Lost in Translation
Reguläre Ausdrücke als Englische Sätze

Tim Bourguignon

tim.bourguignon@mathema.de
www.mathema.de
www.timbourguignon.fr

- Foreword

- * Expression

  - VerbalExpression (Prior-Art)

  - SimpleExpression (Nice try)

  - MagicExpression (Masterpiece)

- Wrap up

- **Why express yourself like this?**

  > **[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}**

- **When you can say it like Shakespeare?**

  > **Thou shall match a string of letters**
  > **follow'd by @**
  > **then some characters**
  > **a dram dot**
  > **and some moo stuff**

- **A more coder friendly version maybe?**

  > **Thou.shallmatch(a-string-of-letters)**
  > **.followdby("@")**
  > **.then.somecharacters()**
  > **.adram(".")**
  > **.and.some(moostuff);**

- **"SimpleExpression"**
  - Syntax close to the English language
  - Build as a fluent API
  - Outputs regular expressions
  - Tailored for newbie's
    - Could it satisfy veterans too?
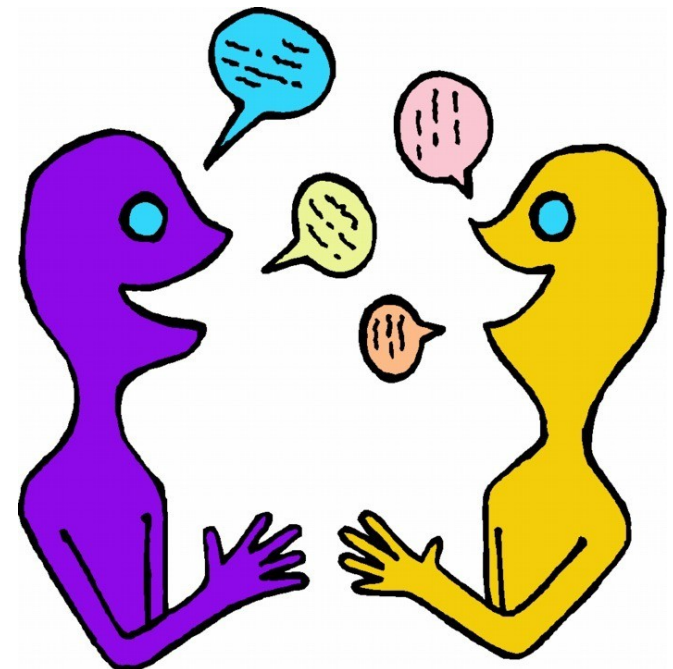
- Example for a "C# dynamics" talk

- Write a real DSL (at least) once

- See if it works…

- Regular Expression knowledge refresh

- ~~Get rich and famous (bitches!)~~

Because I can!

- The more complex the expression, the more surprised and god-like you'll feel when it works

- When you write one that works and you know no-one will ever understand, feel like Houdini mystifying everyone

- You can strip down someone's regex to pieces and yet never figure it out. Which makes you feel like looking at Houdini and God's work combined

- Old regexes (of yours) are like teenage kids, you know they came out of you, but you don't quite get them anymore

