Error-Handling in PEGs

Michael Rüfenacht

m.ruefenacht@students.unibe.ch

Roadmap

1. Error-Detection
2. Error-Recovery
3. Error-Repair
4. Remainder Processing
5. Results
6. Conclusions

Who is who?

Introduction

Parsing Expression Grammars

Formalism to define

- the **lexical** as well as **syntactical**
- elements of a **grammar** as a
- recursive **top-down parser**

Parsing Expression Grammars

```
e_s          ← exprOr EOI

exprOr       ← exprAnd (or exprAnd)*

exprAnd      ← boolean (and boolean)*

boolean      ← _( true / false )_

true         ← `true`

false        ← `false`

or           ← `or`

and          ← `and`

whitespace   ← [ \t]

_            ← whitespace *
```

Parsing Expression Grammars

```
e_s          ← exprOr EOI

exprOr       ← exprAnd (or exprAnd)*

exprAnd      ← boolean (and boolean)*

boolean      ← _( true / false )_

true         ← `true`

false        ← `false`

or           ← `or`

and          ← `and`

whitespace   ← [ \t]

_            ← whitespace *
```

Overview

t r u l e b u t f a l s e

Overview

```
t r u l e   b u t   f a l s e

,,False expected at 0‘‘
```

Overview

t r u l e b u t f a l s e

,,PPError proposing transformations at 3, 6‘‘

Overview

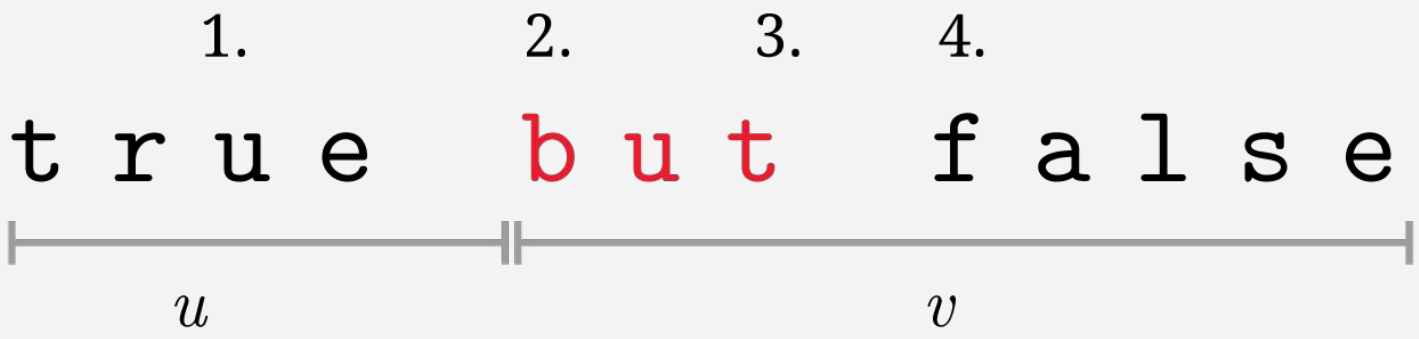
t r u l e b u t f a l s e

,,PPError proposing transformations at 3, 6‘‘

t r u e a n d f a l s e

Overview

Overview



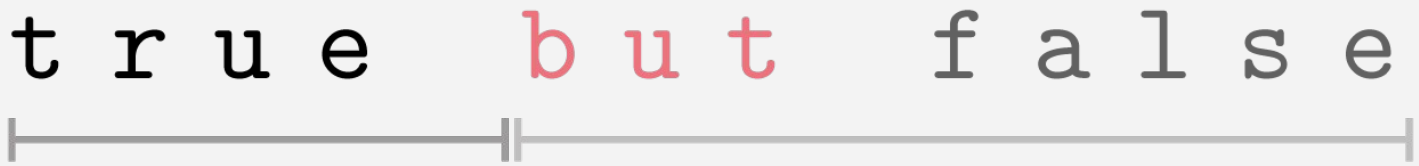
1. Error-Detection
2. Error-Recovery
3. Error-Repair
4. Remainder Processing

A Failure is a Failure is a ...

