УДК 004.4'24 https://doi.org/10.15407/pp2019.02.011

Т.А. Мамедов, А.Ю. Дорошенко

ЗАСІБ НАЛАШТУВАННЯ ПРОГРАМ НА ПЛАТФОРМІ .NET ЗА ДОПОМОГОЮ ПЕРЕПИСУВАЛЬНИХ ПРАВИЛ

Розроблено програмний засіб для оптимізації обчислень, що дозволяє в автоматизованому режимі здійснити оптимізацію програми шляхом підвищення її швидкодії. Для цього реалізовано спеціальний плагін до системи переписувальних правил TermWare, за допомогою якого система здійснює налаштування програм, написаних на платформі .NET. Плагін використовує аналізатор Roslyn, реалізований генератор термів системи TermWare з вихідного коду програми. Програмний засіб проілюстровано на відомому прикладі клітинного автомату «Гра життя» на різних розмірах площини. Під час експериментів проведені виміри швидкодії програми до та після модифікації вихідного коду. Наведене порівняння результатів роботи методу самоналаштування за допомогою TermWare та інструментарію під назвою Eazfuscator.NET. Експерименти розробленого методу та бібліотеки Eazfuscator.NET проведені на персональному комп'ютері.

Ключові слова: клітинний автомат гра життя, автотюнінг, вимірювання швидкодії.

Вступ

Ручна оптимізація займає немалий час в циклі розробки програмних засобів, тому в останні роки розроблюються різні автоматизації цього процесу, зокрема, шляхом самоналаштування програм на цільову платформу. Концепція такого підходу під назвою автотюнінгу вже давно існує в світі інформаційних технологій і з кожним роком стає все більш важливою, оскільки обсяг коду програмних систем і/або кількість оброблюваної інформації зростає все більше і більше, а швидкодія програм стає більш гострою проблемою [1]. На сьогодні створено досить багато систем, які реалізують концепцію автотюнінгу. Серед них, для прикладу, можна назвати такі системи, як ATLAS [2], AbcLibScript [3], TuningGenie [4].

В даній роботі для цієї мети була використана система переписувальних правил TermWare [5], для якої був реалізований parser/printer для мови C#. Також був використаний інструментарій TuningGenie для вимірювання швидкодії програми на різних ділянках коду. Наведено порівняння різних підходів до концепції автотюнінгу, серед яких є і використання такого засобу автоматизації програм на мові C# як Eazfuscator.NET [6].

Матеріал даної роботи організований наступним чином. В розділі 1 описаний алгоритм, який був розроблений на мові C# для експерименту. В розділі 2 коротко описана реалізація плагіну TermWare для роботи з мовою C#. В третьому розділі описано засоби та реалізацію аналізатора мови C#. В останньому розділі описаний експеримент та результати оптимізації програми за допомогою переписувальних правил та інструментарію Eazfuscator.NET.

1. Алгоритм програми «Гра життя»

«Гра життя» являє собою нескінченну, двомірну ортогональну сітку квадратних клітин, кожна з яких знаходиться в одному з двох можливих станів, живих або мертвих (або заселених і ненаселених відповідно) [7]. Кожна клітина взаємодіє зі своїми вісьмома сусідами, які є клітинами, розташованими горизонтально, вертикально або по діагоналі. На кожному кроці відбуваються наступні переходи:

- будь-яка жива клітина з меншою, ніж двома живими сусідами вмирає, як ніби за ненаселення;
- будь-яка жива клітина з двома або трьома живими сусідами живе на наступному поколінні;
- будь-яка жива клітина з більш ніж трьома живими сусідами вмирає через перенаселення;

- будь-яка мертва клітина з трьома живими сусідами стає живою клітиною, за допомогою відтворення.

В реалізації програми для даної роботи на початку вказуються константи, які визначають розмір площини (сітки) та кількість поколінь. Переходячи на наступне покоління, всі правила застосовуються одночасно до всіх клітин на площині та відбувається дискретний момент «народження» або «смерті».