- How to write regular expressions

- Do's and don'ts working with Regexes

- Optimization & performance

- Discuss the use of the RFC822 Email Regex

?
==
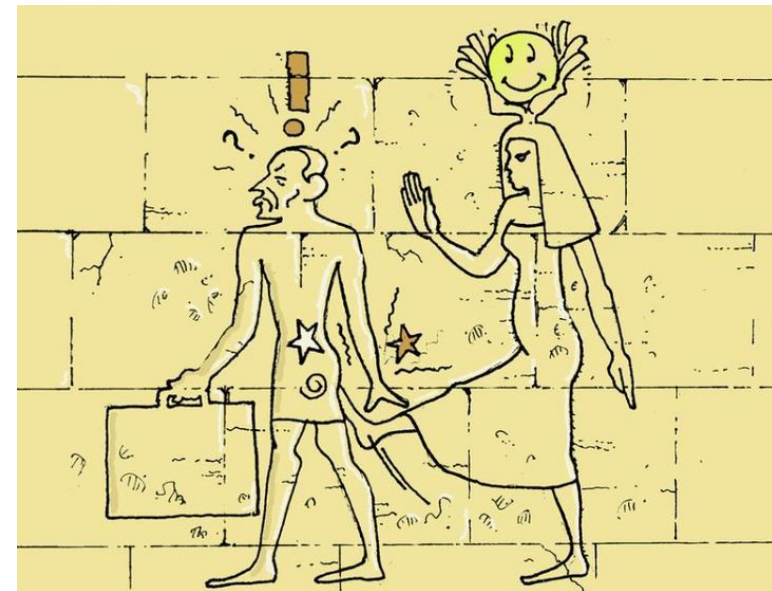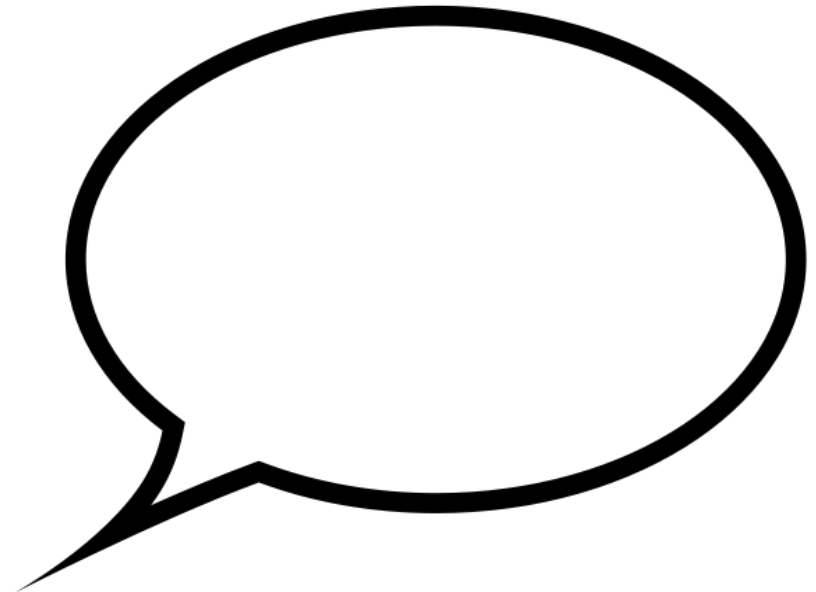
VerbalExpression

- **"JavaScript Regular expressions made easy"**
  - On github (github.com/VerbalExpressions)
  - Forks for: Ruby, C#, Python, Java, Groovy, PHP, Haskell, C++ and Objective-C

```
var tester = VerEx()
    .startOfLine()
    .then( "http" )
    .maybe( "s" )
    .then( "://" )
    .maybe( "www." )
    .anythingBut( " " )
    .endOfLine();
```

- Inconsistencies

```
var expression = VerEx()
        .find( "http" )
        .maybe( "s" )
        .then( "://" )
        .or()
        .then( "ftp://" )
```

- find()
- or().then(X)
- Or() logic

■ Branching

Could you please get me a burger and fries or a pizza?


```
var expression = VerEx()
        .find( "http" ).maybe( "s" ).then( "://" )
        .or()
        .then( "ftp://" )
```

?

```
var expression = VerEx()
    .(find( "http" ).maybe( "s" )
    .then( "://" ))
    .or()
    .(then( "ftp://" ))
```

```
var expression = VerEx()
        .find( "http" ).maybe( "s" )
        .(then( "://" ))
        .or()
        .(then( "ftp://" ))
```

Can you tell what this VerbalExpression does?

```
VerEx().then( "." ).replace( my_paragraph, ". Stop." );
```