1. Error-Detection

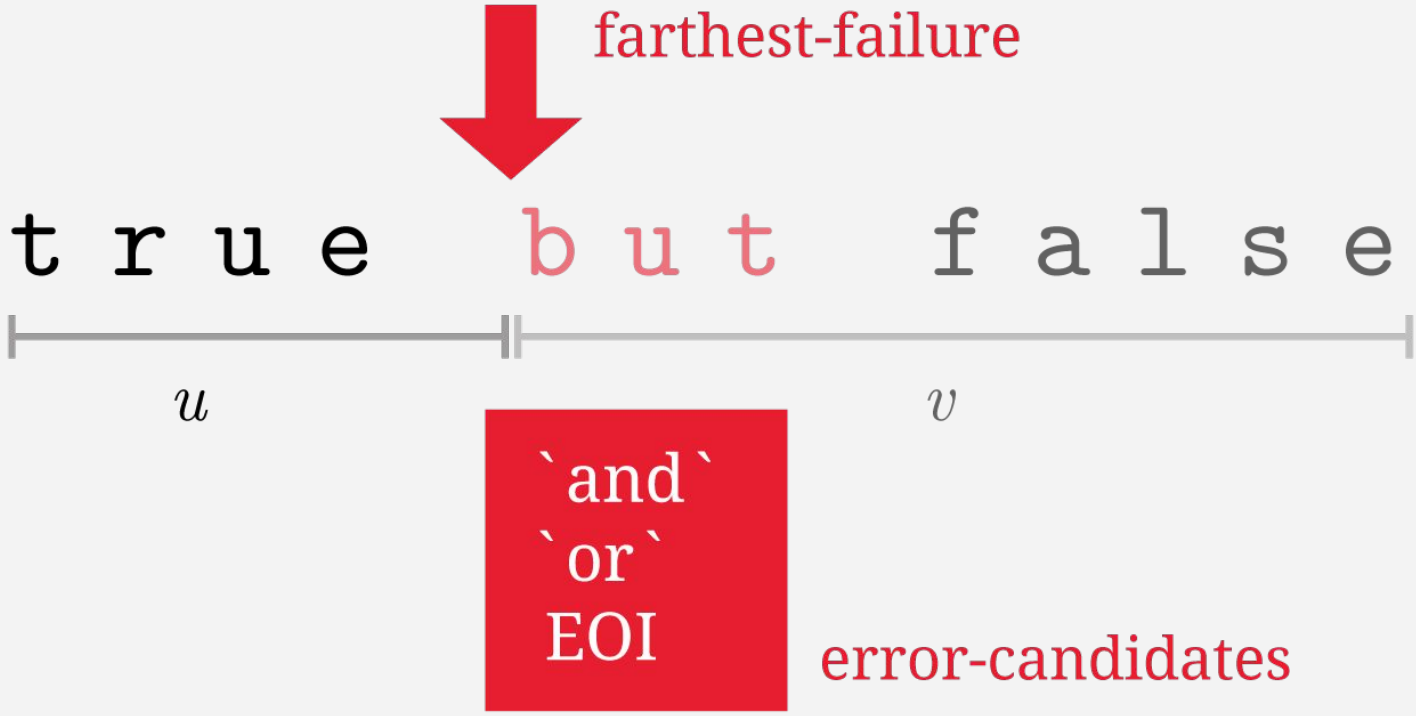
1. Error-Detection

Problems



Problems

Problems


farthest-failure
t r u e
b u t
f a l s e
u
v
`and`
`or`
EOI
error-candidates

Solutions

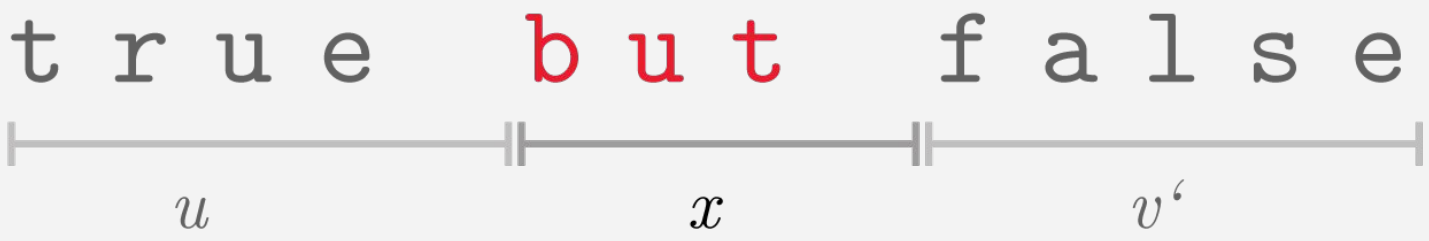
Restart the parsing in a **recording-mode**

- generate **parsing-records**
- detect the **farthest-failures**
- identify **error-candidates**

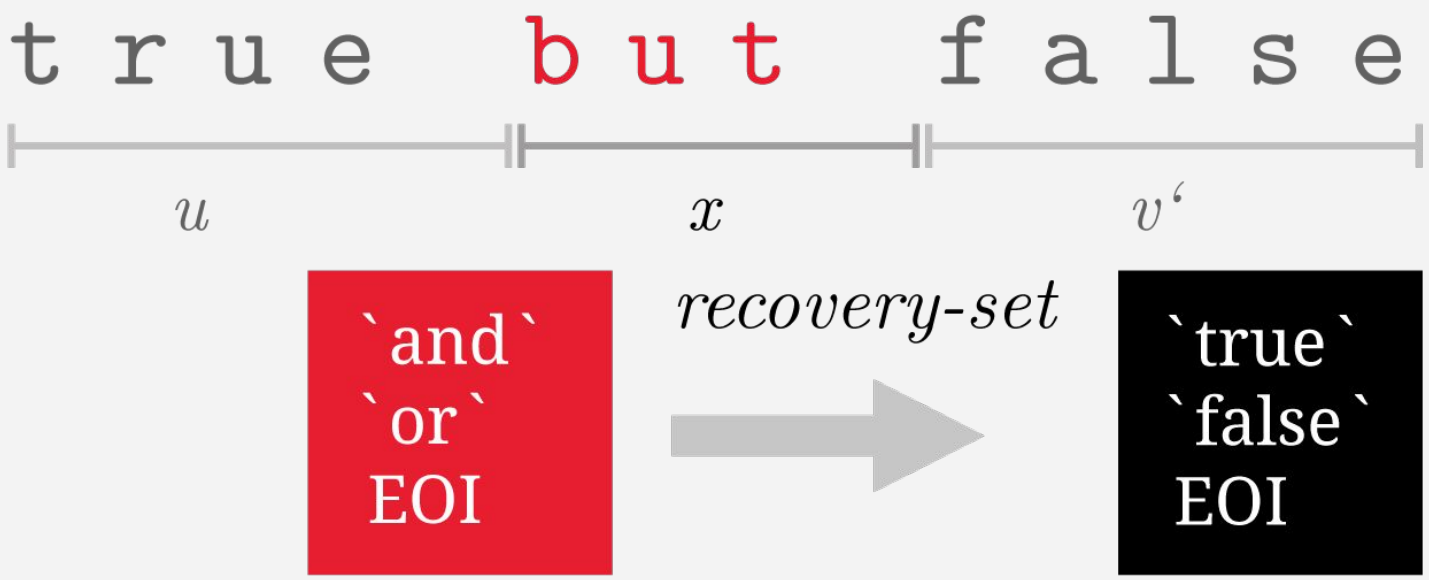
How to skip ...

