В. Ф. ТУРЧИН, В. И. СЕРДОБОЛЬСКИЙ

ЯЗЫК РЕФАЛ И ЕГО ИСПОЛЬЗОВАНИЕ ДЛЯ ПРЕОБРАЗОВАНИЯ АЛГЕБРАИЧЕСКИХ ВЫРАЖЕНИЙ

УДК 681.3.06:51

АЛГОРИТМИЧЕСКИЙ ЯЗЫК РЕКУРСИВНЫХ ФУНКЦИЙ (РЕФАЛ)

Возникает все больше задач (решаемых на вычислительных машинах), которые связаны с выполнением преобразований над последовательностями символов, например: проведение аналитических выкладок, доказательство теорем, переводы с естественных языков, автоматическое программирование и др. Алгоритмы этих преобразований, как правило, могут быть удобно записаны с помощью некоторого числа рекурсивных функций, определенных на множестве всевозможных цепочек (последовательностей) символов и принимающих значения из этого же множества, или — используем более привычный для программистов термин — с помощью рекурсивных процедур над цепочками символов. Существующие машинно-независимые языки программирования, ориентированные на такие задачи (ЛИСП, КОМИТ, СНОБОЛ и др.), в той или иной форме предусматривают возможность записи алгоритма в рекурсивном виде. Метаалгоритмический язык, предложенный в [1], [2], допускает запись алгоритмов преобразования цепочек символов в виде, наиболее близком к математическому языку рекурсивных функций и, как полагают авторы, более удобном и прозрачном, чем другие алгоритмические языки.

В настоящей работе будет использован метаалгоритмический язык в том виде, как он описан в [2], с введением дополнительно следующего соглашения. Левую скобку, следующую непосредственно за символом конкретизации Ж, будем опускать, а соответствующую правую скобку, отмечающую конец области действия символа Ж, заменять точкой. Если область действия символа Ж состоит из одного символа, то после него также будем ставить точку. Таким образом, предложение, которое раньше имело вид

$$\S\ Ж(\varrho E1, E2) \sim Ж(\sigma Ж A + Ж(E1 + E2)),$$

теперь будет записано следующим образом:

$$\S\ Ж\varrho E1, E2. \sim Ж\sigma Ж A. + Ж E1 + E2..$$

Если левая часть предложения начинается со значащего символа, будем называть этот символ *детерминативом* предложения. Например, в приведенном выше предложении ϱ — детерминатив. Понятно, что предложения с различными детерминативами можно переставлять в поле памяти, никак не влияя на работу машины. Следовательно, можно сгруппировать все предложения с данным детерминативом, поместив их одно за другим. Порядок предложений в группе, вообще говоря, существенен, но группы в целом можно переставлять как угодно. Исходя из этого, будем записывать на метаалгоритмическом языке определения (вообще говоря, рекурсивные) различных функций. Вместо $f(\xi_1)$, $g(\xi_2)$ и т. д., где ξ_i — произвольные цепочки символов, будем писать $Жf\xi_1.$, $Жg\xi_2.$ и т. д., а определениями функций f, g будут служить группы предложений с детерминативами f, g и т. д. Различные предложения в группе описывают правила конкретизации (т. е. правила вычисления значения функции) при различных типах аргументов. Наличие знака Ж в записи функции $Жf\xi_1.$ приведет к тому, что машина будет выполнять вычисления до тех пор, пока в поле зрения не останется ни одной функциональной записи. Конструкции типа $f(g(\xi_1, h(\xi_2)))$, включающие функции от функций, будут вычисляться правильно благодаря определению ведущего символа конкретизации, данному при описании метаалгоритмической машины [2]. Если пользоваться только предложениями с детерминативами, а правые части предложений записывать таким образом, чтобы за символом Ж всегда следовал либо символ машинной операции, либо детерминатив, то получим суженный метаалгоритмический язык, который может быть назван алгоритмическим языком рекурсивных функций (РЕФАЛ).