- Is this intuitive?
- Why not something like the following?

```
VerEx().find( "." ).in( my_paragraph).replaceWith(". Stop." );
```

SimpleExpression(s)

- Here's how you use a SimpleExpression

```
dynamic simpex = new SimpleExpression();
simpex.here.I.can.chain.my.commands.Generate();
Console.Write(simpex.Expression);
```

- „dynamic"?

```
dynamic someInt = 4;
someInt.ICanWriteHereWhateverIWantAndItCompiles("doh");
// … but will crash & burn in flames at runtime
```

- DynamicObject

```
dynamic someDynamic = new DynamicObject();
someDynamic.Something();
=> TryInvokeMember("Something")
```

- **Floating point number matching**

```
simpex
    .Maybe('-')
    .Numbers              //Default is „zero or more"
    .One('.')
    .Numbers.AtLeast(1)
    .Generate();
```

- **Hexadecimal Color**

```
simpex
    .One('#')
    .Numbers.And("abcdef").Exactly(3)
        .Or
    .Numbers.And("abcdef").Exactly(6)
    .Generate();
```

```
string allowedChars = @"!#$%&'*+/=?^_`{|}~-";
simpex
    .Group
        .Alphanumerics.And(allowedChars).AtLeast(1)
    .Together.As("beforeAt")
    .One('@')
    .Group
        .Letters.And(allowedChars).AtLeast(1)
        .Group
            .One(".")
            .Alphanumerics.And(allowedChars).AtLeast(1)
        .Together.As("dotAndAfter")
    .Together.As("afterAt")
    .Generate();
```

- Letters, Alphanumerics

- Group.*Cardinal*.X.Together.As()

- Regular Expression Range for 0-255

  - [0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5]

  - e.g. a number in 0-9 or 10-99 or 100-199 or 200-249 or 250-255

- IP Match

```
Simpex
    .NumberInRange("1-255").One('.')
    .NumberInRange("0-255").One('.')
    .NumberInRange("0-255").One('.')
    .NumberInRange("0-255")
    .Generate();
```

- How do you like SimpleExpression sofar?

- Any comments / suggestions / violent rant?

- Let's criticize!

- Isn't the following gorgeous to read? (hint: the answer is **YES** ;)

  ```
  simpex.Letters.AtLeast(3).AtMost(4)
  simpex.Letters.And("-_ ").Except("a")
  simpex.Group.Text("http").Maybe('s').Together.As("protocol")
  ```

- What about the following?

  ```
  simpex.Group.One('0').Letters.Exactly(1).Together.Exactly(2)
  ```

  - Isn't there an 'And' missing?

  - 'Two' or 'Twice'?

  - .Letters.Exactly(1) or .Exactly(1).Letters ?

    - .Group.Exactly(3).X.Together

    - .Exactly(3).Group.X.Together ?

- ## What does the following mean?

Simpex
      .Group.AtLeast(5).Numbers.Exactly(2).Together.One(' ').AtLeast(1)

  - *"Group at least 5 numbers twice, followed by at least one space"*
  - *"At least 5 groups of 2 numbers followed by at least one space"*

- ## The problem here:

Group.AtLeast.X.Exactly.Together.Y.AtLeast

  - ## And no, pushing it after the 'together' wouldn't solve the issue

Group.X.Exactly.Together.AtLeast.Y.AtLeast

- "Stuttering", one of the limits of that prose

```
Simpex
    .Group
        .Group
            .Text(abcd)
            .Group
                .Letters.And("-")
            .Together
        .Together
        .Text("cde")
    .Together
```

- ## Create now, join later

```
var abcd = new SimpleExpression().Text("abcd").Generate();
var efgh = new SimpleExpression().Text("efgh").Generate();

simpex.Sub(abcd).Or.Sub(efgh).Generate();
```

- ## That encapsulated grouping example

```
var innerMostGp = new SimpleExpression()
    .Goup.Letters.And("-").Together.Generate();

var innerGp = new SimpleExpression()
    .Group.Text(abcd).Sub(innerMostGp).Together.Generate();

var outerGp = new SimpleExpression()
    .Group.Sub(innerGp).Text("cde").Together.Generate();
```

Critic: Implicit Cardinality

- What is meant here?

