Российская академия наук
Национальный комитет по автоматическому управлению
Научный совет по теории управляемых процессов и автоматизации
ОЭММПУ РАН
Институт проблем управления им. В.А. Трапезникова РАН
Министерство образования и науки Удмуртской Республики
Удмуртский государственный университет
Удмуртский НОЦ ПУ (на базе УдГУ)
Волгоградский НОЦ ПУ (на базе ВолГУ)
Воронежский НОЦ ПУ (на базе ВГАСУ)
Инновационный НОЦ ПУ (на базе МАИ)
Казанский НОЦ ПУ (на базе КГТУ-КАИ)
Курский НОЦ ПУ (на базе КГТУ)
Липецкий НОЦ ПУ (на базе ЛГТУ)
НОЦ «Системный анализ в управлении» (на базе МИФИ)
Пермский НОЦ ПУ (на базе ПГТУ)
Самарский НОЦ ПУ (на базе СГАУ)
Тверской НОЦ ПУ (на базе ТГТУ)
Старооскольский НОЦ ПУ (на базе СТИ)
Удмуртский Совет ИТ-директоров
Институт логики, когнитологии и развития личности РАН

VI Всероссийская школа-семинар молодых ученых

Управление большими системами

посвящается памяти А.А. Маркова

Том 1

31 августа – 5 сентября 2009
г. Ижевск

УДК 007

VI Всероссийская школа-семинар молодых ученых «Управление большими системами»: Сборник трудов. – Т1.- Ижевск: ООО Информационно-издательский центр «Бон Анца», 2009. – 400 с.

В первый том сборника трудов включены научные статьи молодых ученых по фундаментальным основам теории управления, вопросам управления организационными и социально-экономическими системами, управлению качеством.

ISBN 978-5-903140-57-2

Научное издание осуществлено при поддержке РФФИ
грант № 09-07-06039г

УДК 519.682.3

Некоторое расширение языка Рефал

Исламов М.Ш., islamov.marat@gmail.com
Удмуртский государственный университет, г. Ижевск

Аннотация

В данной статье описывается результат работы в области совершенствования сентенциального языка программирования Рефал [1], с целью его адаптации к актуальным сегодня задачам. Предложено некоторое расширение семантики языка концептуально непротиворечивыми, эффективными конструкциями.

Ключевые слова: Рефал, сентенциальный язык, декларативное программирование, синтаксис, семантика, образец, шаблон, динамические программы, метавычисления, суперкомпиляция

Abstract

This article describes the results of the work on improving the sentential program language Refal [1], with the purpose of its adaptation to the problems actual today. Some extensions of the language semantics by conceptually consistent, effective designs are proposed.

Введение

Рефал – сентенциальный язык программирования [2], созданный в СССР профессором В.Ф. Турчиным в 60-х годах прошлого века, обладающий высокой степенью декларативности и достаточно простой и красивой логикой вычислений. Декларативные языки ускоряют процесс разработки программ, сокращают объем исходного кода, позволяют решать задачи более высокого уровня сложности за счет того, что программисту не приходится искать алгоритм решения самостоятельно – от него требуется лишь описание предметной области и постановка задачи, а поиск решения возлагается на вычислительную машину.

Но, как и у любого «бесплатного сыра», у декларативного программирования есть своя мышеловка, и даже не одна. Развивавшееся еще с 60-х годов, несмотря на большой вклад ученых и специалистов со всего мира, оно не вышло за рамки очень узкоспециализированных задач. Связано это прежде всего с тем, что чем сильнее модель вычисления языка отличается от модели вычислительной машины, тем менее эффективно выполняется программа. Рост аппаратных возможностей вычислительных машин нивелируется ростом уровня сложности решаемых задач и объемом входной информации, поэтому высокая эффективность реализации всегда будет одним из основных требований к языку. Решением проблемы может стать приближение декларативного языка программирования к архитектуре вычислительной машины. Такого приближения можно добиться путем

