DOI: https://doi.org/10.36910/6775-2524-0560-2021-42-19
УДК: 004.4`4
Іваненко Антон Романович, студент
https://orcid.org/0000-0002-9846-537X
Марченко Олександр Іванович, к.т.н., доцент
https://orcid.org/0000-0002-4537-3420
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», м. Київ, Україна

СПОСІБ ТРАНСЛЯЦІЇ КОНКАТЕНАЦІЇ РЯДКОВИХ ВИРАЗІВ МОВИ TYPESCRIPT У ПРОМІЖНУ МОВУ CIL ПЛАТФОРМИ .NET

Іваненко А.Р., Марченко О.І. Спосіб трансляції конкатенації рядкових виразів мови TypeScript у проміжну мову CIL платформи .NET. У статті запропоновано спосіб, який надає можливість генерації ефективніших інструкцій CIL для конкатенації рядкових виразів мови програмування TypeScript. Запропонований спосіб базується на виклику потрібного варіанту перевантаження вбудованої у платформу .NET функції в залежності від кількості операндів конкатенації. Результатом роботи способу є згенерований код, який показує кращу швидкодію, ніж код, створений існуючим аналогом.

Ключові слова: транслятор, конкатенація, дерево розбору, CIL, CLR, .NET, TypeScript.

Іваненко А.Р., Марченко А.И. Способ трансляции конкатенации строковых выражений языка TypeScript в промежуточный язык CIL платформы .NET.. В статье предложен способ, который дает возможность генерации эффективных инструкций CIL для конкатенации строковых выражений языка программирования TypeScript. Предложенный способ основан на вызове нужного варианта перегрузки встроенной в платформу .NET функции в зависимости от количества операндов конкатенации. Результатом работы способа является сгенерированный код, который показывает лучшее быстродействие, чем код, созданный существующим аналогом.

Ключевые слова: транслятор, конкатенация, дерево разбора, CIL, CLR, .NET, TypeScript.

Ivanenko A.R., Marchenko O.I. Translation the concatenation of TypeScript string expressions into Common Intermediate Language of .NET platform. The article proposes a method that enables the generation of more efficient CIL instructions for the concatenation of string expressions of the TypeScript programming language. The proposed method is based on calling a proper variant of overload of the function built into the .NET platform depending on the number of concatenation operands. The result of the method is the generated code, which shows better performance than the code generated by the existing analog.

Keywords: translator, concatenation, parsing tree, CIL, CLR, .NET, TypeScript.

Постановка наукової проблеми.

Робота з рядковими типами є однією з найважливіших функціональних можливостей різних мов програмування. Великі об'єми текстових даних можна зустріти в різних сферах людської діяльності: створення веб-ресурсів, машинне навчання, розробка трансляторів, тощо. Через це, велика увага приділяється швидкодії програм, які використовуються у вище зазначених сферах.

Таким чином, при трансляції мови програмування TypeScript у проміжну мову CIL віртуальної машини .NET постає задача генерації таких інструкцій, які дозволять оптимально використовувати ресурси системи при роботі з текстовими даними, особливо – при їх конкатенації.

Аналіз досліджень.

Задачу трансляції різних мов програмування у проміжну мову CIL платформи .NET досліджено у невеликій кількості статей, серед яких можна виділити опис методів трансляції мови програмування Scheme [1] від французьких дослідників та трансляції байт-коду віртуальної машини Java [2] від корейських вчених.

У статті [1] детально проаналізовано, як згенерувати ефективні інструкції для функцій, звертаючи увагу на наявність таких деталей:

- хвостової рекурсії;
- замикання;
- шаблонів.

Стаття [2] демонструє створення транслятора з байт-коду віртуальної машини Java у проміжну мову CIL. Такий підхід дозволяє транслювати та запускати у віртуальному середовищі .NET без змін програми, що були раніше скомпільовані у байт-код.

Обидві статті порівнюють швидкодію віртуальних середовищ JVM та .NET та підкреслюють кращу швидкодію останнього. З цього можна зробити висновок, що обрана цільова платформа для трансляції мови програмування TypeScript є перспективною.

Жодна з публікацій не розкриває реалізацію операцій з рядковими виразами, що залишає поле для дослідження, описаного у цій статті.

Метою статті є розробка методу трансляції конкатенації рядкових виразів мови програмування TypeScript у проміжну мову CIL платформи .NET та порівняльний аналіз швидкодії згенерованого коду з результатом роботи компілятора JScript, який був обраний аналогом для порівняння у попередній статті [3].

Термінологія.

