Использование парсер-комбинаторов на Sprache

Евгений Романовский

Разработчик команды Диадок

Задача

- «Восстановить» объект из строки
- Шаблонизатор с подстановкой переменных

Подходы к решению

1. Парсеры, написанные вручную

- Подходят, когда решение очевидно (Split, IndexOf, Substring)

```
public class SecurityToken
{
    public string UserId { get; set; }
    public DateTime ValidTo { get; set; }
}
```

Нужно распарсить строку

```
"8cb9f238-e0e4-431c-86a6-9ac52b9cb6b9;2017-04-20T00:00:00Z"
```

2. Регулярные выражения

- Относительно простая грамматика в простых случаях

- Трудно читать и поддерживать
- Невозможно работать с вложенными данными
- Не масштабируется на более сложные грамматики

3. Генераторы парсеров

- Генерирует классы на целевом языке (Java, C#, …) во время сборки проекта, которые способные разбирать грамматику
- Требует времени для изучения и интеграции
- Подходит для очень сложных вещей, где важна скорость разбора

- ANTLR (http://www.antlr.org)

3. Генераторы парсеров (ANTLR)

```
grammar Speak;
/* Parser Rules */
chat        : line line EOF ;
line        : name SAYS word NEWLINE;
name              : WORD ;
word        : WORD ;
/* Lexer Rules */
fragment A          : ('A'|'a') ;
fragment S          : ('S'|'s') ;
fragment Y          : ('Y'|'y') ;

fragment LOWERCASE  : [a-z] ;
fragment UPPERCASE  : [A-Z] ;

SAYS                : S A Y S ;
WORD                : (LOWERCASE | UPPERCASE)+ ;
WHITESPACE  : (' '|'\t')+ -> skip ;
NEWLINE             : ('\r'? '\n' | '\r')+ ;
```

4. Парсер-комбинаторы

- Мини-язык описания
- Логика написана на том же языке, что и остальное приложение (C#)
- Техника пришла из функциональных языков

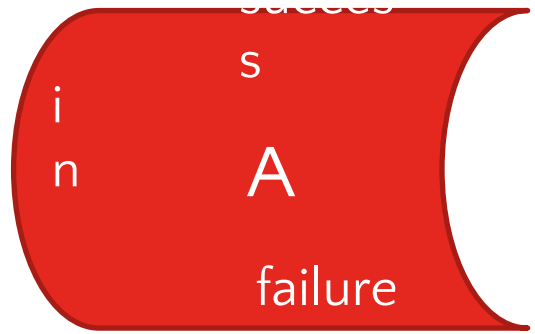
Sprache – C# Parser Framework

- Легко писать
- Легко поддерживать
- Хорошо сочетается с TDD
- Маленький размер библиотеки

https://github.com/sprache/Sprache

Parser

```
public delegate IResult<T> Parser<out T>(IInput input);


public interface IResult<out T>
{
    T Value { get; }
    bool WasSuccessful { get; }
    IInput Remainder { get; }
}
```

