Применение кодогенерации для оптимизации

Игорь Чевдарь

Применение кодогенерации для оптимизации

https://github.com/homuroll/CodeGenPerfTests


E3-E2

Оптимизации

- Выбрать оптимальный алгоритм
- Эффективные структуры данных
- GC и memory traffic
- Низкоуровневые оптимизации
  - Кэш процессора
  - Branch prediction
  - Instruction level parallelism
- Кодогенерация

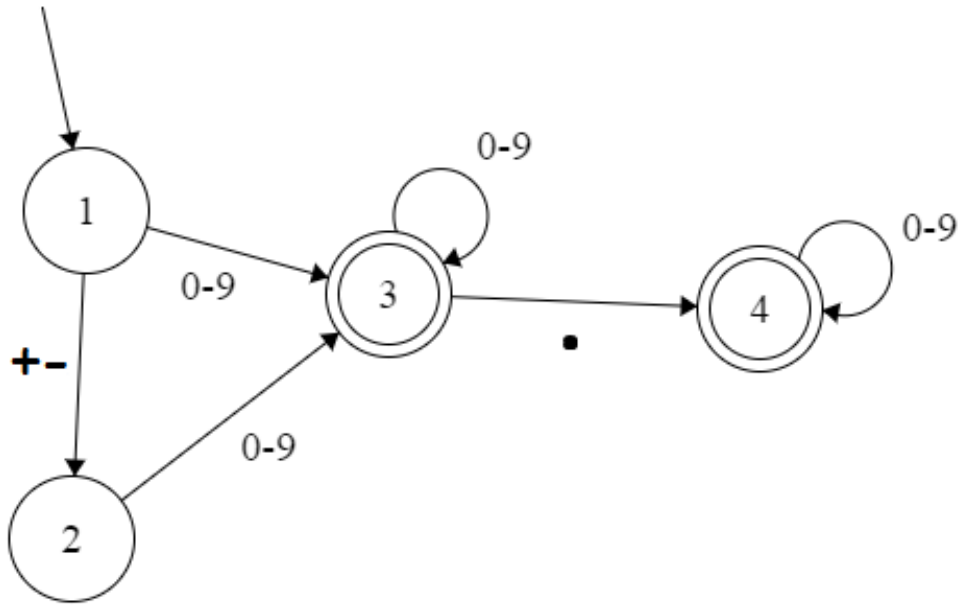
План доклада

- Регулярные выражения
- Валидация xml
- Бинарная сериализация
- Expression’ы
- Вывод

Задача 1. Регулярные выражения

Алгоритм по регулярному выражению строит какую-то автоматоподобную структуру

```
[\-+]?[0-9]+(\.[0-9]*)?
```



Принимает	Отвергает
7 +0. -12 +00.15	+ 0z --0.0 .0

```
public class Node
{
    public Dictionary<char, Node> jumps;
    public bool accepted;
    …
}

bool Match(string str)
{
    Node current = start;
    for (int i = 0; i < str.Length; i++)
    {
        Node next;
        if (!current.jumps.TryGetValue(str[i], out next))
            return false;
        current = next;
    }
    return current.accepted;
}
```

```
public abstract class Node
{
    public bool accepted;
    public abstract Node Jump(char c);
    …
}

public class NodeImpl : Node
{
    private Dictionary<char, Node> jumps;

    public override Node Jump(char c)
    {
        Node result;
        return jumps.TryGetValue(c, out result)
                    ? result
                    : null;
    }
}
```

```
bool Match(string str)
{
    Node current = start;
    for (int i = 0; i < str.Length; i++)
    {
        current = current.Jump(str[i]);
        if (current == null)
            return false;
    }
    return current.accepted;
}
```

Оператор switch по char’ам

- Dictionary
- Несколько if’ов
- Бинарный поиск

Размотанный бинарный поиск

```
if (key == '9') return 4;
if (key > '9')
{
    if (key == '&') return 6;
    if (key > '&')
    {
        if (key == '+') return 7;
        if (key == '-') return 8;
    }
    else if (key == '%') return 5;
}
```

```
else
{
    if (key == 'f') return 1;
    if (key > 'f')
    {
        if (key == '0') return 2;
        if (key == 'z') return 3;
    }
    else if (key == 'a') return 0;
}
```

Оператор switch по char’ам

- Dictionary
- Несколько if’ов
- Бинарный поиск
- Массив
- Несколько небольших отрезков (C#)

Несколько небольших отрезков

- {3, 5, 7, 10, 105, 106, 110, 1001, 1002}
- 3 отрезка: {3..10}, {105..110}, {1001..1002}

Оператор switch по char’ам

- Dictionary
- Несколько if’ов
- Бинарный поиск
- Массив
- Несколько небольших отрезков (C#)
- PerfectHashtable

Perfect hashtable

- Пример:
  1. Находим натуральное число n, такое, что все значения {key % n} различны
  2. Создаем массив arr длины n и записываем arr[key % n] = key
  3. Проверка, что произвольный элемент x лежит в таблице: arr[x % n] == x

- http://cmph.sourceforge.net/papers/chm92.pdf

  An optimal algorithm for generating minimal perfect hash functions (Zbignew Czech, George Havas, Bohdan Majewski)

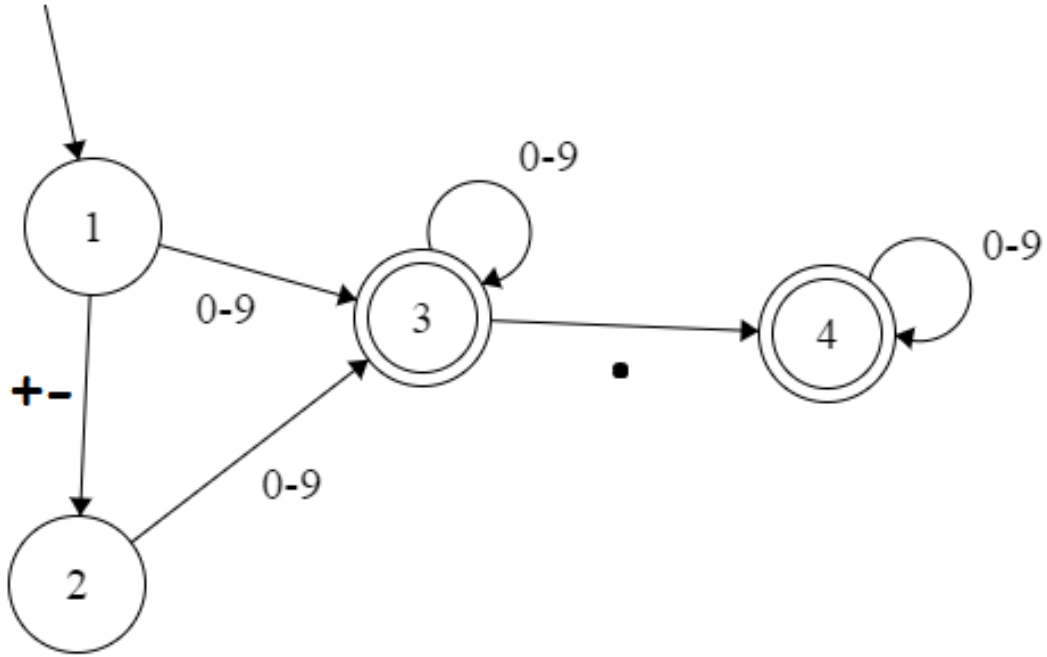
Оператор switch по char’ам

1. Пытаемся разбить множество ключей на небольшое (до 20) количество коротких отрезков
2. Если не вышло, то
   1) Если k <= 50 (150 для Mono), то делаем бинпоиск
   2) Иначе – совершенная хэштаблица

```
bool Match(string str)
{
    Node current = start;
    for (int i = 0; i < str.Length; i++)
    {
        current = current.Jump(str[i]);
        if (current == null)
            return false;
    }
    return current.accepted;
}
```

```
bool Match(string str)
{
    Node current = start;
    for (int i = 0; i < str.Length; i++)
    {
        if (current == s1)
        {
            // Реализация метода Jump для s1
        }
        else if (current == s2)
        {
            // Реализация метода Jump для s2
        }
        …
        else if (current == sn)
        {
            // Реализация метода Jump для sn
        }
    }
    return current.accepted;
}
```