Common Intermediate Language (CIL) – «високорівневий асемблер» віртуальної машини .NET. Проміжна мова, розроблена компанією Microsoft для платформи .NET[4].

Лексичний аналізатор – частина транслятора, що здійснює перетворення вхідного тексту програми, що подана рядком символів, в рядок лексем, поданій в цифровій формі, а також знаходить лексичні помилки[5].

© Іваненко А.Р., Марченко О.І.

Синтаксичний аналізатор – частина транслятора, що здійснює декомпозиції вхідної програми поданої рядком лексем у структурні одиниці мови (оператори, описи, декларації, тощо) [5].

Дерево розбору – внутрішня форма подання вхідної програми, що містить структурні одиниці мови, а також зв'язки між ними[5].

Генератор коду – перетворює вхідну програму, подану одною її внутрішніх форм, в текст (оператори, команди) вихідною мовою на основі семантики вхідної мови [5].

Запропонований спосіб трансляції.

Розглянемо вхідну програму, записану мовою програмування TypeScript, яку потрібно скомпілювати в CIL (рисунок 1).

```
let a: string = 'test a', b: string = 'test b',
    c: string = 'test c', d: string = 'test d',
    e: string = 'test e';
let result: string;

result = a + b + c + d + e;
```

Рис. 1. Вхідна програма на мові TypeScript, яка демонструє конкатенацію рядків.

Після роботи лексичного та синтаксичного аналізатора отримуємо представлення поданої програми у вигляді дерева розбору (рисунок 2).

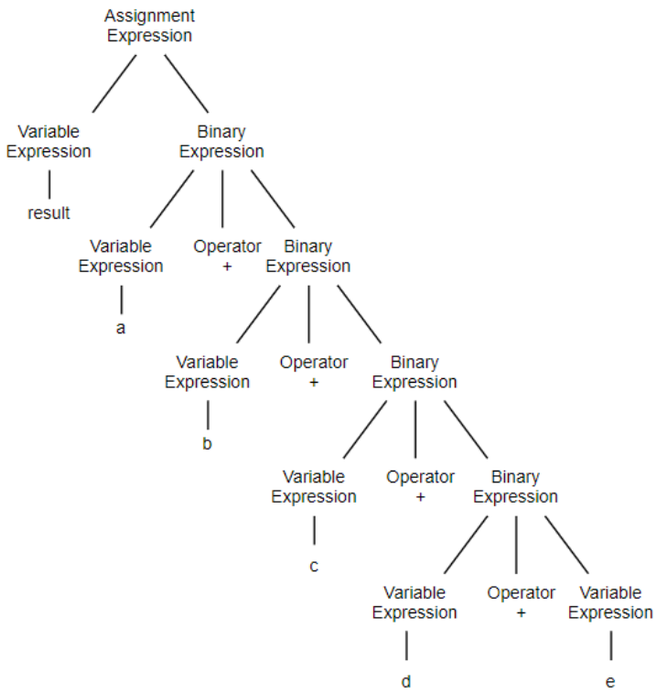

Рис. 2. Частина дерева розбору, що містить конкатенацію рядкових змінних.

Для конкатенації рядків платформа .NET має вбудовану функцію String.Concat [6], яка виконує конкатенацію переданих до неї рядкових аргументів. Звісно, можна згенерувати виклик цієї функції для кожного оператору додавання та передати як аргументи лівий та правий вираз, але така реалізація не буде оптимальною, оскільки доведеться зробити стільки викликів функції, скільки операторів додавання міститься у виразі. У документації зазначено [5], що функція має декілька варіантів виклику з перевантаженнями: для 2 аргументів, для 3 аргументів, для 4 аргументів та для масиву рядків. Тому згенерований код можна оптимізувати викликом потрібного варіанту перевантаження функції.

Запропонований спосіб трансляції конкатенації рядків полягає в наступному:

1. Перетворити у масив дерево бінарних виразів, у яких виконується конкатенація рядків.
2. Якщо розмір масиву менше ніж 5, то викликати потрібний варіант перевантаження функції для 2, 3 чи 4 аргументів та перейти до пункту 5, інакше перейти до пункту 3.
3. Згенерувати інструкції для створення и заповнення масиву рядковими виразами, які необхідно об'єднати.
4. Викликати потрібний варіант перевантаження функції для масиву аргументів.
5. Кінець

Для генерації виконуваних файлів для середовища .NET було використано бібліотеку Mono.Cecil [7], яка надає зручний інтерфейс для запису інструкцій CIL. Дана бібліотека розроблена співробітником компанії Microsoft Джейбі Ейваном та активно підтримується спільнотою.