2. Error-Recovery

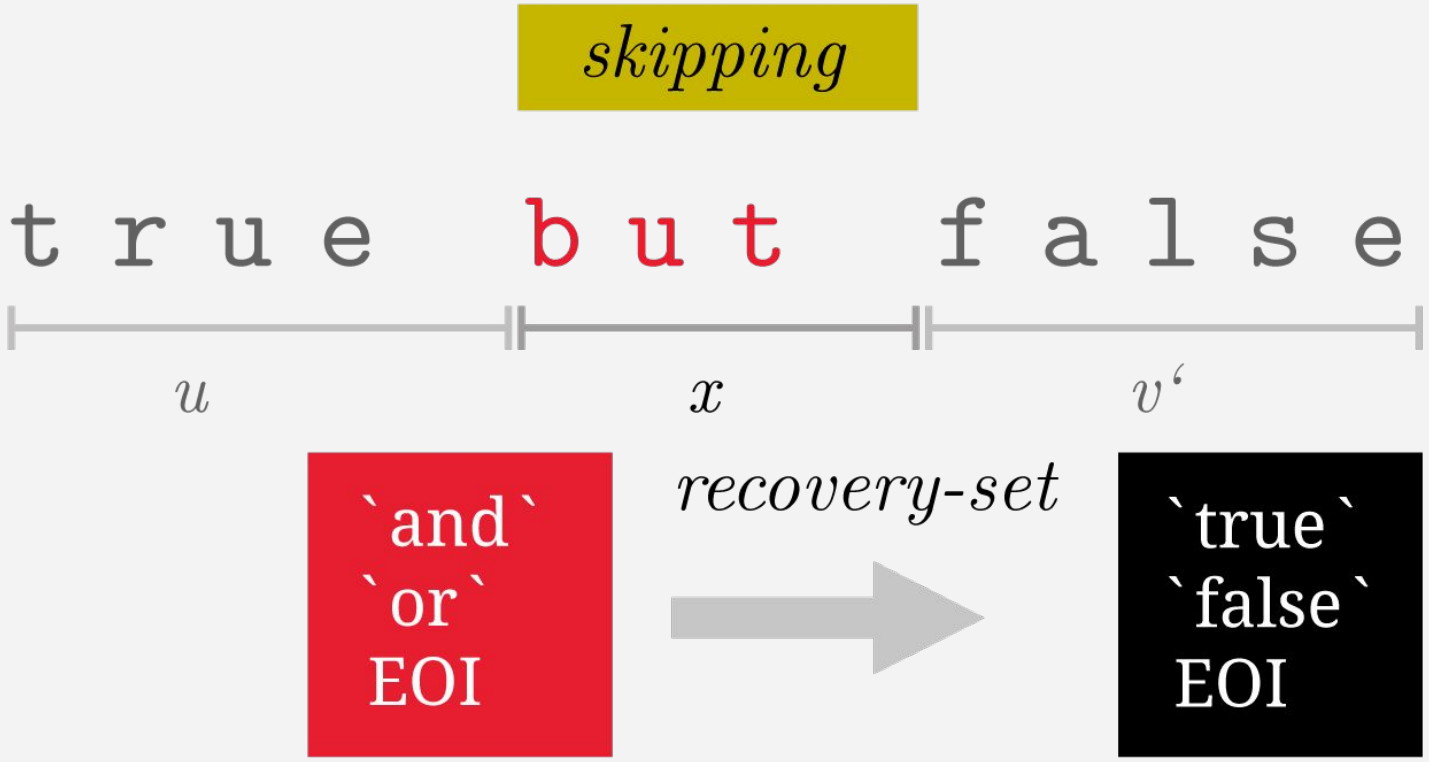
Problems



Problems



Problems


skipping
t r u e b u t f a l s e
u
x
v'
`and`
`or`
EOI
recovery-set
`true`
`false`
EOI

2. Error-Recovery

Solutions

- compute **follow-set**
- create a **recovery-set**
- use a **skipping-parser** to detect the **erroneous area**