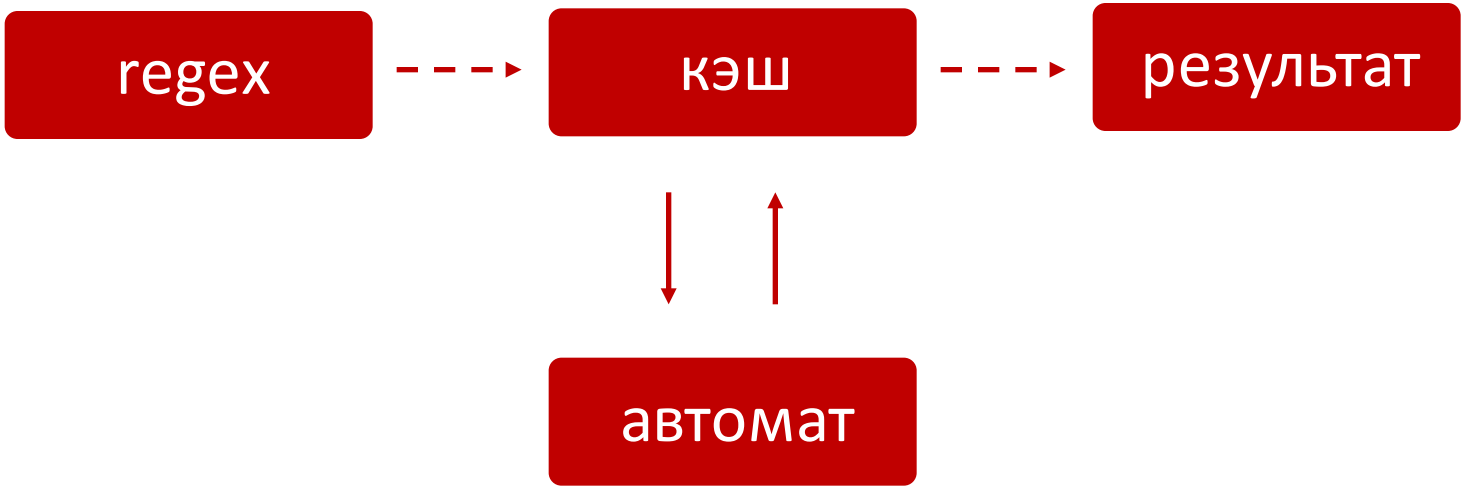
```
[\-+]?[0-9]+(\.[0-9]*)?
```



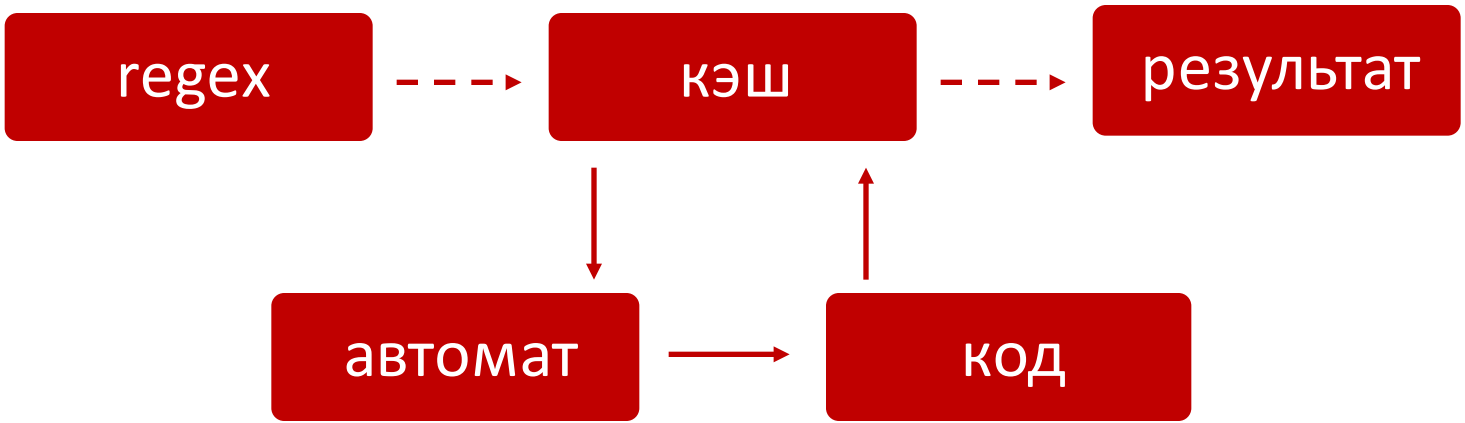
```
bool Match(string str)
{
      int idx = 0;
      int len = str.Length;
      char c;

_1:   if (idx >= len)
          return false;
      c = str[idx++];
      if (c == '+' || c == '-')
          goto _2;
      if (c >= '0' && c <= '9')
          goto _3;
      return false;

_2:   if (idx >= len)
          return false;
      c = str[idx++];
      if (c >= '0' && c <= '9')
          goto _3;
      return false;
}
```

```
_3:   if (idx >= len)
          return true;
      c = str[idx++];
      if (c >= '0' && c <= '9')
          goto _3;
      if (c == '.')
          goto _4;
      return false;

_4:   if (idx >= len)
          return true;
      c = str[idx++];
      if (c >= '0' && c <= '9')
          goto _4;
      return false;
```