Лістинг функції, яка відповідає цьому способу показано на рисунку 3.

Результат виконання цієї функції для обраного прикладу зображено на рисунку 4.

© Іваненко А.Р., Марченко О.І.

```
EmitStringConcat(ILProcessor il, BinaryExpression node)
{
    var nodes = Flatten(node);
    switch (nodes.Count)
    {
        case 0:
            il.Emit(OpCodes.Ldstr, string.Empty);
            break;
        case 1:
            EmitExpression(il, nodes[0]);
            break;
        case 2:
            EmitExpression(il, nodes[0]);
            EmitExpression(il, nodes[1]);
            il.Emit(OpCodes.Call, _stringConcat2);
            break;
        case 3:
            EmitExpression(il, nodes[0]);
            EmitExpression(il, nodes[1]);
            EmitExpression(il, nodes[2]);
            il.Emit(OpCodes.Call, _stringConcat3);
            break;
        case 4:
            EmitExpression(il, nodes[0]);
            EmitExpression(il, nodes[1]);
            EmitExpression(il, nodes[2]);
            EmitExpression(il, nodes[3]);
            il.Emit(OpCodes.Call, _stringConcat4);
            break;
        default:
            il.Emit(OpCodes.Ldc_I4, nodes.Count);
            il.Emit(OpCodes.Newarr, _stringTypeRef);
            for (var i = 0; i < nodes.Count; i++)
            {
                il.Emit(OpCodes.Dup);
                il.Emit(OpCodes.Ldc_I4, i);
                EmitExpression(il, nodes[i]);
                il.Emit(OpCodes.Stelem_Ref);
            }
            il.Emit(OpCodes.Call, _stringConcatArray);
            break;
    }
}
```

Рис. 3. Лістинг функції, яка генерує IL інструкції для конкатенації

```
IL_001f: ldc.i4.5
IL_0020: newarr       [System.Runtime]System.String
IL_0025: dup
IL_0026: ldc.i4.0
IL_0027: ldloc.0      // V_0
IL_0028: stelem.ref
IL_0029: dup
IL_002a: ldc.i4.1
IL_002b: ldloc.1      // V_1
IL_002c: stelem.ref
IL_002d: dup
IL_002e: ldc.i4.2
IL_002f: ldloc.2      // V_2
IL_0030: stelem.ref
IL_0031: dup
IL_0032: ldc.i4.3
IL_0033: ldloc.3      // V_3
IL_0034: stelem.ref
IL_0035: dup
IL_0036: ldc.i4.4
IL_0037: ldloc.s      V_4
IL_0039: stelem.ref
IL_003a: call         string [System.Runtime]System.String::Concat(string[])
IL_003f: dup
IL_0040: stloc.s      V_5
```

Рис. 4. Згенеровані інструкції для конкатенації рядкових змінних за запропонованим способом для програми, показаної на рис.1.

© Іваненко А.Р., Марченко О.І.

Аналіз отриманих результатів.

Для тестування було розроблено консольну програму на мові програмування C# (рисунок 5), яка запускала виконувані файли, що були скомпільовані компіляторами, які порівнюються. Для зменшення ефекту похибок при запуску програми та більшої наочності, виміри швидкості коду прикладів конкатенації, згенерованого різними компіляторами, виконувались в циклі на 1000000 ітерацій (рисунок 6).

Виміри проводилися для конкатенації 3, 4 та 5 рядкових виразів. Отримані результати показано на рисунку 7. Як видно з діаграми, код, що згенерований згідно запропонованого способу, працює швидше, ніж існуючий аналог.

```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        var sw = new Stopwatch();
        foreach (var arg in args)
        {
            using (Process myProcess = new Process())
            {
                myProcess.StartInfo.UseShellExecute = false;
                myProcess.StartInfo.FileName = arg;
                myProcess.StartInfo.CreateNoWindow = true;
                Console.WriteLine("Starting process: {0}", myProcess.StartInfo.FileName);
                myProcess.Start();
                sw.Start();
                myProcess.WaitForExit();
                sw.Stop();
                Console.WriteLine("Ms: {0}", sw.ElapsedMilliseconds);
            }
        }

    }
}
```

Рис. 5. Лістинг консольної програми, яка запускає виконуючі файли.

```
function main(): void {
    let a: string = 'test a', b: string = 'test b',
        c: string = 'test c', d: string = 'test d',
        e: string = 'test e';

    let result: string;
    for (let i = 0; i < 1000000; i = i + 1)
        result = a + b + c + d + e;
}
```

Рис. 6. Вхідна програма на мові TypeScript, яка тестувалась.