ПРЕОБРАЗОВАНИЕ АЛГЕБРАИЧЕСКИХ ВЫРАЖЕНИЙ

Задачу упрощения алгебраических выражений можно разделить на две части — «рутинную» и «эвристическую». Рутинная часть включает в себя применение к алгебраическому выражению некоторого числа алгоритмов, заведомо упрощающих, в случае своей применимости, выражение. Сюда относится снятие лишних (син-

таксически) скобок, приведение подобных членов, замены типа $A \times 0 = 0$ и т. п. Эвристическая часть предполагает введение некоторой меры сложности выражения (например, суммарное число знаков) и выполнение какого-то числа попыток уменьшить сложность выражения путем тождественных преобразований, которые могут привести (но могут и не привести) к упрощению, а также путем преобразований, для которых проверка применимости рутинным способом слишком громоздка и которые поэтому требуют эвристического программирования в собственном смысле слова.

Ясно, что более простая рутинная часть является необходимой основой для перехода к эвристической части. В настоящей работе описан алгоритм «нормализации» алгебраического выражения, выполняющий всю рутинную часть задачи упрощения, за исключением вынесения за скобки общего множителя, которое вместе с попыткой разложить сумму на множители (и, конечно, с раскрытием скобок) мы сочли более удобным отнести ко второй, эвристической части.

В общих чертах алгоритм нормализации заключается в следующем. Прежде всего исходное алгебраическое выражение приводится к определенному стандартному виду, отличному от обычной алгебраической записи в следующих отношениях.

1. Все числа записываются в десятичном виде, снабжаются признаком ν и заключаются в скобки, например (ν 1), (ν 58), (ν — 0.00137).

2. Операция возведения в степень записывается в виде ξ ($\uparrow$ η), где ξ — простое арифметическое выражение, а η — арифметическое выражение (в смысле синтаксиса АЛГОЛа).

3. Функциональные обозначения вроде $f(x)$, $\varphi(x, y, z)$ и т. п. записываются в виде (f ⟨от⟩ x), (φ ⟨от⟩ x, y, z) и т. д.

Допускается также операторная запись (S ⟨от⟩ ξ), где оператор S должен быть термом в смысле синтаксиса метаалгоритмического языка, например $((\partial/\partial x)$ ⟨от⟩ (f ⟨от⟩ x)).

4. Операции вычитания и деления заменяются сложением и умножением с введением множителя или показателя—1. Эта замена, без которой, в принципе, можно обойтись, значительно упрощает основные алгоритмы.

5. Знак умножения × не может быть опущен.

Снабжение чисел признаком ν иллюстрирует наш метод решения проблемы выделения синтаксических понятий. Для описания преобразований над цепочками символов обычно бывает необходимо различать некоторые синтаксические категории, например в нашем случае — числа и переменные (изображаемые буквами). Если заставлять машину каждый раз распознавать данную цепочку символов как число (пользуясь, например, бэкусовским синтаксисом), то это повлечет существенную потерю эффективности. При нашем подходе выделение синтаксических понятий производится один раз — при первом просмотре, в результате которого в текст вставляются признаки синтаксических категорий, по которым в дальнейшем они сразу распознаются. Снабдив все числа признаком ν, мы в левых частях правил конкретизации можем записывать произвольные числа в виде (νE1), (νE2) и т. д. Устранение операции — и / избавляет нас от необходимости различать аддитивные (+, —) и мультипликативные (×, /) операции, но если бы мы пожелали сохранить — и /, то можно было бы всюду заменить +, —, ×, / на α +, α —, μ ×, μ / соответственно. Тогда произвольные аддитивные операции изображались бы в виде αS1, αS2 и т. д., а мультипликативные операции — в виде μS1, μS2, μS3, . . . и т. д.).

Применение процедуры (функции) нормализации к алгебраическому выражению в стандартной форме сводится к последовательности процедур, которую легко можно усмотреть из формального описания алгоритма (см. ниже). Детерминативами процедур служат символы (в смысле метаалгоритмического языка), образованные путем сокращений, которые нетрудно расшифровать. Укажем наименее очевидные сокращения: < с. л. с. > — снятие лишних скобок, < п. п. с. > — приведение подобных сомножителей, < п. п. ч. > — приведение подобных членов. Для экономного приведения подобных членов и сомножителей используем процедуру упорядочения, которая устраняет неоднозначность записи выражений, возникающую вследствие коммутативности сложения и умножения. После выполнения упорядочения подобные члены или сомножители оказываются рядом. Процедура упорядочения опирается на отношение следования алгебраических выражений, определенное отдельной группой предложений.