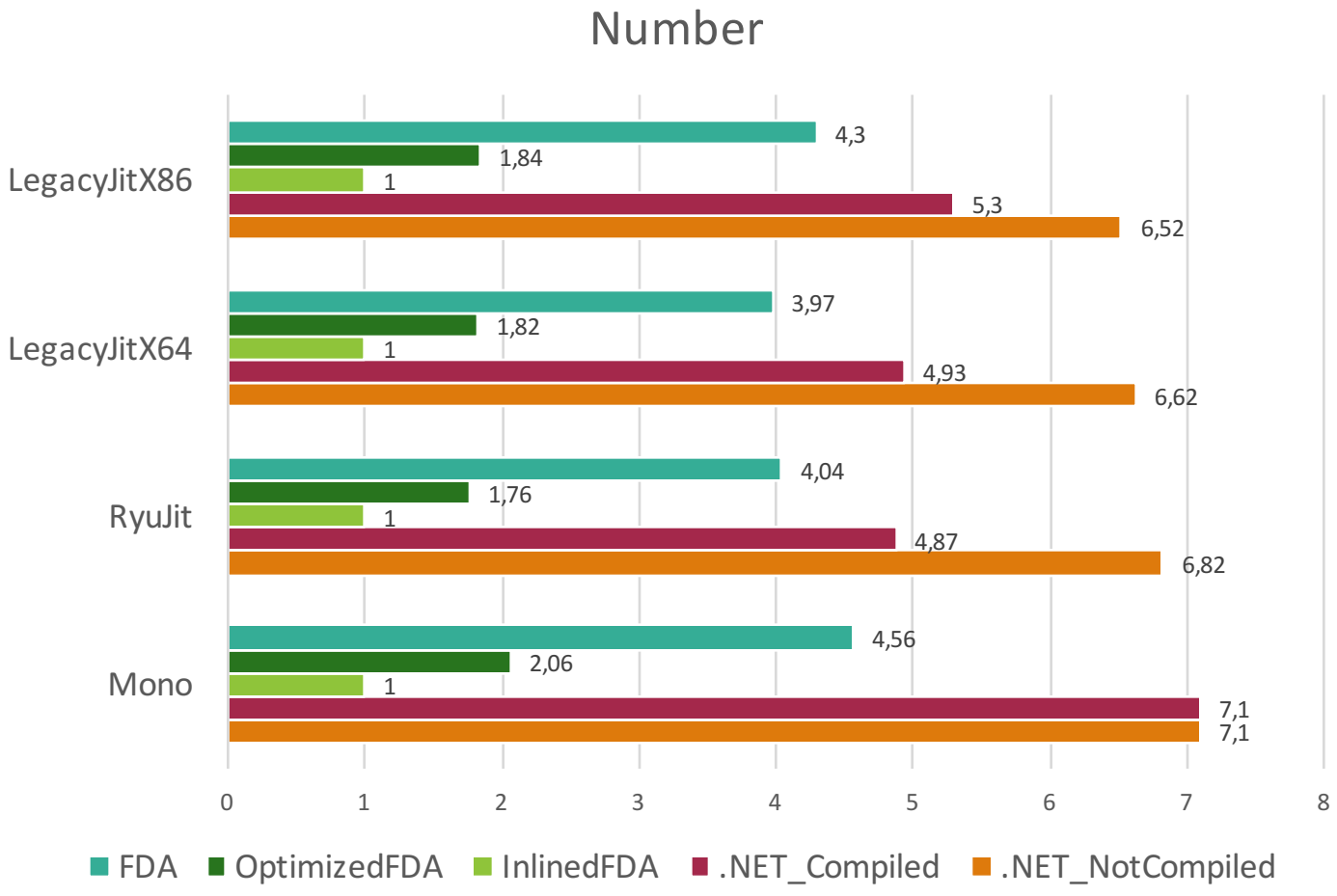
Без кодогенерации

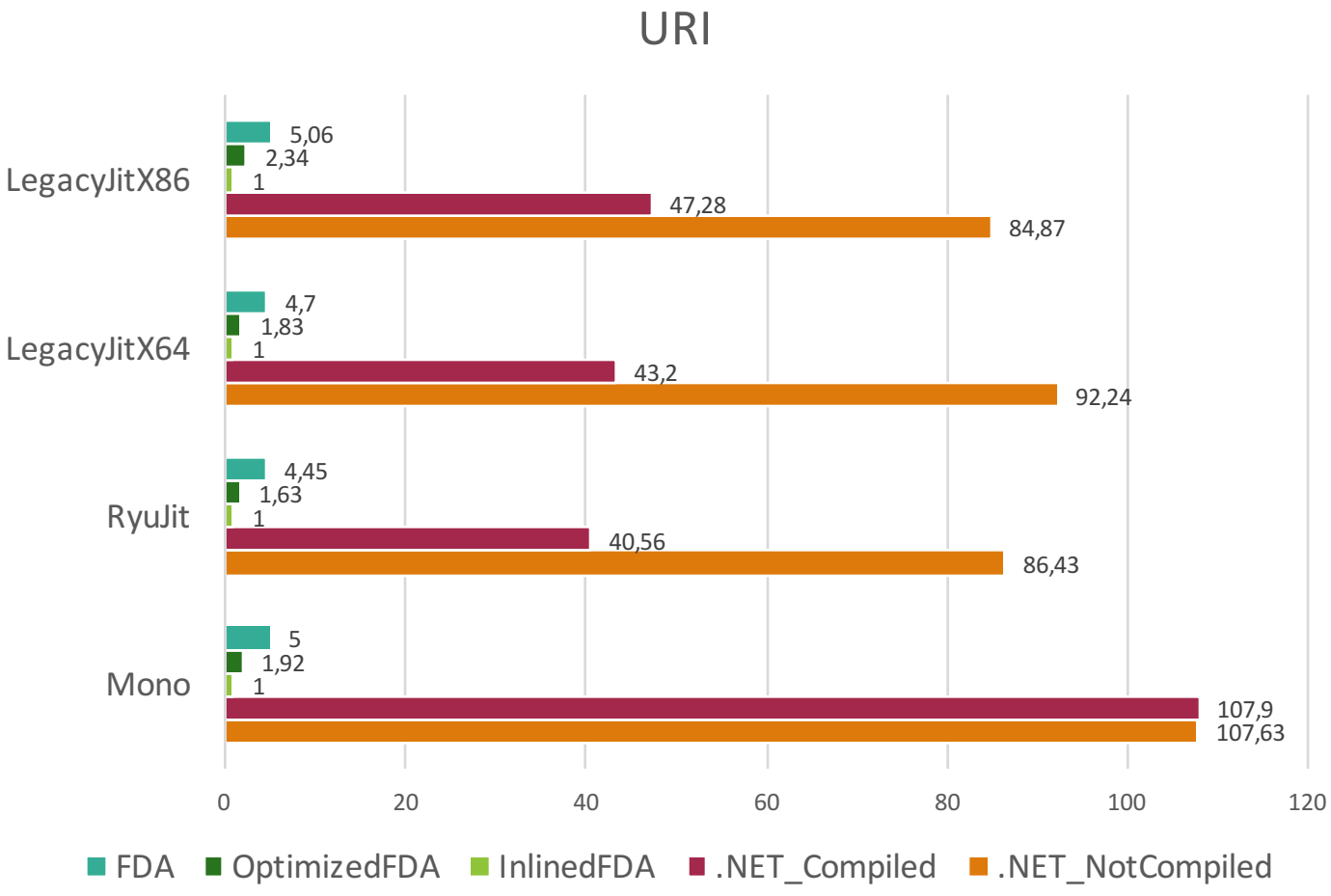


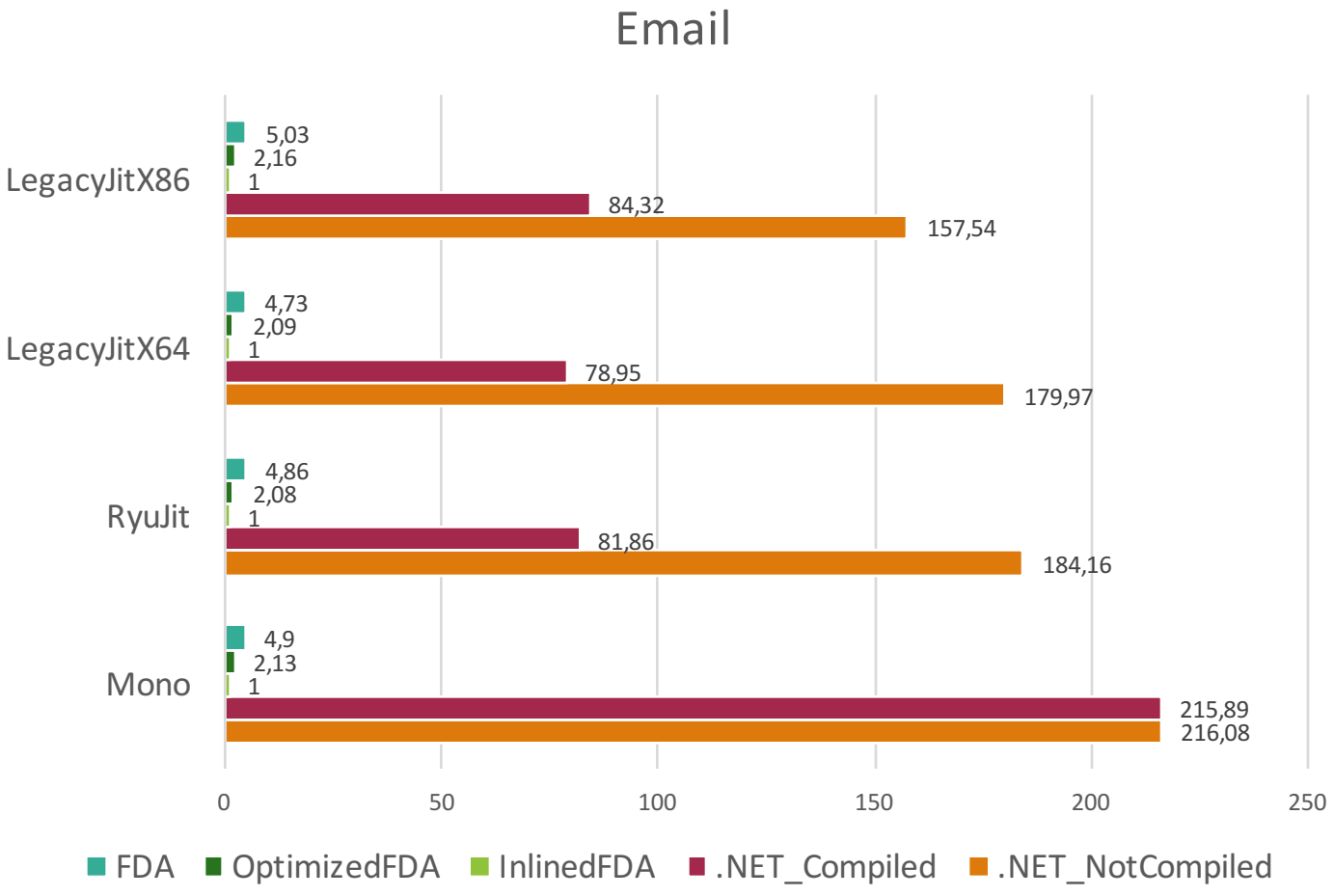
С кодогенерацией



Например, IL-код с помощью Reflection.Emit
https://github.com/homuroll/GrEmit
https://www.nuget.org/packages/GrEmit


Number
LegacyJitX86
4,3
1,84
1
5,3
6,52
LegacyJitX64
3,97
1,82
1
4,93
6,62
RyuJit
4,04
1,76
1
4,87
6,82
Mono
4,56
2,06
1
7,1
7,1
0
1
2
3
4
5
6
7
8
FDA
OptimizedFDA
InlinedFDA
.NET_Compiled
.NET_NotCompiled


URI
LegacyJitX86
5,06
2,34
1
47,28
84,87
LegacyJitX64
4,7
1,83
1
43,2
92,24
RyuJit
4,45
1,63
1
40,56
86,43
Mono
5
1,92
1
107,9
107,63
0
20
40
60
80
100
120
FDA
OptimizedFDA
InlinedFDA
.NET_Compiled
.NET_NotCompiled


Email
LegacyJitX86
5,03
2,16
1
84,32
157,54
LegacyJitX64
4,73
2,09
1
78,95
179,97
RyuJit
4,86
2,08
1
81,86
184,16
Mono
4,9
2,13
1
215,89
216,08
0
50
100
150
200
250
FDA
OptimizedFDA
InlinedFDA
.NET_Compiled
.NET_NotCompiled

Из-за чего regex в .NET медленнее?