ограничения синтаксиса и семантики, а также использованием оптимизаторов и метавычислителей, рассмотрение которых выходит за рамки данной статьи. Среди распространенных реализаций диалектов сентенциальных и функциональных языков фактическими стандартами стали именно те, которые отличаются компактной, концептуально точной и однозначной семантикой (Common Lisp, Scheme, Refal-5, Gnu-Prolog, Swi-Prolog, Glasgow Haskell и некоторые другие), что позволяет быстро писать достаточно эффективные программы с помощью удобных и выразительных языковых средств. Диалекты, которые содержат лишние правила и не дают ощутимых преимуществ в выполнении программы или в ее выразительности, не пользуются значительной популярностью (Турбо-Пролог, некоторые версии Рефала). В качестве доказательства вышесказанного стоит обратить внимание на Лисп-машину – именно простота и компактность языка позволила реализовать его аппаратно, несмотря на то, что он относится к языкам высокого уровня.

Руководствуясь сказанным выше, при проектировании и разработке очередного диалекта языка Рефал принцип концептуальной непротиворечивости автор статьи поставил на первое место.

Диалекты языка Рефал

В основе языка лежит идея алгорифмов Маркова [3], широко применяемая в инструментах прототипирования. В первоначальном виде Рефал являлся метаязыком, предназначенным для описания других алгоритмических языков [4, 5], но в результате появления достаточно эффективных реализаций для ЭВМ он стал находить практическое использование в качестве языка символьных преобразований. На сегодняшний день существует несколько диалектов языка [1, 6–10], которые достаточно сильно отличаются друг от друга уже на уровне семантики, но содержат одно общее подмножество – Базисный Рефал [1]. Фактическим стандартом среди них признан Рефал-5 [8], разработанный самим Турчиным в 1989 году.

Добавленные в Рефал-5 расширения Базисного Рефала были представлены автором языка как инструменты, сокращающие классическую запись программ, однако фактически они явились направляющими дальнейшего развития языка. Вспомним, что Рефал-5 отличается от базисного наличием with-контрукций (блоков), where-конструкций (условий) и стека глобальных значений. Следующие версии языка (Рефал-6, Рефал Плюс) явились углубленным развитием этих конструкций, в результате чего в диалектах появились откаты блоков и функций, предложения были заменены тропами с заборами и отсечениями, было введено некоторое по-

добие объектов ООП. Последнее, скорее всего, явилось данью моде (примеры значимого применения автору статьи не известны), остальное – это инструменты управления сопоставлением. Таким образом, вместо декларативного описания постановки подзадач пользователю предлагается программировать процесс сопоставления, что с одной стороны повышает гибкость языка, с другой – приводит к отклонению от заложенного Турчиным основополагающего принципа: язык должен быть ориентирован в первую очередь на модель человеческого мышления, а не на модель вычислительной машины [4, 5].

Поэтому в работе по дальнейшему развитию языка в сентенциальном направлении с сохранением базовых концепций автор статьи отталкивался от диалекта Рефал-5.

D-Refal. Базис

Как и в языке-предшественнике, для представления любых данных в D-Refal'е используются объектные выражения, а основными операциями являются сопоставление объектного выражения с образцом и подстановка. От стандартного Рефала новый диалект унаследовал способ определения функции в виде списка предложений с левой и правой частью. Рефал-условия также вошли в базис нового диалекта, в отличие от рефал-блоков, которые нарушают структуру рефал-предложения и могут быть заменены другими конструкциями языка.

Программы на языке D-Refal выполняет специальная рефал-машина, принципы работы которой подробно описаны в [8].