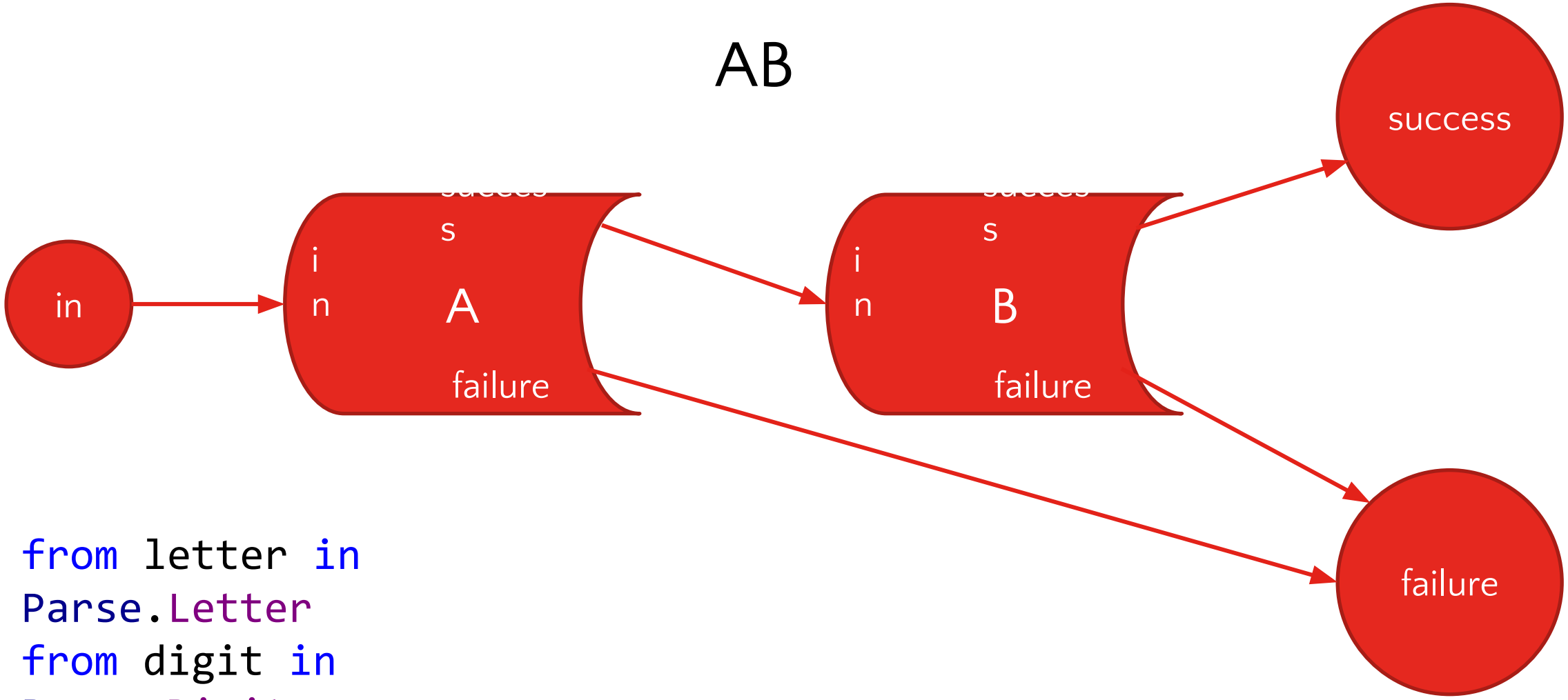


Primitive parsers

```
Parse.Char('<')

Parse.Digit

Parse.WhiteSpace

Parse.Numeric

Parse.AnyChar

Parse.LineEnd

Parse.LetterOrDigit
```