- Более сложная структура данных, чем автомат
- Вызов общей медленной функции `RegexRunner.CharInClass`

Задача 2. Валидация xml с помощью xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="Квартира">
        <xs:sequence minOccurs="1" maxOccurs="5">
            <xs:choice minOccurs="1" maxOccurs="4">
                <xs:element name="Дверь"/>
                <xs:element name="Окно"/>
            </xs:choice>
            <xs:element name="Площадь"/>
        </xs:sequence>
    </xs:element>
</xs:schema>
```

Валидации по схеме

По сути у нас язык с операциями

- `<a/>` (element)
- AB (sequence)
- A|B (choice)
- A* (maxOccurs = "unbounded")
- A{k, l} (minOccurs = "k", maxOccurs = "l")

Схема

```
<element name="Квартира">
    <sequence minOccurs="1" maxOccurs="5">
        <choice minOccurs="1" maxOccurs="4">
            <element name="Дверь"/>
            <element name="Окно"/>
        </choice>
        <element name="Площадь"/>
    </sequence>
</element>
```

Регулярное выражение

```
R((a|b){1,5}c){1,4}r

R = <Квартира>
r = </Квартира>
a = <Дверь/>
b = <Окно/>
c = <Площадь/>
```

Оператор switch по string’ам

- Dictionary (старый C#)
- Несколько if’ов
- Бинпоиск по хэшкодам (Roslyn)
- Бор
- PerfectHashtable

Быстрая хэш-функция

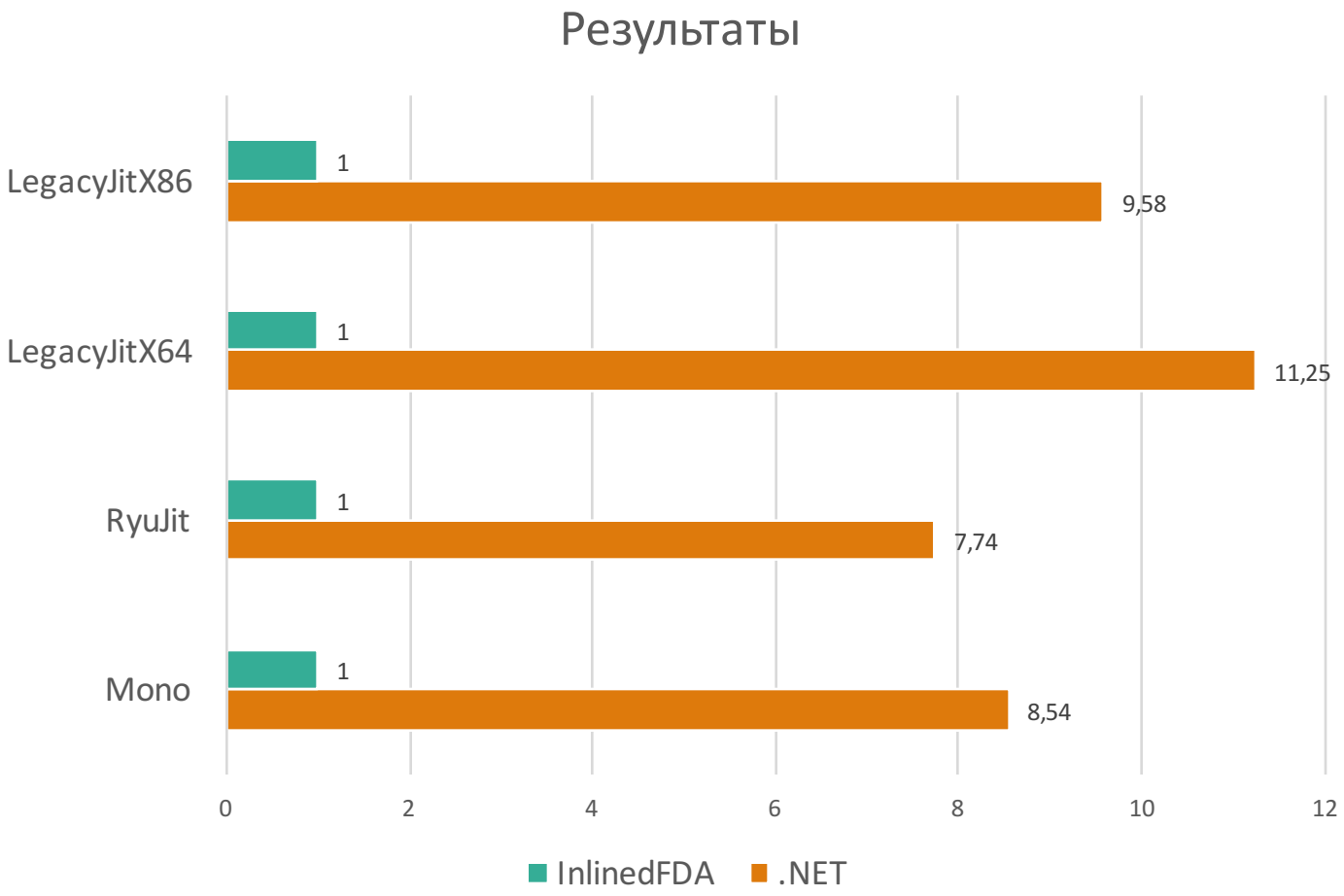
1. Ищем k: {key[k]} различны
2. Если нет, то ищем (k, l): {key[k], key[l]} различны
3. Если нет, ищем тройку

Не вышло, тогда можно так (Roslyn):

```
int res = 5478361;
for (int i = 0; i < s.Length; i++)
   res = (res ^ s[i]) * 1237681;
```

Оператор switch по string’ам

- Если количество ключей до 7 (40 для Mono), то строим бор
- Иначе – совершенная хэш-таблица


Результаты
LegacyJitX86
LegacyJitX64
RyuJit
Mono
1
9,58
1
11,25
1
7,74
1
8,54
0
2
4
6
8
10
12
InlinedFDA
.NET

Задача 3. Бинарная сериализация

- Нужно сохранить контракт данных в массив байт и наоборот
- Например, это делает сериализатор ProtoBuf

```
public unsafe interface IWriter
{
    void Write(object obj, byte* data, ref int idx);
}

public unsafe interface IReader
{
    object Read(byte* data, ref int idx);
}
```

```
public unsafe class Int32Writer : IWriter
{
    public void Write(object obj, byte* data, ref int idx)
    {
        *(int*)(data + idx) = (int)obj;
        idx += 4;
    }
}

public unsafe class Int32Reader : IReader
{
    public object Read(byte* data, ref int idx)
    {
        var result = *(int*)(data + idx);
        idx += 4;
        return result;
    }
}
```

unbox

box

```
public unsafe class ClassWriter : IWriter
{
    // Инициализируем в конструкторе
    private readonly FieldInfo[] fields;
    private readonly IWriter[] fieldWriters;

    public void Write(object obj, byte* data, ref int idx)
    {
        for (int i = 0; i < fields.Length; i++)
        {
            var fieldValue = fields[i].GetValue(obj);
            fieldWriters[i].Write(fieldValue, data, ref idx);
        }
    }
}
```

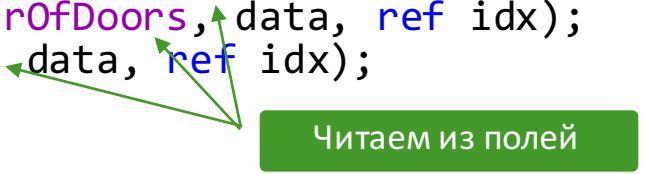
```
public unsafe class ClassReader : IReader
{
    // Инициализируем в конструкторе
    private readonly Type type;
    private readonly FieldInfo[] fields;
    private readonly IReader[] fieldReaders;

    public object Read(byte* data, ref int idx)
    {
        var result = Activator.CreateInstance(type);
        for (int i = 0; i < fields.Length; i++)
        {
            var fieldValue = fieldReaders[i].Read(data, ref idx);
            fields[i].SetValue(result, fieldValue);
        }
        return result;
    }
}
```

Reflection

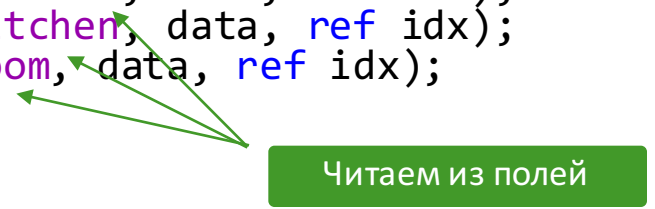
Reflection

```
public class Flat
{
    public int Number;
    public Room Kitchen;
    public Room Room;
}

public class Room
{
    public int NumberOfWindows;
    public int NumberOfDoors;
    public int Area;
}
```

