ВЫВОД ТИПОВ ПЕРЕМЕННЫХ, НАПРАВЛЕННЫЙ НА ПОВЫШЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ ПРОГРАММ

Третьяк Александр Викторович
старший преподаватель департамента информационной безопасности,
Дальневосточный федеральный университет,
РФ, г. Владивосток

TYPE INFERENCE AIMED AT IMPROVING PROGRAM PERFORMANCE

Tretyak Alexander Viktorovich
senior teacher, Department of Information Security,
Far Eastern Federal University,
Russia, Vladivostok

АННОТАЦИЯ

Идея вывода типов не нова, однако существующие решения делают акцент на упрощении исходного кода, а не на повышении производительности программы.

Фактически, в существующих языках программирования применяется только два вида вывода типов: «простой» (когда тип переменной/аргумента определяется непосредственно из присваиваемого ей выражения) и вывод на основе использования (для которого практически всегда используется алгоритм Хиндли — Милнера [1]). В данной работе предлагается альтернатива второго вида вывода типов.

ABSTRACT

The idea of type inference is not new, but existing solutions focus on simplifying source code rather than improving program performance.

In fact, existing programming languages use only two kinds of type inference: "simple" (when the type of a variable/argument is determined directly from the expression assigned to it) and usage-based inference (for which the Hindley-Milner algorithm [1] is almost always used). This paper proposes an alternative to the second kind of type inference.

Ключевые слова: языки программирования, вывод типов, компилятор

Keywords: programming languages, type inference, compiler

Рассмотрим такую конструкцию:

```
var s = <строковый_литерал>
```

Здесь объявляется переменная s и ей присваивается некоторый строковый литерал. Во всех существующих языках программирования, поддерживающих вывод типов, переменная s при таком объявлении является строковой переменной (то есть её тип — строка). Однако в новом языке программирования 111 предлагается определять тип переменной s по её использованию в последующем (после её объявления) программном коде.

Изначально для s назначается тип PseudoString, который поддерживает все операции и методы типа String, то есть с точки зрения программиста-пользователя переменная s выглядит как обычная строковая переменная. Однако фактический тип s определяется компилятором и далеко не всегда PseudoString превращается в String.

Вот лишь некоторые типы, которые может принять переменная s:

- Char
- StringView
- StringBuilder
- Set/HashSet
- String

Рассмотрим более подробно, в каких случаях для переменной s будет выбран и назначен каждый из этих типов.

Тип Char

Если строковый литерал, назначаемый s, состоит только из одного символа, то компилятор пытается назначить переменной s тип Char. Если это возможно, то есть если к s не применяются операции, которые не поддерживает

тип Char (например, конкатенация или присвоение строке), тогда переменной s назначается тип Char.

Вот пример кода, иллюстрирующий такой случай:

```
var s = "-"
if ...
   s = "--" // наличие любой из этих двух строк приводит
   s += "+" // к тому, что `s` не может иметь тип `Char`
print(s)
```

Тип StringView

Данный тип состоит из двух значений: указателя на начальный символ строки и целого числа, обозначающего длину этой строки в символах. Если к s применяются только константные операции (такие как преобразование к числу, получение i-го символа, выделение подстроки, передача в качестве входного аргумента какой-либо функции), тогда переменной s назначается тип StringView.

Тип StringBuilder

Данный тип представлен во многих популярных языках программирования (например, в Java [2] и C# [3]) и предназначен для оптимизации операции накопления строки-результата в процессе работы некоторого алгоритма. В сети Интернет можно встретить множество статей ([4], [5], [6]), посвященных проблеме производительности конкатенации строк внутри цикла в Java с рекомендацией использовать StringBuilder для результирующей строки вместо типа String.

Вот соответствующий пример кода на Java (код использует тип String):

```
String result = "";
for (int i=0; i<1e6; i++) {
    result += "some more data";
}
processString(result);
```

(Данный код имеет вычислительную сложность $O(n^2)$.)

При использовании StringBuilder получается такой код:

```
var result = new StringBuilder();
for (int i=0; i<1e6; i++) {
    result.append("some more data");
}
processString(result.toString());
```

(Данный код имеет вычислительную сложность O(n).)

Ручная замена String на StringBuilder остаётся актуальной даже в последних версиях Java, так как Java-компилятор автоматически заменяет String на StringBuilder только в простых случаях и не может этого сделать для кода с циклами, как в вышестоящем примере.