Псевдокод для визначення всіх сусідів для клітини.

```
BEGIN
NUMBER numOfAliveNeighbors=0, x, y, heigth, width;
ARRAY OF ARRAYS cells
FOR (i = x-1; i < x + 2; i++)
{
  FOR (j = y-1; j < y + 2; j++)
  {
    IF (!((i < 0 || j < 0) || (i >= heigth || j >= width)))
    {
      if (cells[i, j] == true)
      {
        numOfAliveNeighbors++;
      }
    }
  }
}
END
```

Псевдокод основної дії (перехід на наступне покоління) наведений далі.

```
BEGIN
NUMBER heigth, width;
ARRAY OF ARRAYS cells;
FOR (i = 0; i < heigth; i++)
{
  FOR (j = 0; j < width; j++)
  {
    numOfAliveNeighbors=CalcNeigh(i, j);
    IF (cells[i, j] == TRUE)
    {
      IF (numOfAliveNeighbors < 2 AND numOfAliveNeighbors > 3)
      {
        cells[i, j] = FALSE;
      }
    }
    ELSE
    {
      IF (numOfAliveNeighbors == 3)
      {
        cells[i, j] = TRUE;
      }
    }
  }
}
END
```

2. Система Termware

Для здійснення всіх перетворень використовується платформа переписувальних правил Termware, докладно описана в [5]. Termware призначена для опису перетворення над термами, тобто виразами виду $f\ (t_1 \ldots t_n)$. Для завдання перетворень використовуються правила Termware, тобто конструкції виду *source [condition]* –> *destination [action]*.

Тут *source* – вихідний терм (зразок для пошуку), *condition* – умова застосування правила, *destination* – перетворений терм, *action* – додаткове дія при спрацьовуванні правила. Кожен з 4 компонентів правила може містити змінні (які записуються у вигляді $ var), що забезпечує спільність правил. Компоненти *condition* і *action* є необов'язковими. Вони можуть виконувати довільний процедурний код, зокрема використовувати додаткові дані про програму.

Застосування правила відбувається наступним чином: спочатку знаходиться підтерм вхідного терма (дерева програми), який підходить під *source*. Далі перевіряється умова застосування (якщо вона є). Якщо умова виконується, відбувається заміна *source* на *destination*. При цьому змінні в *destination* замінюються відповідними значеннями з *source*. Також виконується дія *action* (якщо воно було присутнє).

Кожне перетворення задається системою правил, тобто набором правил, які послідовно застосовуються до даного тер-

му (дереву програми). Порядок застосування правил визначається стратегією. У систему Termware вбудовані кілька основних стратегій, таких як TopDown, BottomUp, FirstTop. Крім того, можливе створення додаткових стратегій.

3. Реалізація плагіну для TermWare для роботи з мовою C#

3.1. Аналізатори мови C#. Існує досить багато аналізаторів для мови C#, такі як Metaspec C# parser library та інші, описані в роботі [6]. Але проблематика аналізаторів мови C# полягає у тому, що мова має вже вісім версій. У 2015 році вийшла платформа Roslyn [8] з відкритим вихідним кодом, що розробляється корпорацією Microsoft, і містить у собі компілятори і засоби для розбору і аналізу коду, написаного на мовах програмування C # і Visual Basic.

Різні нововведення на зразок code fixes реалізуються саме за рахунок використання Roslyn.

За допомогою засобів аналізу, наданими платформою Roslyn можна проводити повний розбір коду, аналізуючи всі підтримувані конструкції мови.

Середовище Visual Studio дозволяє створювати на основі Roslyn як вбудовані в саму IDE інструменти (розширення Visual Studio), так і незалежні додатки (standalone інструменти).

3.2. Реалізація. В даній роботі був використаний фреймворк TermWare, в якому розроблений концепт термінальних систем об'єктних структур та переписувальних правил. TermWare написаний на мові Java, але використовує окремо плагін Parser/Printer. Тому незважаючи на те, що фреймворк написаний на одній технології, він надає можливість мультиплатформності. Для цього був створений екземпляр TermWareInstance – це, образно говорячи, стрижень, який містить у собі ієрархічну систему доменів і словники імен парсерів мов та принтерів. Для роботи з будь-яким іншим стеком технологій треба реалізувати 2 інтерфейси: IParser, IParserFactory.

