Declarative Syntax Definition

pretty printing

Guido Wachsmuth

Assessment

last lecture

Assessment

last lecture

Compare SDF with the plain formalism of context-free grammars.

Assessment

last lecture

Compare SDF with the plain formalism of context-free grammars.

Discuss additional description means provided by SDF and why they are needed in compiler construction.

Assessment

last lecture

Compare SDF with the plain formalism of context-free grammars.

Discuss additional description means provided by SDF and why they are needed in compiler construction.

- regular expressions

Assessment

last lecture

Compare SDF with the plain formalism of context-free grammars.

Discuss additional description means provided by SDF and why they are needed in compiler construction.

- regular expressions
- layout

Assessment

last lecture

Compare SDF with the plain formalism of context-free grammars.

Discuss additional description means provided by SDF and why they are needed in compiler construction.

- regular expressions
- layout
- AST constructors

Assessment

last lecture

Compare SDF with the plain formalism of context-free grammars.

Discuss additional description means provided by SDF and why they are needed in compiler construction.

- regular expressions
- layout
- AST constructors
- follow restrictions

Assessment

last lecture

Compare SDF with the plain formalism of context-free grammars.

Discuss additional description means provided by SDF and why they are needed in compiler construction.

- regular expressions
- layout
- AST constructors
- follow restrictions
- annotations for associativity

Assessment

last lecture

Compare SDF with the plain formalism of context-free grammars.

Discuss additional description means provided by SDF and why they are needed in compiler construction.

- regular expressions
- layout
- AST constructors
- follow restrictions
- annotations for associativity
- priorities

Overview

today's lecture

Overview

today's lecture

plagues of traditional parsing algorithms

- paradise lost
- paradise regained

Overview

today's lecture

plagues of traditional parsing algorithms

- paradise lost
- paradise regained

pretty-printing

- from trees to text
- box layout in pretty-print tables

Overview

today's lecture

plagues of traditional parsing algorithms

- paradise lost
- paradise regained

pretty-printing

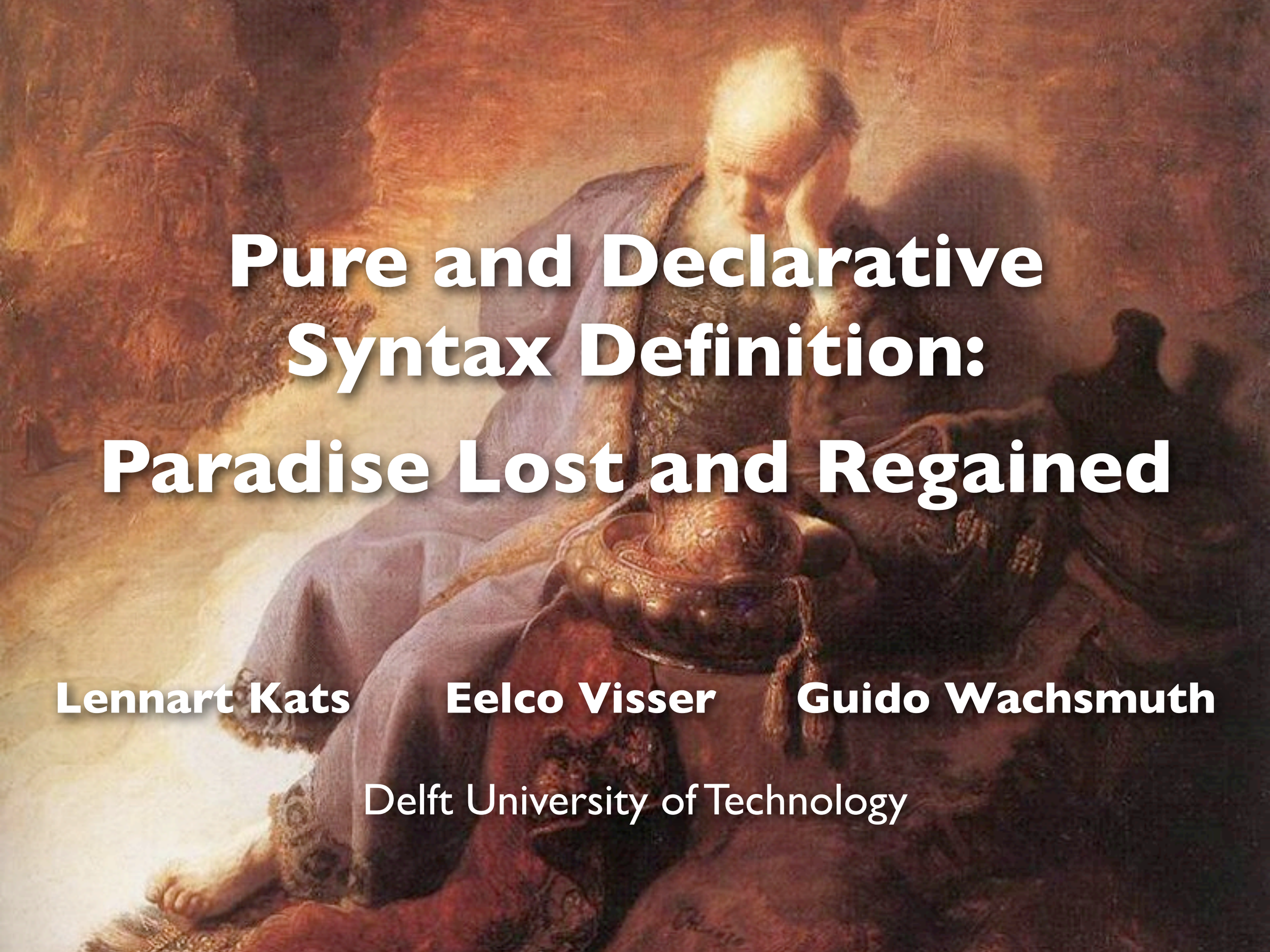
- from trees to text
- box layout in pretty-print tables

template language

- alternative to SDF
- generation of pretty-printing strategies
- generation of completion templates

I

paradise lost & regained



Pure and Declarative Syntax Definition: Paradise Lost and Regained

Lennart Kats **Eelco Visser** **Guido Wachsmuth**

Delft University of Technology

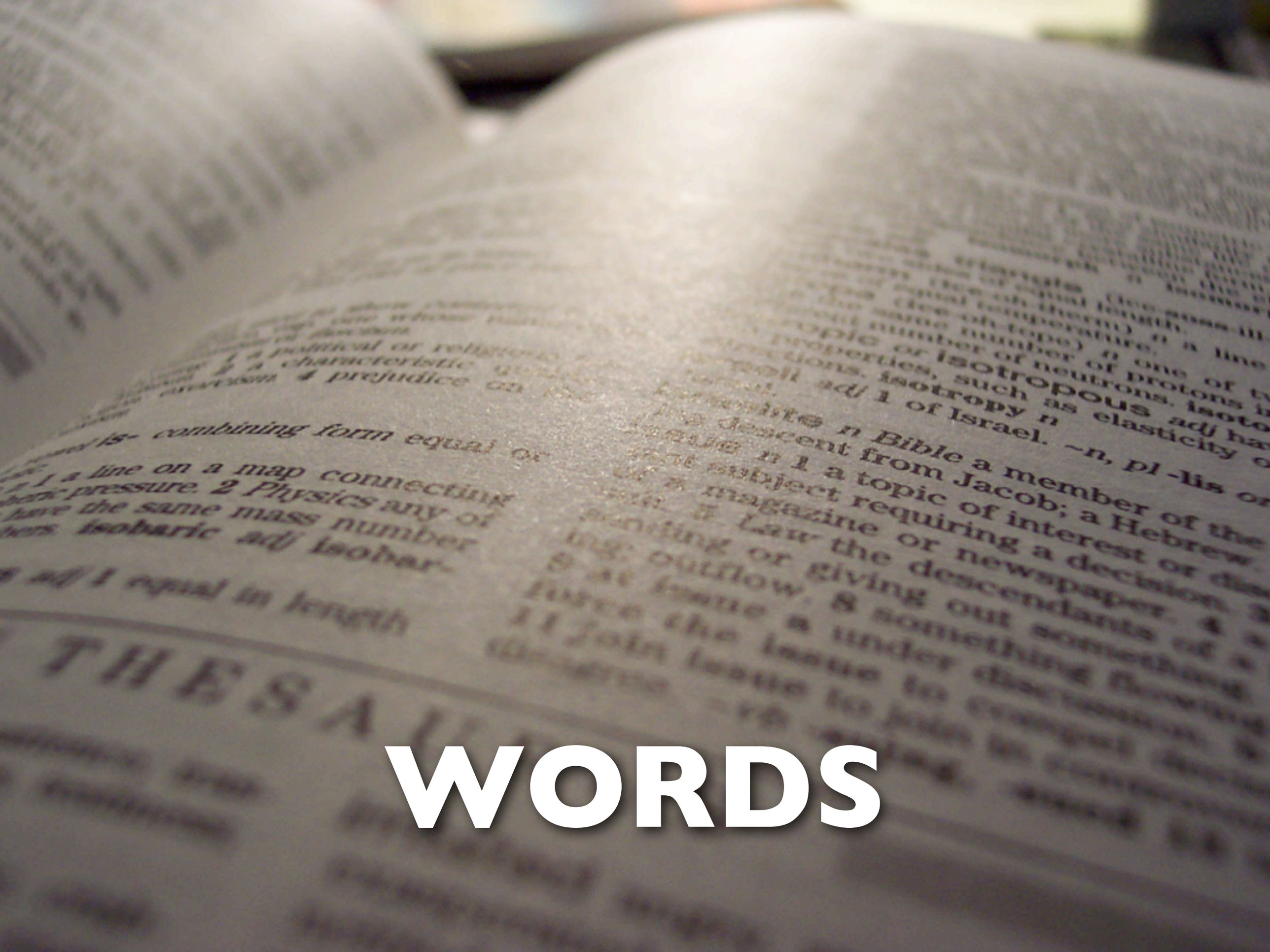
PARADISE

PARADISE **LOST**

PARADISE REGAINED

PARADISE **DENIED**

PARADISE


WORDS

TREES

LA DEUXIÈME ANNÉE

DE LATIN

Théorie 250 pages.

227 Exercices 100 pages.

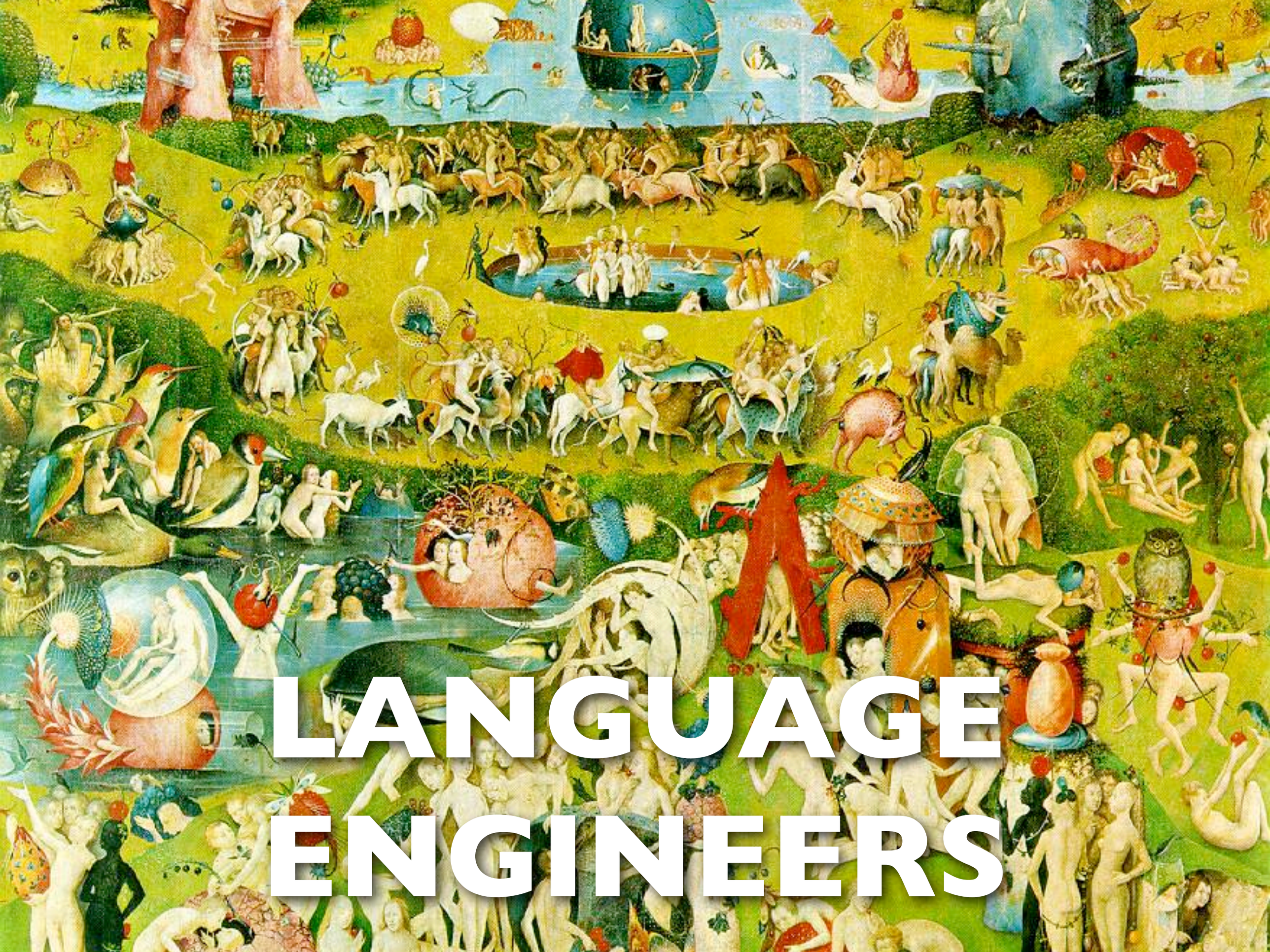
Lexiques

24 pages blanches pour notes et 4 cartes

Troisième édition.