How to modify the input

3. Error-Repair

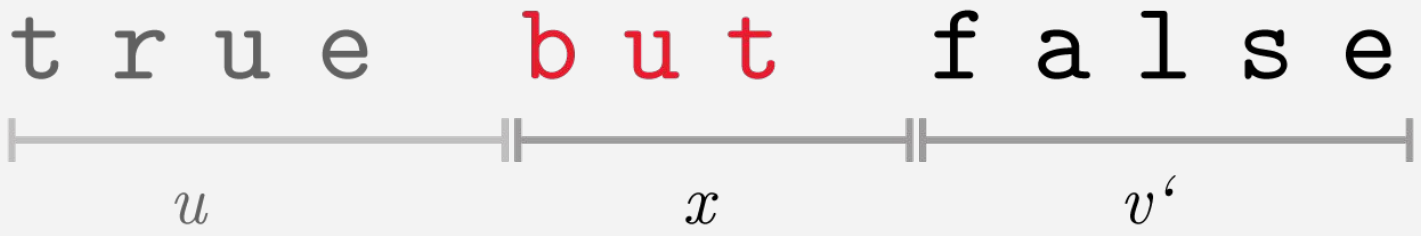
3. Error-Repair

Problems

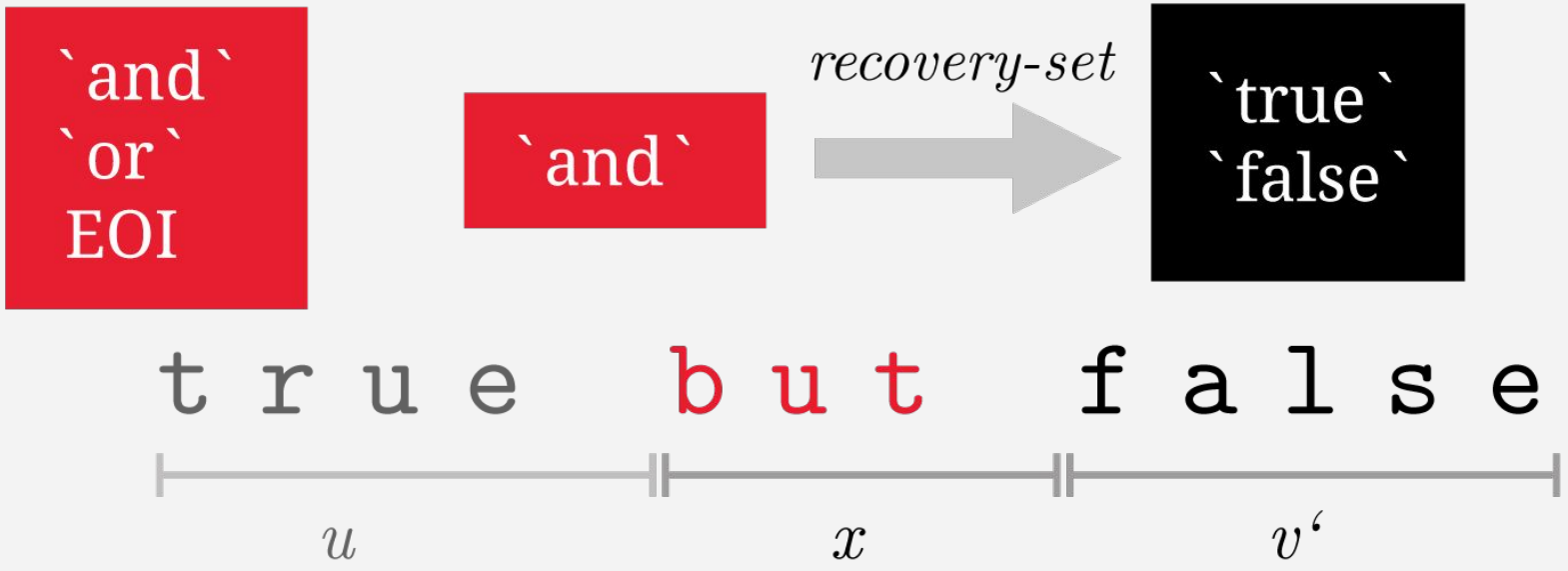
Three types of errors (transformations):

- Addition (Insertion)
- Omission (Removal)
- Substitiution (Substitution)
- how to **classify**

Problems

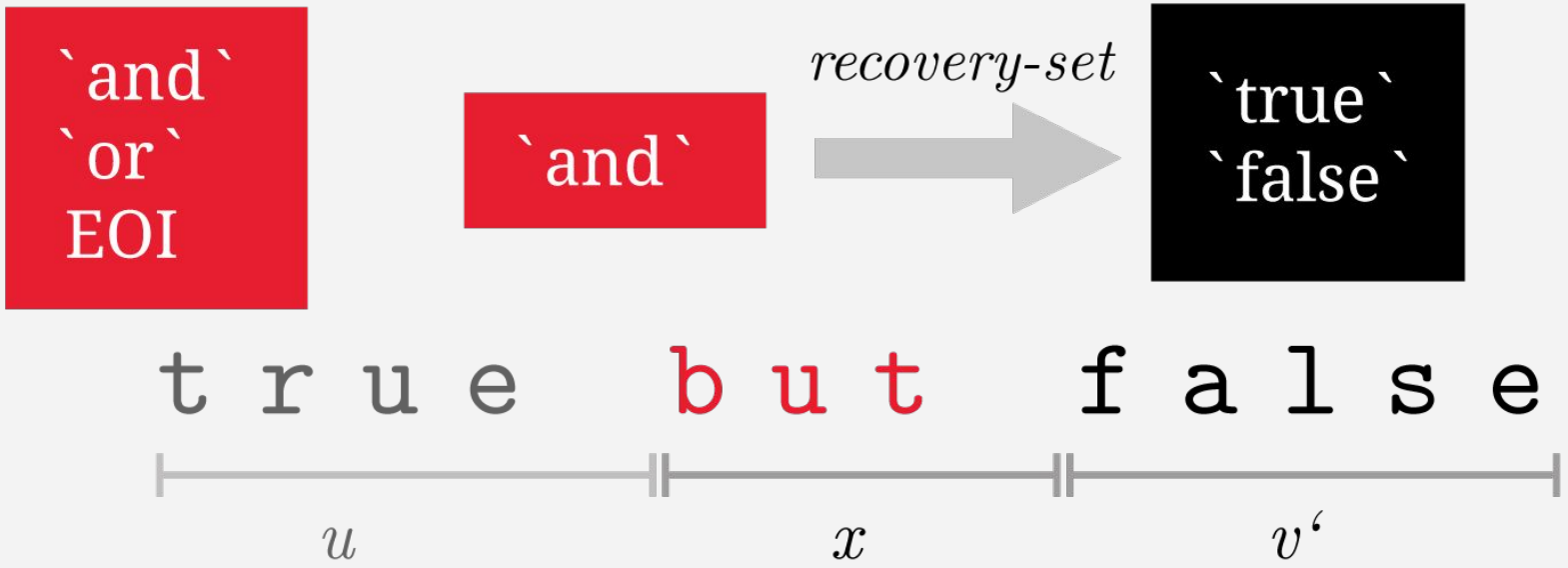


Problems



3. Error-Repair

Problems



It seems ***plausible*** to ***substitute*** *x* with ***and***.
It seems ***plausible*** to ***substitute*** *x* with ***or***.