Загальний паттерн використання цих інтерфейсів наступний:

- програміст визначає парсер мови, відповідний інтерфейсу IParser і фабрику IParserFactory;
- перед використанням розбирача цієї мови повинна бути зареєстрована своя фабрика парсерів для відповідної мови, використовуючи метод TermWareInstance. addParserFactory (String language, IParserFactory factory).

Після цього виклик loadFile(fname, X) буде використовувати парсер відповідної мови X.

Для синтаксичного аналізу програм було використано можливості відкритого програмного забезпечення – компілятора Roslyn. За допомогою компілятора можна отримати синтаксичне дерево реалізованої програми, яке і потрібно для парсеру.

Оскільки Roslyn працює на CLR, а парсер/принтер повинен працювати на JVM, для розв'язування цієї проблеми був використаний міст jni4net, який дозволяє реєструвати будь-який dll файл, згенерований за допомогою інструменту proxygen, який створює обгортку до будь-якої бібліотеки [9]. Це дозволяє використовувати та створювати екземпляри класів з C# на Java та навпаки з Java на C#.

Також наведена спрощена UML-діаграма класів відповідного парсеру на рис. 1.

Далі буде наведений приклад терму для методу NextGeneration().

```
For(Assignment(i,0), i<=Heigth, Increment(i),
For(Assignment(j,0), i<=Width, Increment(j),
[DeclarationAssignment(numOfAliveNeighb
ors, int, [MethodCall(CalcNeighbor, i, j)],
If(ArrayElement(ArrayElement(cells, i), j), [
        If (numOfAliveNeighbors<2, [
Assignment(ArrayElement(ArrayElement(cell
s, i), j), false)])
        If (numOfAliveNeighbors>3, [
Assignment(ArrayElement(ArrayElement(cell
s, i), j), false)])])
        If(NotEqual(ArrayElement(ArrayEle
ment(cells, i), j)), true),
        [If (Equal(numOfAliveNeighbors, 3)),
[Assignment(ArrayElement(ArrayElement(ce
lls, i), j), true)
]))])])]
```

