M. M. Mikhnev, K. N. Polyaev

DEVELOPMENT OF FILE SERVERS FOR AUTOMATION SYSTEMS OF TECHNOLOGICAL PREPARATION OF PRODUCTION

The authors consider problems of development of file servers for automation systems of technological preproduction, for example, the technical solutions implemented in the file server of information retrieval system of technological support.

Keywords: technological preparation, information system, a file server.

УДК 681.142.2

П. Б. Могнонов, Д. Н. Чимитов, Н. В. Ярышкина

РЕАЛИЗАЦИЯ ИНТЕРПРЕТАТОРА С ЗАДЕРЖАННЫМ ВЫЧИСЛЕНИЕМ

Рассматривается один из способов оптимизации работы алгоритмов посредством интерпретатора с задержанным вычислением. Делается попытка реализовать алгоритм работы такого интерпретатора на базе расширенного функционального языка ЛИСП. Показывается значение задержанных вычислений для адаптации процедуры.

Ключевые слова: задержанные вычисления, интерпретатор, ламбда-исчисление, ЛИСП, полиморфизм.

Под интерпретатором с задержанным вычислением понимается такой интерпретатор, который вычисляет значения аргументов по необходимости, т. е. в тот момент работы программы, когда действительно необходимо значение вычисляемого выражения. Таким образом, в ходе своей работы при вызове какой-либо процедуры интерпретатор с задержанным вычислением определяет, какие аргументы требуется вычислить, а какие задержать. Задержанные аргументы при этом не вычисляются, а преобразуются в объекты, называемые рецептами [1].

В данной статье сделана попытка показать, как можно использовать интерпретатор с задержанным вычислением U для автоматического решения следующей задачи: пусть в банке процедур B имеется относительно обобщённая (или полиморфная) процедура $P(x_1, \ldots, x_n)$. Тогда интерпретатор U, вызывая $P(x_1, \ldots, x_n)$ и имея контекстные условия C на текущий момент t процесса вычисления, может задержать немедленное исполнение процедуры $P(x_1, \ldots, x_n)$ на время, необходимое для конкретизации процедуры $P(x_1, \ldots, x_n)$, или, другими словами, для адаптации процедуры $P(x_1, ,\ldots, x_n)$ к текущему контексту C вычислительного процесса. Во время адаптации (т. е. во время задержки и интерпретации) процедуры P интерпретатор U определяет фактические значения некоторых параметров $\{x_{i1}, x_{i2}, \ldots, x_{im}\}$, где $(m \le n)$ исходя из контекста C. Процедура P с вычисленными фактическими параметрами определяет производную процедуру P_1. Можно сказать, что процедура P является родителем процедуры P_1, а P_1 – потомком P. Отметим, что в банке процедур B все процедуры частично упорядочены отношением полиморфизма и образуют полную решетку (или декартово-замкнутую категорию Скотта [2; 3]), что отражает структуру знаний, записанную в банке процедур B.

Разработка структуры знаний банка процедур B в данной работе не затрагивается и предполагается, что эта задача инженерии знаний конкретной предметной области. Таким образом, имеем следующую схему работы интерпретатора U: ***исходные данные:*** [1] $P(x_1, \ldots, x_n)$; 2) контекст динамического процесса вычисления C; 3) задержка, необходимая интерпретатору U: $C \rightarrow \{x_{i1}, \ldots, x_{im}\}$ для определения фактических параметров процедуры P] $\rightarrow$ ***результат:*** конкретизированная (или адаптированная) процедура P_1.

Рассмотрим варианты реализации алгоритма работы интерпретатора с задержанным вычислением, разработанного на базе расширенного функционального языка ЛИСП.

Реализация интерпретатора с задержанным вычислением для количественной оптимизации процесса вычислений. В качестве примера рассмотрим программу, выполняющую функцию формирования суммы и произведения бесконечного списка чисел:

```
сумпроизвспис(n) ≡ сумпроизв(список(1, n))
сумпроизв(x) ≡ {сп (x, 0, 1)
                гдерек сп = λ(x, s, p)
                если равно(x, НИЛ) то двучлен (s, p)
                иначе сп (cdr(x), s+car(x), p*car(x))}
список(m, n) ≡ если m>n то НИЛ иначе
                cons (m, список(m+1,n))
двучлен(x, y) = cons(x, cons(y, НИЛ))
```

Данная программа строит список натуральных чисел и вычисляет их сумму и произведение. Пусть нам дан некоторый бесконечный список натуральных чисел, который формируется в программе при помощи функции список(m, n). При нормальном порядке выполнения программа сначала должна построить весь список, а затем вычислить сумму и произведение его членов. Однако если список будет бесконечным, произойдет зацикливание программы на этапе формирования списка и её работа завершится неудачей. Существуют два способа решения данной проблемы.