Armand COLIN & Cie

GRAMMARS



LANGUAGE ENGINEERS

LANGUAGES

Software Construction and Data Structures with Ada 95
ADDISON WESLEY
KERNIGHAN · RITCHIE
THE C PROGRAMMING LANGUAGE
SECOND EDITION
PTR
STROUSTRUP
C++
THE C++ PROGRAMMING LANGUAGE
THIRD EDITION
Apple PRESS
The Dylan Reference Manual
Shalit
Addison Wesley
The Java™ Programming Language
Second Edition
Arnold Gosling
JAVA
The Little MLer
Felleisen and Friedman
ULLMAN
ELEMENTS OF ML PROGRAMMING
EDITION
Miranda™
The Craft of Functional Programming
Thompson
THIRD EDITION
Programming Perl
Wall, Christiansen & Orwant
Learning Python
& Ascher
Nelson
Systems Programming with Modula-3
DYBVIG
THE SCHEME PROGRAMMING LANGUAGE
ANSI SCHEME
SECOND EDITION
PTR
Guzdial
Rose
Squeak
Open Personal Computing and Multimedia

LA DEUXIÈME ANNÉE

DE LATIN

Théorie 250 pages.

227 Exercices 100 pages.

Lexiques

24 pages blanches pour notes et 4 cartes

Troisième édition.

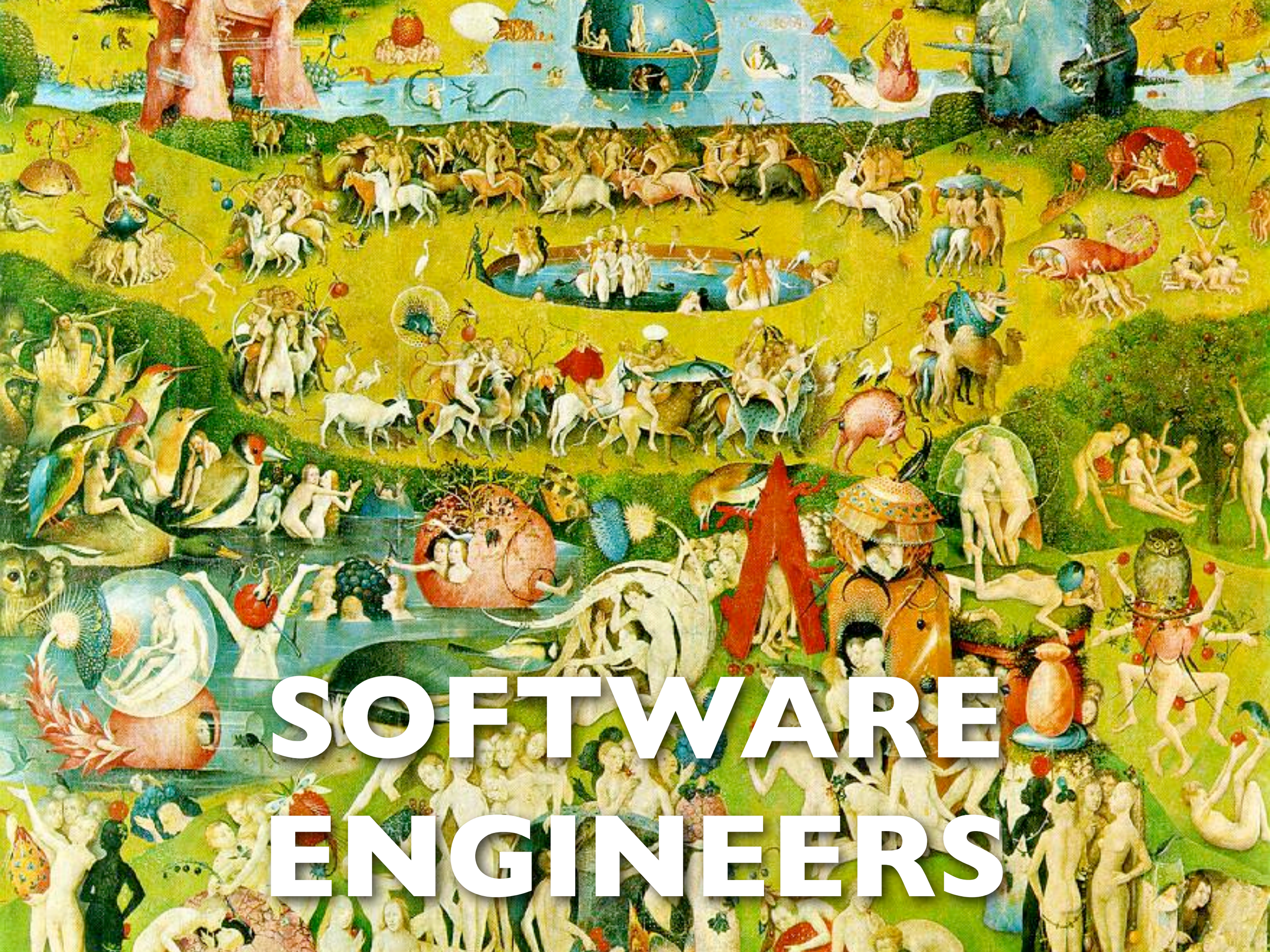
Armand COLIN & Cie

GRAMMARS

NATURAL

PURE

BEAUTIFUL



SOFTWARE ENGINEERS

LANGUAGE SOFTWARE

NOT
NATURAL

NOT
PURE

NOT
BEAUTIFUL

LA DEUXIÈME ANNÉE

DE LATIN

Théorie 250 pages.