Типы данных

Стандартными символами языка являются: текстовые символы (characters), текстовые термы (compound symbols), целые числа (integer numbers), вещественные числа (real numbers), байты (bytes). Вместе со структурными скобками (structure brackets) они служат для образования объектных выражений языка. Для работы с текстовыми символами используется ставший уже стандартом формат Юникод. Поскольку юникод-символы не тождественны байтам, последние были выделены в отдельный тип данных языка. Текстовые термы, как и в языке-предшественнике, определяются с помощью двойных кавычек.

Переменные

Рефал-5 имеет три типа переменных: s, t и e. В описываемом диалекте присутствует еще один тип переменных – это тип «жадная e-переменная»

(обозначается E). При инициализации E-переменная захватывает самое длинное допустимое подвыражение и, в случае отката, укорачивает область захвата на один терм влево. Это дает программисту очень гибкий инструмент управления шаблонами и полностью заменяет механизм сопоставления справа налево, ведь он аналогичен замене всех e-переменных левой части на переменные типа E.

Кроме стандартных типов переменных, в диалекте имеются еще несколько дополнительных типов: integer (для целых чисел), real (для вещественных чисел), number (для любых чисел), letter (для текстовых символов), byte (для байтов).

В любом диалекте языка Рефал переменная может быть открытой (свободной) или закрытой (ссылающейся на открытую). В языке D-Refal для определения закрытых переменных может быть использовано специальное обозначение – символ «@» вместо имени типа. Данное обозначение применимо к любым закрытым переменным, но оно является необходимым для случаев, когда соответствующая открытая переменная не имеет имени типа (ниже будут подробнее рассмотрены такие случаи).

Чтобы избежать загромождения текста программ лишней информацией, переменные, которые используются в предложении только один раз, могут быть описаны без имени (но точка в конце обязательна). Такие переменные далее называются безымянными. Любая безымянная переменная является открытой, то есть два идентичных объявления безымянной переменной являются объявлениями различных переменных.

Пример 1. Рефал-предложение с безымянными переменными и @-переменной.

e. *'REFAL-'* s.version e. = @.version.

Спецификаторы

Самым концептуально чистым диалектом является, пожалуй, Базисный Рефал. Созданный В.Ф. Турчиным диалект воплотил в себе все принципы и идеи описания алгоритмов способом, удобным для человека, а не для машины, но практическое применение языка показало его недостаточность. Решение какой-нибудь несложной подзадачи требовало использования вспомогательных функций, что разрушало целостность описания алгоритма, затрудняло чтение кода программы и вынуждало программиста совершать лишние действия. Поэтому появление различных расширений Базисного Рефала было закономерным явлением. Рефал-условия и рефал-блоки, бесспорно, сделали Рефал-5 на порядок практичнее, однако при решении большинства стандартных задач, связанных с обработкой текста, программисты предпочитают использовать языки с более

удобными регулярными выражениями, выделяя отсутствие в Рефале подобных спецификаторов как недостаток языка.

Для устранения этого недостатка в D-Refal были добавлены три конструкции, расширяющие описание образцов: группы, альтернативы и повторители.

Группой называется описание переменной, тип которой определяется некоторым образцом. В конкретном синтаксисе группы обозначаются фигурными скобками, внутри которых задается образец. Этот образец может содержать как закрытые переменные (определенные левее группы), так и открытые, однако все эти открытые переменные не видны в левой части рефал-предложения за пределами группы, и для исключения путаницы их использование там запрещено.

Пример 2. Образец с группой.

e. { *'<Tag>'* e. *'</Tag>'* }.tag e.

После определения группы в левой части рефал-предложения могут использоваться закрытые переменные, ссылающиеся на значение переменной этой группы. Определяются они с помощью описанного выше символа «@».

Для большей выразительности программ на языке D-Refal точка у безымянных групповых переменных не указывается.

Альтернативой языка D-Refal называется группа, имеющая несколько определяющих образцов (вариантов). В конкретном синтаксисе эти образцы разделяются символом «|» и имеют приоритет согласно порядку перечисления.

Пример 3. Образец с альтернативой.