Первый способ заключается в том, чтобы ограничить количество элементов списка до начала его формирования, введя в программу дополнительные функции либо жёсткие ограничения. Однако при этом ухудшатся функциональные характеристики программы – её универсальность.

Второй способ более выгоден и основан на применении метода задержанных вычислений. Суть его заключается в том, что формируется не полный список чисел, а только его часть, которая в данный момент используется функцией сумпроизв(x). Причем согласно правилам связывания, функция формирования списка список(m, n) будет выдавать элементы именно в том порядке, в каком они требуются для функции сумпроизв(x) [1].

Реализация этого метода основана на организации задержки при формировании следующего элемента списка в функции список(m, n), которая примет следующий вид:

```
список(m, n) ≡ если m>n то НИЛ иначе
                cons (m, задержать(список(m+1,n)))
```

либо с использованием функции без параметров:

```
список(m, n) ≡ если m>n то НИЛ иначе
                cons (m, λ( ) список(m+1,n))
```

Возобновление же вычислений должно производиться в функции сумпроизв(x) при обращении к следующему элементу списка:

```
сумпроизв(x) ≡ {сп (x, 0, 1)
                gдерек сп = λ(x, s, p)
                если равно(x, НИЛ) то двучлен (s, p)
                иначе сп (возобновить cdr(x), s+car(x)
                p*car(x))}
```

либо с использованием функции без параметров:

```
сумпроизв(x) ≡ {сп (x, 0, 1)
                gдерек сп = λ(x, s, p)
                если равно(x, НИЛ) то двучлен (s, p)
                иначе сп (cdr(x) ( ), s+car(x), p*car(x))}
```

Таким образом, согласно предлагаемому методу, вычисления должны выполняться в такой последовательности.

Шаг 1. Первый вызов функции сумпроизвспис(n) вызывает функцию список(1, n), результатом выполнения которой будет список(1, n) = cons (1, список(2, n)), где второй параметр функции cons – список(2, n) – задержан и представляет собой рецепт и может быть использован как аргумент в функции сумпроизв(x), которая выполняется далее и результат её работы отображается следующим образом: так как x ≠ НИЛ, то сумпроизв(1) = сп (cdr(x), 0+1, 1*1) = сп (список(2, n), 1, 1), где cdr(x) – следующий элемент списка, генерируемый при необходимости, вместо которого и подставляется невычисленный рецепт.

Шаг 2. Теперь требуется, чтобы задержанное вычисление список(2, n) было возобновлено при задержке вычисления список(3, n), в результате чего получаем

список(2, n) = cons (2, список(3, n))

так как

x ≠ НИЛ, то сумпроизв(3) = сп (cdr(x), 1+2, 1*2) = сп (список(3, n), 3, 2)

Шаг 3. список(3, n) = cons (3, список(4, n))

так как

x ≠ НИЛ, то сумпроизв(4) = сп (cdr(x), 3+3, 2*3) = сп (список(4, n), 6, 6) и т. д.

Таким образом, мы видим, что организация задержанных вычислений позволяет избежать зацикливания программы и получить некоторый конечный результат без сообщений об ошибке. Иными словами, данный алгоритм будет работать при любых значениях количества элементов списка, и мы получаем количественную оптимизацию вычислений.

Реализация интерпретатора с задержанным вычислением для полиморфных вычислений. В качестве примера рассмотрим функцию увеличения всех элементов списка на некоторое число *N*:

```
увелич(x) ≡ если равно(x, НИЛ) то НИЛ иначе
            cons(car(x) + N, увелич(cdr (x)))
```

Если эту функцию применить к списку целых чисел, то она выдает список такой же длины, но с элементами, на единицу увеличенными по сравнению с исходным списком.

Рассмотрим также аналогичную предыдущей функцию умножения всех элементов списка на некоторое число *N*:

```
умнож(x) ≡ если равно(x, НИЛ) то НИЛ иначе
           cons(car(x) * N, умнож(cdr(x))),
```

Эта функция вычисляет по исходному списку *x* список, каждый элемент которого умножен на некоторое число *N*.

Семантико-алгоритмическая часть рассматриваемых функций практически одинаковая, различие лишь в операции, совершаемой над элементами списка, т. е. в первом случае это сложение, во втором – умножение. Данный факт приводит нас к решению задачи объединения подобных функций в одну обобщённую функцию, в которой операция, выполняемая над элементами списка, будет являться параметром самой функции. Такая универсальная функция может быть записана в виде следующей программы:

```
вычисл (f) = λ (x) если равно(x, НИЛ) то НИЛ иначе
             cons( f(car(x)) (вычисл (f))(cdr(x)))
```

Здесь *f* – функция, производящая операции над элементами списка x.