При программировании на языке РЕФАЛ можно некоторые процедуры не описывать, а объявить их машинными процедурами, которые при трансляции будут выполняться на уровне машинных команд. В данном случае мы относим к машинным процедурам следующие: 1) операции над числами, объединенные в одну процедуру с детерминативом < вычисл. >; 2) функцию с детерминативом < отрицат . >, принимающую значение «*И*» для отрицательных чисел и «*Л*» —

для всех остальных цепочек символов; 3) функцию с детерминативом < маш. порядок > такую, что Я < маш. порядок > S1S2. есть «Л» в том и только в том случае, если символ S1 следует за символом S2 в некоторой фиксированной последовательности, упорядочивающей все допустимые символы.

Процедура нормализации рассматривает каждую пару скобок (кроме лишних, подлежащих снятию) как «закрытую» и не входит внутрь этих скобок. Через эту процедуру определяется процедура сквозной нормализации, которая проникает во все скобки и начинает выполняться изнутри.

В формальном описании алгоритма, которое приводится в следующем разделе, мы пользуемся функцией с детерминативом ⟨ если ⟩. Она определяется несколько иначе, чем в [2], а именно:

§ Я ⟨если⟩ E*a* ⟨то⟩ E1 ⟨иначе⟩ E2.
∼ Я ⟨если⟩ Я E*a*. ⟨то⟩ E1 ⟨иначе⟩ E2.
§ Я ⟨если⟩ *И* ⟨то⟩ E1 ⟨иначе⟩ E2. ∼ Я μ E1.
§ Я ⟨если⟩ *Л* ⟨то⟩ E1 ⟨иначе⟩ E2. ∼ Я μ E2.
§ Я μ E1. Я → *A* ∼ E1. Я ≈ Я A..
§ Я ≈ E A. ∼ Я ∼ E *A*.

Отличие заключается в том, что прежде чем заменять условное выражение на выражение, стоящее после символа ⟨ то ⟩ или ⟨ иначе ⟩, это последнее выражение подвергается мета-преобразованию μ-(детерминатив μ) (см. [2]). В результате μ - преобразования пассивная группа Я′ переходит в активный символ конкретизации Я. Последние два параграфа служат для определения μ - преобразования как обычной процедуры. Необходимый эффект достигается путем временного занесения аргумента в память и последующей его выборки из памяти.

Кроме условного оператора, предполагаются определенными остальные понятия, описанные в [2] как «основные» понятия программирования. Определения совпадают с приведенными в [2].

ФОРМАЛЬНОЕ ОПИСАНИЕ АЛГОРИТМА

Первая группа предложений определяет процедуру нормализации:

§ 1.1. Я ⟨норм.⟩ E1. ∼ Я ⟨преобр. сум.⟩ Я ⟨преобр. терм.⟩ Я ⟨с. л. с.⟩ E1...
§ 1.2. Я ⟨норм.⟩ (νE1) S2 (νE3). ∼ Я ⟨вычисл.⟩ (νE1) S2 (νE3).
§ 1.3. Я ⟨норм.⟩ (νE1). ∼(νE1)
§ 1.4. Я ⟨норм.⟩ S1. ∼ S1

Чтобы понять, как работает эта процедура, надо читать предложения снизу вверх: в том порядке, в котором они просматриваются машиной. Предложение 1.4. говорит, что применение процедуры нормализации к отдельному символу дает этот самый символ. Предложение 1.3 говорит то же о произвольном числе. Предложение 1.2 приводит к немедленному выполнению арифметических действий над двумя числами. Эти три предложения выделяют простейшие частные случаи. Если ни одно из них не оказалось применимым, то сработает предложение 1.1, которое описывает общий случай и сводит выполнение нормализации к последовательному выполнению следующих трех процедур: снятия лишних скобок, преобразования термов и преобразования суммы.