{ '/*' e. '*/' | '//' e. '\n' | '\n*' e. '\n' }.comment E.other

После описания типа переменной (например с помощью имени типа, группы или альтернативы) может стоять пара квадратных скобок с двумя целыми числами или закрытыми integer-переменными. Такая конструкция называется **повторителем типа** переменной D-Refal'а и является аналогом квантификатора повторения в регулярных выражениях языка Perl. В общем случае описание переменной с повторителем имеет следующий вид:

<TypeDefinition> [*<from>* .. *<to>*] . *<varName>*

Тут *<TypeDefinition>* – это описание типа; *<from>* и *<to>* могут быть числами или переменными типа integer, *<varName>* – имя переменной. Значение *<from>* должно быть не больше значения *<to>*, и оба они должны быть неотрицательными – иначе выражение считается некорректным и приводит к ошибке. Данная конструкция определяет переменную, эквивалентную цепочке безымянных переменных типа *<TypeDefinition>* длиной от *<from>* до *<to>* элементов, заключенной в пару групповых скобок.

Стоит отдельно разобрать случай, когда *<TypeDefinition>* является альтернативой. В альтернативе с повторителем приоритет в первую очередь отдается выбору варианта, и только потом определению количества элементов. Так, например, для переменной {'X'|'Y'|'Z'}[2..4].xyz список возможных значений, упорядоченных по приоритету, выглядит следующим образом:

'XX' 'XXX' 'XXXX' 'XXXY' 'XXXZ' 'XXY' 'XXYX' 'XXYY' ... 'XXZZ' 'XY' 'XYX' 'XYXX' 'XYXY' ... 'XZZZ' ... 'YX' 'YXX' 'YXXX' ... 'ZZZZ'

Это верно и для случаев, когда *<TypeDefinition>* содержит одну или несколько альтернатив.

Для определения повторителей, не ограниченных сверху, используется запись вида:

<TypeDefinition> **|** *<from>* **... |** . *<varName>*

Для случая, когда *<from>* всегда совпадает с *<to>*, в языке предусмотрено сокращение:

<TypeDefinition> **|** *<from>* **|** . *<varName>*

Как и в группах, безымянные переменные с повторителем определяются без точки в конце.

Пример 4. Образец с повторителем.
{ 'a' | 'b' | 'c' | 'd' }**[1..256]**.x

Пример 5. Безымянный образец с повторителем.
{ 'a' | 'b' | 'c' | 'd' }**[1..256]**

Рефал-условия

Рефал-условия впервые появились в Рефале-4 и доказали свою состоятельность во всех последующих диалектах. В D-Refal'е они имеют тот же синтаксис и семантику, что и в Рефале-5:

, *<результатное выражение>* **:** *<образец>*

Одним из самых примечательных расширений языка является отрицательное условие. Отрицательные условия определяются с помощью модификатора **$NOT**, который может стоять перед результатным выражением условия или перед образцом условия (обе формы записи эквивалентны). Если левая часть имеет отрицательные условия, то она сопоставима с объектным выражением только в тех случаях, когда все эти условия, взятые без модификатора $NOT, не выполняются.

Пример 6. Левая часть для выражения, не содержащего цифр.

e.nodig ,
$NOT e.nodig : e. *{'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'|'0'}* e.

Пользовательские шаблоны

Уже к появлению первой компьютерной реализации Рефала была выделена проблема гибкости описания образцов, для решения которой в диалекте Рефал-2 появились спецификации переменной и возможность определять их за пределами места применения. Однако эти спецификации позволяли программисту лишь накладывать ограничения на область допустимых значений переменной.

Более гибким и выразительным является механизм построения пользовательских структур шаблонов из более простых структур. Общий вид пользовательской структуры шаблона в диалекте D-Refal выглядит следующим образом:

Template *<имя>* **::=** *<описание>* ; (*)
<описание> ::= *<образец>* { ,*<условие>* }[0...]