3. Error-Repair

Solutions

- match the **recovery-set**
- decide which **transformations are plausible**

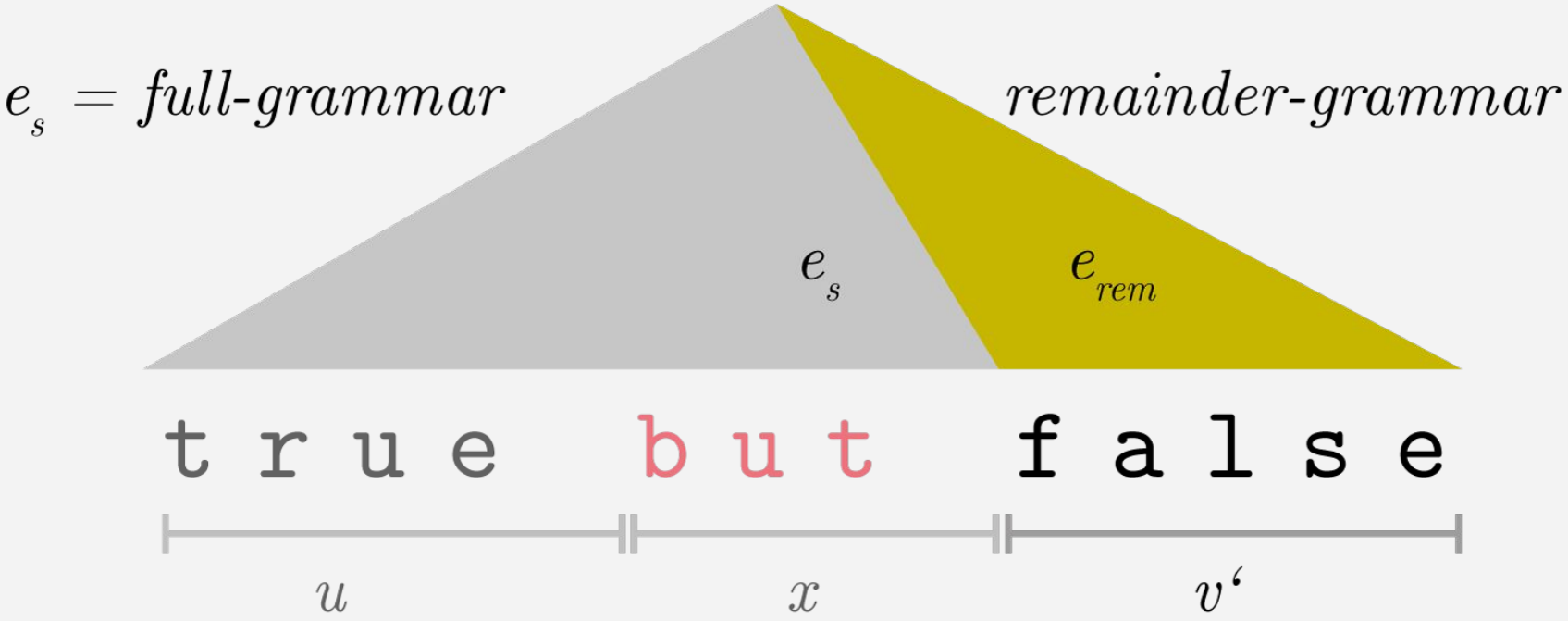
How to resume the parsing...

4. Remainder Processing

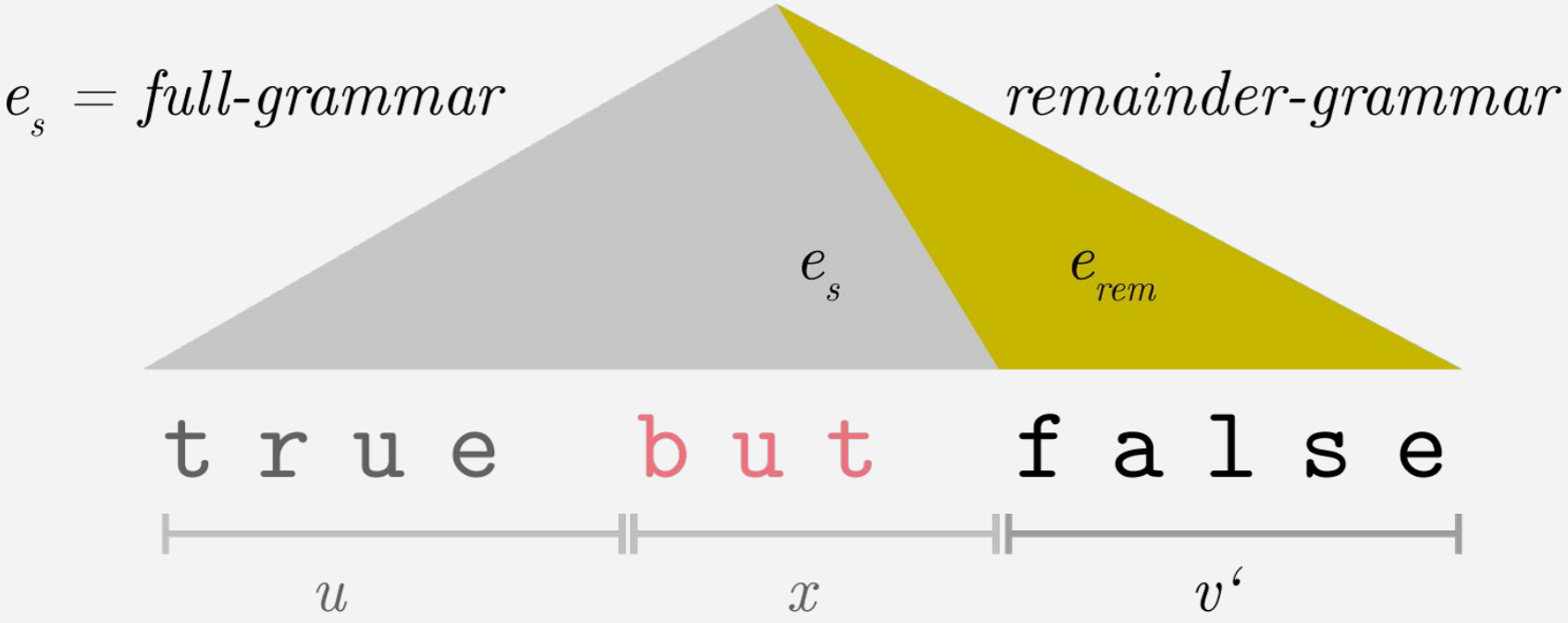
Problems

4. Remainder Processing

Problems



Problems



It was ***correct*** to ***substitute*** but with and.