227 Exercices 100 pages.

Lexiques

SYNTAX DEFINITIONS

Armand COLIN & Cie

NATURAL

PURE

BEAUTIFUL

THE **FALL**

PARSER
DEFINITIONS

PARADISE **LOST**

PAIN

SWEAT

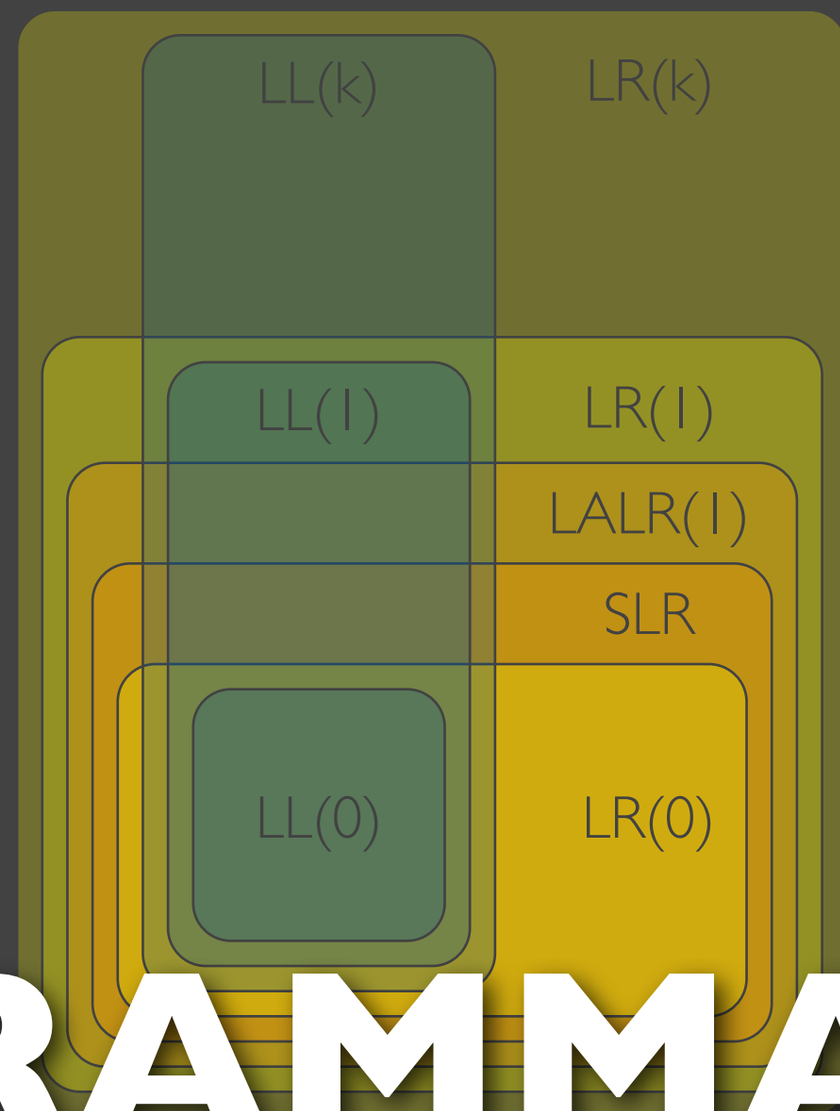
NOT
NATURAL

NOT
PURE

NOT
BEAUTIFUL

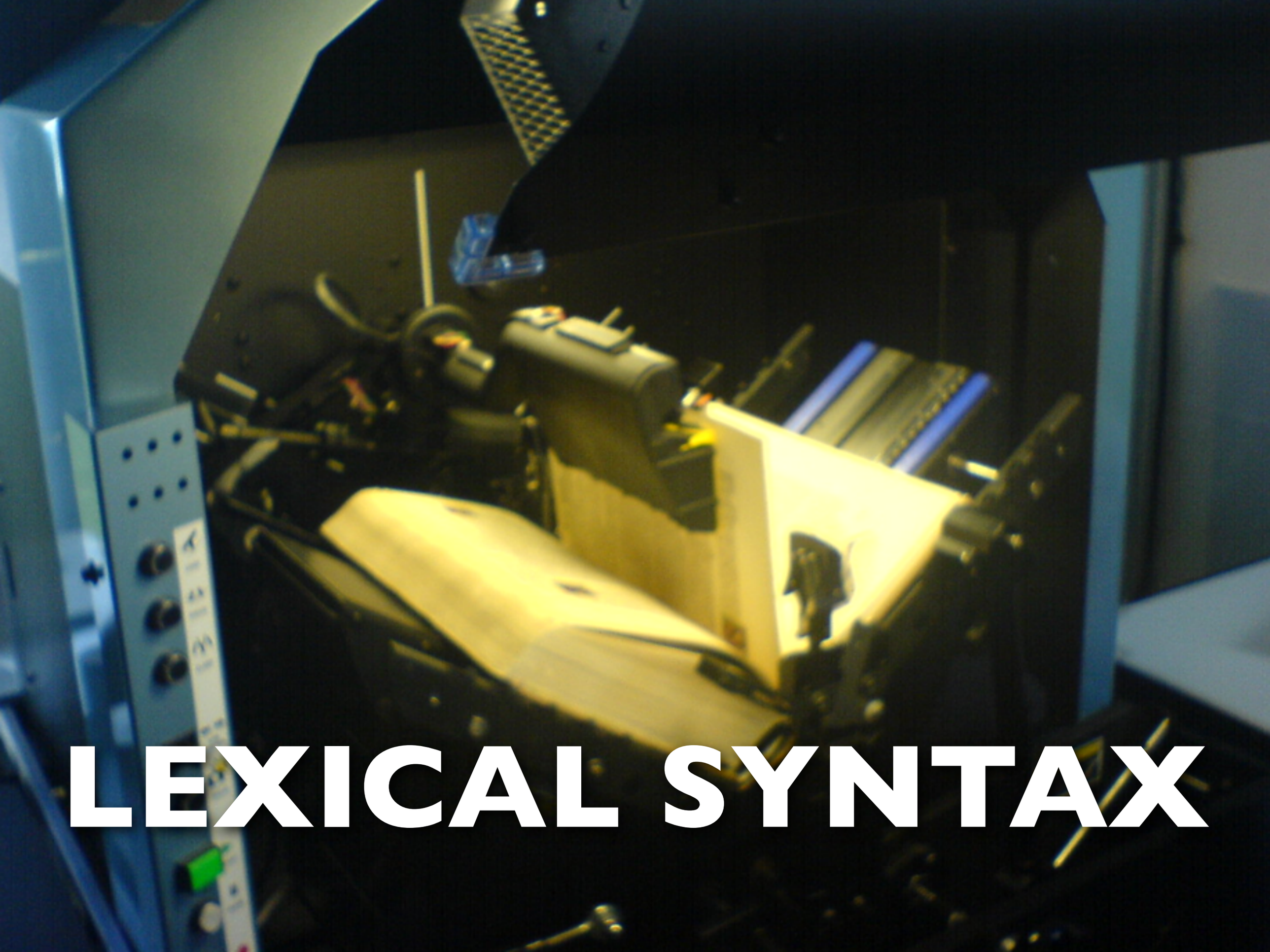
THE **PLAGUES**

context-free grammars



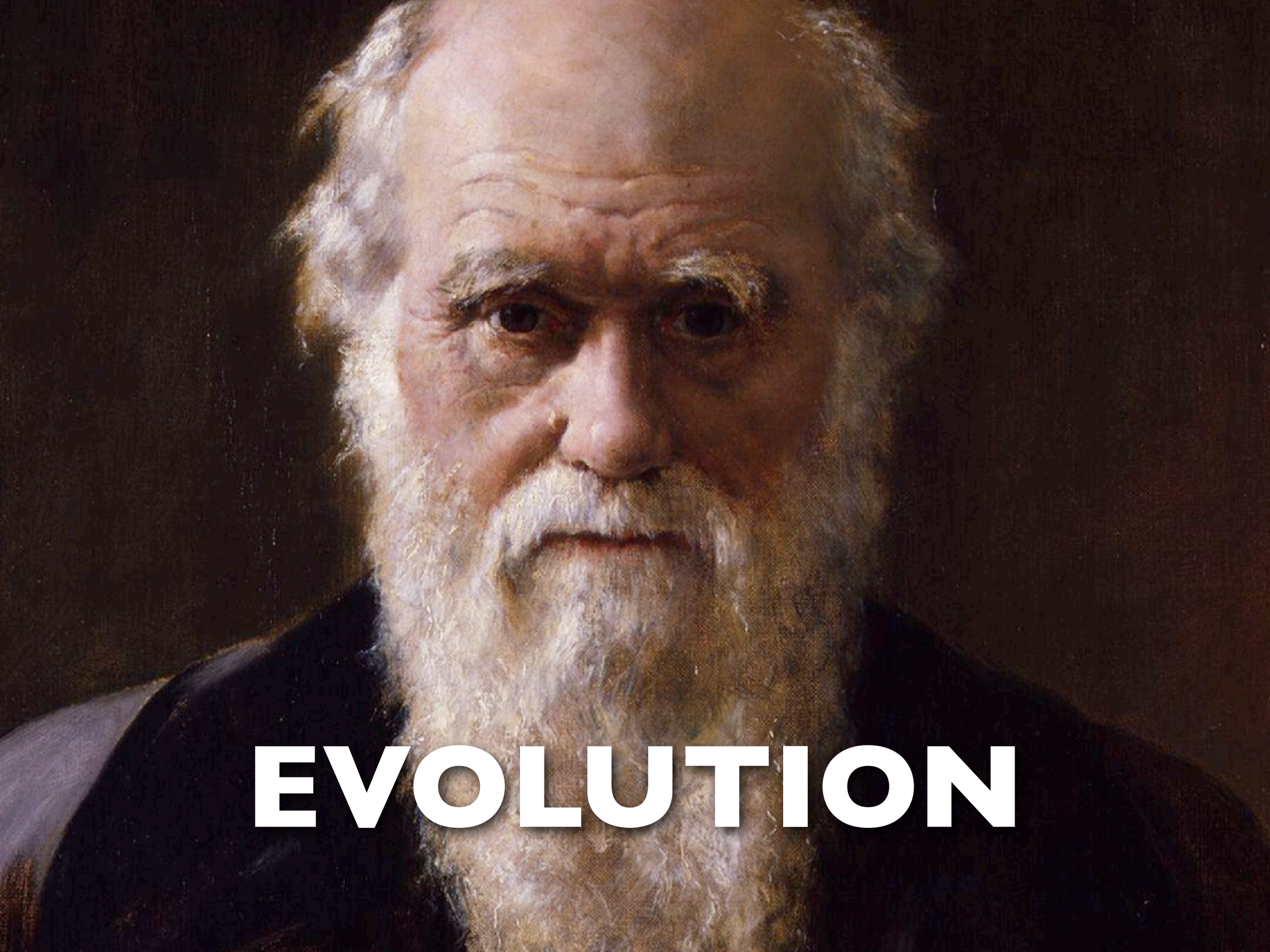
GRAMMAR CLASSES

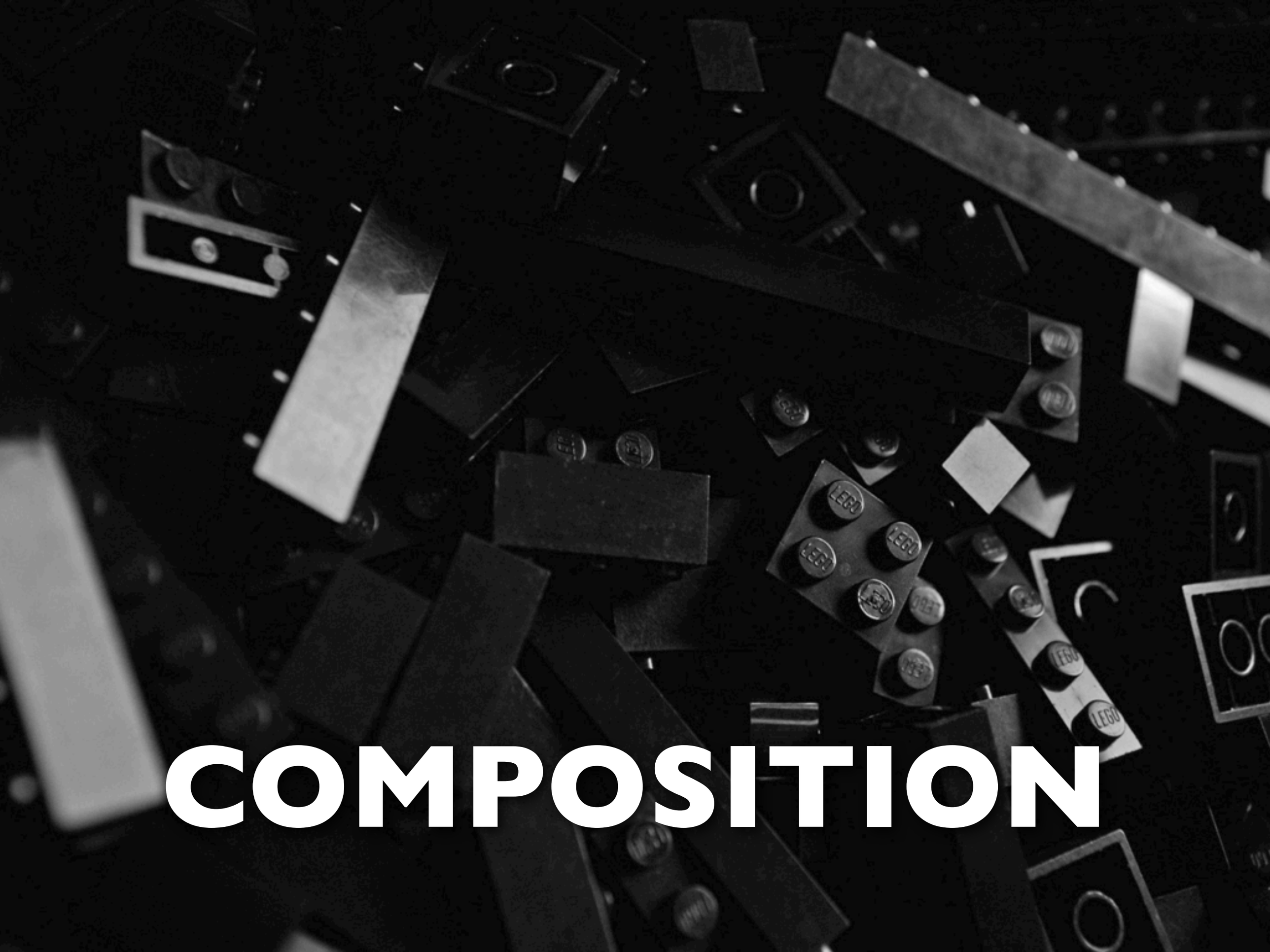
DISAMBIGUATION



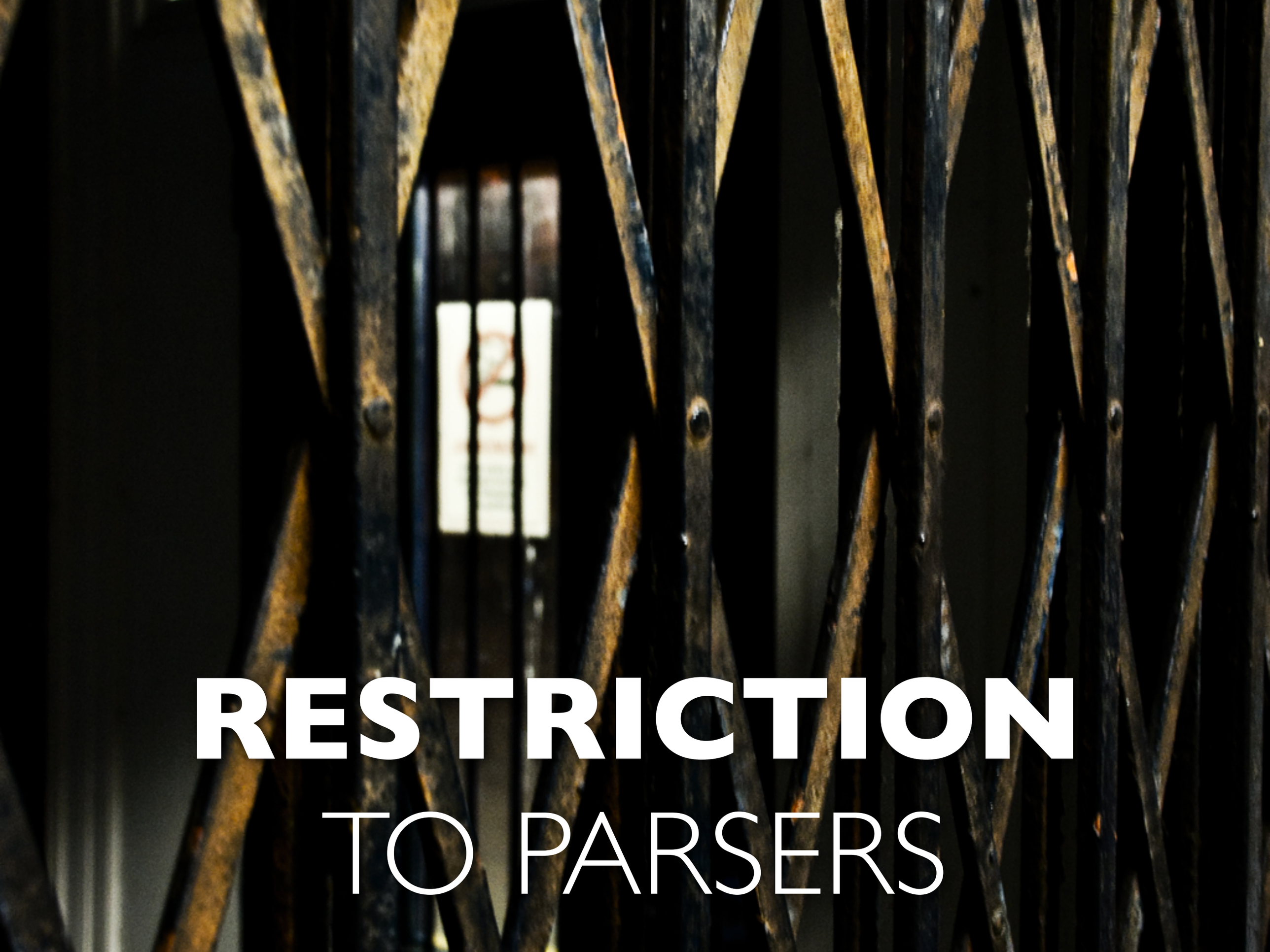
LEXICAL SYNTAX

TREE CONSTRUCTION


EVOLUTION

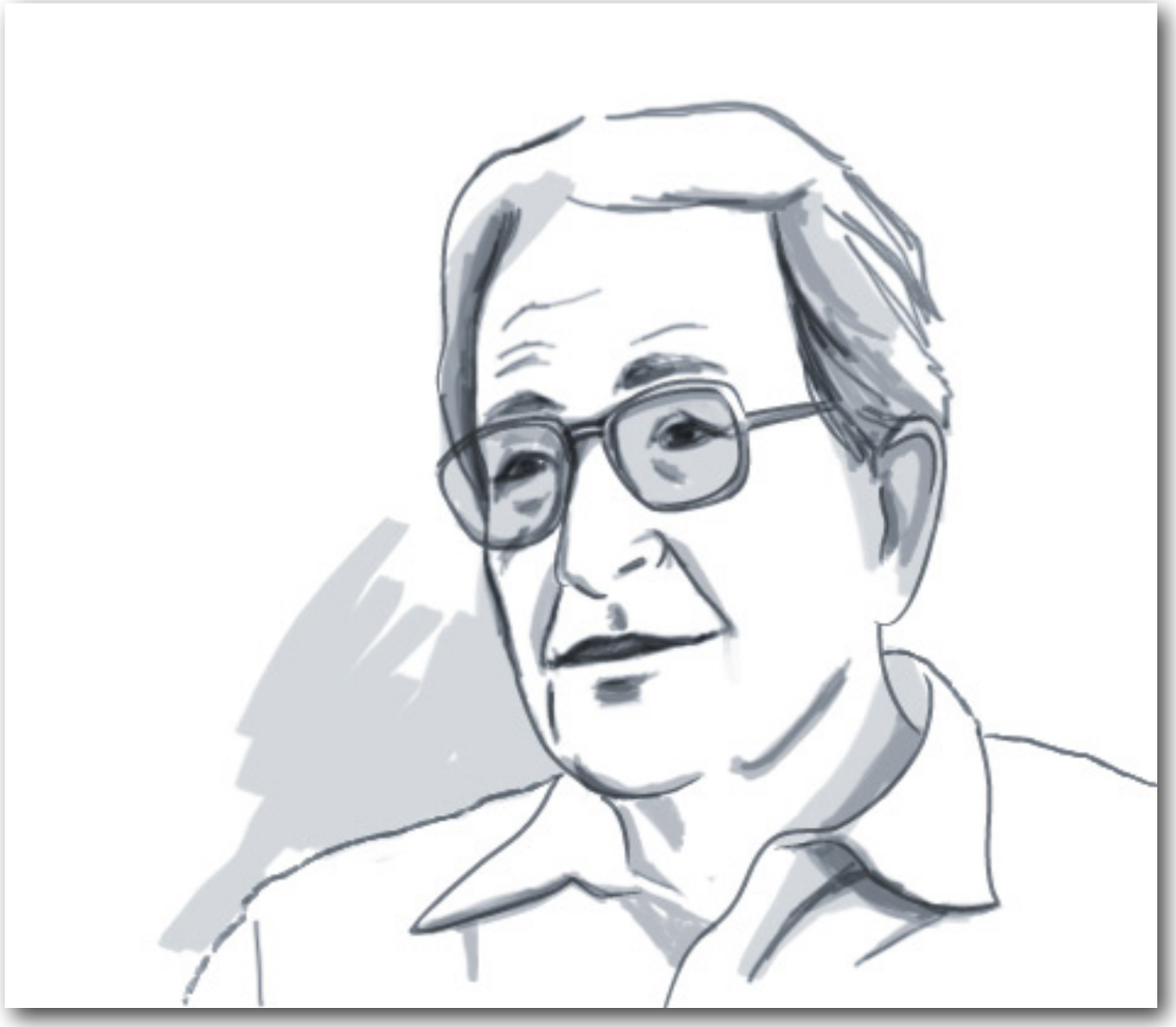


COMPOSITION



RESTRICTION
TO PARSERS

PARADISE



WORDS WERE MADE THROUGH

GRAMMARS

LA DEUXIÈME ANNÉE

DE LATIN

Théorie 250 pages.

227 Exercices 100 pages.

Lexiques

24 pages blanches pour notes et 4 cartes

Troisième édition.