  `simpex.EitherOf("a|b|c").AtLeast(2)`

  - *"a, b or c, at least two of them"*
  - *"twice a or twice b or twice c"*
    - How do I do I express the other one?

Critic: Experts, get lost!

- Regular expression "experts" fall back onto what they know

- Does this create a class? A Group? Capturing or not?

  > **simpex.Letters.Except("aeiou").And("§$%&").AtLeast(2).AtMost(4)**

- How can I do Backward & Forward Lookup?

  - Well you can't

- The more you know the more disturbing SimpleExpression is

- **All functions are known at compile time**

  - Fully "implement-able" via a Fluent API

- **Unnecessary lack of Intellisense support**

- SimpleExpression's commands cannot be linearly parsed

- Simple repeat count

```
Simpex.One("x").AtLeast(3).AtMost("5")

// x => x{3,} => x{3,5}
```
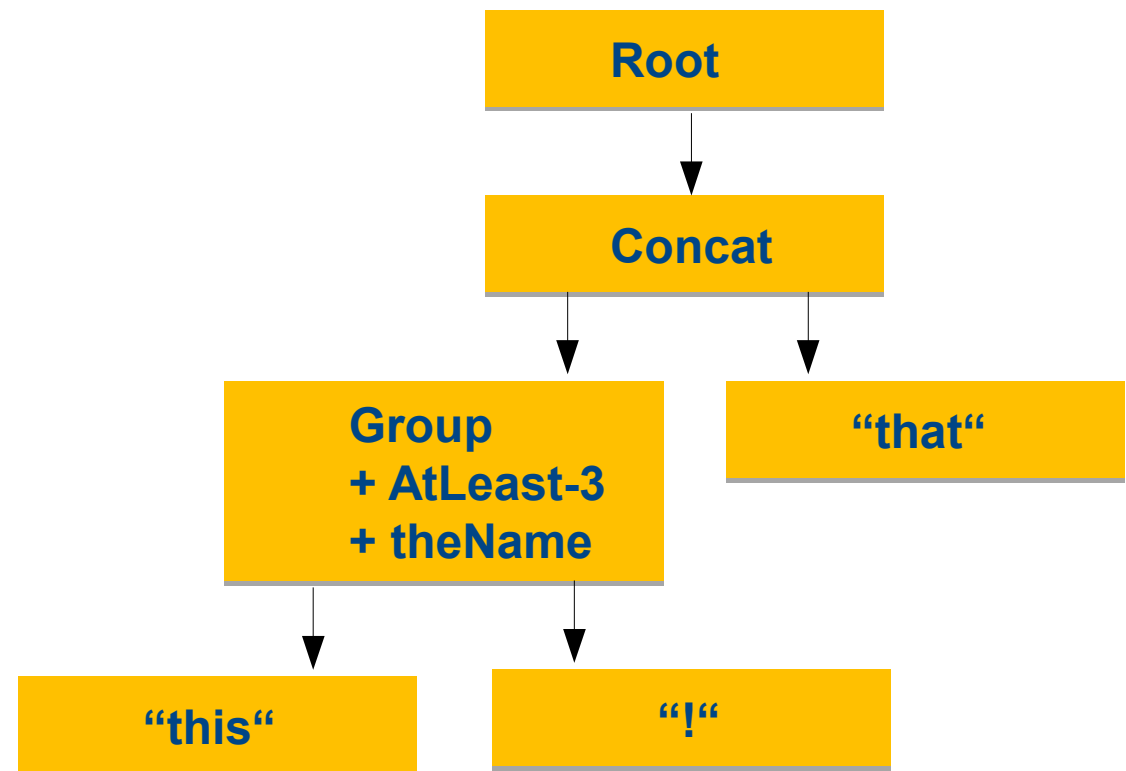
- Inversion of named groups and repeat count

```
simpex
    .Group.AtLeast(3).Text("something").Together.As("theName")

// (<theName>something){3,}
```

```
simpex
    .Group.AtLeast(3)
        .Text("this").One("!")
    .Together.As("theName")
    .Text("that")
    .Generate();
```