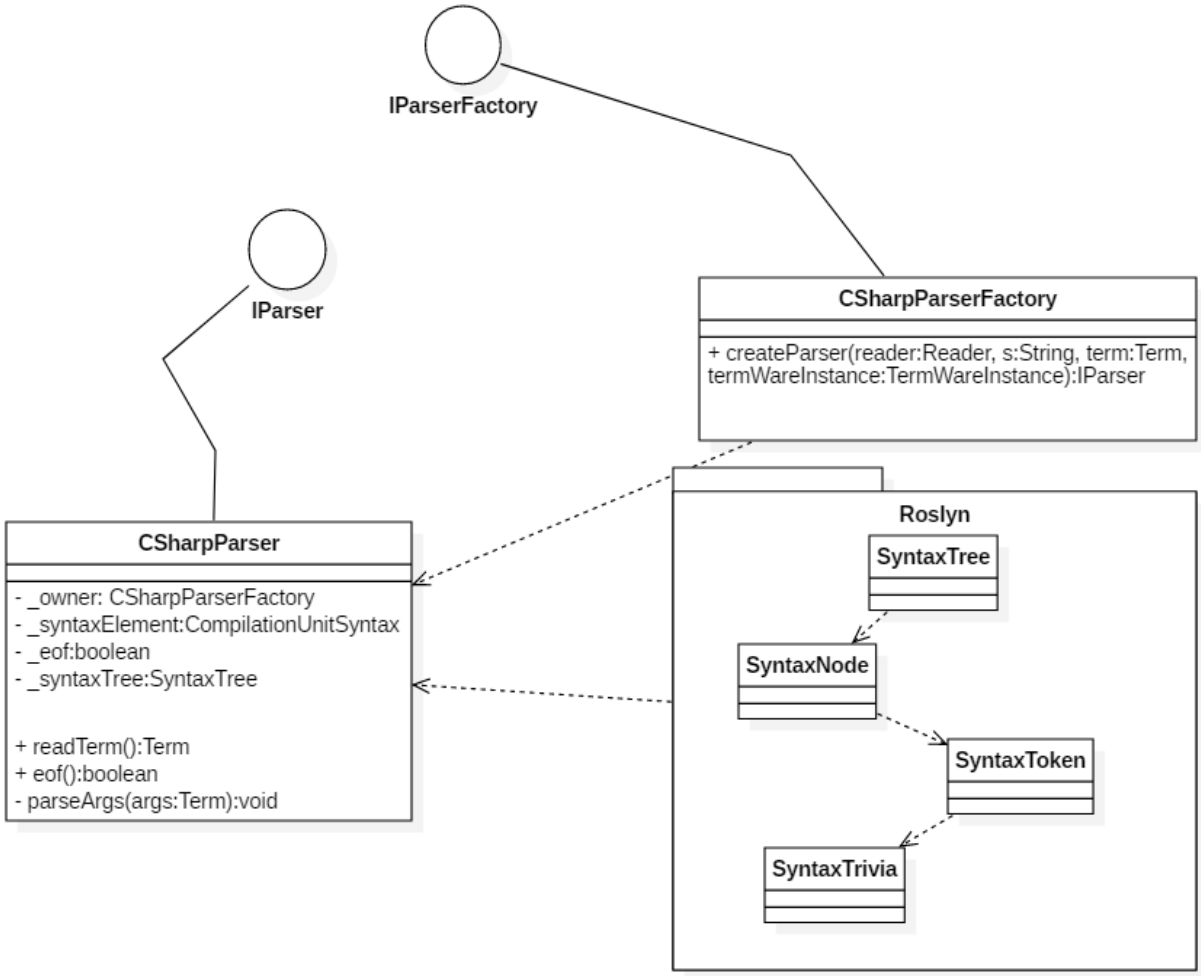

Рис. 1. Спрощена діаграма класів плагіну для TermWare

Наступний код – це приклад терму для методу визначення сусідів клітини CalcNeighbor(x, y).

```
[DeclarationAssignment(numOfAliveNeighb
ors, int, 0),
For(Assignment(i, x – 1), i < x + 2,
Increment(i),
For(Assignment(j, y – 1), j < y + 2,
Increment(j),
    If(Not(Or(Or(i<0,j<0),Or(i>=Heigth,j
>=Width))),
[If(ArrayElement(ArrayElement(cells, i), j),
[Increment(numOfAliveNeighbors)]
)])]
```

4. Експеримент

Для визначення ефективності програмних перетворень проведено експеримент із застосуванням двох видів автоматичної оптимізації програми.

Експеримент складався з порівняння результатів виконання програм, отриманих за допомогою описаних систем.

Експеримент проводився з використанням комп’ютера, оснащеної процесором Intel Core i7-7700HQ 2.80GHz з обсягом оперативної пам'яті в 16 Гб.

Масштабування дослідної задачі регулюється за допомогою вибору розміру площини, на якому здійснюється «Гра життя», а також кількість поколінь, які повинні пройти перед закінченням виконання програми.

Експеримент проводився на різних розмірах площини, починаючи від 300 і закінчуючи розмірами в 1200000 клітин.

Результат експериментів виводиться на Excel файл та показаний на рис. 2, де можна побачити, що застосування системи TermWare є більш ефективним з точки зору швидкодії в порівнянні з бібліотекою Eazfuscator.NET.