```
public static unsafe class RoomWriter
{
    public static void Write(Room room, byte* data, ref int idx)
    {
        Int32Writer.Write(room.NumberOfWindows, data, ref idx);
        Int32Writer.Write(room.NumberOfDoors, data, ref idx);
        Int32Writer.Write(room.Area, data, ref idx);
    }
}
```




```
public static unsafe class FlatWriter
{
    public static void Write(Flat flat, byte* data, ref int idx)
    {
        Int32Writer.Write(flat.Number, data, ref idx);
        RoomWriter.Write(flat.Kitchen, data, ref idx);
        RoomWriter.Write(flat.Room, data, ref idx);
    }
}
```



```
public static unsafe class RoomReader
{
    public static Room Read(byte* data, ref int idx)
    {
        var room = new Room();
        room.NumberOfWindows = Int32Reader.Read(data, ref idx);
        room.NumberOfDoors = Int32Reader.Read(data, ref idx);
        room.Area = Int32Reader.Read(data, ref idx);
        return room;
    }
}
```

Вызов конструктора

Пишем в поля

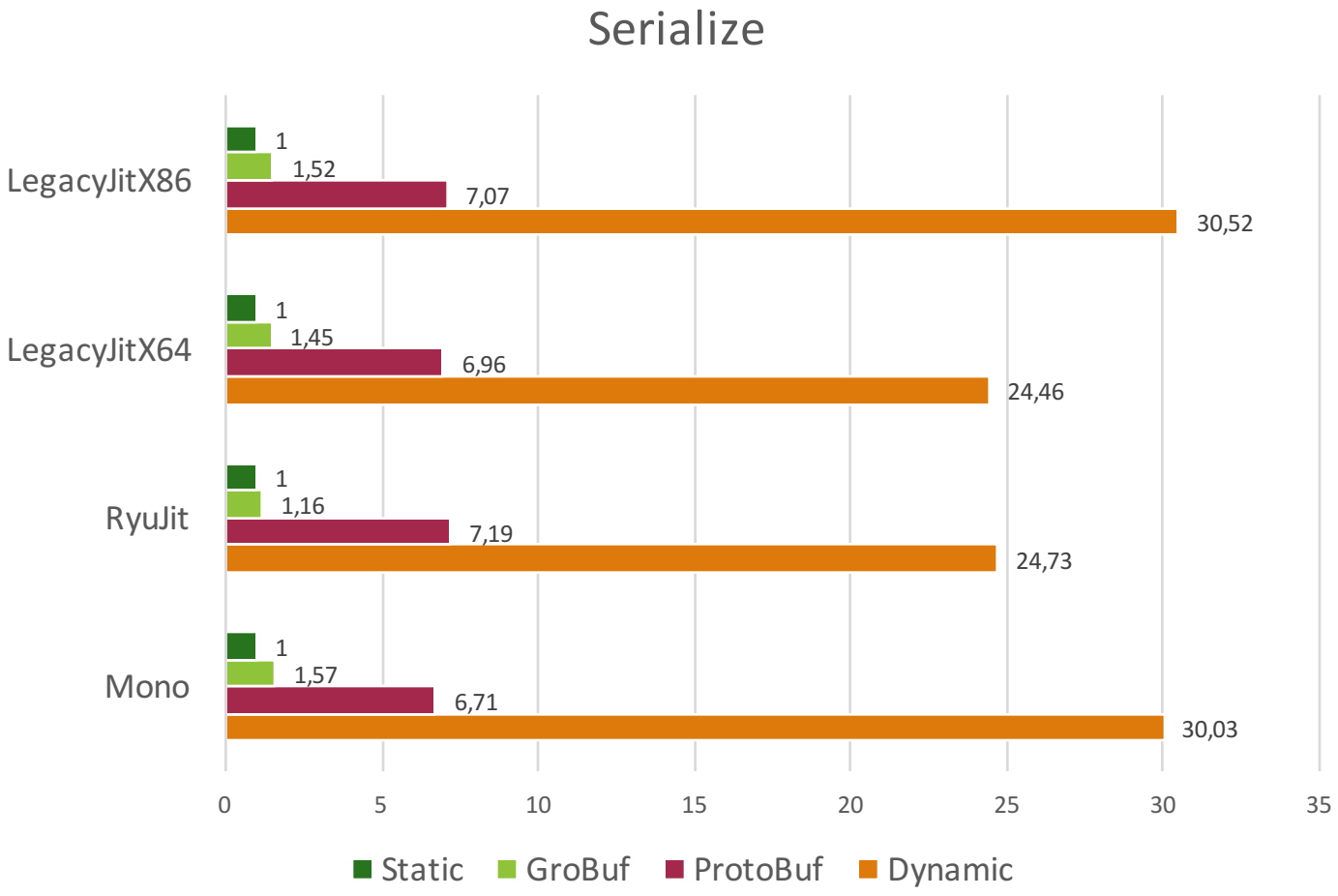
```
public static unsafe class FlatReader
{
    public static Flat Read(byte* data, ref int idx)
    {
        var flat = new Flat();
        flat.Number = Int32Reader.Read(data, ref idx);
        flat.Kitchen = RoomReader.Read(data, ref idx);
        flat.Room = RoomReader.Read(data, ref idx);