Armand COLIN & Cie

GRAMMARS

Num → Digit Num

Num → Digit

Digit → “0”

Digit → “1”

Num → Digit Num

Num → Digit

Digit → “0”

Digit → “1”

production rules

Num → Digit Num

Num → Digit

Digit → “**0**”

Digit → “**1**”

terminal symbols

production rules

Num → Digit Num

Num → Digit

Digit → “0”

Digit → “1”

nonterminal symbols

terminal symbols

production rules

Num → Digit **Num**

Num → Digit

Digit → “0”

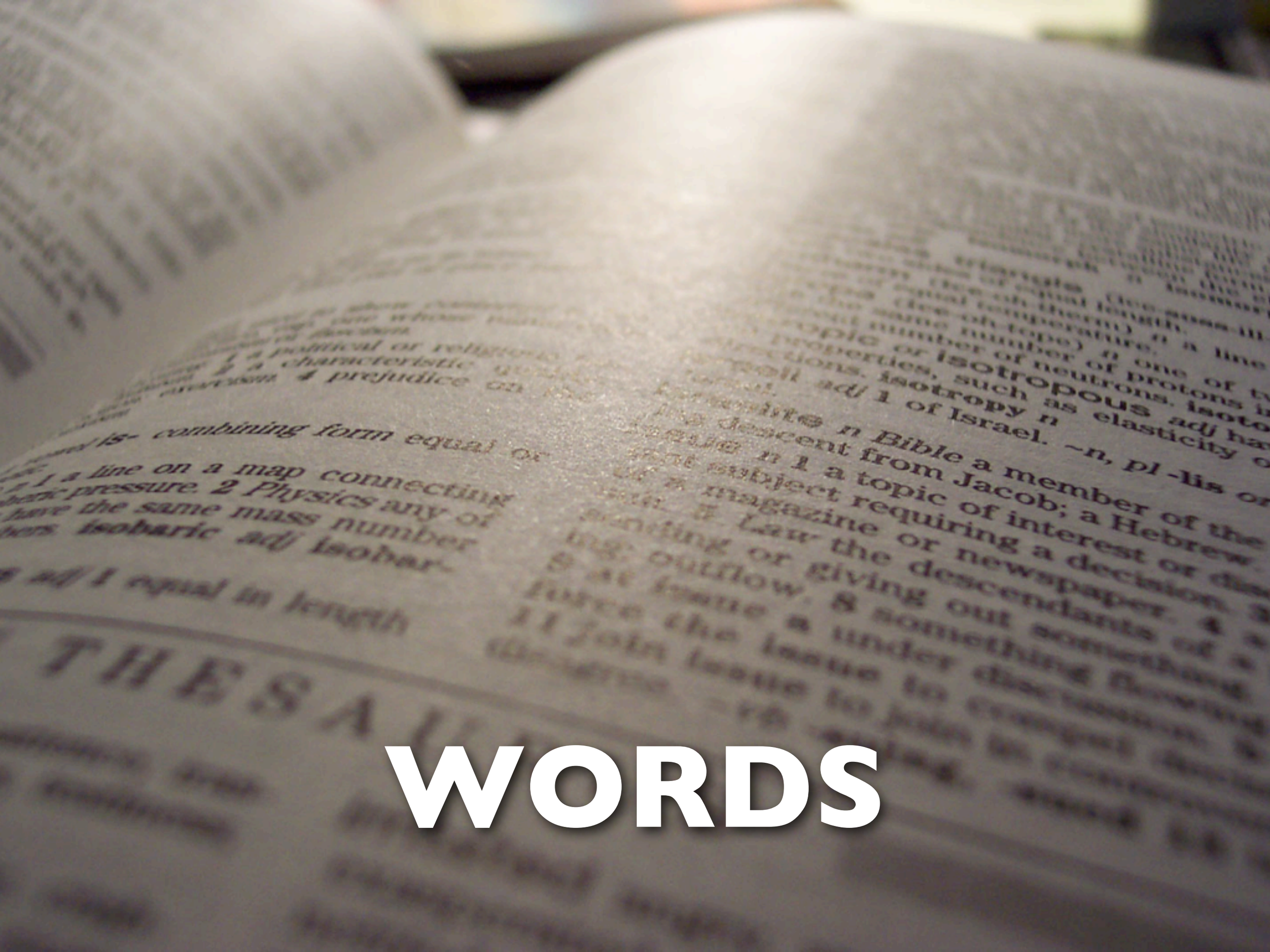
Digit → “1”

nonterminal symbols

terminal symbols

production rules

start symbol


WORDS

Num

Num

Digit Num

Num → Digit Num

Num

Digit **Num**

Digit Digit

Num → Digit Num

Num → Digit

Num

Digit Num

Digit **Digit**

Digit 0

Num → Digit Num

Num → Digit

Digit → “0”

Num

Digit Num

Digit Digit

Digit 0

| 0

Num → Digit Num

Num → Digit

Digit → “0”

Digit → “|”

SENTENCES

Exp → Exp "+" Exp

Exp → Exp "*" Exp

Exp → Num

production rules

Exp → Exp “**+**” Exp

Exp → Exp “*” Exp

Exp → **Num**

terminal symbols

production rules

Exp → Exp "+" Exp

Exp → Exp "*" Exp

Exp → Num

nonterminal symbols

terminal symbols

production rules

Exp → Exp "+" Exp

Exp → Exp "*" Exp

Exp → Num

nonterminal symbols

terminal symbols

production rules

start symbol

Exp

Exp

Exp + Exp

Exp → Exp “+” Exp

Exp

Exp + Exp

Exp * Exp + Exp

Exp → Exp “+” Exp

Exp → Exp “*” Exp

Exp

Exp + Exp

Exp * Exp + Exp

3 * Exp + Exp

Exp → Exp "+" Exp

Exp → Exp "*" Exp

Exp → Num

Exp

Exp + Exp

Exp * Exp + Exp

3 * **Exp** + Exp

3 * 7 + Exp

Exp → Exp "+" Exp

Exp → Exp "*" Exp

Exp → Num

Exp → Num

Exp	Exp → Exp “+” Exp
Exp + Exp	Exp → Exp “*” Exp
Exp * Exp + Exp	Exp → Num
3 * Exp + Exp	Exp → Num
3 * 7 + **Exp**	Exp → Num
3 * 7 + 21	

THEY MADE **LANGUAGES**
BY MAKING GRAMMARS

LA DEUXIÈME ANNÉE

DE LATIN

Théorie 250 pages.

227 Exercices 100 pages.

Lexiques

24 pages blanches pour notes et 4 cartes

GRAMMAR

Troisième édition.

Armand COLIN & Cie

nuā assistit. Exemplū acci
pite frēs meī malī ⁊ lon
ganimitatis laboris ⁊ pa
tiētiae: ꝓphetas q̄ locuti sūt
ī nōīe dñi. Ecce beatificā
mus eos qui sustinuerūt.
Sufferentiā iob audistis:
⁊ finē dñi vidistis: q̄m mi
sericors est dñs ⁊ miserator.
Ante oīa autē frēs mei no
te iurare neq; p celū neq;
terrā: neq; aliud q̄ēcūq; iu
ramentū. Sit autē sermo ve
ster est est nō nō: ut nō sub
dicio cadatis. Tristatur
q̄ aliq̄s vestrūm: oret equo
animo ⁊ psallat. Infirmā
tur quis in vobis: inducat
os ecclesie ⁊ orent super e
um ungētes eū oleo ī nōīe
dñi: ⁊ oratio fidei saluabit in
firmū: ⁊ alleuiabit eū dñs:
et si peccata fuerit dimittentur
ei. Confitemini ergo alterū

menes ei. Et rursū orauit
⁊ celū dedit pluuiā: ⁊ terra de
dit fructū suū. Frēs mei si
quis ex vobis errauerit a
veritate ⁊ cōuerterit quis
eū: scire debet q̄m qui con
uertit peccatorem ab errō
re viē sue saluabit aiām
eius a morte: ⁊ operit multi
tudinē peccatorū. Explicit
epla bēi iacobi apli. ¶ In
cipit epla bēi pet̄i apli primā

Petrus apo .vij.
stolus ihe
su xpi ele
ctis adue
nis disp
sionis pō
ti galatie cappadocie asye ⁊
bithynie secūdm ꝓsciencia
dei patris in sanctificatione
spiritus in obedientiam et aspersionem
sanguinis ihesu xpi gratia vobis et
pax multiplicetur. Benedi

.viij.

dinate incorruptibile ⁊ in
tacinuatā ⁊ immarcesibilem
cōseruatā in celis ī vobis
qui in virtute dei custodi
ri per fidem ī salutē paratā
lari ī tempore nouissimo
quo exultabitis modi
nūc si oportet cōtristari
uariis temptationibus:
probatio vestre fidei multo
ior sit auro q̄ per ignem
batur. Inueniat in la
dem ⁊ honorē in reue
one ihu xpi: quē cū non
dentes diligitis. In quē
quoq; nō videntes cre
dentes autem exultabitis
ienarrabili ⁊ glorifi
portātes finē fidei vestre
ē aiārū. De qua salu
sierūt atq; scrutati
te q̄ de futura ī vobis
phetauerūt: scrutā
dō vel quale tempus
ficaret ī eis spūs xpi
ans eis q̄ in xpo s

TRUTH

Exp → Exp "+" Exp
Exp → Exp "*" Exp
Exp → Num

productive

Exp "+" Exp → Exp
Exp "*" Exp → Exp
Num → Exp

reductive

$3 * 7 + 21$

3 * 7 + **21**

3 * 7 + Exp

Num → Exp

3 * 7 + 21	Num → Exp
3 * **7** + Exp	Num → Exp
3 * Exp + Exp	

3 * 7 + 21	Num → Exp
3 * 7 + Exp	Num → Exp
3 * Exp + Exp	Num → Exp
Exp * Exp + Exp	

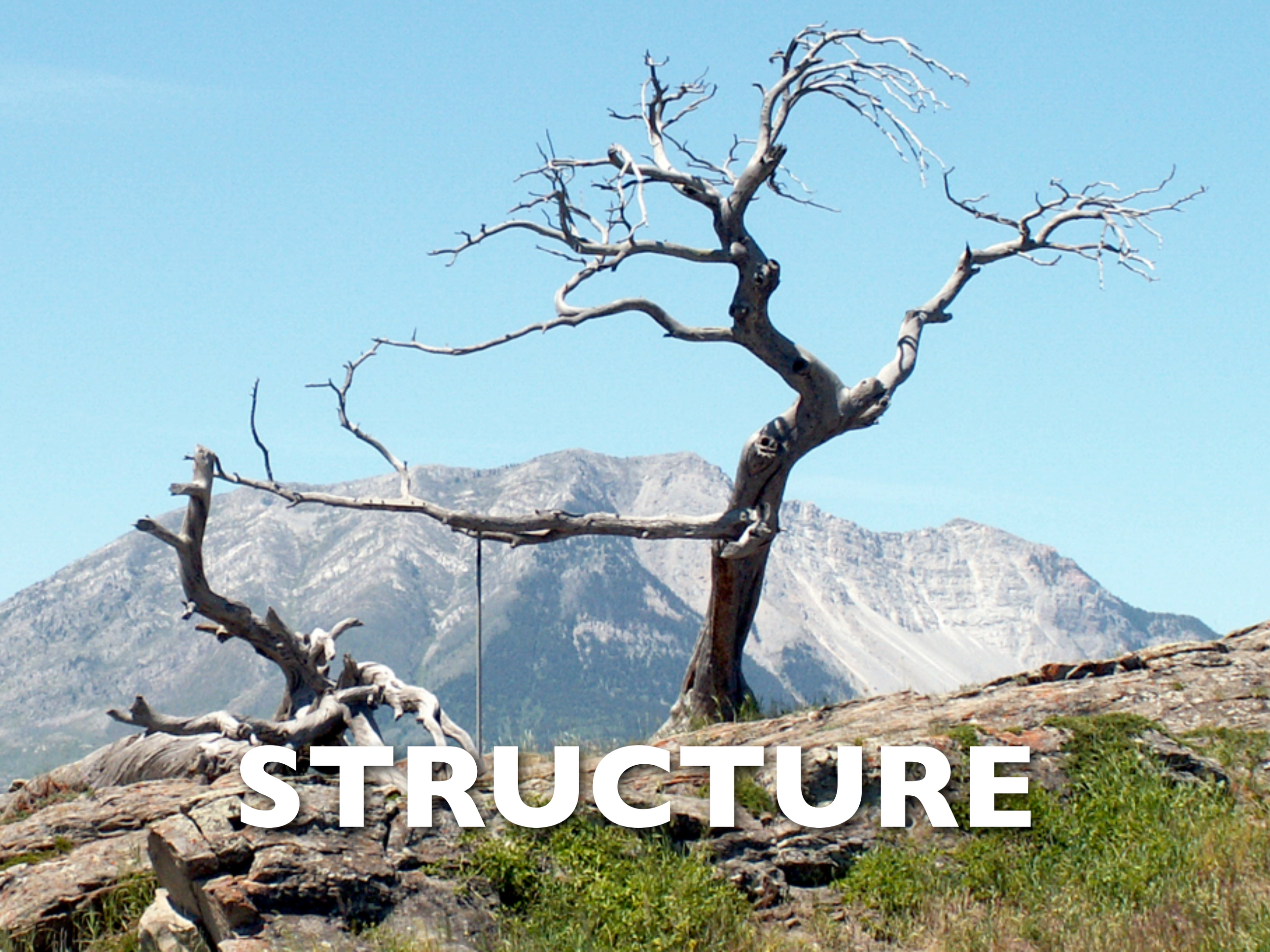
3 * 7 + 21	Num → Exp
3 * 7 + Exp	Num → Exp
3 * Exp + Exp	Num → Exp
Exp * **Exp** + Exp	Exp “*” Exp → Exp
Exp + Exp	

3 * 7 + 21	Num → Exp
3 * 7 + Exp	Num → Exp
3 * Exp + Exp	Num → Exp
Exp * Exp + Exp	Exp "*" Exp → Exp
Exp + Exp	Exp "+" Exp → Exp
Exp	