Снятие лишних скобок описывается группой предложений 2.1—2.5:

§ 2.1. Я ⟨с. л. с.⟩ E1. ∼ E1
§ 2.2. Я ⟨с. л. с.⟩ E1 (E3) E5. ∼ E1 Я ⟨если⟩ ((оканч. на ×) E1) ∨ ((нач. на ×) E5)) & (⟨сумма⟩ E3) ⟨то⟩ (E3) Я′ ⟨с. л. с.⟩ E5. ⟨иначе⟩ Я′ ⟨с. л. с.⟩ E3E5..
§ 2. 3. Я ⟨с. л. с.⟩ E1 (W3 ⟨от⟩ E4) E5. ∼ Я ⟨с. л. с.⟩ E1. (W3 ⟨от⟩ E4) Я ⟨с. л. с.⟩ E5.
§ 2.4. Я ⟨с. л. с.⟩ E1 (νE3) E5. ∼ Я ⟨с. л. с.⟩ E1. (νE3) Я ⟨с. л. с.⟩ E5.
§ 2. 5. Я ⟨с. л. с.⟩ E1W3 (↑E4) E5. ∼ Я ⟨с.л.с.⟩ E1W3 (↑E4) Я ⟨с. л. с.⟩ E5.

Предложения 2.5, 2.4, 2.3 описывают случаи, когда скобки заведомо снимать нельзя, ибо они служат для выделения степени, числа и функции (соответственно). Эти предложения выносят соответствующие скобки и их содержимое из области действия процедуры < с. л. с. >. Предложение 2.2, поскольку оно стоит выше, будет применяться только в тех случаях, когда скобки служат для указания порядка действий. Скобки надо сохранить лишь в том случае, если выражение в скобках содержит знак плюс, не заключенный в другую пару скобок (т. е. является суммой) и умножается слева или справа на какое-либо другое выражение. Это правило и изображается предложением 2.2, для чего вводятся три вспомогательных предиката: оканчиваться на знак умножения, начинаться со знака умножения и быть суммой. Заметим, что так как наша процедура нормализации не входит внутрь скобок, выражение E3 остается вне сферы действия процедуры < с. л. с. >, когда скобки остаются, и входит в нее, когда скобки снимаются. На предложение 2.1 мы попадаем только тогда, когда в аргументе нет ни одной скобки. Процедура < с. л. с. > в этом случае заканчивается.

Следующие шесть предложений описывают

вспомогательные предикаты, использованные в § 2.2. Они вряд ли нуждаются в пояснениях.

§ 3.1. Ж⟨оканч. на×⟩E1.~*Л*
§ 3.2. Ж⟨оканч. на ×⟩E1×.~*И*
§ 4.1. Ж⟨нач. на ×⟩E1.~*Л*
§ 4.2. Ж⟨нач. на ×⟩×E1.~*И*
§ 5.1. Ж⟨сумма⟩ E1.~*Л*
§ 5.2. Ж⟨сумма⟩ E1 + E2.~*И*

Процедура преобразования термов описывается следующей группой предложений:

§ 6.0. Ж⟨преобр. терм.⟩ E1.~Ж⟨дей.×⟩ Ж⟨упор. по ×⟩:Ж⟨дей.↑⟩Ж⟨п.п.с.⟩ Ж⟨ввести↑1⟩Ж⟨упор. по×⟩:E1... ...
§ 6.1. Ж⟨преобр. терм.⟩ W1.~W1
§ 6.2. Ж⟨преобр. терм.⟩ E1 +E2.~Ж(преобр. терм.) E1. + Ж⟨преобр. терм.⟩ E2.

Предложение 6.2. расщепляет обрабатываемую цепочку на отдельные термы (в арифметическом смысле). Если арифметический терм является также термом в смысле метаалгоритмического языка, т. е. одним символом или выражением в скобках, то предложение 6.1 приведет к окончанию процедуры. Если он состоит из нескольких сомножителей, то будет действовать предложение 6.0, которое подписывает последовательное выполнение следующих шести процедур.

1. Упорядочение по умножению. Оно приводит к тому, что подобные сомножители (т. е. сомножители, различающиеся только показателем степени) оказываются рядом. Знак :, вводимый в аргумент, служит в процессе выполнения процедуры указателем, отделяющим уже упорядоченную часть последовательности от еще не упорядоченной.

2. Введение единичной степени в сомножители, не содержащие показателя степени. Это вспомогательная процедура, служащая для упрощения приведения подобных сомножителей.