        return flat;
    }
}
```

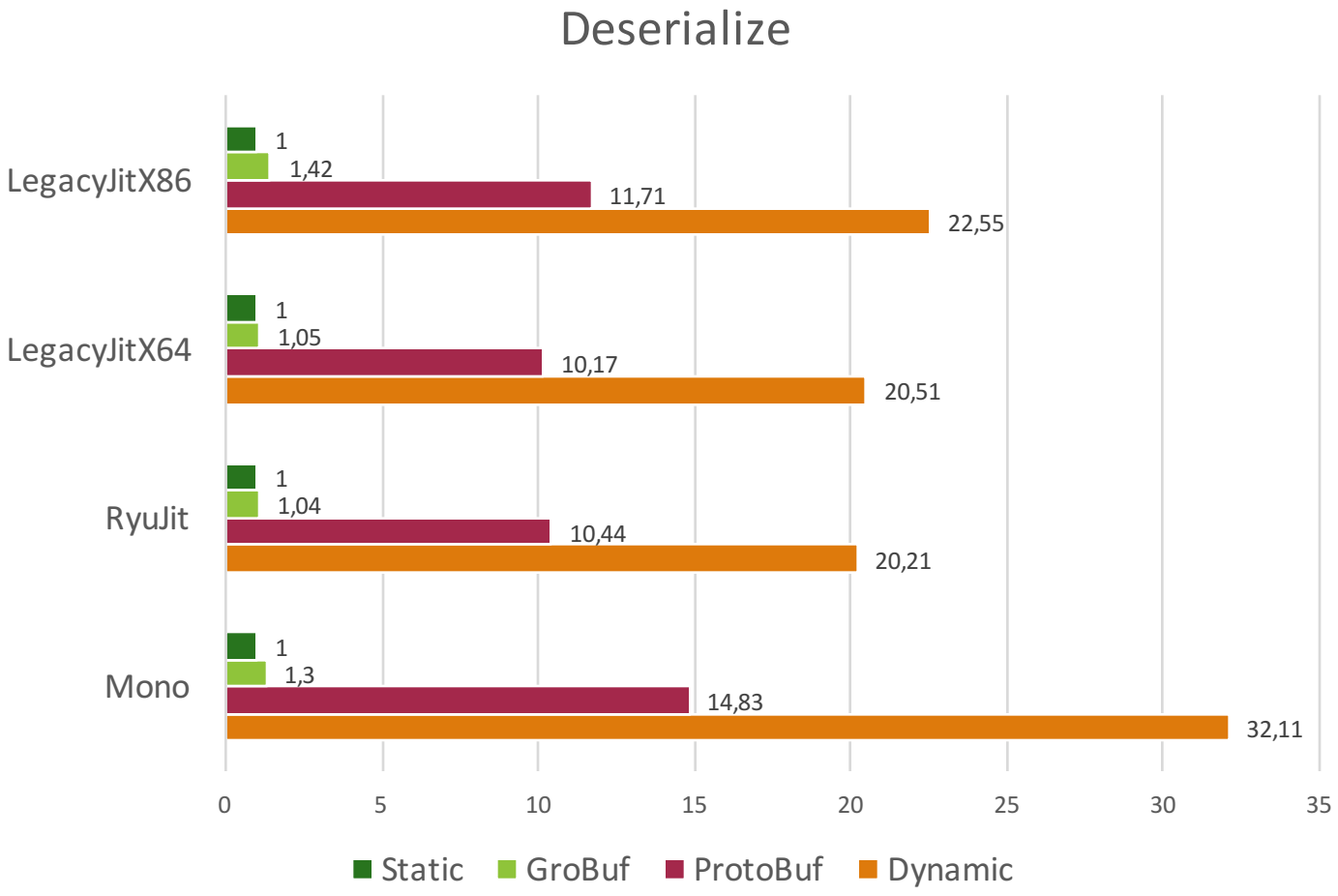
Вызов конструктора

Пишем в поля

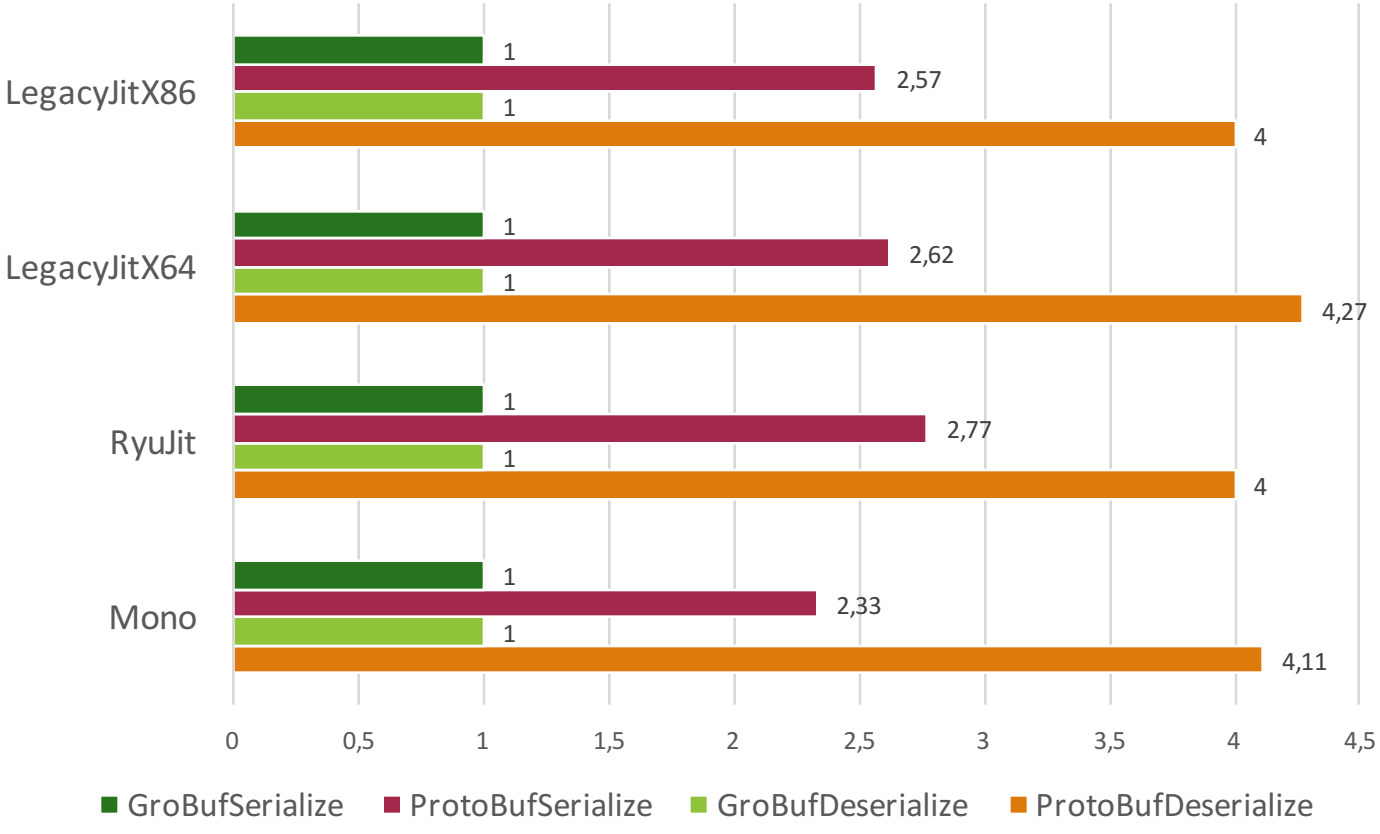
- Бинарный сериализатор GroBuf, использующий кодогенерацию на IL
  https://github.com/homuroll/GroBuf
  https://www.nuget.org/packages/GroBuf

- Есть поддержка старых версий контрактов
- Сравнение с ProtoBuf:
  - Более сырой и простой формат – работает быстрее
  - Но зато размер сериализованных данных больше


Serialize
LegacyJitX86
1
1,52
7,07
30,52
LegacyJitX64
1
1,45
6,96
24,46
RyuJit
1
1,16
7,19
24,73
Mono
1
1,57
6,71
30,03
0
5
10
15
20
25
30
35
Static
GroBuf
ProtoBuf
Dynamic


Deserialize
LegacyJitX86
1
1,42
11,71
22,55
LegacyJitX64
1
1,05
10,17
20,51
RyuJit
1
1,04
10,44
20,21
Mono
1
1,3
14,83
32,11
0
5
10
15
20
25
30
35
Static
GroBuf
ProtoBuf
Dynamic

Более сложный контракт



Задача 4. Сериализация JSON

JIL – сериализатор JSON, написанный с применением кодогенерации на IL

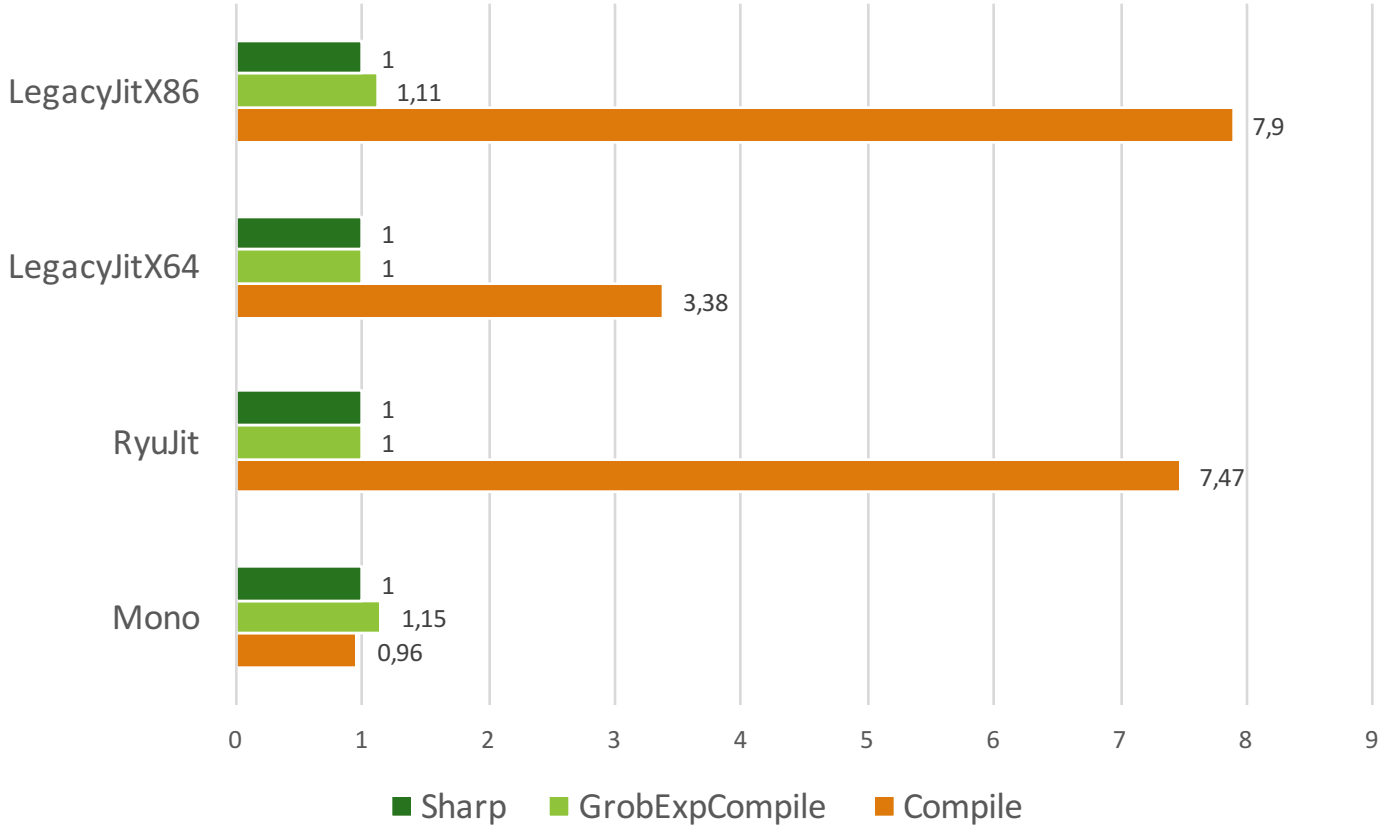
https://github.com/kevin-montrose/Jil

Задача 5. Компиляция expression’ов

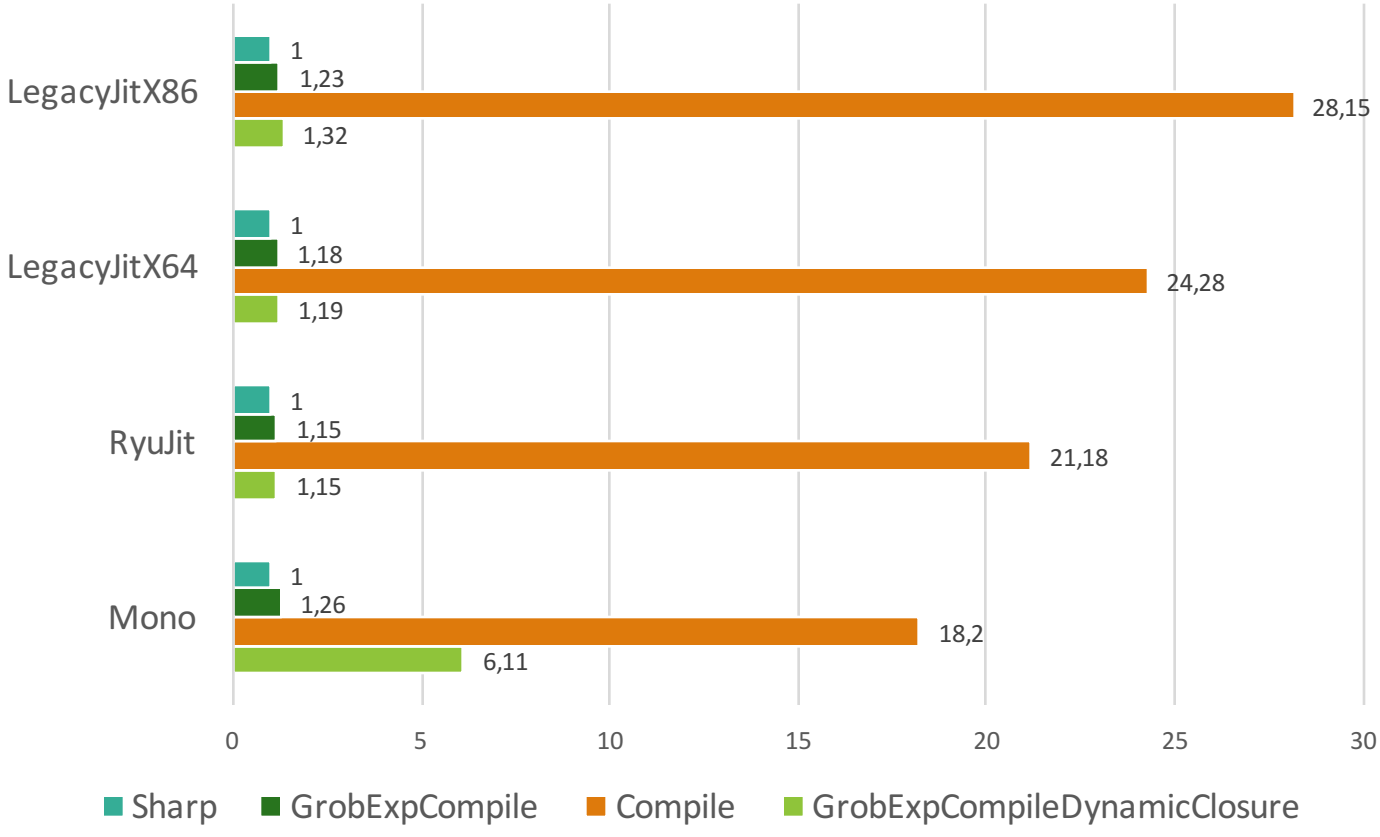
- Бурно использовались Expression’ы
- Натолкнулись на проблемы с производительностью
- Решение: свой компилятор
  https://github.com/homuroll/GrobExp.Compiler
  https://www.nuget.org/packages/GrobExp.Compiler