Root

Concat

**Group
+ AtLeast-3
+ theName**

"that"

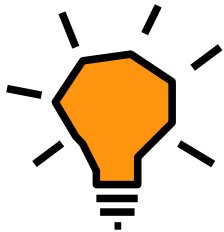**"this"**

"!"

```
//Recursively generated expression:
(<theName>this!){3,})that
```

- SimpleExpression's semantic is quite nice and in some cases can actually be helpful

- Many edge cases where the grammatic doesn't fit that well and tend to pull down the concept as a whole ; it is like *"death by 1000 paper cuts"*

- Using parenthesis & reordering elements logically instead of grammatically

  → Losing some readability for the sake of precision?

- Compared to SimpleExpression

  - Loses the dynamics for a fluent API

  - Less-funky but less ambiguous functions

  - No more cumbersome Abstract Syntax Tree

  - Way more functions!

- **Install via Nuget**

```
Install-Package MagicExpression
```

- **Instanciation**

```
var magicWand = Magex.New();

magicWand.The.Functions.Here;        //no lame .Generate() here

Console.WriteLine(magicWand.Expression);
```

```
var magicWand = Magex.New();

MagicWand
    .Character('-').Repeat.AtMostOnce()
    .CharacterIn(Characters.Numeral).Repeat.Any()
    .Character('.')
    .CharacterIn(Characters.Numeral).Repeat.AtLeastOnce();

// Creates the following regex:     -?[0-9]*\.[0-9]+
// Matches "1.234", "-1.234", "0.0", ".01"
// Doesn't Match "0", "1,234", "0x234", "#1a4f66"
```

```
Simpex
    .Maybe('-')
    .Numbers
    .One('.')
    .Numbers.AtLeast(1)
    .Generate();
```

- Character() & CharacterIn()

- .Repeat trigger

- Optional block handled via .AtMostOnce()

    - .Any() or .Between(0, uint) would also do the trick

```
var magicWand = Magex.New()
    .Character('<')
    .CaptureAs("tag",
        x => x.CharacterNotIn('>').Repeat.AtLeastOnce())
    .Character('>')
    .Character().Repeat.Any().Lazy()
    .String("</")
    .BackReference("tag")
    .Character('>');

// Matches "<strong>hello world</strong>" & "<h1>A title</h1>"
// Doesn't match "<h1>A tag mismatch</strong>"
```

- Group() → Non-capturing group

- Capture() → Capturing group

- CaptureAs() → Named capturing group

- BackReference(string) → Back reference on a named group

```
var magicWand = Magex.New();
magicWand.Options = RegexOptions.IgnoreCase;
const string allowedChars = @"!#$%&'*+/=?^_`{|}~-";

MagicWand
    .Alternative(
        Magex.New().String("http"),
        Magex.New().String("ftp"))
    .Character('s').Repeat.AtMostOnce()
    .String("://")
    .Group(Magex.New().String("www."))
        .Repeat.AtMostOnce()
    .CharacterIn(Characters.Alphanumeric, allowedChars);
```

- Alternative(params Magex[])

- CharacterIn(params char[])

MATHEMA campus 2014

- „0x12e5ad"

```
Magex.New()
    .Character('0')
    .CharacterIn("xX")
    .CharacterIn(Characters.Numeral, "abcdefABCDEF").Repeat.Times(6);
```