3. Приведение подобных сомножителей.

4. Упрощения, связанные с действием возведения в степень, отражающие такие его свойства, как $x^1 = x$, $x^0 = 1$ и т. д., а также действия над числами.

5. Повторное упорядочение по умножению. Оно необходимо, потому что приведение подобных сомножителей может разрушить порядок по умножению. В то же время все термы должны быть преобразованы к стандартной форме, чтобы было возможно приведение подобных членов при обработке суммы.

6. Упрощения, связанные с действием умножения: $x \times 1 = x$ и т. п., а также действия над числами.

Мы не будем приводить описания всех этих процедур, ограничимся лишь процедурой приведения подобных сомножителей:

§ 11.1. Ж⟨п. п. с.⟩ E1.~E1
§ 11.2. Ж⟨п. п. с.⟩ E1W2(↑E3)×W2(↑E4)E5. ~Ж⟨п. п. с.⟩ E1W2(↑Ж⟨норм.⟩ E3 + E4.)E5.

Преобразование суммы описывается аналогично преобразованию термов.

Упорядочение выражений как по умножению, так и по сложению, опирается на двухместный предикат следования, который каждой упорядоченной паре выражений сопоставляет истинностное значение *И* (порядок правильный) или *Л* (порядок неправильный). Этот предикат описывается следующим набором предложений:

§ *C*1. Ж⟨след.⟩⟨за⟩ E1.~*Л*
§ *C*2. Ж⟨след.⟩ E1⟨за⟩.~*И*
§ *C*3. Ж⟨след.⟩(E1)⟨за⟩(E4).~Ж⟨след.⟩E1⟨за⟩E4.
§ *C*4. Ж⟨след.⟩ W1E2⟨за⟩W3E4.~Ж⟨след.⟩W1⟨за⟩W3.
§ *C*5. Ж⟨след.⟩ W1E2⟨за⟩ W1E4.~Ж⟨след.⟩E2⟨за⟩E4.
§ *C*6. Ж⟨след.⟩ S1⟨за⟩S2.~Ж⟨маш. порядок⟩S2S1.
§ *C*7. Ж⟨след.⟩ S1⟨за⟩(E2).~*Л*
§ *C*8. Ж⟨след.⟩(E1)⟨за⟩S2.~*И*
§ *C*9. Ж⟨след.⟩(νE1) за E2.~*И*
§ *C*10. Ж⟨след.⟩ E1⟨за⟩(νE2).~*Л*
§ *C*11. Ж⟨след.⟩(νE1)⟨за⟩(νE2).~Ж¬К⟨отрицат.⟩ Ж⟨вычисл.⟩(νE2)—(νE1)...

Через процедуру нормализации, которая, как указывалось, действует только на верхнем уровне скобочной структуры, определяется процедура сквозной нормализации (на всех уровнях). Ее описание включает вспомогательную процедуру внесения оператора нормализации в скобки (< войти — в — ск >) и выглядит следующим образом:

§ 21.1. Ж⟨скв. норм.⟩ E1.~Ж⟨норм.⟩ Ж⟨войти — в — ск.⟩ E1..
§ 21.2. Ж⟨скв. норм.⟩W1⟨от⟩E2.~W1⟨от⟩ Ж⟨скв. норм.⟩E2.
§ 21.3. Ж⟨скв. норм.⟩↑E1.~↑Ж⟨скв. норм.⟩E1.
§ 21.4. Ж⟨скв. норм.⟩νE1.~νE1
§ 22.1. Ж⟨войти — в — ск.⟩ E1.~E1
§ 22.2. Ж⟨войти — в — ск.⟩ E1.(E3)E5. ~E1(Ж⟨скв. норм.⟩ E3.) Ж⟨войти — в —ск.⟩E5.

В общей сложности описание алгоритма содержит 71 предложение.