THEY TURNED WORDS INTO

TREES

SENTENCES



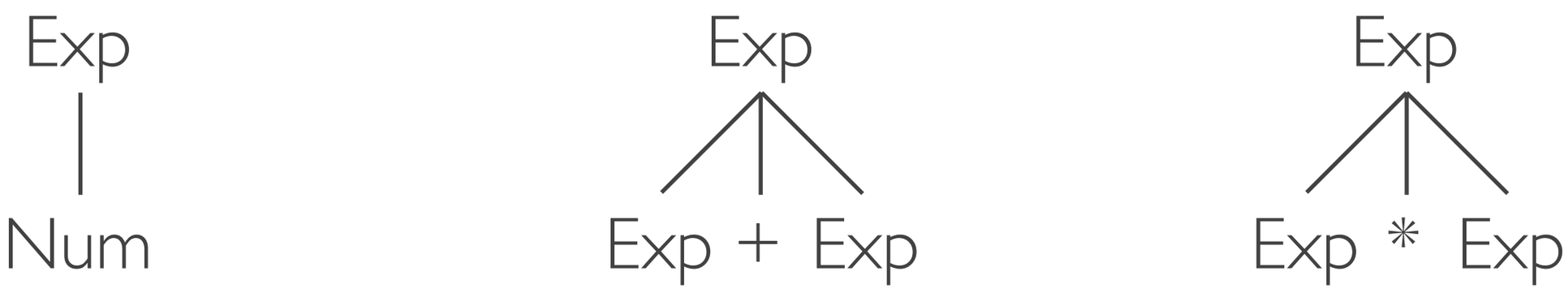
STRUCTURE

Exp → Exp "+" Exp
Exp → Exp "*" Exp
Exp → Num

productive

Exp "+" Exp → Exp
Exp "*" Exp → Exp
Num → Exp

reductive

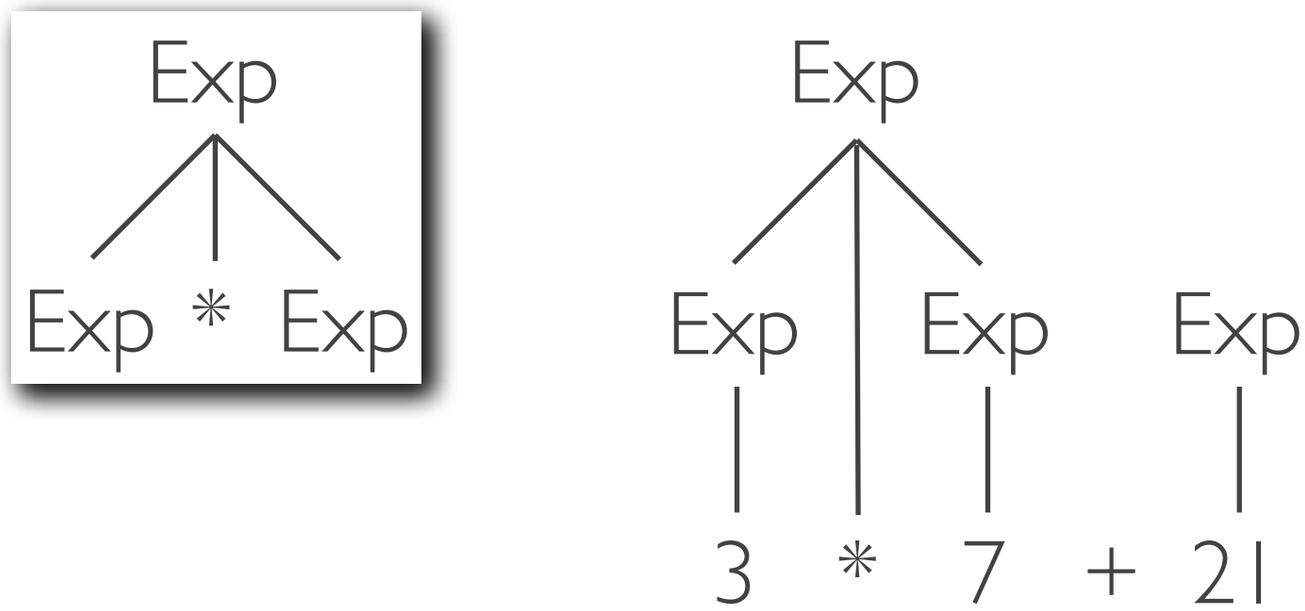


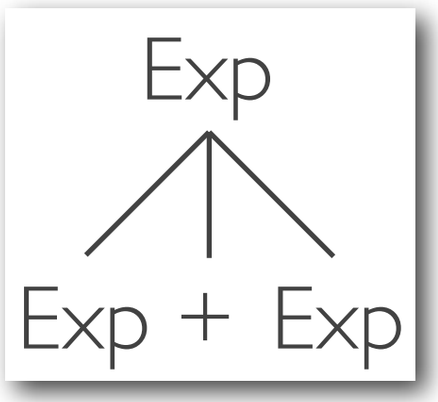
tree construction

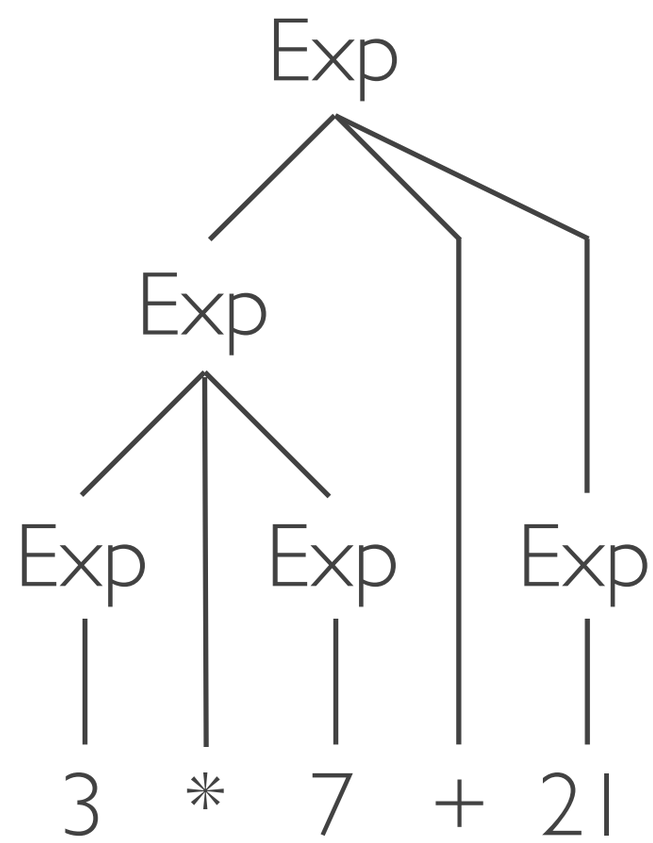
$$3 * 7 + 21$$

Exp
|
Num

Exp Exp Exp
| | |
3 * 7 + 21


Exp
Exp * Exp
Exp
Exp Exp Exp
3 * 7 + 21


Exp
Exp + Exp


Exp
Exp
Exp
Exp
Exp
3 * 7 + 21

ONE FORMALISM

THREE READINGS

PURE

DECLARATIVE

BEAUTIFUL

PARADISE **LOST**

EFFICIENCY

THE FIRST PLAGUE WERE

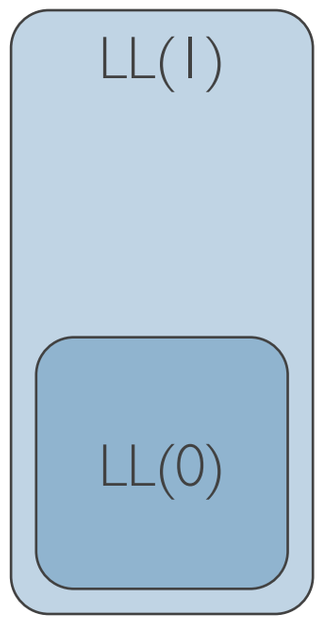
GRAMMAR CLASSES

context-free grammars

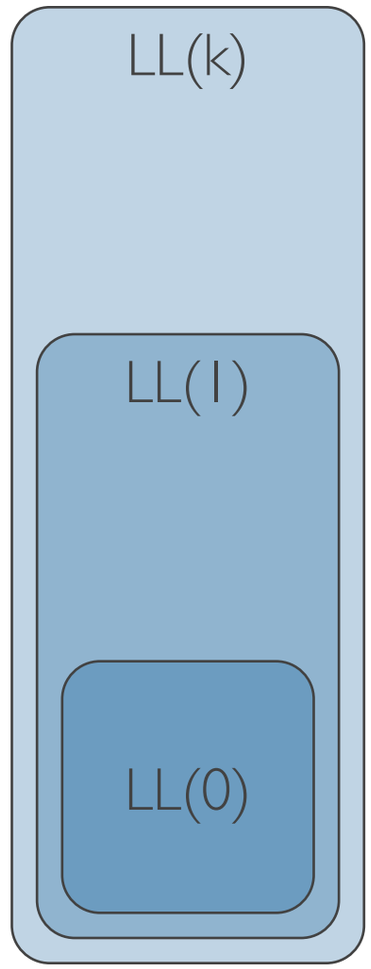
context-free grammars

LL(0)