- Hex Color

```
Magex.New()
    .Character('#')
    .Alternative(
        Magex.New().CharacterIn(Characters.Numeral, "abcdefABCDEF")
            .Repeat.Times(6),
        Magex.New().CharacterIn(Characters.Numeral, "abcdefABCDEF")
            .Repeat.Times(3).EndOfLine());
```

```
Magex.New()
    .Builder.NumericRange(1, 255).Character('.')
    .Builder.NumericRange(0, 255).Character('.')
    .Builder.NumericRange(0, 255).Character('.')
    .Builder.NumericRange(0, 255);
```

- Builder property to help you with predefined functions
  - Currently only NumericRange()
- Literal(string) function to add a predefined regular expression

- Other functions?
  - Email? Date with pseudo variable format → „yyyy-MM-dd" ?
  - Hex, Floating point number... ? Any ideas? Wishes?

Doc?

- Readme

    - https://github.com/ghusse/magicexpression
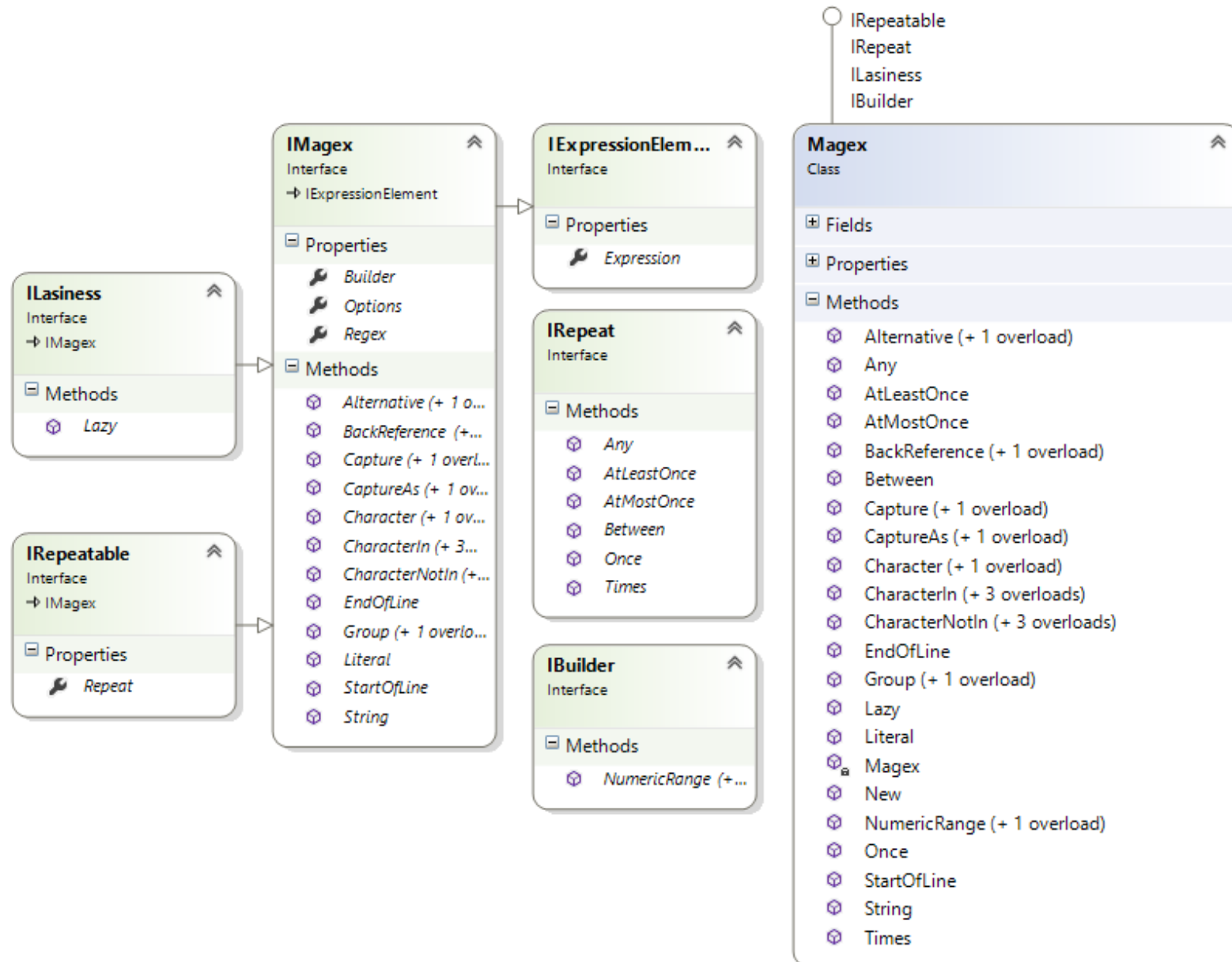
magex.Character('s').Repeat.AtMostOnce().Character...