```
public class TestClassA
{
    public string S { get; set; }
    public TestClassB[] ArrayB { get; set; }
}

public class TestClassB
{
    public string S { get; set; }
}

Expression<Func<TestClassA, string>>
    benchmark1 = a => a.ArrayB[0].S;

Expression<Func<TestClassA, bool>>
    benchmark2 = a => a.ArrayB.Any(b => b.S == a.S);
```

```
a => a.ArrayB[0].S
```


LegacyJitX86
1
1,11
7,9
LegacyJitX64
1
1
3,38
RyuJit
1
1
7,47
Mono
1
1,15
0,96
0
1
2
3
4
5
6
7
8
9
Sharp
GrobExpCompile
Compile

```
a => a.ArrayB.Any(b => b.S == a.S)
```



Задача 6. Оптимизация LINQ

LINQ часто более выразительный

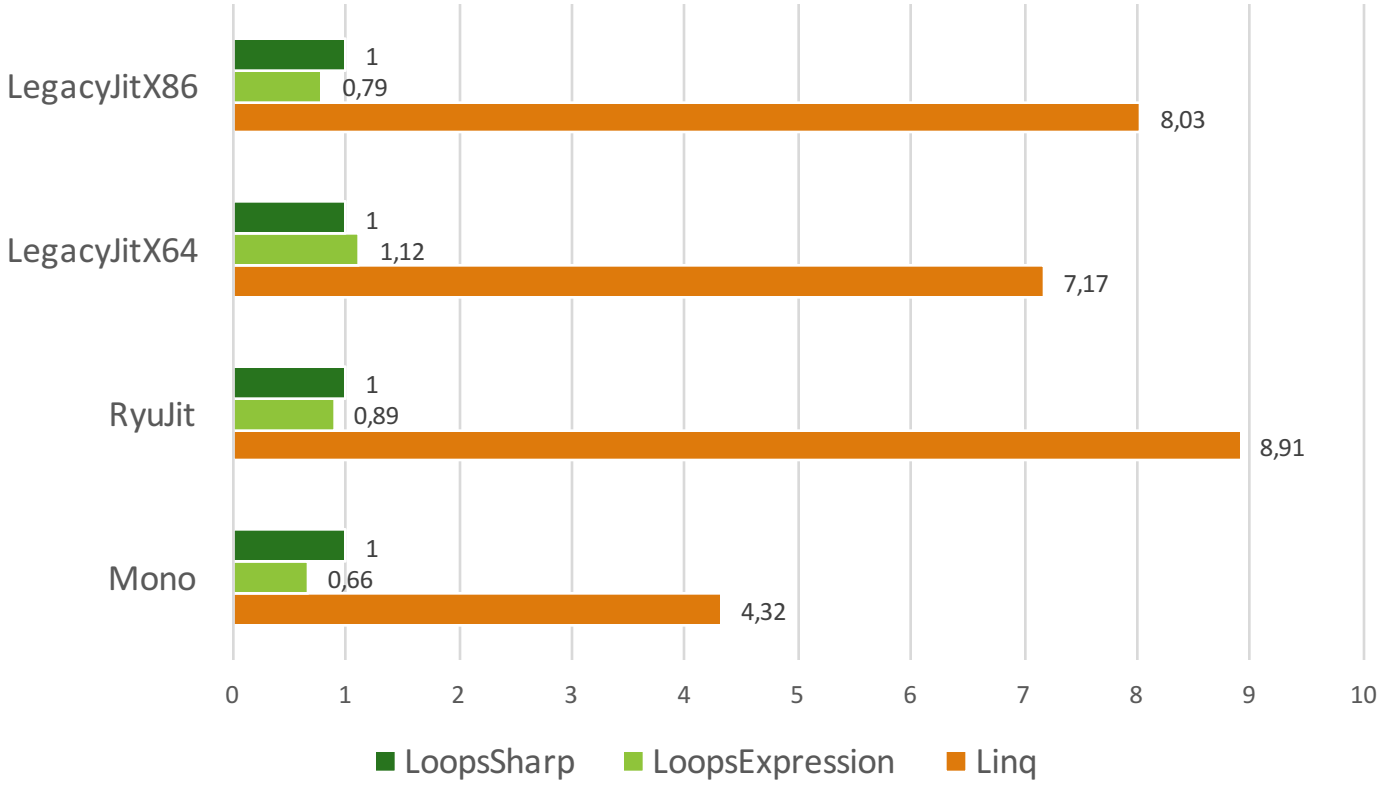
```
data.CurrencyCode =
    (from sg7 in message.SG7
     from details in sg7.Currencies.Details
     where details.UsageCodeQualifier == "2"
           && details.TypeCodeQualifier == "4"
     select
details.IdentificationCode).FirstOrDefault();
```

Но работает медленнее

- Перед компиляцией развёртываем LINQ в циклы
- Посмотреть можно в проекте CodeGenPerfTests.Linq

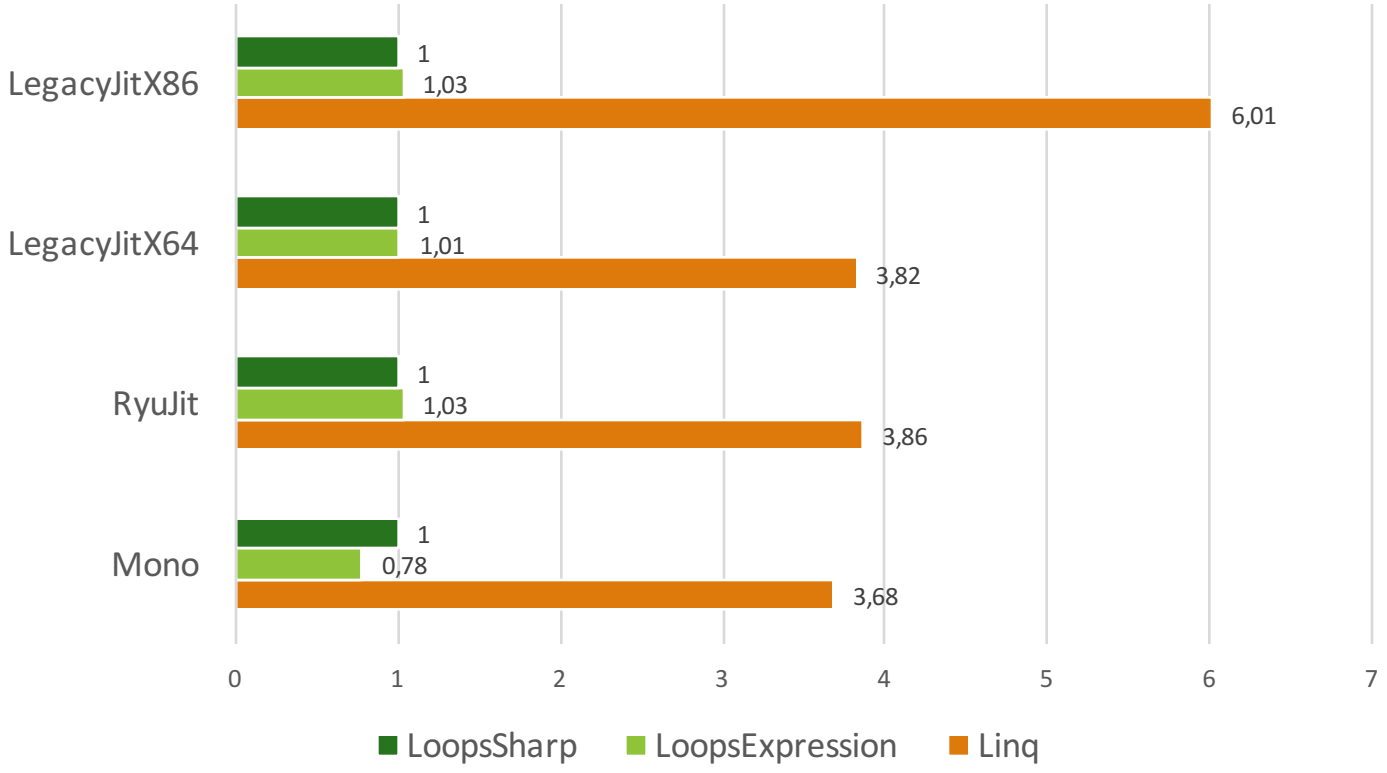
```
string result = null;
foreach (var sg7 in message.SG7)
{
    foreach (var details in sg7.Currencies.Details)
    {
        if (details.UsageCodeQualifier == "2"
            && details.TypeCodeQualifier == "4")
        {
            result = details.IdentificationCode;
            goto _found;
        }
    }
}
_found:
```