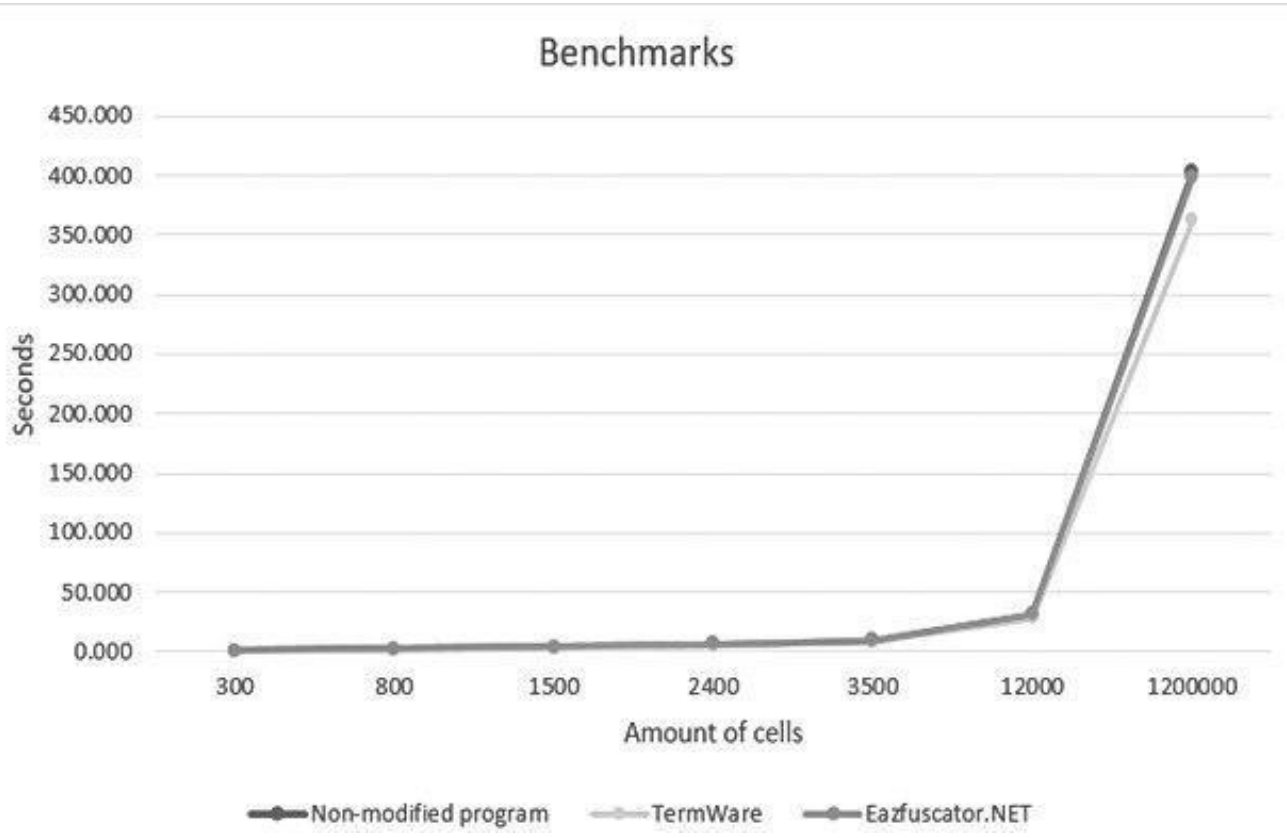

Рис. 2. Результат тестів програми

Як видно з графіка, система TermWare випереджає Eazfuscator.NET, коефіцієнт прискорення порівняно з немодифікованою програмою і TermWare складає приблизно на 1.34, а з бібліотекою Eazfuscator.NET – на 1.15.

З іншого боку, якщо порівнювати довжину коду, то в цьому випадку Eazfuscator.NET не потребує ніяких модифікацій коду для застосування, тому в цьому випадку ця бібліотека краща за використання TermWare.

Висновок

Розроблено та реалізовано плагін системи переписувальних правил TermWare для роботи з мовою C#, що дозволяє оптимізувати програми, написані цією мовою, в атоматизованому режимі у напрямку покращення їх швидкодії. Проведено порівняння з фреймворком Eazfuscator.NET на відомому академічному прикладі клітинного автомата «Гра життя».

Література

1. Naono K., Teranishi K., Cavazos J., Suda R. (2010), Software Automatic Tuning From Concepts to State-of-the-Art Results. – Berlin: Springer.
2. R Clinton Whaley, Jack J Dongarra, ATLAS. Encyclopedia of Parallel Computing. 2011. С. 95–101.
3. Katagiri T., Kise K., Honda H., Yuba T., AbcLibScript: A directive to support specification of an auto-tuning facility for numerical software. Parallel Computing. 2006. С. 92–112.
4. Pavlo A. Ivanenko, Anatoliy Y. Doroshenko, and Kostiantyn A. Zhereb, TuningGenie: Auto-Tuning Framework Based on Rewriting Rules // in: 10th International Conference, ICTERI 2014, Kherson, Ukraine, June 9-12, 2014, Revised Selected Papers, Series: Communications in Computer and Information Science, (Ermolayev, V., Mayr, H.C., Nikitchenko, M., Spivakovsky, A., Zholtkevych, G. (Eds.)), Springer. CCIS Vol. 469. 2014. P. 139–160.
5. TermWare [Електронний ресурс]. Режим доступу до ресурсу: http://www.gradsoft.ua/products/termware_rus.html
6. Eazfuscator.NET [Online]. Available from: https://www.gapotchenko.com/eazfuscator.net