Solutions

- compute a **remainder-grammar**
- **recursively invoke the error-handling** on the remaining input

Achievements

5. Results

Concrete Error-Handling-Schemes

We created **two implementations**

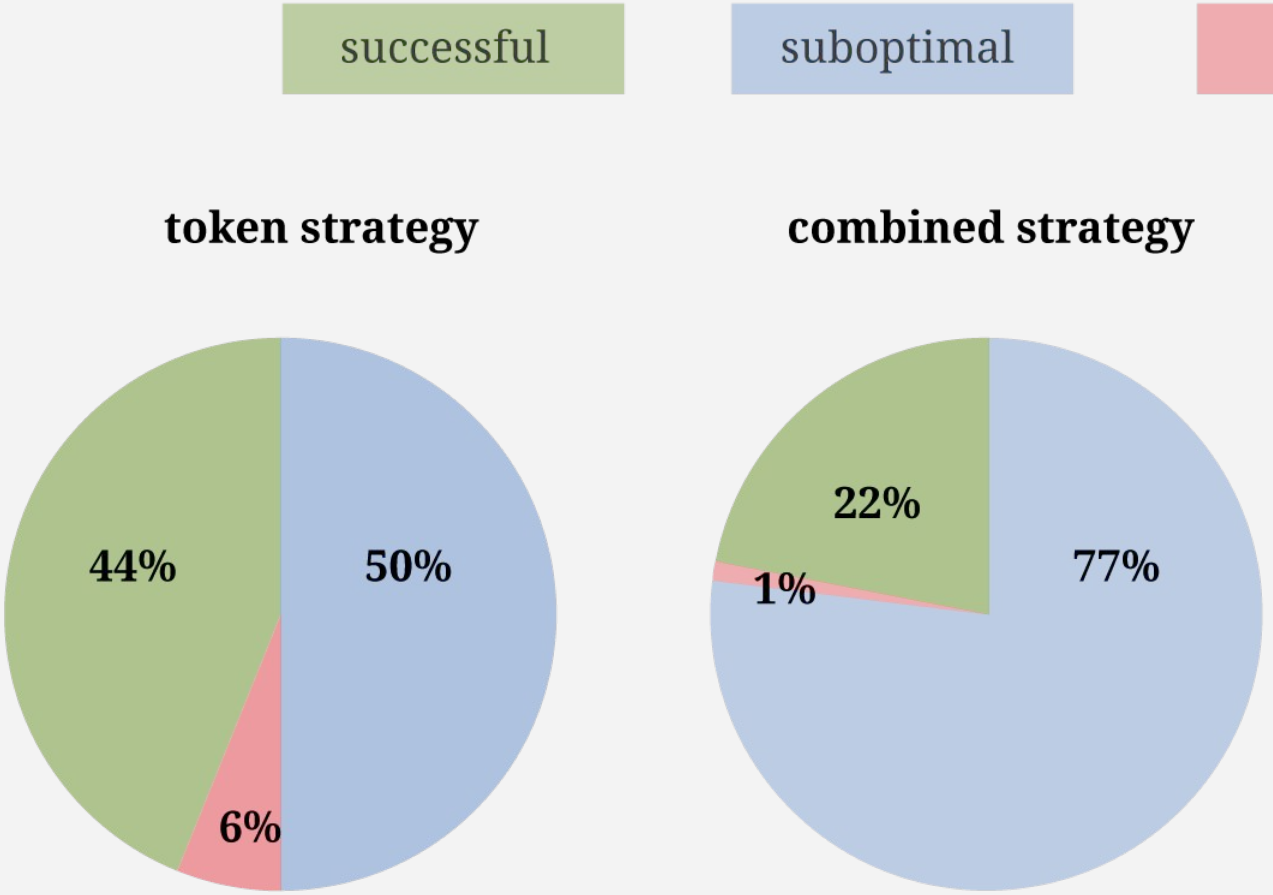
- a **lexical** error-handling-scheme
- a **token** error-handling-scheme
- we **combined both**

Numbers

Evaluated on a corpus of **Json files** with an **arbitrary number of errors**

	lexical strategy			token strategy			combined strategy		
	successful	suboptimal	failed	successful	suboptimal	failed	successful	suboptimal	failed
lexical errors	51	137	92	125	138	17	69	209	2
token errors				128	149	11	61	226	1
mixed errors				127	143	15	58	225	2

Numbers


successful
suboptimal
failed
token strategy
44%
50%
6%
combined strategy
22%
1%
77%

Achievements and Caveats

6. Conclusion

Achievements

- error-handling for **an arbitrary number** of **non-consecutive errors**
- work around **PEGs' limitations**
- ways to express parsing-combinators' **state** and **partial grammars**

6. Conclusion

Caveats & Future Work

- **consecutive errors**
- **prioritization** of repair
- **performance**
- no **editor integration**

Bring it on!