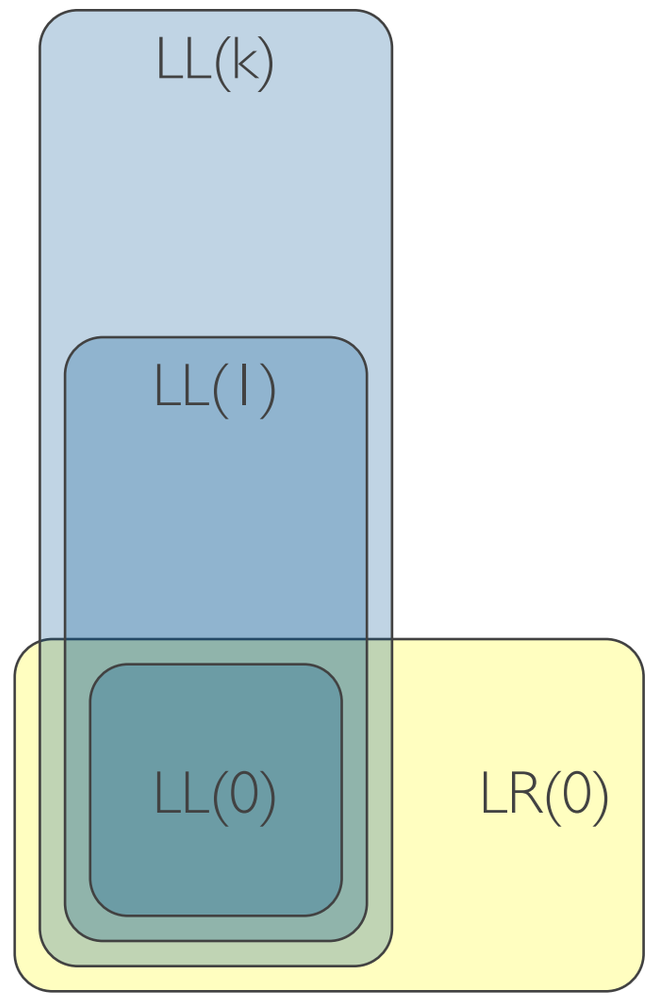
context-free grammars



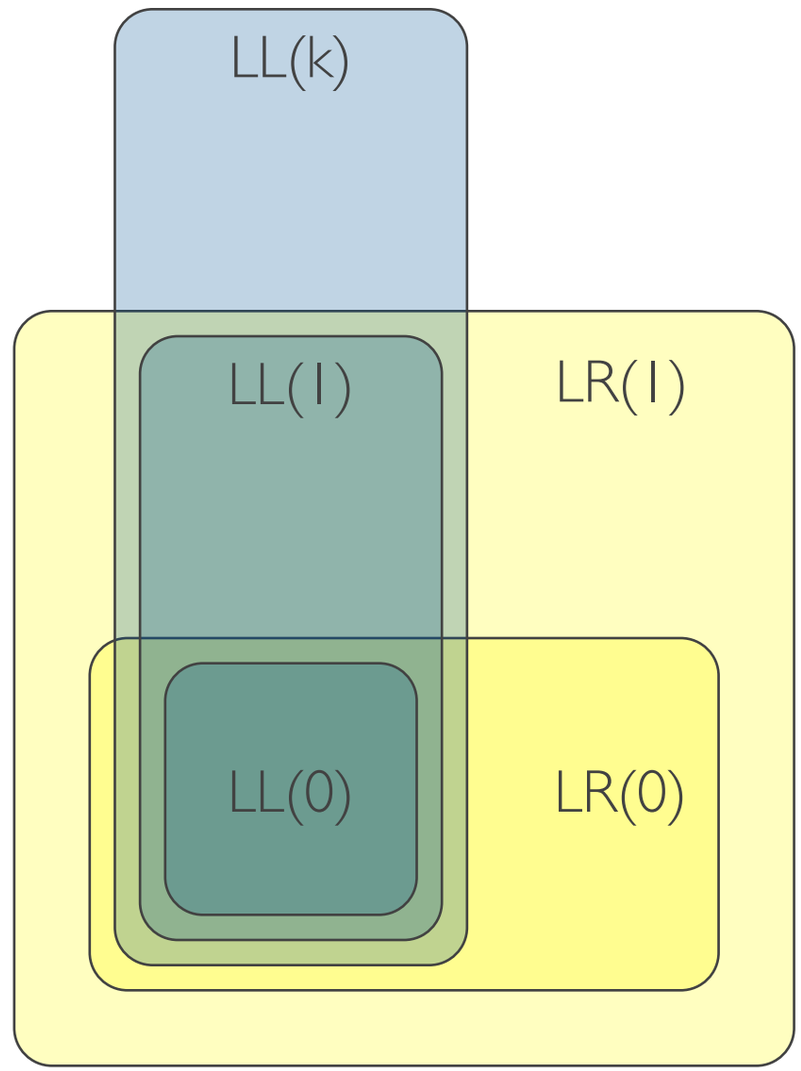
context-free grammars



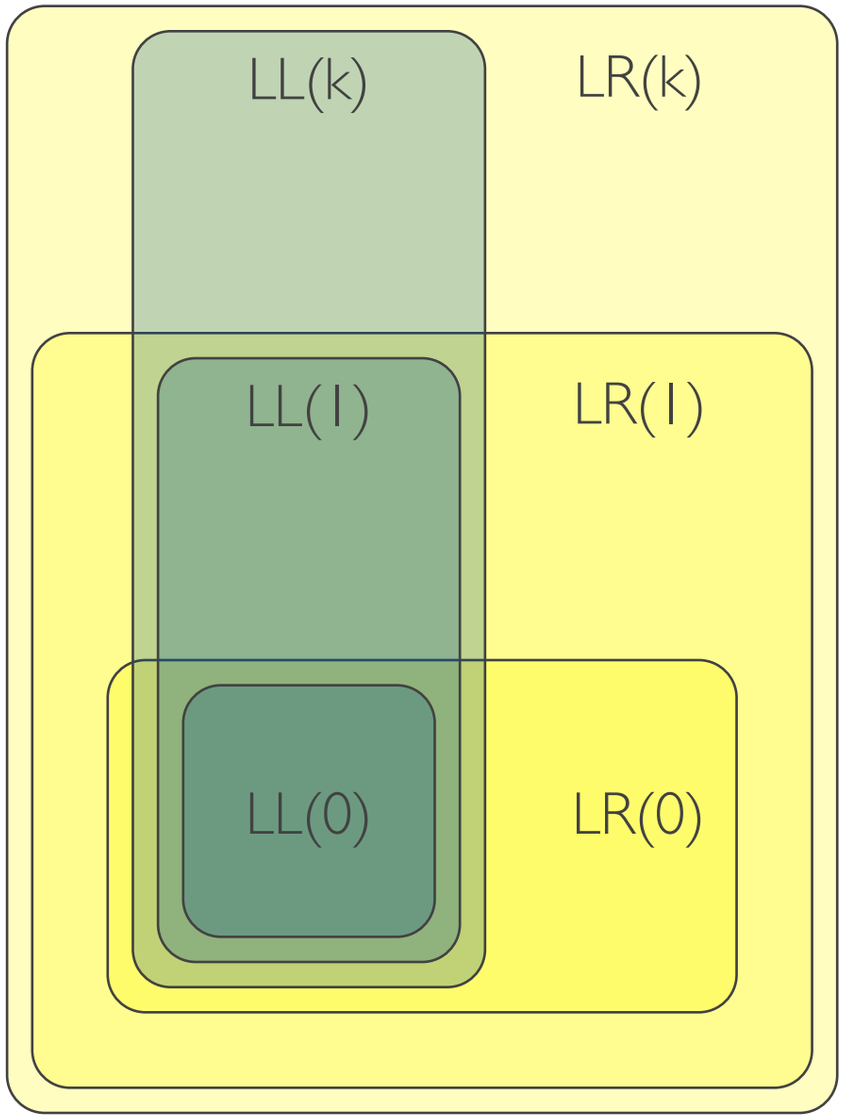
context-free grammars



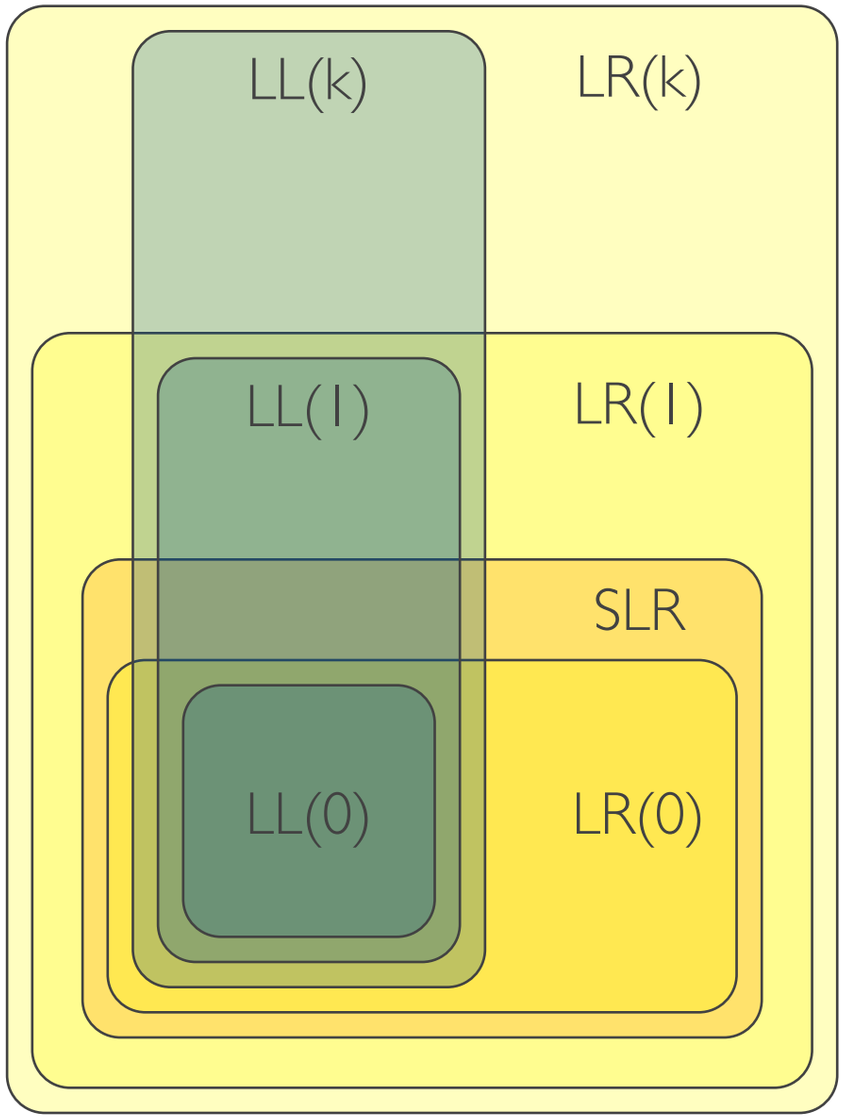
context-free grammars



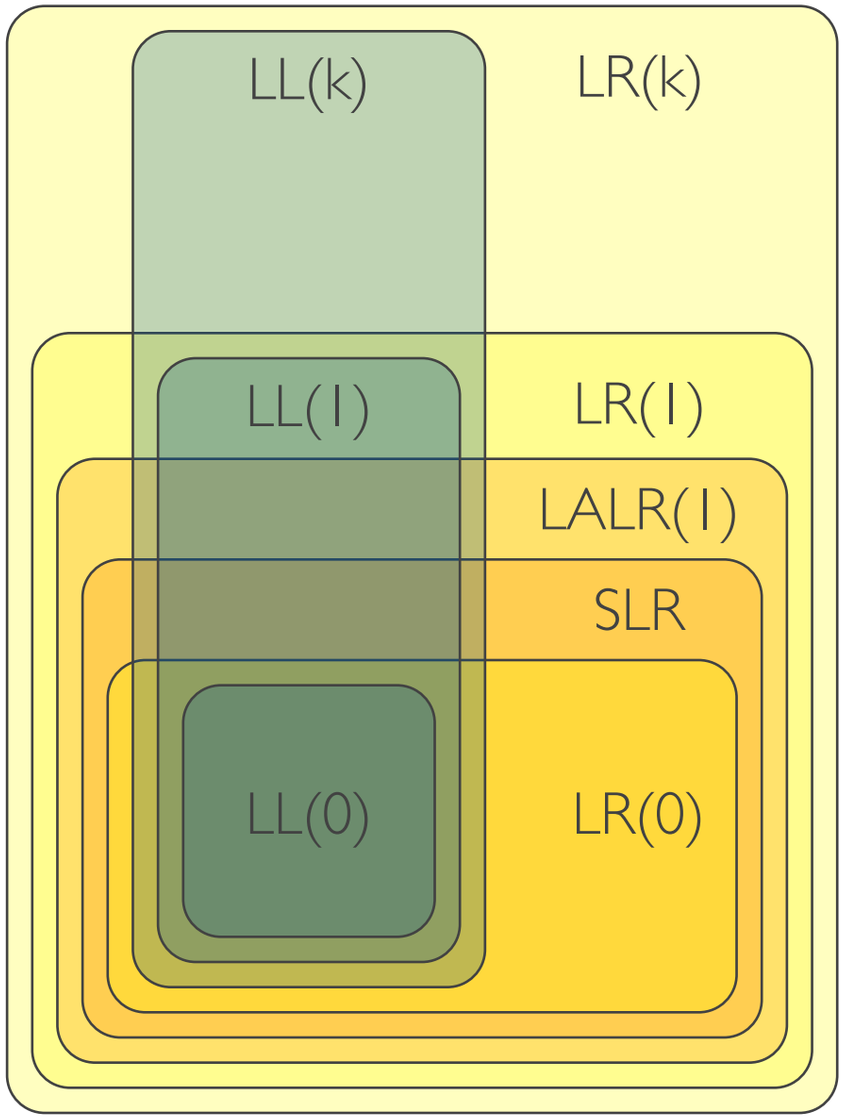
context-free grammars



context-free grammars



context-free grammars

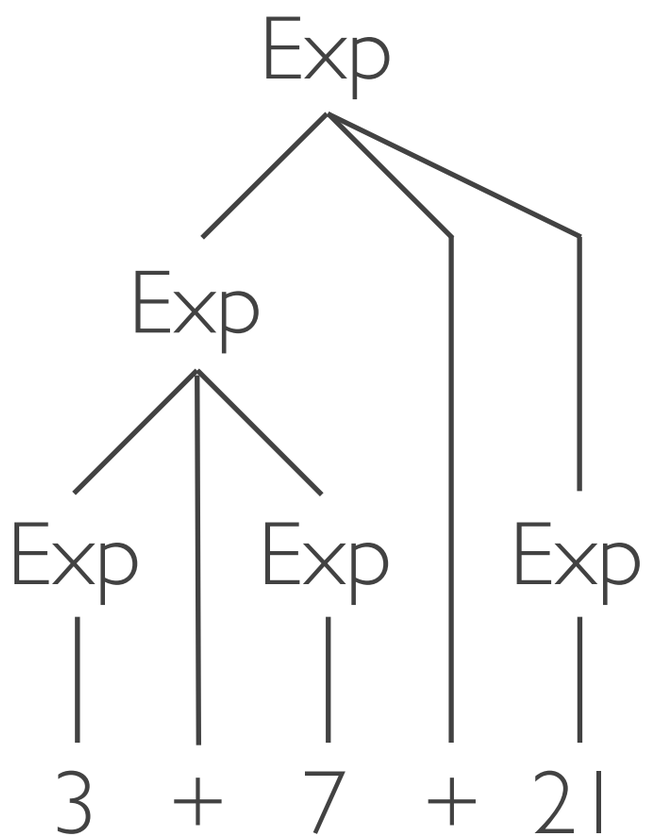


Exp "+" Exp → Exp
Exp "*" Exp → Exp
Num → Exp

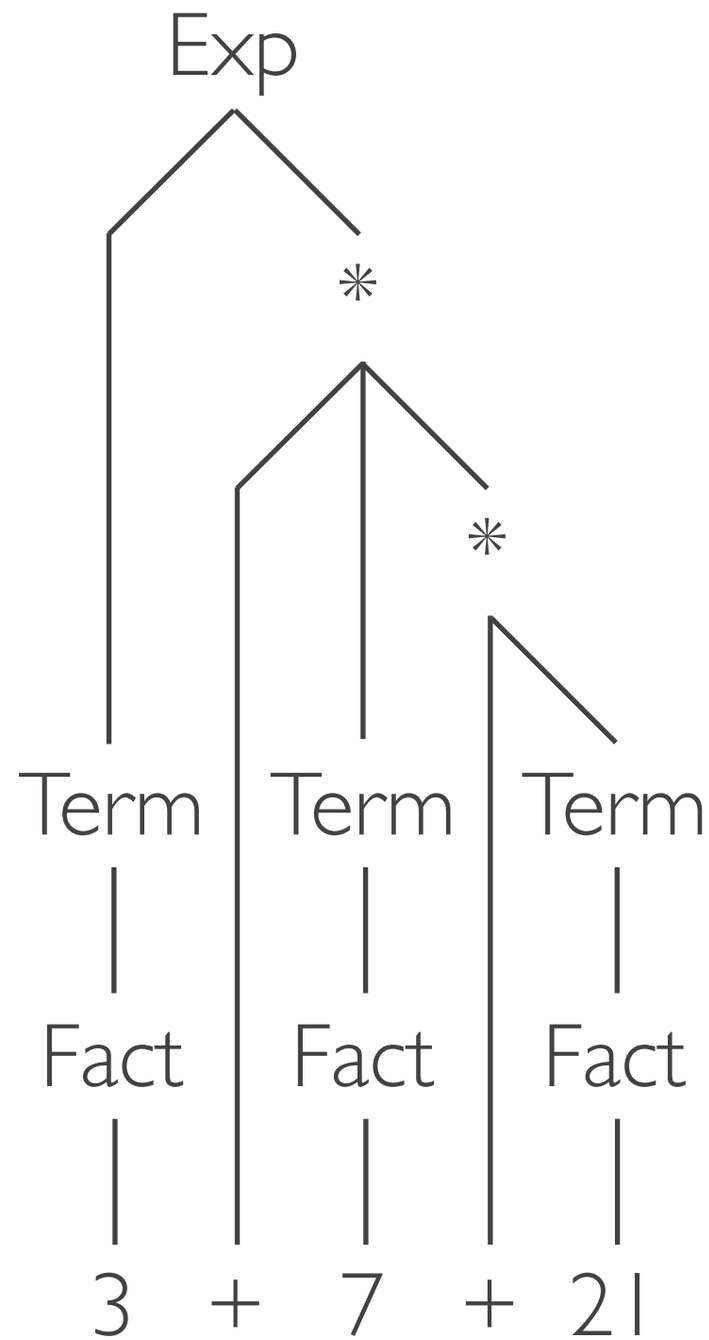
Term ("+" Term)* → Exp
Fact ("*" Fact)* → Term
Num → Fact