где *<имя>* – это идентификатор (за исключением имен встроенных типов переменных), а *<описание>* рефал-образец, возможно, с условиями.

Объявление в программе такой структуры является объявлением нового типа переменной. С такой переменной может быть сопоставлено лишь то объектное выражение, которое соответствует образцу и условиям в описании структуры (*). Переменная, описание которой задается именем пользовательской структуры шаблона, далее называется *переменной пользовательского типа*, а сама структура – *пользовательским шаблоном*. Стоит отметить, что при описании пользовательского шаблона в качестве типов переменных могут быть использованы любые другие пользовательские шаблоны, включая описываемый шаблон. Это добавляет в алгоритмический язык рекурсивных функций еще один вид рекурсии – рекурсию описания пользовательских шаблонов.

Пример 7. Шаблон для правильного списка скобок.

```
Template GoodBrackets ::= { $empty | '(' GoodBrackets. ')' GoodBrackets. }
```

В примере 7 определяется пользовательский шаблон для выражения, состоящего из литер-скобок, расставленных в правильном порядке. В программе с таким определением шаблона становится возможным использование переменных типа GoodBrackets:

```
Go {
  $empty,
    <Card> : GoodBrackets.input = <Prout 'Ok'>;
  $empty = <Prout 'Not right'>;
}
```

Для удобной работы с переменными пользовательского типа в D-Refal'е реализована операция разыменования (::), позволяющая получить доступ к переменной внутри описания шаблона.

<закрытая_переменная>::<имя_вложенной_переменной>

Синтаксис операции разыменования.

Такая ссылка является закрытой переменной, а значит, и к ней применима операция разыменования, что дает возможность строить цепочки разыменований для доступа к вложенным переменным.

Пример 8. Шаблон всех внутренних тегов, не содержащих вложенных тегов, и рефал-предложение с разыменованием.

```
Template Tags ::= '<' id.name  attribs.attributeList '>' e.body '</' id.name '>',
                  $NOT e.body : e. Tags.ins e. ;

...
e. Tags.xyz e. =
  <Prout 'Tag [' Tags.xyz::name ']'s first attribute is '
    Tags.xyz::attributeList::first::name      >;
...
```

Стоит отметить, что кроме возможности писать выразительные программы, пользовательские шаблоны также дают возможность создавать неэффективные алгоритмы (как, впрочем, и любой язык программирования). Неправильно спроектированный пользовательский шаблон может привести к зацикливанию сопоставления пользовательской переменной, например, с пустым выражением, а неопределенность границ сопоставляемого с переменной объектного выражения повышает вероятность возникновения проблем с e- и E-переменными. Для того чтобы исключить большинство таких проблем, на определение пользовательских шаблонов были наложены некоторые ограничения. Первым ограничением является запрет на использование E-переменных в образце пользовательского шаблона (но не в его условиях). Поскольку такие переменные в начале сопоставления захватывают наибольшее выражение, их жадность может распространяться на все активное поле зрения, размеры которого могут быть известны только во время выполнения программы, а не во время написания этого шаблона. Когда же дело касается условий в описании шаблона, то в этом случае границы сопоставляемого объектного выражения определяются конкретным результатным выражением и полностью контролируются программистом, что позволяет безопасно использовать E-переменные в условиях. Другим ограничением является запрет на использование e-переменных по краям образца описания пользовательского шаблона, поскольку в противном случае область допустимых значений пользовательской переменной

значительно увеличивается, как и время сопоставления. Конечно, программист может обойти последнее ограничение, маскируя произвольные выражения спецификаторами, однако в этом случае подразумевается, что он знает, что делает.