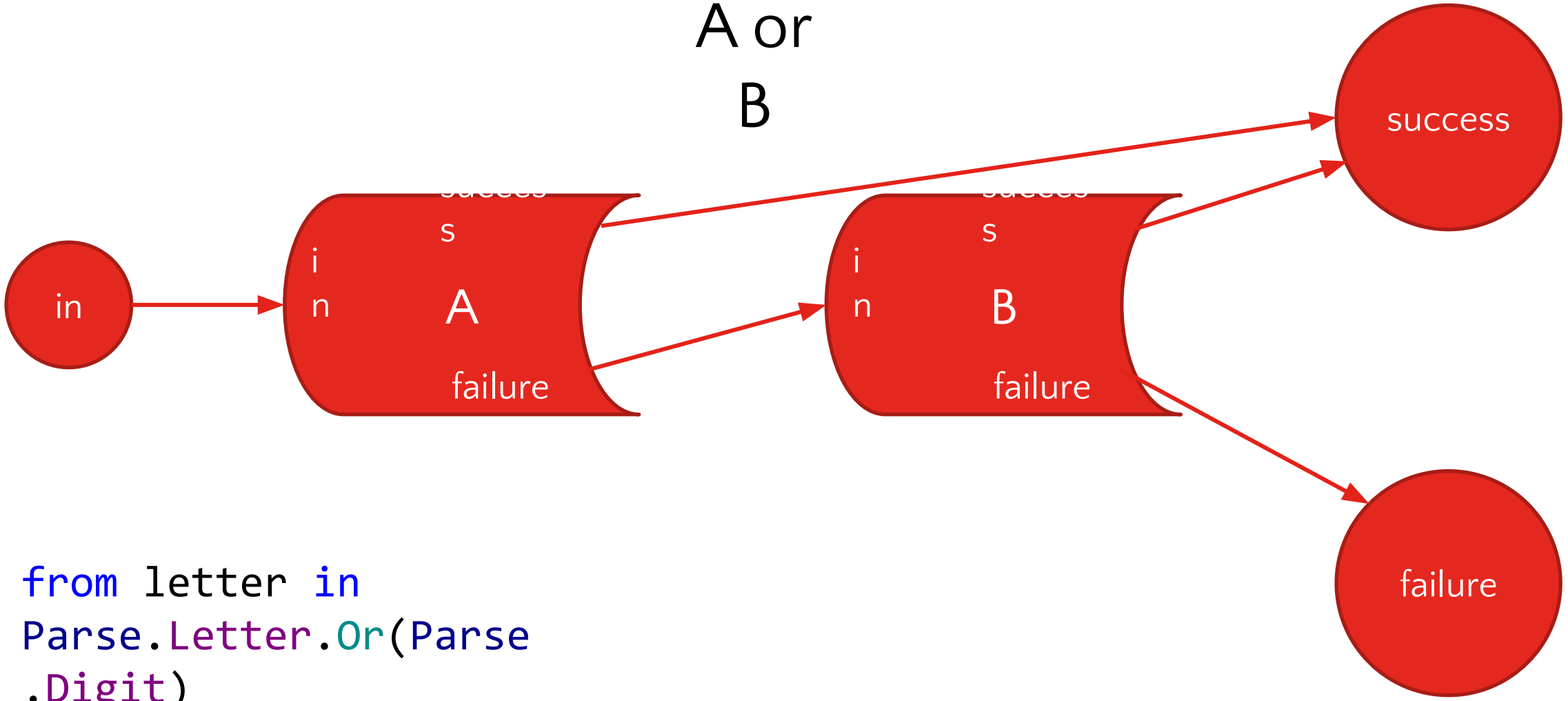
…

Parsers and combinators – Sequence



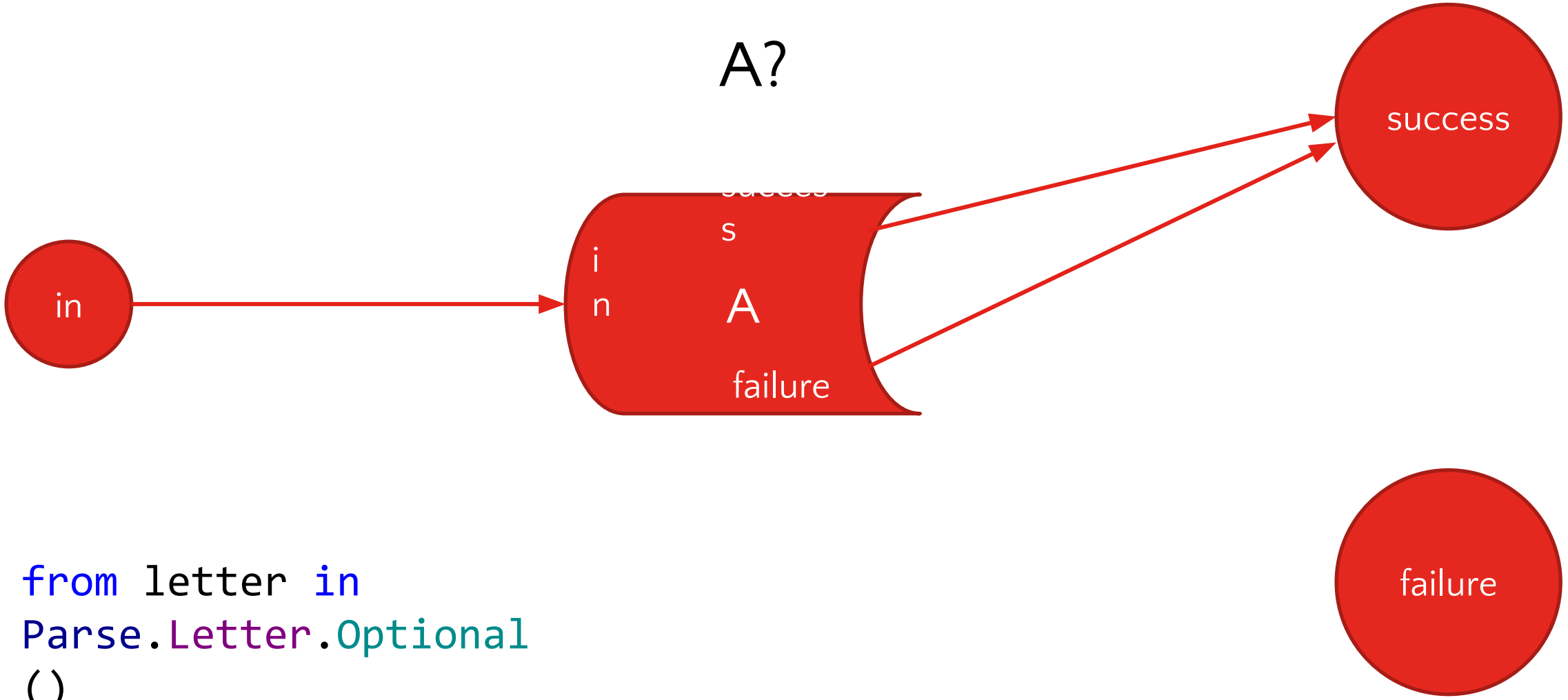

```
from letter in
Parse.Letter
from digit in
Parse.Digit
```