IMagex

Methods	
▷ Alternative	IRepeatable
▷ Alternative	IRepeatable
▷ BackReference	IRepeatable
▷ BackReference	IRepeatable
▷ Capture	IRepeatable
▷ Capture	IRepeatable
▷ CaptureAs	IRepeatable
▷ CaptureAs	IRepeatable
▷ Character	IRepeatable
▷ Character	IRepeatable
▷ CharacterIn	IRepeatable
▷ CharacterIn	IRepeatable
▷ CharacterIn	IRepeatable
▷ CharacterIn	IRepeatable
▷ CharacterNotIn	IRepeatable
▷ CharacterNotIn	IRepeatable
▷ CharacterNotIn	IRepeatable
▷ CharacterNotIn	IRepeatable
▷ EndOfLine	IMagex
▷ Group	IRepeatable
▷ Group	IRepeatable
▷ Literal	IMagex
▷ StartOfLine	IMagex
▷ String	IMagex

IRepeatable

Properties	
🔧 Repeat	IRepeat

Only those methods are available after .Repeat

IRepeat

Methods	
▷ Any	ILasiness
▷ AtLeastOnce	ILasiness
▷ AtMostOnce	ILasiness
▷ Between	ILasiness
▷ Once	IMagex
▷ Times	IMagex

IMagex
Interface
↦ IExpressionElement

ILasiness
Interface
↦ IMagex

IMagex gives us access to all those Methods...
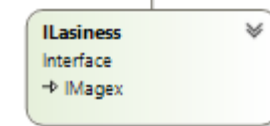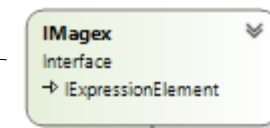
Gives an ILasiness Back, i.e. an IMagex

Let's wrap up
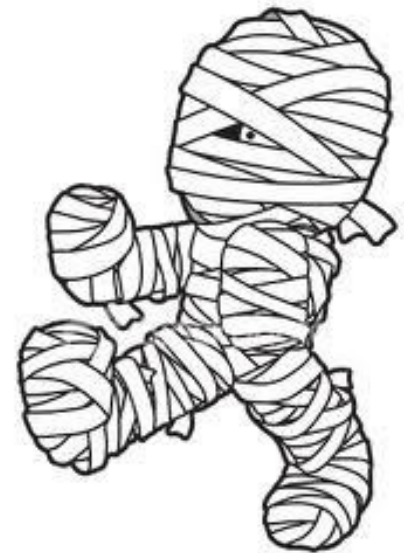
- **It is possible to write such a DSL!**
  - ~~I'm going to be rich and famous~~
  - Our languages are not always a good thing to immitate
  - But (in this case) a pinch of DSL doesn't hurt

- **SimpleExpression**
  - Semantically attractive, but not viable as is

- **MagicExpression**
  - Less sexy but useful
  - Next big feature → Reverse engineer regular expressions?

Fork me on GitHub

MATHEMA campus 2014

QUESTIONS?
SUGGESTIONS?
IDEAS?

THANKS!

- www.timbourguignon.fr
- tim.bourguignon@mathema.de