D-Refal. Расширение

Базис языка D-Refal содержит в себе языковые конструкции, которые в полной мере позволяют создавать выразительные декларативные программы. Однако, как было сказано в начале статьи, за выразительность декларативного языка приходится расплачиваться эффективностью программы, и D-Refal тут не является исключением. Насколько бы ни была понятна и проста программа, если она выполняет относительно простые действия неоправданно долго, ее применение на практике будет нецелесообразным. В данной части статьи дается описание дополнительных конструкций диалекта, с помощью которых программист может повысить эффективность своих программ без ущерба для выразительности.

Дополнительные типы переменных

Для того чтобы выяснить, к какому типу данных относится некоторый терм объектного выражения, в предшествующем диалекте – Рефале-5 – имеется функция Type, которая возвращает определенные текстовые символы для каждого типа данных. Как правило, программисту приходится писать для ее использования вспомогательную функцию либо использовать рефал-блоки.

В диалекте D-Refal данная задача решается более просто. В языке определено несколько дополнительных встроенных типов переменных для каждого вида рефал-символов:

integer. – переменная для целого числа;
real. – переменная для вещественного числа;
number. – переменная для любого числа;
symbol. – переменная для текстового символа;
alpha. ; – переменная для латинской буквы;
digit. – переменная для текстового символа-цифры;
byte. – переменная для байта.

Использование встроенных типов переменных значительно повышает эффективность программ, так как исключает дополнительные вызовы функций. Типы переменных для более специфических видов данных программист может определить с помощью пользовательских шаблонов.

Расширенные варианты

Несмотря на озвученный выше отказ от рефал-блоков, в диалекте D-Refal имеется похожий, более ограниченный механизм – расширенные варианты. Если альтернатива является описанием типа переменной, то любому варианту (или вариантам) этой альтернативы может быть приписана подстановка – некоторое объектное выражение. Если такой вариант окажется успешным при сопоставлении переменной-альтернативы, то в каждом результатном выражении с ссылкой на эту переменную ее значение будет заменено соответствующей подстановкой.

В конкретном синтаксисе вариант и его подстановка разделяются символом «=».

Пример 9. Использование расширенного варианта.

```
UnEscape {
    '\'{'\\' | 'n'='\n' | 't'='\t' | '"' | '\''}.sym E.next = @.sym <UnEscape E.next>;
    '\'s.sym E.next = <MakeError>;
    s.s E.next = s.s <UnEscape E.next>;
}
```

Расширенные альтернативы не нарушают структуру предложения, одновременно с этим они играют такую же полезную роль, как и рефал-блоки в Рефале-5. Стоит отметить, что в роли подстановок могли бы выступать не объектные, а результатные выражения (выражения с вызовами функций), что сделало бы язык более мощным, однако автор отказался от данной идеи в пользу выразительности программ.

Ссылки на объектные выражения

Достаточно часто при написании рефал-программ возникает ситуация, когда после распознавания некоторой достаточно сложной структуры в левой части рефал-предложения, в правой части происходит вызов функции, которой в качестве аргумента передается часть распознанной структуры или вся структура целиком. Поскольку для работы с этим аргументом необходимо его повторное сопоставление с похожим (а иногда таким же образцом), многие операции повторяются вновь. В случае рекурсивных вычислений количество ненужных операций возрастает. Для устранения такой избыточности в языке D-Refal предусмотрены ссылки на объектные выражения. Ссылки – это рефал-символы, которые содержат информацию о структуре объектного выражения и предоставляют быстрый и эффективный доступ к его содержимому. Структура такого объектного выражения определяется пользовательским шаблоном, а ссылка – переменной пользовательского типа, оснащенной специальной меткой (здесь и далее такие

переменные называются ссылочными). В конкретном синтаксисе этой меткой является символ «&» перед именем типа пользовательской переменной.

Пример 10. Рефал-предложение с определением ссылки (ссылочной переменной) на объектное выражение в правой части.

```
e. MyTemplate.x e. = <X &MyTemplate.x >;
```

Типом ссылки и типом ссылочной переменной будем называть шаблон, определяющий эту ссылку и ссылочную переменную.