Parsers and combinators – Or



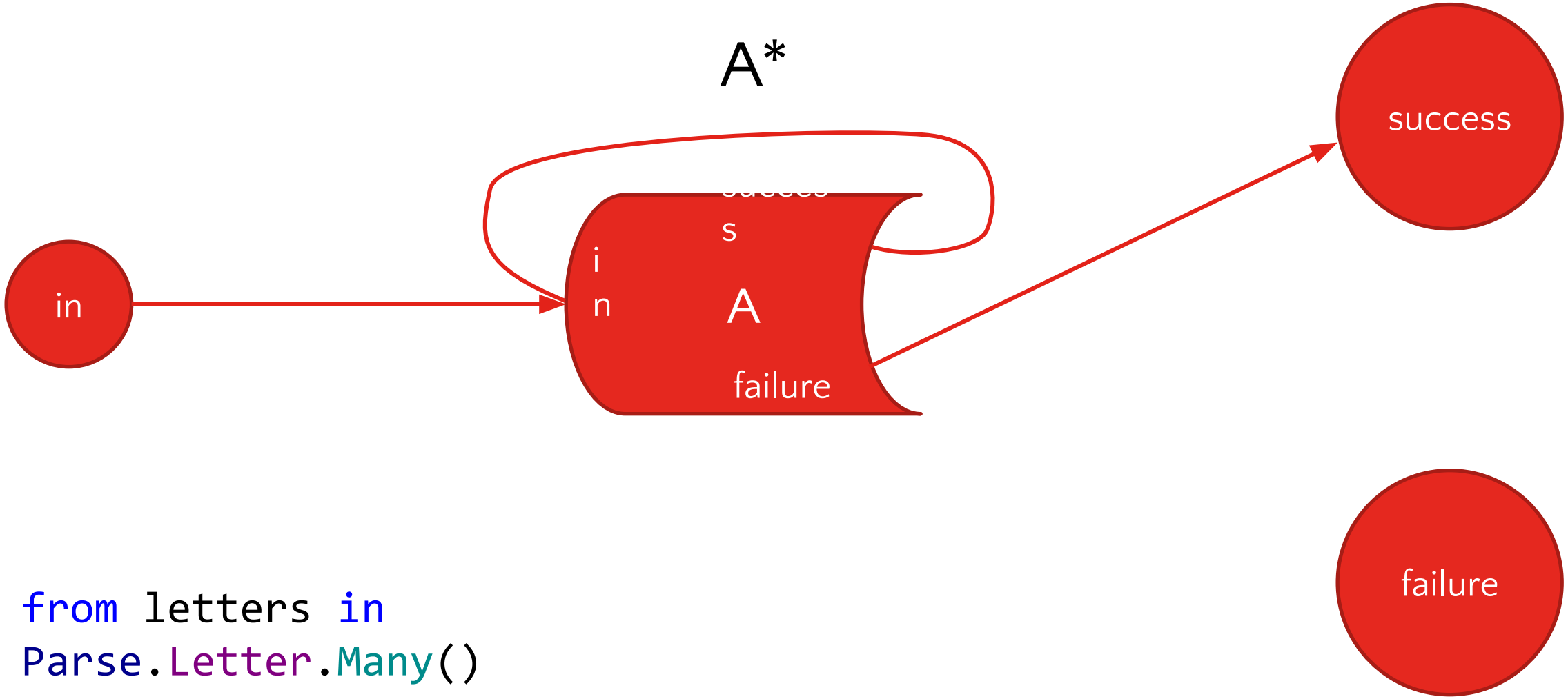

```
from letter in
Parse.Letter.Or(Parse
.Digit)
```