7. Doroshenko A., Shevchenko R. A rewriting framework for rule-based programming dynamic applications. Fundamenta Informaticae. 2006. Vol. 72, N 1–3. P. 95–108.
8. Roslyn [Електронний ресурс]. Режим доступу до ресурсу: https://github.com/dotnet/roslyn
9. Жереб К.А., Программный инструментарий, основанный на правилах для автоматизации разработки приложений на платформе Microsoft .NET. *Управляющие системы и машины.* 2009, №4. С. 51–59.
10. JNI4NET [Електронний ресурс]. – Режим доступу до ресурсу: http://jni4net.com/
11. Application of Rewriting Term System for Source Code Analysis [Електронний ресурс]. Режим доступу до ресурсу: http://www.gradsoft.ua/eng/whitepapers/secr2008/secr2008-1.html

References

1. Naono, K., Teranishi, K., Cavazos, J., Suda, R. (2010), Software Automatic Tuning From Concepts to State-of-the-Art Results. – Berlin: Springer.
2. R Clinton Whaley, Jack J Dongarra, ATLAS. Encyclopedia of Parallel Computing. 2011. P. 95–101.
3. Katagiri T., Kise K., Honda H., Yuba T., AbcLibScript: A directive to support specification of an auto-tuning facility for numerical software. Parallel Computing. 2006. P. 92–112.
4. Pavlo A. Ivanenko, Anatoliy Y. Doroshenko, and Kostiantyn A. Zhereb, TuningGenie: Auto-Tuning Framework Based on Rewriting Rules // in: 10th International Conference, ICTERI 2014, Kherson, Ukraine, June 9-12, 2014, Revised Selected Papers, Series: Communications in Computer and Information Science, (Ermolayev, V., Mayr, H.C., Nikitchenko, M., Spivakovsky, A., Zholtkevych, G. (Eds.)), Springer, CCIS Vol. 469, 2014. P. 139–160.
5. TermWare [Online]. Available from: http://www.gradsoft.ua/products/termware_rus.html
6. Eazfuscator.NET [Електронний ресурс]. – Режим доступу до ресурсу: https://www.gapotchenko.com/eazfuscator.net
7. Doroshenko A., Shevchenko R. A rewriting framework for rule-based programming dynamic applications. Fundamenta Informaticae. – 2006. – Vol. 72, No. 1–3. – P. 95–108.
8. Roslyn [Online]. Available from: https://github.com/dotnet/roslyn
9. Zhereb K., Программный инструментарий, основанный на правилах для автоматизации разработки приложений на платформе Microsoft .NET // Control systems and computers.(4). P. 51–59. (in Russian).
10. JNI4NET [Online]. Available from: http://jni4net.com/
11. Application of Rewriting Term System for Source Code Analysis [Online]. Available from: http://www.gradsoft.ua/eng/whitepapers/secr2008/secr2008-1.html

Одержано 03.05.2019

Про авторів:

Мамедов Турал Алірзайович,
студент 2 курсу магістратури в Київському національному університеті імені Тараса Шевченка.
Кількість наукових публікацій в українських виданнях – 2.
https://orcid.org/0000-0003-3029-5834,

Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук,
професор, завідувач відділу теорії комп'ютерних обчислень,
професор кафедри автоматики та управління в технічних системах
НТУУ "КПІ імені Ігоря Сікорського".
Кількість наукових публікацій в українських виданнях – понад 150.
Кількість наукових публікацій в зарубіжних виданнях – понад 50.
Індекс Хірша – 5.
http://orcid.org/0000-0002-8435-1451

Місце роботи авторів:

Київський національний університет імені Тараса Шевченка, кафедра теорії та технології програмування.

Національний технічний університет України "КПІ імені Ігоря Сікорського" та Інститут програмних систем
НАН України,
03187, м. Київ-187,
проспект Академіка Глушкова, 40.
Тел.: (044) 526 3559.
E-mail: tural.mamedov1@gmail.com,
doroshenkoanatoliy2@gmail.com