При работе программы функция, будучи применена к *x*, порождает новый список, каждый элемент которого получается применением соответствующей

функции к x. Однако в случае когда f не будет известна заранее, программа завершится неудачей. Во избежание этого применим для данной программы метод задержанных вычислений, в результате программа принимает следующий вид:

```
вычисл (f) = λ (x) ( ) если равно(x, НИЛ)
то НИЛ иначе
        cons(f(car(x)), λ ( ) (вычисл (f))(cdr(x)))
```

Задержка аргумента при помощи функции без параметров λ( ) (вычисл (f))(cdr(x)) позволяет интерпретатору с задержанным вычислением обобщить вызываемые процедуры, обеспечивая таким образом универсальность программы, которая заключается в том, что она будет работать при любых функциях и производить требуемые операции над элементами исходного списка. Вычисление будет производиться, как только становится известной выполняемая функция.

На основании изложенного можно заключить, что применение интерпретатора с задержанными вычислениями даёт возможность пользоваться обобщенными процедурами из базы процедур B: из каждой обобщенной процедуры P, принадлежащей B, можно получить при помощи задержанных вычислений производную процедуру P_i от P, что улучшает емкостные характеристики, а что самое главное – улучшает логико-структурную и интеллектуальную мощь системы.

Адаптация процедуры с помощью интерпретатора с задержанным вычислением. В качестве примера рассмотрим следующие предложения русского языка.

1. Павел разбил чашку.
2. Павел разбил сквер.

В компьютерном словаре слово «разбил» толкуется при помощи базисного слова лингвистики как деструкция или разрушение. Однако в отдельных контекстах слово «разбил» имеет прямо противоположный смысл и означает созидание или креативность. Если рассмотреть слово «разбил» как оболочку полиморфной функции, то в контексте функция «разбил» обретает конкретное значение деструкция или креативности:

```
разбил(X, чашку):= деструкция;
разбил(X, сквер):= креативность.
```

В различных вычислительных процессах довольно часто возникает необходимость конкретизировать (найти значения) некоторые объекты (функции, процедуры, автоматные структуры), а затем после конкретизации возобновить вычислительный процесс. Для конкретизации объекта приходится задержать вычислительный процесс и начать интерпретировать указанный объект в контексте задержанного вычислительного процесса.

Итак, мы рассмотрели методы оптимизации работы алгоритмов, используя интерпретатор с задержанным вычислением.

Интерпретатор с задержанным вычислением U можно использовать как автоматический генератор родственных процедур (процедур, обладающих «сходством»), который, работая в режиме динамической интерпретации, существенно повышает интеллектуальную мощь системы программирования. Указанное свойство интерпретатора можно понимать как одну из необходимых составляющих систем баз знаний.

В связи с этим введём определение: под динамическим режимом интерпретации будем понимать адаптацию обобщённой процедуры из обобщённой базы процедур к текущему вычислительному контексту под управлением интерпретатора с задержанным вычислением.

Также следует отметить одно важное обстоятельство: в ламбда-исчислении аргумент функции может превратиться в функцию (занять активную позицию), а функция, наоборот, превратиться в аргумент (занять пассивную позицию).

Используя указанный интерпретатор, можно реализовать данную идею ламбда-исчисления в программировании. Идея принципа активизации данных [4; 5] в программировании имеет чрезвычайно важную роль для ликвидации технологии ручного программирования и переходу к автоматизации программирования для повышения производительности труда программистов и снижения себестоимости программных продуктов.

Интерпретатор U выполняет роль метапроцедуры относительно процедуры $P(x_1, \ldots, x_n)$: исходными данными для U являются неполное тело процедуры P, множество значений для параметров $\{x_{j1}, \ldots, x_{jk}\} \subset \{x_{i1}, \ldots, x_{im}\}$. Для построения процедуры адаптации можно использовать конечную совокупность $\{G_t\}$ контекстно-свободных грамматик, построив формальную модель предметной области в виде уравнений, использующих нетерминальные символы соответствующих грамматик. Для построения уравнений можно использовать операцию сопоставления образца с шаблоном [6, с. 318]. В следующей статье авторы продемонстрируют сказанное на примерах.

Библиографические ссылки

1. Хендерсон П. Функциональное программирование. Применение и реализация : пер. с англ. М. : Мир, 1983.