Parsers and combinators – Optional




```
from letter in
Parse.Letter.Optional
()
```

Parsers and combinators – Many




```
from letters in
Parse.Letter.Many()
```

Строим свои парсеры

```
Parser<string> identifier =
    from leading in Parse.WhiteSpace.Many()
    from first in Parse.Letter.Once()
    from rest in Parse.LetterOrDigit.Many()
    from trailing in Parse.WhiteSpace.Many()
    select new string(first.Concat(rest).ToArray());

var id = identifier.Parse(" abc123  ");
Assert.AreEqual("abc123", id);
```

Off-topic: Select, SelectMany, Where C#

```
public static IEnumerable<U> Select<T, U>(this
IEnumerable<T> source, Func<T, U> selector)
```

```
public static IEnumerable<U> SelectMany<T, U, V>(this
IEnumerable<U> parser, Func<T, Parser<U>> selector,
Func<T, U, V> projector)
```

```
public static IEnumerable<T> Where<T>(this IEnumerable<T>
source, Func<T, bool> predicate)
```

Off-topic: Select, SelectMany, Where C#

```
public static Parser<U> Select<T, U>(this Parser<T>
parser, Func<T, U> convert)
```

```
public static Parser<V> SelectMany<T, U, V>(this
Parser<T> parser, Func<T, Parser<U>> selector, Func<T, U,
V> projector)
```

```
public static Parser<T> Where<T>(this Parser<T> parser,
Func<T, bool> predicate)
```

Примеры задач

1. Парсинг из текста

```
public class Person
{
    public Person(...)
    {
        Type = type;
        Properties = properties;
    }
    public string Type { get; }
    public IEnumerable<Property> Properties { get; }
}
```

```
public class Property
{
    public Property(...)
    {
        Name = name;
        Value = value;
    }
    public string Name { get; }
    public string Value { get; }
}
```

1. Парсинг из текста

```
magician
[
   name       ‘Merlin’
   age   100500
]
```

Demo 1 (парсинг из текста)

2. Шаблонизатор

```
var props = new Dictionary<string, object>
{
    {"Number", "MX-123"},
    {"Date", new DateTime(2017, 04, 19)},
    {"Caption", "Счет"}
};

string template = "{{Caption}} №{{Number}} от {{Date}}";
```

```
"Счет №MX-123 от 19.04.17"
```

Demo 2 (шаблонизатор)

3. WWW-Authenticate challenge

```
HTTP 401 Unauthorized

# Basic challenge
WWW-Authenticate: Basic realm="FooCorp"

# OAuth 2.0 challenge after sending an expired token
WWW-Authenticate: Bearer realm="FooCorp", error=invalid_token,
error_description="The access token has expired"
```

3. WWW-Authenticate challenge

```
# from RFC-2617 (HTTP Basic and Digest authentication)

challenge       = auth-scheme 1*SP 1#auth-param
auth-scheme     = token
auth-param      = token "=" ( token | quoted-string )

# from RFC2616 (HTTP/1.1)

token           = 1*<any CHAR except CTLs or separators>
separators      = "(" | ")" | "<" | ">" | "@"
                | "," | ";" | ":" | "\" | <">
                | "/" | "[" | "]" | "?" | "="
                | "{" | "}" | SP | HT
quoted-string   = ( <"> *(qdtext | quoted-pair ) <"> )
qdtext          = <any TEXT except <">>
quoted-pair     = "\" CHAR
```

Demo 3 (WWW-Authenticate challenge)

- Serilog
- Stateless
- Autofac
- ...

Автор библиотеки
Nicholas Blumhardt

Контрибьютор
других известных
библиотек

Парсеры, построенные на Sprache

- The template parser in Octostache
- The XAML binding expression parser in OmniXaml
- The query parser in Seq
- The connection string parser in EasyNetQ
- Sprache appears in the credits for JetBrains ReSharper

Вопросы?

http://bit.ly/dotnet_feedback

Евгений Романовский

infoman@skbkontur.ru

kontur.ru