Ссылочные переменные могут быть как открытыми, так и закрытыми. Открытые ссылочные переменные успешно сопоставляются только с ссылками того же типа. Закрытые — определяют *идентичную ссылку*. Две ссылки являются *идентичными*, если они указывают на одно и то же объектное выражение.

На примере 10 продемонстрировано рефал-предложение, в левой части которого присутствует переменная типа MyTemplate, определенного пользователем, а в правой части – определение ссылки на значение этой переменной. Когда управление программой будет передано функции X, ее аргументом будет всего один рефал-символ – ссылка типа MyTemplate. Для доступа к значениям переменных шаблона MyTemplate в этой ссылке, программист может использовать операцию разыменования («::»). Принцип ее действия для ссылочных переменных аналогичен принципу действия для переменных пользовательского типа. Это становится возможным благодаря тому, что в рефал-ссылке сохраняется не только ссылка на объектное выражение, но и ссылка на результат его сопоставления.

Пример 11. Использование операции разыменования для ссылочной переменной.

```
X {
    &MyTemplate.x e. = &MyTemplate.x::subvar::subsubvar ;
}
```

Практическое применение ссылок не вызывает трудностей, однако суть использования одноименных пользовательских и ссылочных переменных в образцах не очевидна, поэтому ниже приводятся примеры образцов с пояснениями.

Образец	Значение
&Map.1 (Map.1)	ссылка на объектное выражение типа Map и такое же выражение в скобках
Map.1 &Map.1	выражение и ссылка типа Map на такое же выражение
&Map.1 &Map.1	две ссылки типа Map на идентичные объектные выражения

Как и другие рефал-символы, ссылки могут быть сопоставлены с s-переменными.

Отсечение

Пользовательские шаблоны и спецификаторы предоставляют удобный и достаточно эффективный способ распознавания ожидаемой структуры выражения. Однако для нетривиальных программ большая часть времени выполнения приходится на поиск *подходящего* образца среди *неподходящих*. Пользовательские шаблоны и спецификаторы могут увеличить время сопоставления выражения с неподходящим образцом, поэтому конструкции, позволяющие ускорить этот процесс, вполне востребованы в языке. Одной из таких конструкций является *отсечение*.

Отсечением называется точка в образце (обозначается «!»), которая объявляет всю левую от нее часть образца единожды сопоставимой. Технически это означает, что при возникновении отката в этой точке весь текущий образец объявляется неприменимым к сопоставляемому объектному выражению.

Пример 12. Использование отсечения.

```
Template theTAG  ::=
                    '<TAG name="' e.tagname '">' ! e.value '</TAG>',
                    < CorrectTags e.value> : true ;
```

В примере 12 рефал-машина не будет пытаться наращивать значение переменной e.tagname после того, как активной в сопоставлении станет переменная e.value. Если же попытка рефал-машины сопоставить переменную e.value с аргументом будет неудачна, то результат сопоставления всей пользовательской переменной типа theTAG будет признан неуспешным. Таким образом, значение параметра *name* тега *TAG* может быть только выражением внутри кавычек (''), не содержащим других кавычек ('').

Динамическое доопределение программ

В самом первом диалекте языка Рефал, который тогда еще назывался Метаалгоритмическим языком, была определена возможность занесения из поля зрения в поле памяти любого набора рефал-предложений, что фактически равносильно самомодификации программы. Компьютерные реализации языка такой возможности не имели, однако в связи с активным применением языка в метавычислениях и задачах искусственного интеллекта, динамическое изменение рефал-программ становится все более необходимым.

В диалекте D-Refal имеется возможность переопределять функции и пользовательские шаблоны программы во время ее выполнения. Это достигается с помощью встроенной функции <Refal e.>, которой в качестве аргумента передается текст определения функции или шаблона. Поскольку

этот текст может генерироваться во время выполнения программы – у программиста появляется возможность написания полиморфного рефал-кода.