ПРИМЕР РАБОТЫ АЛГОРИТМА

Для проверки алгоритмов, написанных на метаалгоритмическом языке, был написан транслятор «КИТ—1» для машины М—20, работающий в режиме интерпретации, иначе говоря, моделирующий на машине М—20 работу метаалгоритмической машины. Транслятор налагает некоторые ограничения на язык, из которых самое существенное — возможность занесения в память только таких предложений, у которых левая часть состоит из одного (значащего) символа. Транслятор содержит около 2000_8 команд. Ввод осуществляется путем ручной кодировки символов цифрами и последующей пробивки. Результаты выводятся на буквенно-цифровую «широкую» печать машины «Урал». После окончания работы печатается число шагов метаалгоритмической машины, выполненное за время работы (при этом подстановка выполняется за два шага).

Для отладки алгоритма нормализации через транслятор «КИТ-1» было пропущено несколько примеров. Исходное выражение вводилось в машину в алголоподобном виде. Затем над ним производилось входное преобразование, приводящее его к стандартному виду, требуемому алгоритмом нормализации. (Входное преобразование также было записано на языке РЕФАЛ и выполнялось через «КИТ-1».) Затем над этим выражением выполнялась процедура сквозной нормализации. В текст алгоритма было введено несколько процедур вывода (отсутствующих в приведенном выше формальном описании), вследствие чего машина по ходу работы выдавала промежуточные результаты. В одном из примеров задание состояло в сквозной нормализации выражения

$$2\times((A+B+C)-C)/(B+A)+A-2.$$

Приводим текст машинной выдачи. Символ v (признак числа) печатается в виде звездочки *.

нормализ. $2\times((A+B+C)-C)/(B+A)+A-2$
вх. преобр. $(*2)\times((A+B+C)+(*-1)\times C)\times$
$\times(B+A)(\uparrow(*-1))+A+(*-1)\times(*2)$
нормализ. $A+B+C$
⟨упор. по $+$⟩ $A+B+C$
нормализ. результат $A+B+C$
нормализ. $(A+B+C)+(*-1)\times C$
$(A+B+C)\to A+B+C$
⟨упор. по $+$⟩ $A+B+C+C\times(*-1)$
$C\times(*1)+C\times(*-1)=C\times(*0)$
$C\times(*0)=(*0)$
$A+B+(*0)=A+B$
нормализ. результат $A+B$
Нормализ. $B+A$
⟨упор. по $+$⟩ $A+B$
Нормализ. результат $A+B$
нормализ. $(*2)\times(A+B)\times(A+B)(\uparrow(*-1))+A+$
$+(*-1)\times(*2)$
$(A+B)(\uparrow(*1))\times(A+B)(\uparrow(*-1))=(A+$
$+B)(\uparrow(*1)+(*-1))$
$(*1)+(*-1)=(*0)$
$(A+B)(\uparrow(*0))=(*1)$
$(*2)\times(*1)=(*2)$
$(*2)\times(*-1)=(*-2)$
⟨упор. по $+$⟩ $A+(*2)+(*-2)$
$(*2)+(*-2)=(*0)$
$A+(*0)=A$
нормализ. результат A
Результат A
Задание выполнено. Число шагов 1010.

ЗАКЛЮЧЕНИЕ

С точки зрения авторов язык РЕФАЛ, а также более общий метаалгоритмический язык, весьма удобны для записи алгоритмов преобразования цепочек символов. При выполнении алгоритма, записанного на РЕФАЛе, с помощью транслятора «КИТ-1», получается большая потеря эффективности (по сравнению с программой, написанной вручную) из-за того, что «КИТ-1» в буквальном соответствии с описанием метаалгоритмической машины [1], [2] совершает много излишних просмотров и переписей. Но можно написать транслятор [3], свободный от этого недостатка, и при соблюдении определенных правил конструирования предложений потеря эффективности при реализации алгоритмов не будет превышать разумного предела. Поэтому РЕФАЛ (или метаалгоритмический язык) может явиться языком массового пользования для задач преобразования информации, задаваемой в виде последовательностей символов.

ЛИТЕРАТУРА

1. В. Ф. Турчин, в сб. «Цифровая вычислительная техника и программирование», изд-во «Советское радио», 1966, стр. 116.
2. В. Ф. Турчин, Метаалгоритмический язык, журн. «Кибернетика», № 4, К., 1968.
3. Рефал—интерпретатор (доклад), Первая Всесоюзная конференция по программированию, Киев, ноябрь 1968.

Поступила в редакцию 31.V 1967