Questions

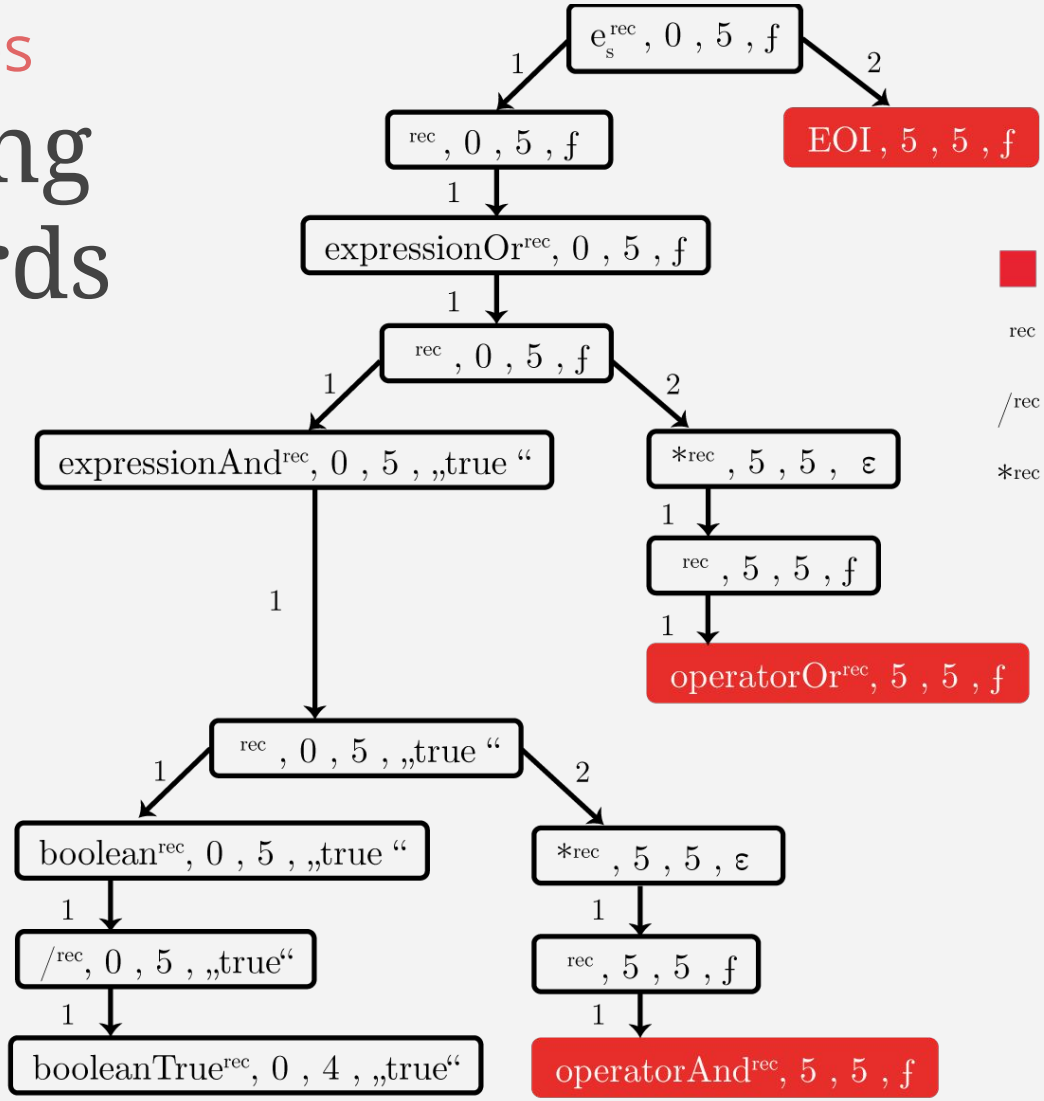
Examples

Failure Location

```
{  ← failure: " '[' expected at position 0"
      "object" : {
        "key" : "valuePairs"
      }
    , "string" : "hello"
    , "boolean" : true
    , "null"    : null
    , "integer": 100
    , "float"   : 10.3
    , "array"   : [
          "comma"
        , "separated"
          "values"  ← farthest-failure: " ',' expected ... "
    ]
}
```