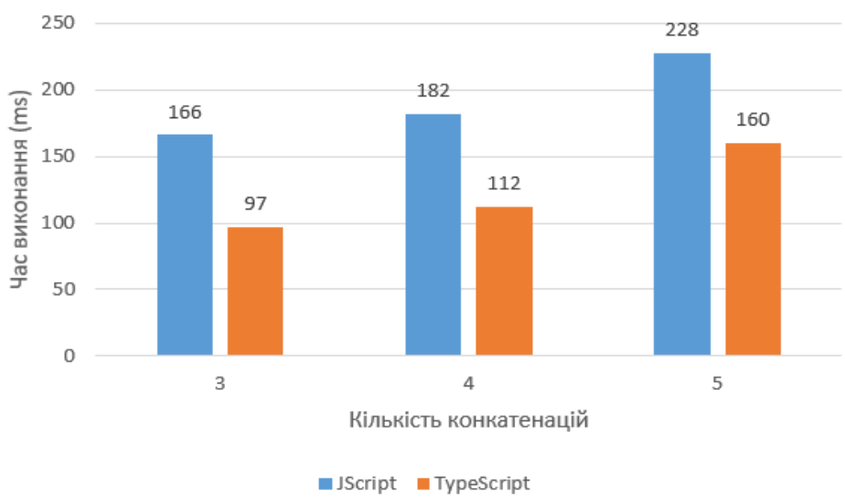

Рис. 7. Графік порівняння швидкодії згенерованих інструкцій для конкатенації компілятором JScript та компілятором TypeScript за запропонованим способом.

© Іваненко А.Р., Марченко О.І.

Висновки.

На першому етапі дослідження способів трансляції мови TypeScript у проміжну мову CIL платформи .NET особливу увагу було приділено роботі з рядковими типами даних, зокрема операції конкатенації рядків.

При розробці власного компілятора мови TypeScript було використано спосіб трансляції, який запропоновано у даній статті, який під час тестування та порівняльного аналізу продемонстрував кращу швидкодію згенерованого коду в порівнянні з існуючим аналогом – компілятором JScript.

Таким чином, враховуючи отримані результати, можна підкреслити висновок, зроблений у попередній статті [1], а саме, що створення прямого компілятора мови програмування TypeScript у проміжний код CIL платформи .NET є доцільним і має як наукову новизну, так і практичну цінність.

Список бібліографічного опису

1. Yannis Bres, Bernard Serpette, Manuel Serrano, Compiling Scheme programs to .NET Common Intermediate Language [Електроний ресурс] – Режим доступу: https://www.researchgate.net/publication/213885643_Compiling_Scheme_programs_to_NET_Common_Intermediate_Language.
2. YangSun Lee, SeungWon Na, DaeHoon Hwang, Language Translator for Execution Java programs in .NET [Електроний ресурс] – Режим доступу: https://www.koreascience.or.kr/article/JAKO200411922370411.pdf
3. Марченко О.І., Іваненко А.Р. Аналіз способів трансляції мови TypeScript у проміжну мову CIL платформи .NET
4. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд., СПб.:Питер, 2013
5. Марченко О.І. Конспект лекцій з дисципліни «Іженерія програмного забезпечення-1. Основи проектування трансляторів», Київ 2013.
6. Документація Microsoft [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/dotnet/api/system.string.concat
7. Документація Mono.Cecil [Електронний ресурс] – Режим доступу: https://cecil.pe/

References

1. Yannis Bres, Bernard Serpette, Manuel Serrano, Compiling Scheme programs to .NET Common Intermediate Language [Electronic resource] – Access mode: https://www.researchgate.net/publication/213885643_Compiling_Scheme_programs_to_NET_Common_Intermediate_Language.
2. YangSun Lee, SeungWon Na, DaeHoon Hwang, Language Translator for Execution Java programs in .NET [Electronic resource] – Access mode: https://www.koreascience.or.kr/article/JAKO200411922370411.pdf
3. Marchenko O.I., Ivanenko A.R. Analysis of TypeScript translation methods into Common Intermediate Language of .NET platform
4. Jeffrey Richter. CLR via C#. 2012
5. Marchenko O.I Synopsis of lectures on the subject "Software Engineering-1. Basics of translator design », Kyiv 2013.
6. Microsoft documentation [Electronic resource] – Access mode: https://docs.microsoft.com/en-us/dotnet/api/system.string.concat
7. Mono.Cecil documentation [Electronic resource] – Access mode: https://cecil.pe/

© Іваненко А.Р., Марченко О.І.