paradise

grammar classes



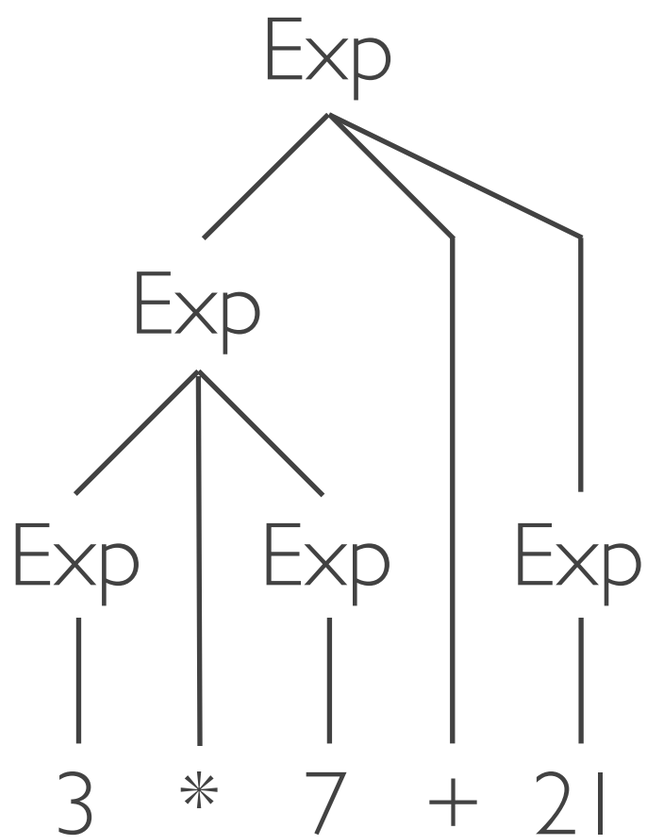
paradise

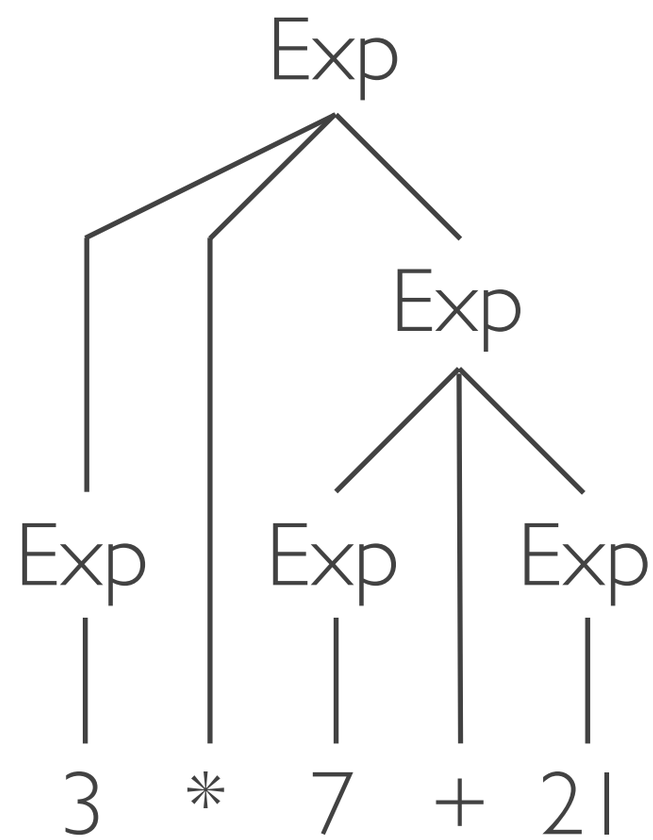


grammar classes

THE SECOND PLAGUE WAS

DISAMBIGUATION

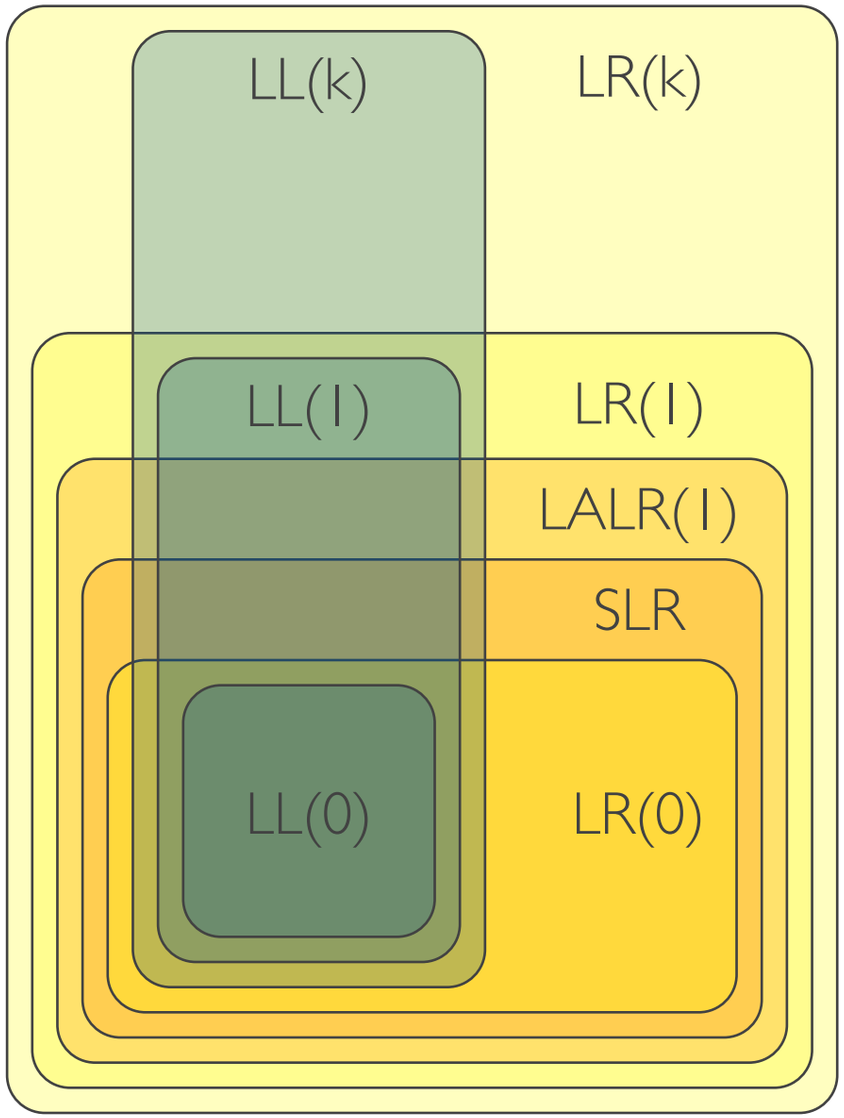

Exp
Exp
Exp
Exp
Exp
3
*
7
+
21

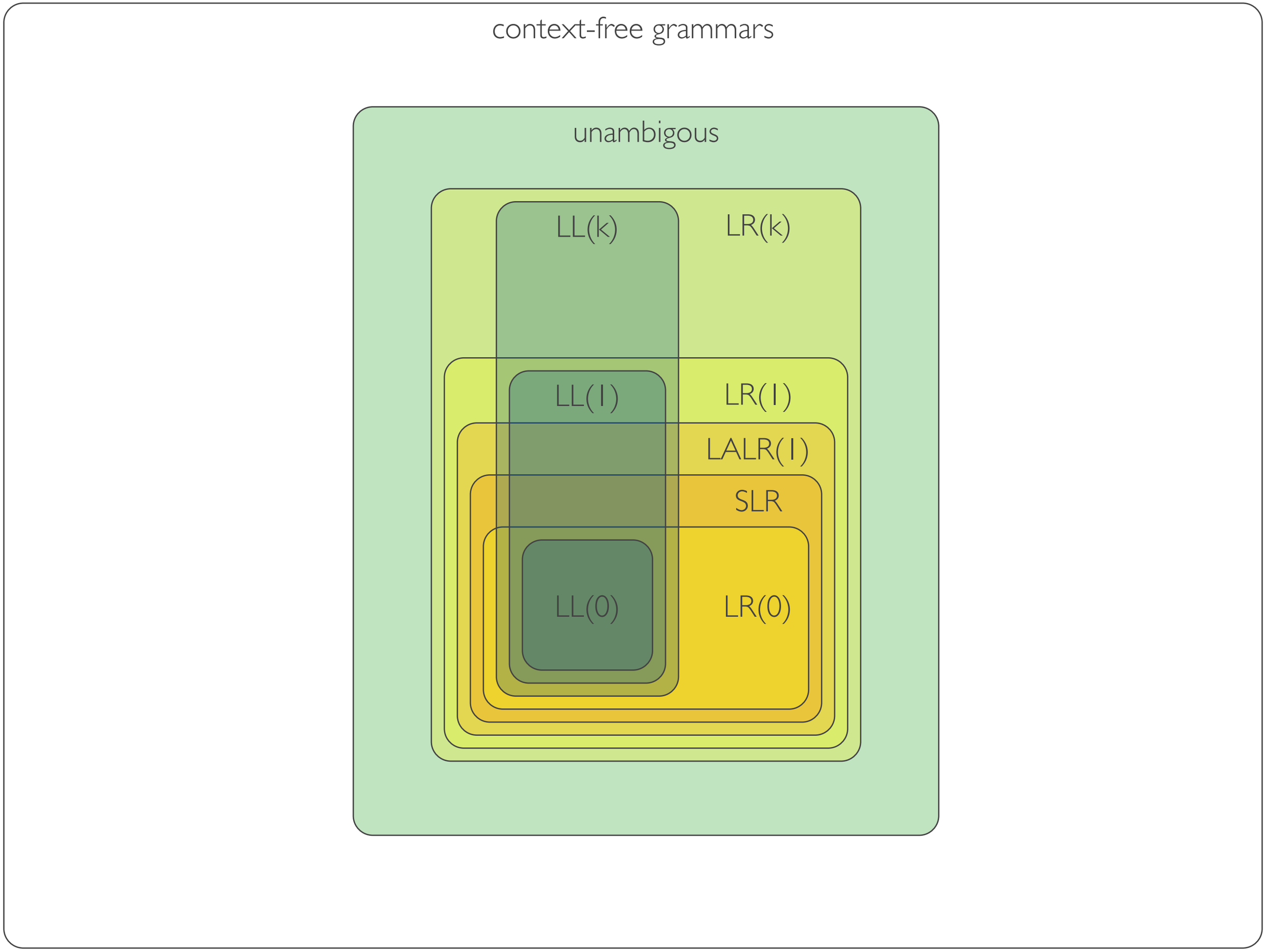

Exp
Exp
Exp
Exp
Exp
3
*
7
+
21

precedence	operators	associativity
1	( ), [ ]	non-associative
2	new	non-associative
3	.	left-associative
4	++, --	non-associative
5	-, +, !, ~, ++, --, (*type*)	right-associative
6	*, /, %	left-associative
7	+, -	left-associative
...	...	...

text books

context-free grammars




context-free grammars
unambigous
LL(k)
LR(k)
LL(1)
LR(1)
LALR(1)
SLR
LL(0)
LR(0)