```
Expression<Func<List<int>, int>> benchmark1 =
    x => x.Count(z => z > 10000);
```

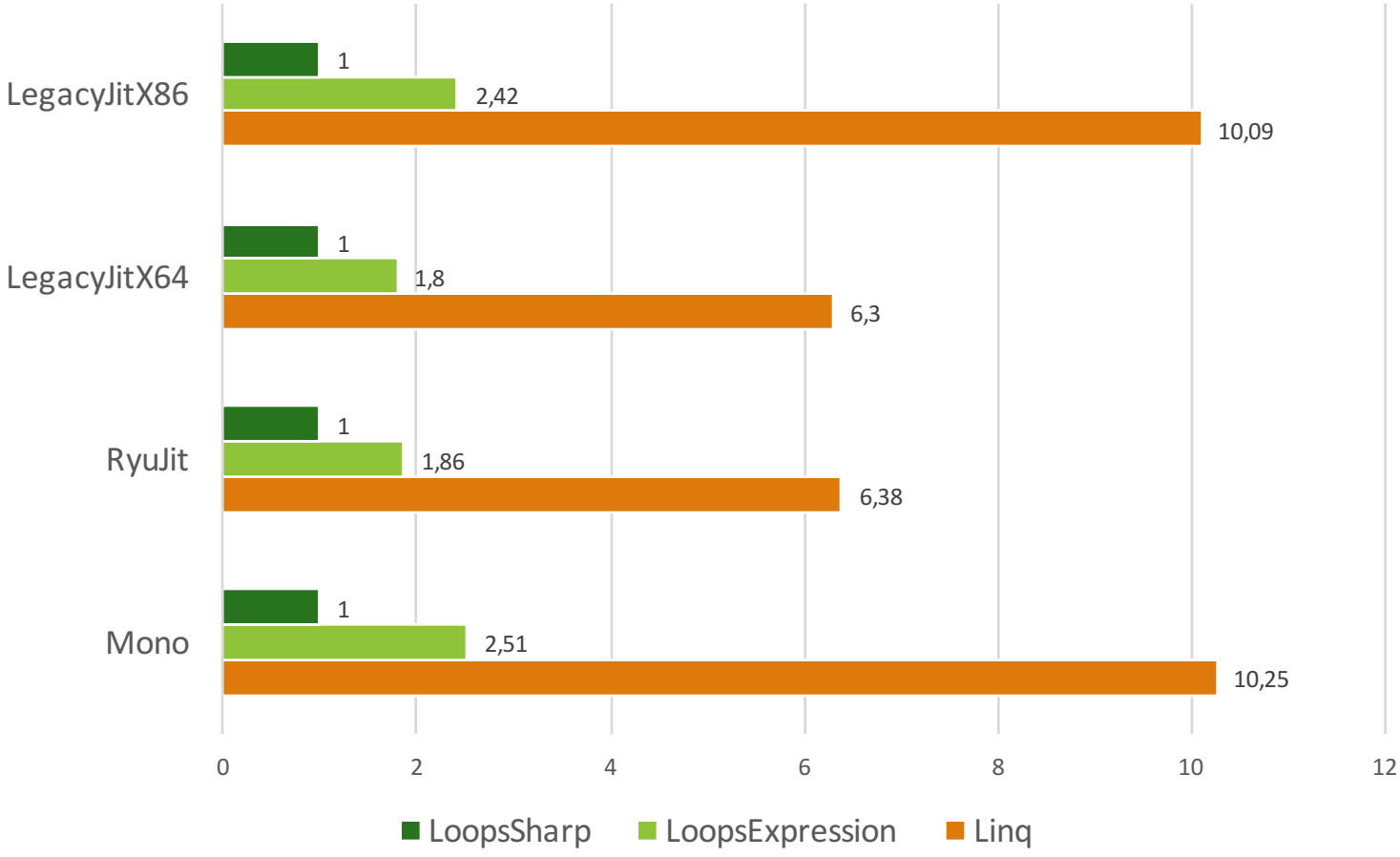


```
public class TestData
{
    public int X { get; set; }
}

Expression<Func<List<TestData>, int>> benchmark2 =
    x => x.Aggregate(0, (s, o) => s + o.X);
```

```
x => x.Aggregate(0, (s, o) => s + o.X);
```



```
Expression<Func<List<Message>, string>> benchmark3 =
    x => (from message in x
          from sg7 in message.SG7
          from details in sg7.Currencies.Details
          where details.UsageCodeQualifier == "2"
                && details.TypeCodeQualifier == "4"
          select details.IdentificationCode).FirstOrDefault();
```


LegacyJitX86
1
2,42
10,09
LegacyJitX64
1
1,8
6,3
RyuJit
1
1,86
6,38
Mono
1
2,51
10,25
0
2
4
6
8
10
12
LoopsSharp
LoopsExpression
Linq

Вывод

Преимущества

- Прирост производительности
- Код красивый и оптимальный

Недостатки

- Технически сложная задача
- Большее время разогрева
- Возможен прирост использования памяти

ВОПРОСЫ?

Игорь Чевдарь

ichevdar@kontur.ru

homuroll@yandex.ru