В отличие от обычной строки, StringBuilder как правило не предоставляет методы, присутствующие в типе String (такие как find(), replace(), split(), lowercase(), trim() и пр.), и, по сути, поддерживает лишь две операции: добавление и преобразование к строке.

Соответственно, если к s применяются только операции добавления, после которых следует одно или несколько обращений к s, тогда типом s назначается StringBuilder.

Тип Set/HashSet

Если к s применяется только оператор `` `in` `` (в форме `` `<символ> in s` ``), тогда типом s назначается Set/HashSet. При этом, на основе того факта, что s не изменяется, компилятор может подобрать более эффективную (по сравнению с универсальной) реализацию Set или HashSet: например, возможно использовать идеальную хеш-функцию (perfect hash function [7]), то есть такую, которая при заданном заранее известном ограниченном наборе возможных ключей (а ключом в данном случае является символ) гарантированно не даёт коллизий. Это значительно упрощает реализацию HashSet: не требуется поддерживать списки коллизий, а также можно использовать более простую и производительную хеш-функцию.

Тип String

Тип String назначается переменной s в том случае, если для s не удалось назначить ни один из типов, перечисленных выше.

Заключение

На примере типа Set/HashSet хорошо видно главное преимущество автоматического вывода типа переменной s, которое заключается в предоставлении большей свободы компилятору в отношении оптимизации генерируемого машинного кода. То есть, если написать var s = Set(<строковый_литерал>) или var s = HashSet(<строковый_литерал>), то будет жёсткая привязка к реализации конкретного типа. Хотя на самом деле тут, возможно, и не нужен никакой Set/HashSet — если символов в строковом литерале всего 2 или 3, то быстрее будет просто проверить условие (`c == s[0] or c == s[1] or c == s[2]`). Кроме того, если количество символов в строковом литерале небольшое, то компилятор может использовать инструкции расширений процессора SSE2/AVX2/AVX-512, чтобы проверять на равенство некоторому символу все элементы строкового литерала параллельно. Аналогичная оптимизация уже применяется в компиляторах C/C++ для реализации функции memchr [8]. В языках программирования без вывода типов, направленного на повышение производительности программ, пришлось бы дать название такому типу (что-то вроде OptimizedReadonlySet), либо функции, которая бы вызывалась вместо оператора `in`. А в языках с таким выводом типов программист избавлен от необходимости углубляться в детали реализации, и может полностью сконцентрироваться на алгоритме решения задачи (например, есть некий набор символов и необходимо в цикле проверить, что некоторый символ `c` входит в этот набор символов или не входит, а уж об оптимальной реализации такой проверки для конкретной целевой архитектуры процессора [и конкретного набора символов] пусть думает компилятор).

Список литературы:

1. Вывод типов — Википедия / [Электронный ресурс]. – Режим доступа: URL: https://ru.wikipedia.org/wiki/Вывод_типов (дата обращения: 06.08.2023).
2. StringBuilder (Java Platform SE 8) / [Электронный ресурс]. – Режим доступа: URL: https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuilder.html (дата обращения: 06.08.2023).
3. StringBuilder Class (System.Text) | Microsoft Learn / [Электронный ресурс]. – Режим доступа: URL: https://learn.microsoft.com/en-us/dotnet/api/system.text.stringbuilder?view=net-7.0 (дата обращения: 06.08.2023).
4. String concatenation with Java 8 / [Электронный ресурс]. – Режим доступа: URL: https://pellegrino.link/2015/08/22/string-concatenation-with-java-8.html (дата обращения: 06.08.2023).
5. We Don't Need StringBuilder for Simple Concatenation – Dzone / [Электронный ресурс]. – Режим доступа: URL: https://dzone.com/articles/string-concatenation-performacne-improvement-in-ja (дата обращения: 06.08.2023).
6. Improve the performance of your Java programs with string builders | by Zakarie A | Level Up Coding / [Электронный ресурс]. – Режим доступа: URL: https://levelup.gitconnected.com/improve-the-performance-of-your-java-programs-with-string-builders-4c3316ccad55 (дата обращения: 06.08.2023).
7. Perfect hash function - Wikipedia / [Электронный ресурс]. – Режим доступа: URL: https://en.wikipedia.org/wiki/Perfect_hash_function (дата обращения: 06.08.2023).
8. std::find() and memchr() Optimizations | Georg's Log / [Электронный ресурс]. – Режим доступа: URL: https://gms.tf/stdfind-and-memchr-optimizations.html (дата обращения: 06.08.2023).