Другой важной функцией диалекта для динамической модификации программы является встроенная функция <MetaQuery t.name>, которая возвращает текст пользовательской функции или пользовательского шаблона по ее имени. В случае, когда функция MetaQuery вызывается с пустым аргументом, результатом ее работы будет текст всей программы. Если запрашиваемая функция пользователем не определена (встроенная или неизвестная), результатом будет пустое выражение.

Заключение

D-Refal – это предложенное автором статьи решение проблемы отсутствия современной и концептуально непротиворечивой реализации диалекта Рефала, необходимость в которой назрела уже давно. Задачи обработки текстовых данных на уровне глубокого анализа их структур если и решаются, то средствами, предназначенными для области численных решений. Описанный в статье язык программирования позволяет работать со сложными структурами более естественным способом – с помощью конкретизации с гибкой системой шаблонов.

Одним из приоритетных направлений, на которое в первую очередь обращал внимание автор при проектировании и реализации диалекта, являются метавычисления. Рефал изначально ориентирован на работу с алгоритмическими языками, однако его практическое применение выявило ряд слабых мест, включая слабый способ определения образцов. Спецификаторы и пользовательские шаблоны расширяют Рефал выразительными инструментами, сохраняя основные концепции языка, а функции динамического доопределения программ дают пользователю возможность создавать более гибкие алгоритмы. Внешние шаблоны с условиями, очевидно, упрощают описание сложных инвариантов в метапрограммировании, что является крайне востребованным для суперкомпиляции. О результатах применения диалекта D-Refal в этой области будет сообщено в следующих работах.

Список литературы

1. *Турчин В.Ф.* Базисный РЕФАЛ. Описание языка и основные приемы программирования (методические рекомендации) // Фонд алгоритмов и программ в отрасли «Строительство». Т. 5. № 33. ЦНИПИАСС. Москва, 1974.

2. *Непейвода Н.Н., Скопин И.Н.* Основания программирования. – Москва Ижевск: Институт компьютерных исследований, 2003. 868 с.

3. *Марков А.А.* Теория алгорифмов // Труды Математического института им. В.А. Стеклова. Изд-во АН СССР, 1954.

4. *Турчин В.Ф.* Метаалгоритмический язык // Кибернетика. 1968. № 4. С. 45 54.

5. *Турчин В.Ф.* Метаязык для формального описания алгоритмических языков // Цифровая вычислительная техника и программирование. М.: Сов. радио, 1966. С. 116 124.

6. *Климов Ан.В., Романенко С.А.* Система программирования Рефал-2 для ЕС ЭВМ. Описание входного языка. М.: ИШ им. М.В. Келдыша АН СССР, 1987.

7. *Романенко С.А.* Рефал-4 – расширение Рефала-2, обеспечивающее выразимость результатов прогонки. – М.: ИПМ им. М.В. Келдыша АН СССР, 1987, препринт № 147. 27 с.

8. *Turchin V.* REFAL-5, Programming Guide and Reference Manual. New England Publishing Co. Holyoke, 1989.

9. *Klimov Ark.V.* Refal-6, 2003, (*http://www.refal.org/~arklimov/refal6*).

10. *Гурин Р.Ф., Романенко С.А.* Язык программирования Рефал Плюс. – М.: Интертех, 1991. С. 183.

11. *Турчин В.Ф.* Эквивалентные преобразования программ на Рефале // Автоматизированная система управления строительством. М.: ЦНИПИАСС, 1974. С. 36 68.

12. *Климов Ан.В., Романенко С.А., Турчин В.Ф.* Компилятор с языка Рефал. – М.: ИПМ АН СССР, 1972. – 74 с.

13. *Робин Хантер.* Основные концепции компиляторов. Пер. с англ. – М.: Издательский дом «Вильямс», 2002. – 256 с. : ил. – парал. Тит. Англ.