up program execution several times.

Keywords: *refal-5, refal-5e, compilation, translation, interpretation, backtracking.*

Гошев Владимир Андреевич,
аспирант кафедры информатики
математико-механического
факультета СПбГУ,
sunx@sunx.cc