Examples

Parsing Records



error-candidates

rec recording sequence

/rec recording choice

*rec recording repetition

Examples

Larger Errors

```
{ “string”                }
```

Examples

Valid Prefixes

```
"string", true ]
```

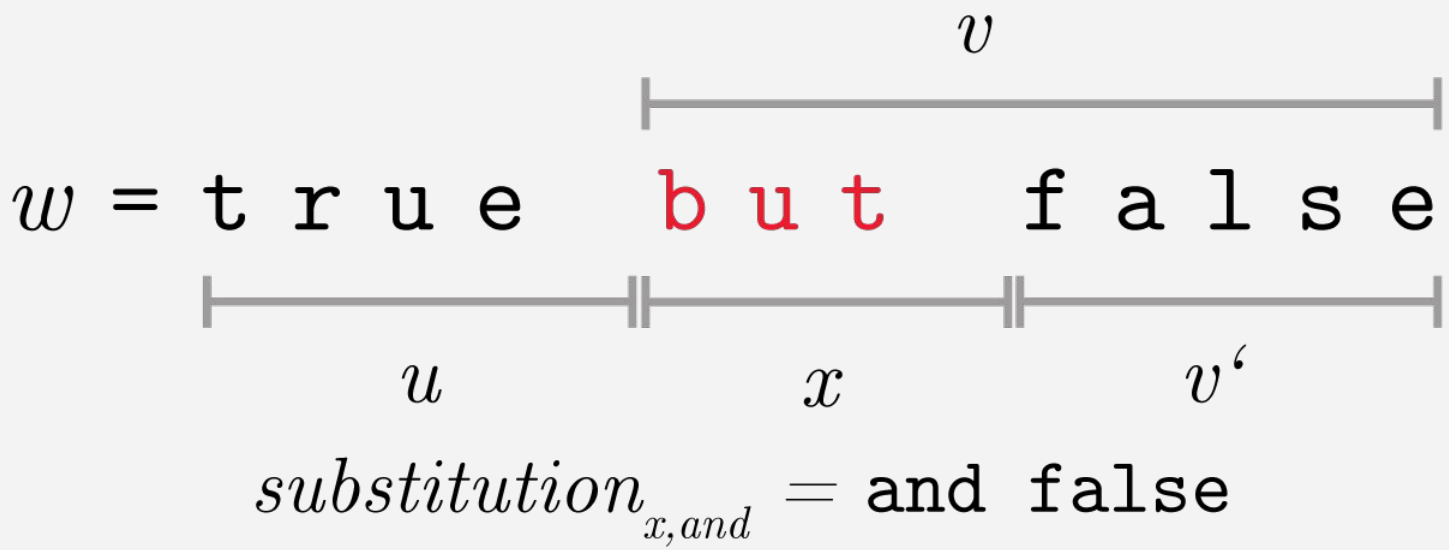
Examples

Man vs. Machine

```
"string" ]
```

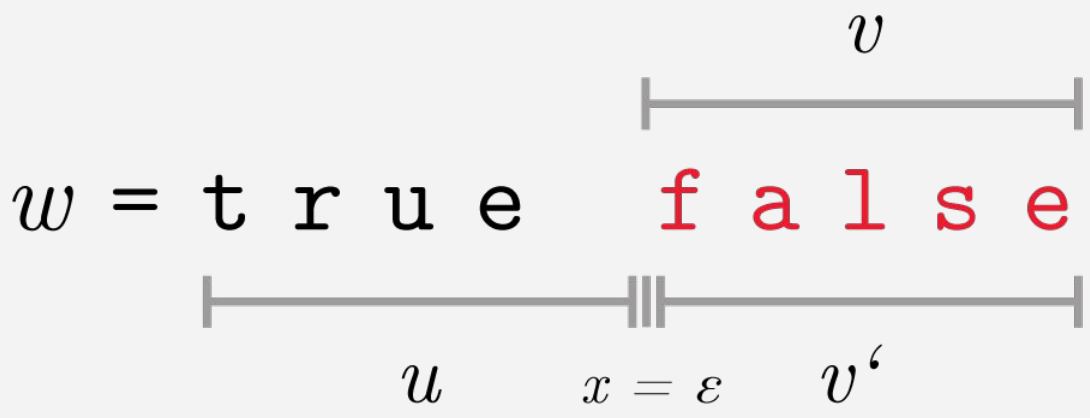
Examples

Substitution



Examples

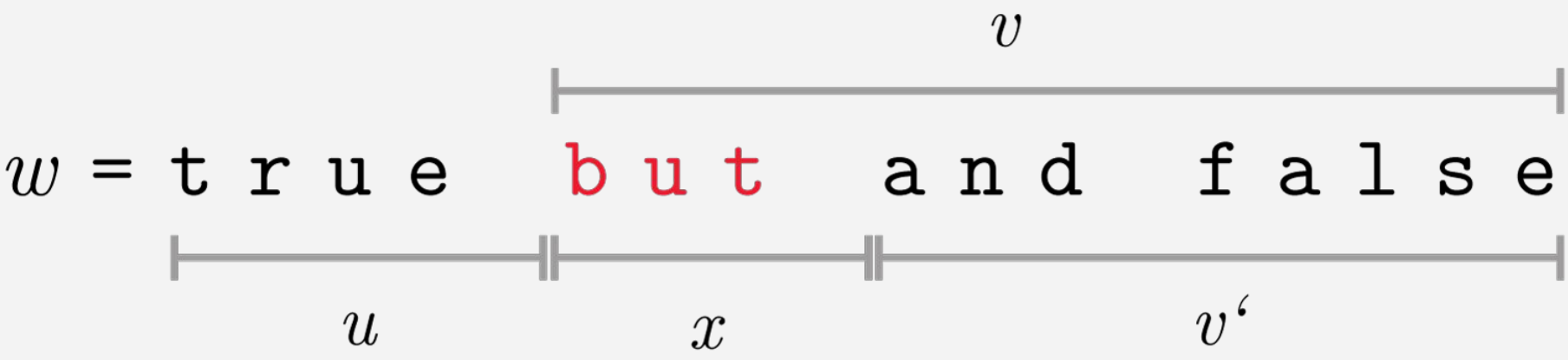
Insertion



$$insertion_{and} = substitution_{x,and} = \texttt{and false}$$

Examples

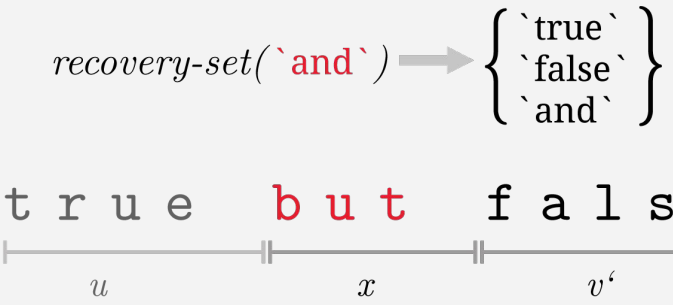
Removal



$$removal_x = substitution_{x,\varepsilon} = v‘ = \texttt{and false}$$

Examples

Configuration



choice

sequence

repetition

BooleanLogicGrammar

expressionOr

EOI

expressionAnd

operatorOr

boolean

booleanTrue

booleanFalse

operatorAnd

Examples

Remainder-Grammar

remainder-grammar	c
rem(e_s, c')	$1\ c'$
rem(expressionOr EOI, c')	$1\ c'$
rem(expressionOr, c') EOI	$1\ c'$
rem(expressionAnd (operatorOr expressionAnd)*, c') EOI	$1\ c'$
rem(expressionAnd, c') (operatorOr expressionAnd)* EOI	$1\ c'$
rem(boolean (operatorAnd boolean)*, c') (operatorOr expressionAnd)* EOI	$2\ c'$
rem((operatorAnd boolean)*, c') (operatorOr expressionAnd)* EOI	$1\ c'$
rem(operatorAnd boolean, c') (operatorAnd boolean)* (operatorOr expressionAnd)* EOI	$2\ c'$
rem(boolean, c') (operatorAnd boolean)* (operatorOr expressionAnd)* EOI	$1\ c'$
rem(booleanTrue / booleanFalse, c') (operatorAnd boolean)* (operatorOr expressionAnd)* EOI	$2\ c'$
rem(booleanFalse, c') (operatorAnd boolean)* (operatorOr expressionAnd)* EOI	$()$
booleanFalse (operatorAnd boolean)* (operatorOr expressionAnd)* EOI	

Examples

Compound Strategy

```
t r u l e   b u t   f a l s e
```

Examples

Compound Strategy

t r u l e b u t f a l s e

1. Lexical-Strategy fails

t r u e b u t f a l s e

Examples

Compound Strategy

```
t r u l e   b u t   f a l s e
```

1. Lexical-Strategy fails

```
t r u   e   b u t   f a l s e
```

2. Token-Strategy succeeds

```
t r u e
```

Examples

Compound Strategy

```
t r u l e   b u t   f a l s e
```

1. Lexical-Strategy fails

```
t r u   e   b u t   f a l s e
```

2. Token-Strategy succeeds

```
t r u e
```

3. Compound-Strategy succeeds

```
t r u   e   a n d   f a l s e
```

Examples

Why do we fail?

```
{
  “key” : true
  “error”
}
```

Examples

Why do we fail?

```
{
  "key" : true
  "error" ',' or '}'
}
```

Examples

Performance: The call stack

```
{
   “object” : {
      “object” : {
         “object” : {
            “object” : {
               “object” : “cool”
            }
         }
      }
   }
}
```