2. Scott D. Lattice Theory, Data Types and Semantics // Formal Semantics of Programming Languages / ed. R. Rustin. Englewood Cliffs, N. J. : Prentice-Hall, 1972.

3. Барендрегт Х. Ламбда-исчисление. Синтаксис и семантика. М. : Мир, 1981.

4. Тузов В. А. Математическая модель языка. Л. : Изд-во Ленингр. ун-та, 1984.

5. Могнонов П. Б., Чимитов Д. Н., Ярышкина Н. В. Формальное определение семантики языка ASPLE // Математика, ее приложение и мат. образование : материалы 3-й Всерос. конф. с междунар. участием. Ч. 1. Улан-Удэ : Изд-во ВСГТУ, 2008. С. 211–222.

6. Глушков В. М., Ющенко Е. Л., Цейтлин Г. Е. Алгебра. Языки. Программирование. Киев : Наукова думка, 1978.

P. B. Mognonov, D. N. Chimitov, N. V. Yaryshkina

IMPLEMENTATION OF THE INTERPRETER WITH THE DELAYED CALCULATION

One of methods of optimization of operation of algorithms, by means of the interpreter with the delayed calculation, is considered. The authors make an attempt to implement algorithm of operation of such interpreter on the basis of expanded functional language LISP. Value of the delayed calculations for procedure adaptation is shown.

Keywords: the delayed calculations, the interpreter, lambda-calculation, LISP, polymorphism.

УДК 004.75

Н. А. Распопин, М. В. Карасева, П. В. Зеленков, Е. В. Каюков, И. В. Ковалев

МОДЕЛИ И МЕТОДЫ ОПТИМИЗАЦИИ СБОРА И ОБРАБОТКИ ИНФОРМАЦИИ*

Рассмотрена структура представления данных, отвечающих определенным требованиям. Данная дополнительная структура данных разработана исходя из особенностей поисковой системы и требуемой функциональности системы.

Ключевые слова: оптимизация, поисковая система, расширенная база, сбор и обработка информации.

Сегодня редко можно встретить человека, который не использовал бы поисковые технологии. Поиск информации в информационных сетях начал зарождаться задолго до появления глобальной сети. Прообразом первых сетевых поисковиков считается картотечная система наподобие той, что мы встречали в библиотеках, когда большое количество документов классифицируется по особому признаку, каковым может быть жанр произведения, отрасль науки, автор, название произведения; используется и упорядочение по алфавиту. Классификация может быть одноуровневая, когда мы производим ее только по одному признаку, или многоуровневая, когда признаков несколько. Например, мы классифицируем всю литературу по отраслям науки, а потом каждую отрасль науки классифицируем по направлению или по автору и т. д. Система каталогов просуществовала в бумажном виде долгое время, а впоследствии она была перенесена в электронный вариант. В Интернете, в частности, можно было встретить сайты-каталоги, где все многообразие веб-страниц было строго сортировано по тематике, и для того чтобы найти «нужный» сайт, приходилось просматривать десятки, а то и сотни ресурсов. С тех пор в сфере поисковых технологий изменилось очень многое.

Первые поисковые системы сталкивались с большим количеством проблем – прежде всего технического характера [1]. В то время как математические модели позволяли реализовывать достаточно сложные по содержанию алгоритмы, которые могли обеспечить приемлемый для того времени уровень сервиса, аппаратные средства не могли предоставить платформы для реализации подобного рода идей. В значительной степени не хватало мощности процессора для обеспечения производительности, однако основной проблемой была нехватка памяти как оперативной – для реализации самого процесса поиска, так и дисковой – для хранения пространства данных. Разработчики были вынуждены прибегать к многократной оптимизации, что сказывалось на качестве конечного продукта – приходилось чем-то жертвовать. С развитием вычислительной техники подобного рода проблемы ушли на второй план, уступив место новым.

Главная из них – огромный объем информации, которая ко всему прочему имеет очень низкий уровень формализации, что в значительной степени затрудняет поиск на пространстве данных [1]. Под уровнем формализации понимается степень смысловой и любой иной неоднозначности данных. Также необходимо учитывать и лингвистическую неоднозначность. Это прежде всего грамматические, синтаксические, семантические ошибки, которые искажают пространство данных, приводя в конечном счете к ошибочным результатам работы поисковой системы.

Лавинообразное накопление цифровой информации требует совершенно новых подходов к решению проблемы поиска по информационному пространству. Под информационным пространством понимается вся совокупность данных в цифровом виде, доступная посредством различных носителей информации, а также местных локальных и глобальных сетей [2].

*Работа выполнена при финансовой поддержке ФЦП «Научные и научно-педагогические кадры инновационной России на 2009–2013 гг.»