Exp "+" Term → Exp

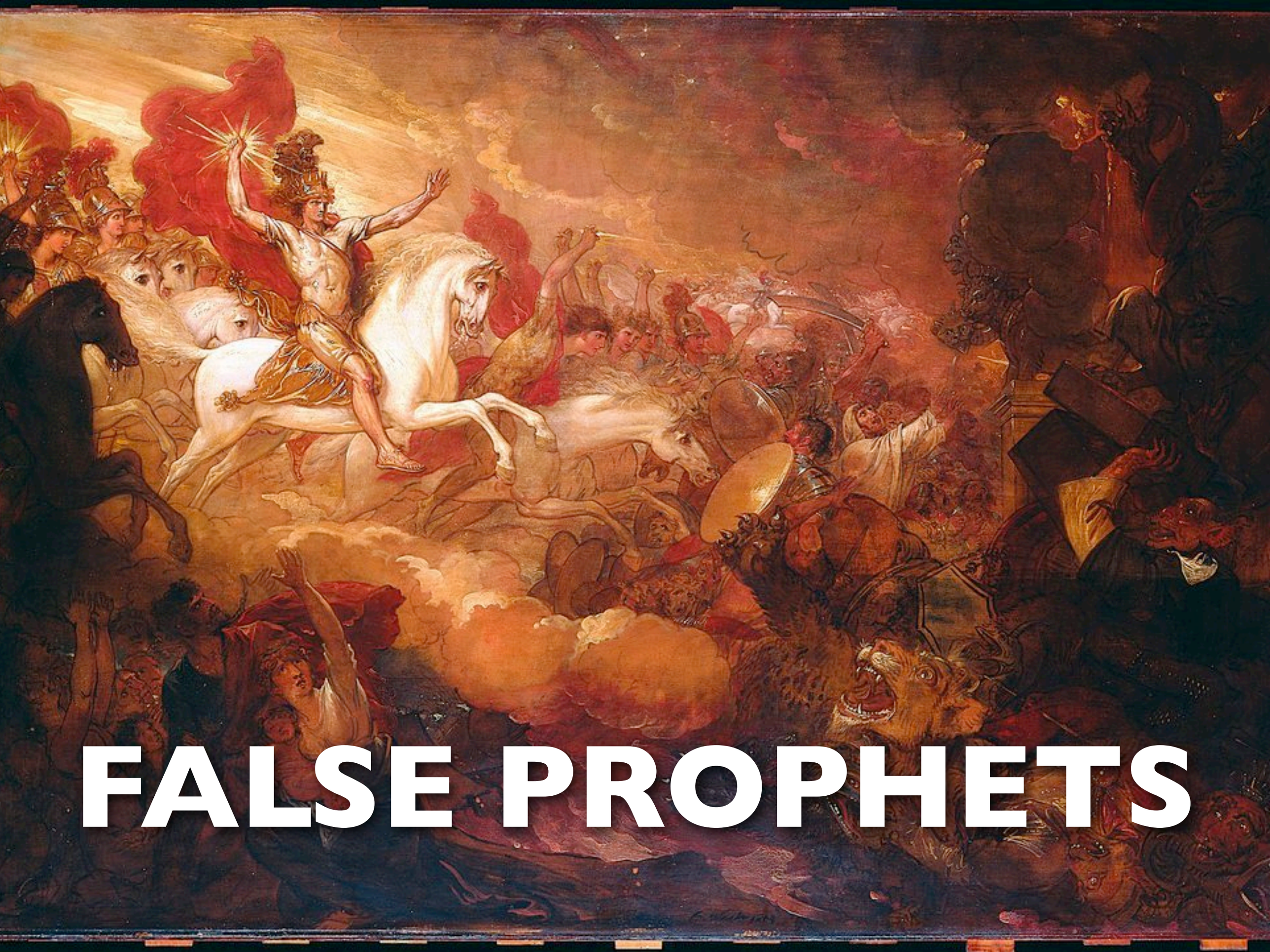
Term → Exp

Term "*" Fact → Term

Fact → Term

Num → Fact

grammar classes



FALSE PROPHETS

L = {ab, a}

“a” “b” → A

“a” → A

paradise

L = {ab, a}

“a” “b” /

“a” → A

PEGs

L = {ab, a}

“a” → A

“a” “b” → A

paradise

L = {a}

“a” /

“a” “b” → A

PEGs

if c_1 **then if** c_2 **then** s_1 **else** s_2

dangling else

“if” E “then” S “else” S / “if” E “then” S /

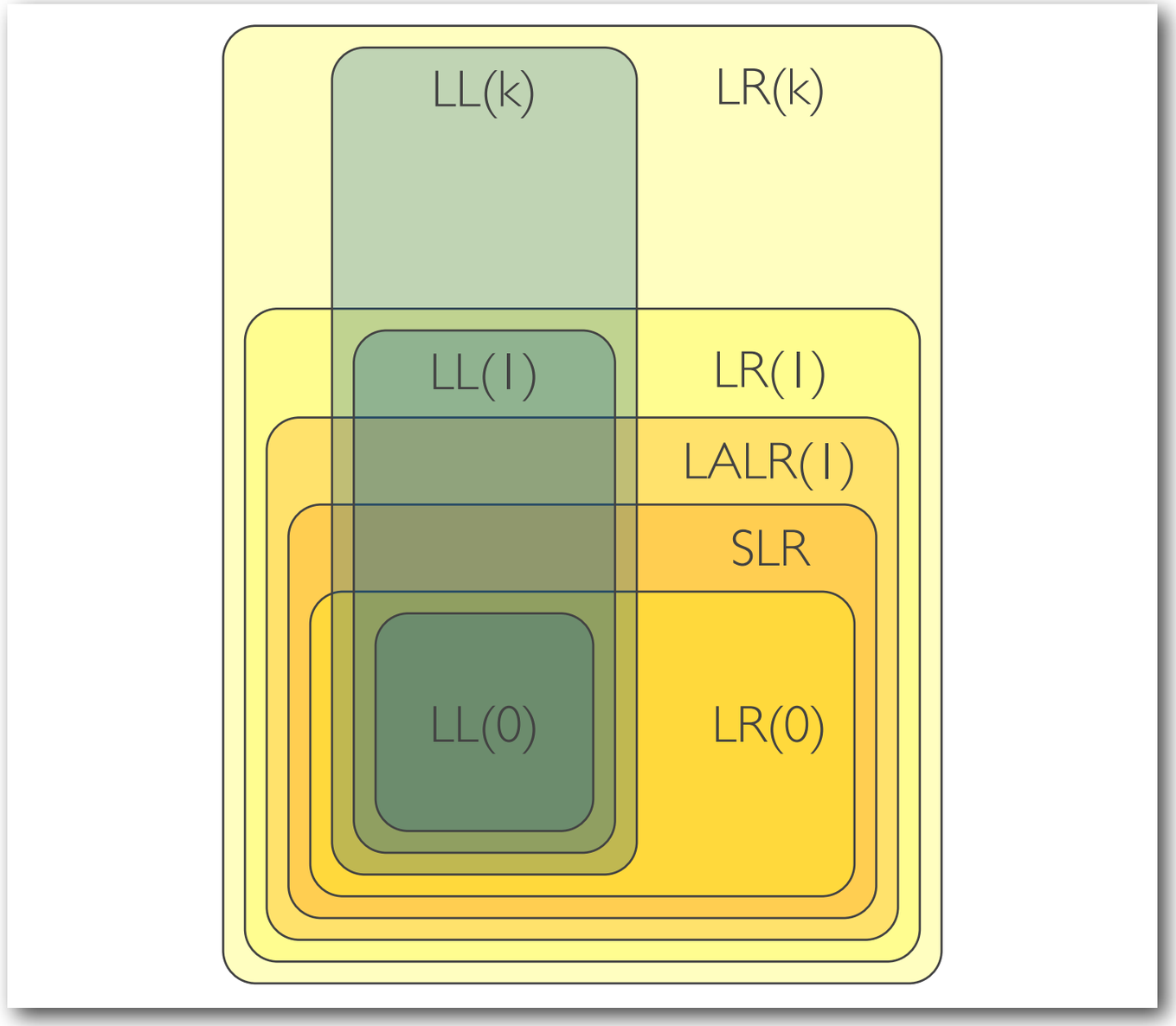
“if” E “then” S → S “if” E “then” S “else” S → S

PEGs

THE THIRD PLAGUE WAS

LEXICAL SYNTAX

morphology & syntax



limited look-ahead

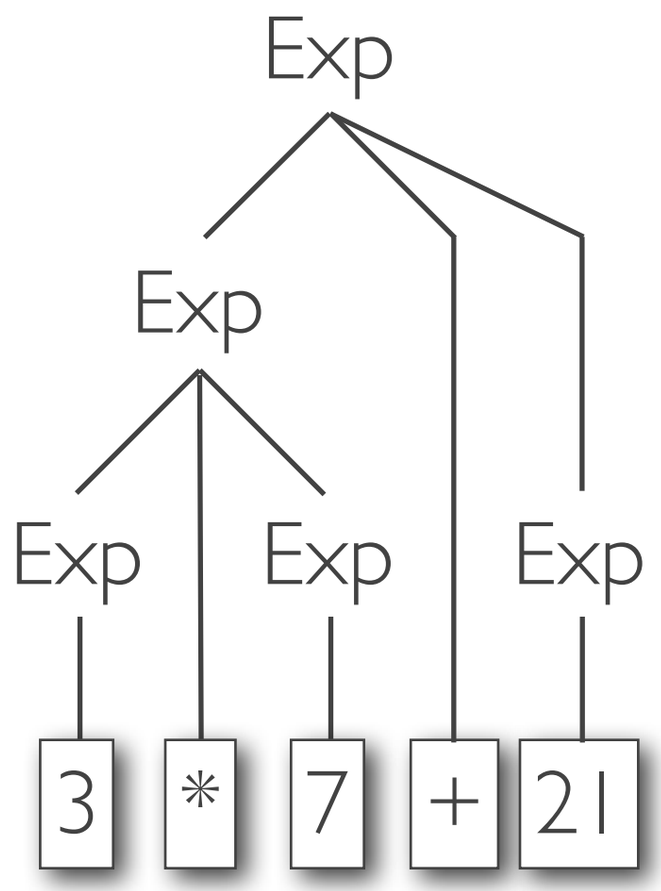
scanners

$$3 * 7 + 21$$

3 * 7 + 21

parsers

3 * 7 + 21


Exp
Exp
Exp
Exp
Exp
3
*
7
+
21

```
x = 1. * .10

y : array [ 1 .. 10 ] of integer
```

x = 1. * .10

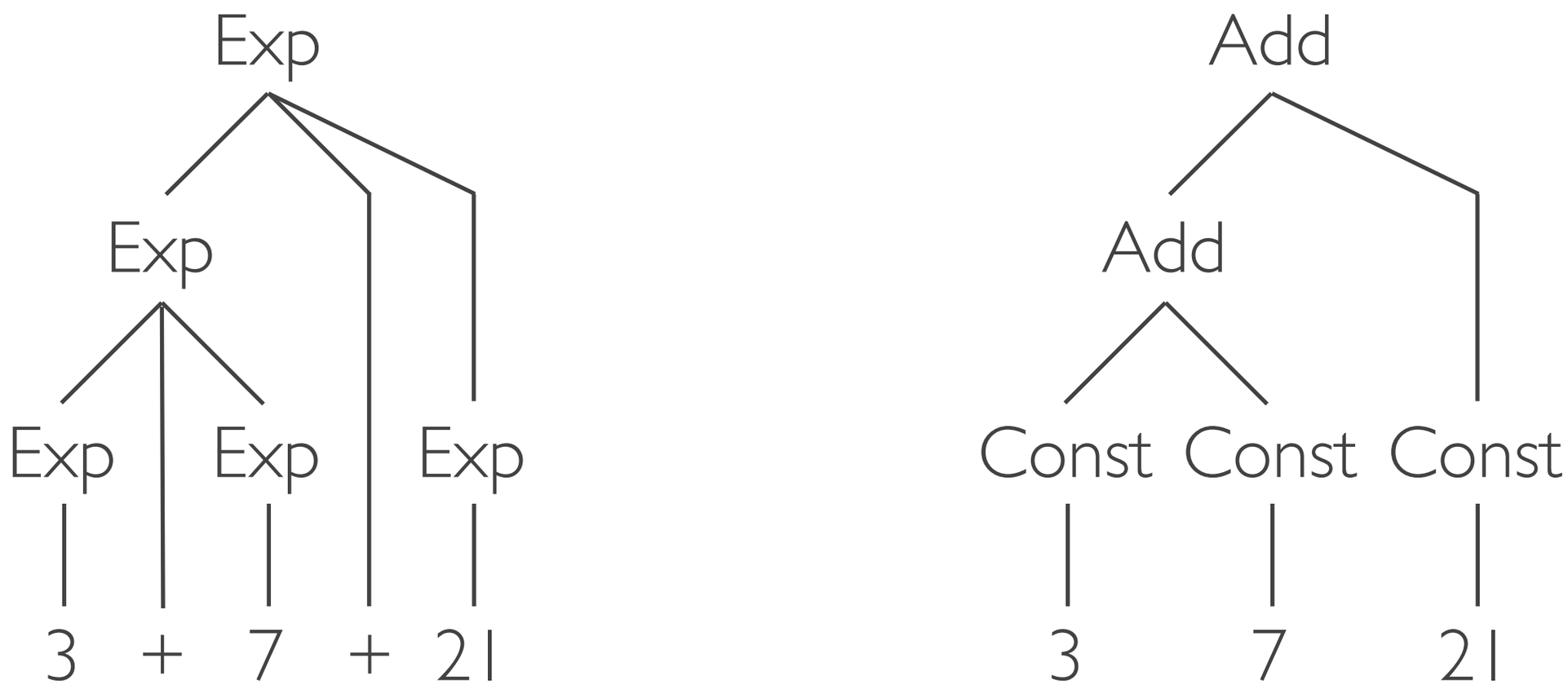
y : array [1 .. 10] of integer

x = 1. * .10

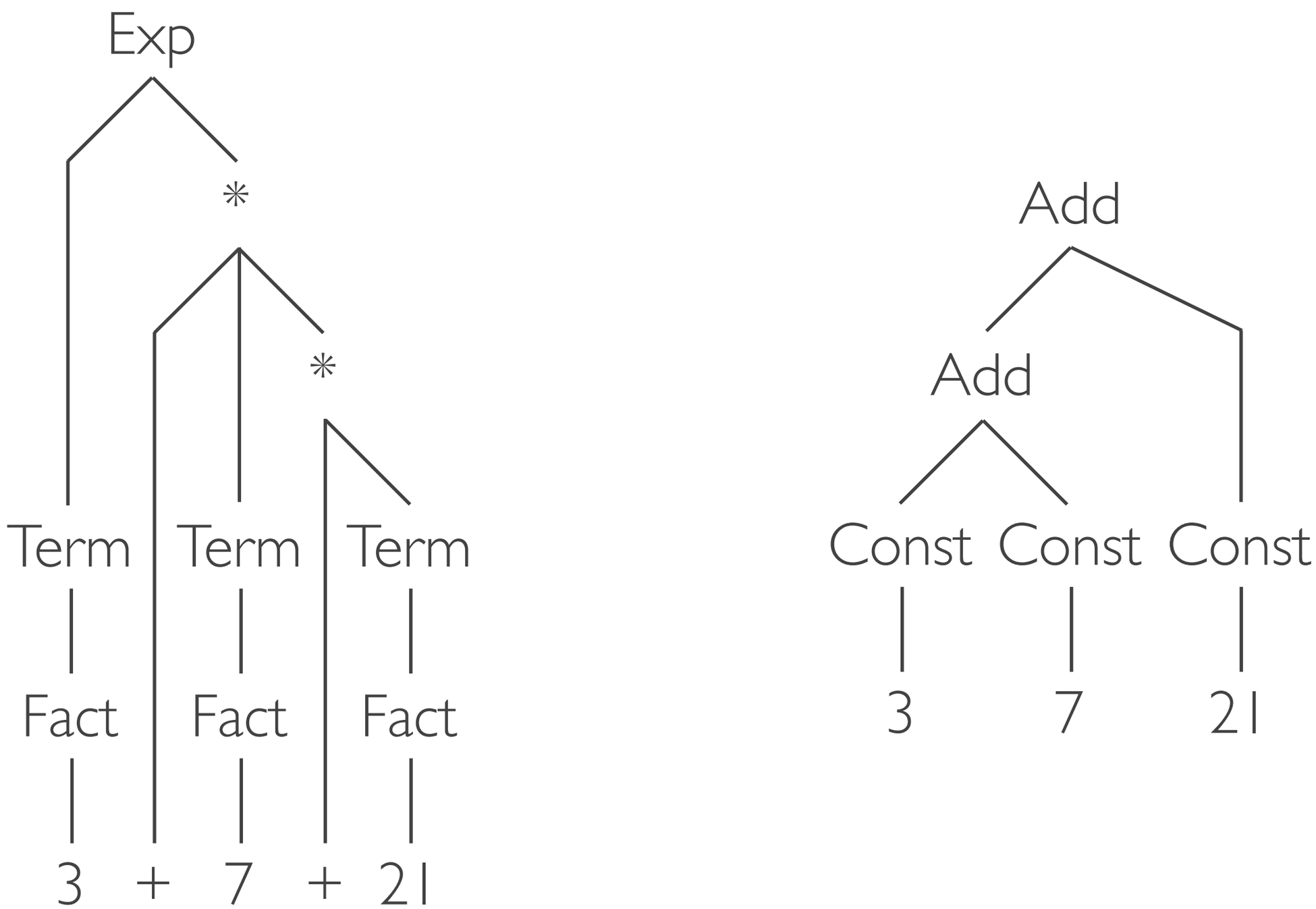
y : array [1. .10] of integer

THE FOURTH PLAGUE WAS

TREE CONSTRUCTION



paradise



grammar classes

```
expr:
  INTEGER            { $$ = con($1); }
| expr '+' expr { $$ = opr('+', 2, $1, $3);}
| expr '*' expr { $$ = opr('*', 2, $1, $3);}
;
```

semantic actions

THE FIFTH PLAGUE WAS

EVOLUTION

Exp "+" Exp → Exp
Exp "*" Exp → Exp
Num → Exp

paradise

Exp "+" Term → Exp
Term → Exp
Term "*" Fact → Term
Fact → Term
Num → Fact

grammar classes

Exp "+" Exp → Exp
Exp "*" Exp → Exp
Num → Exp

Exp "=" Exp → Exp
Exp "<" Exp → Exp
Exp ">" Exp → Exp

paradise

CExp "+" Term → **CExp**
Term → **CExp**
Term "*" Fact → Term
Fact → Term
Num → Fact

Exp "=" CExp → Exp
Exp "<" CExp → Exp
Exp ">" CExp → Exp
CExp → Exp

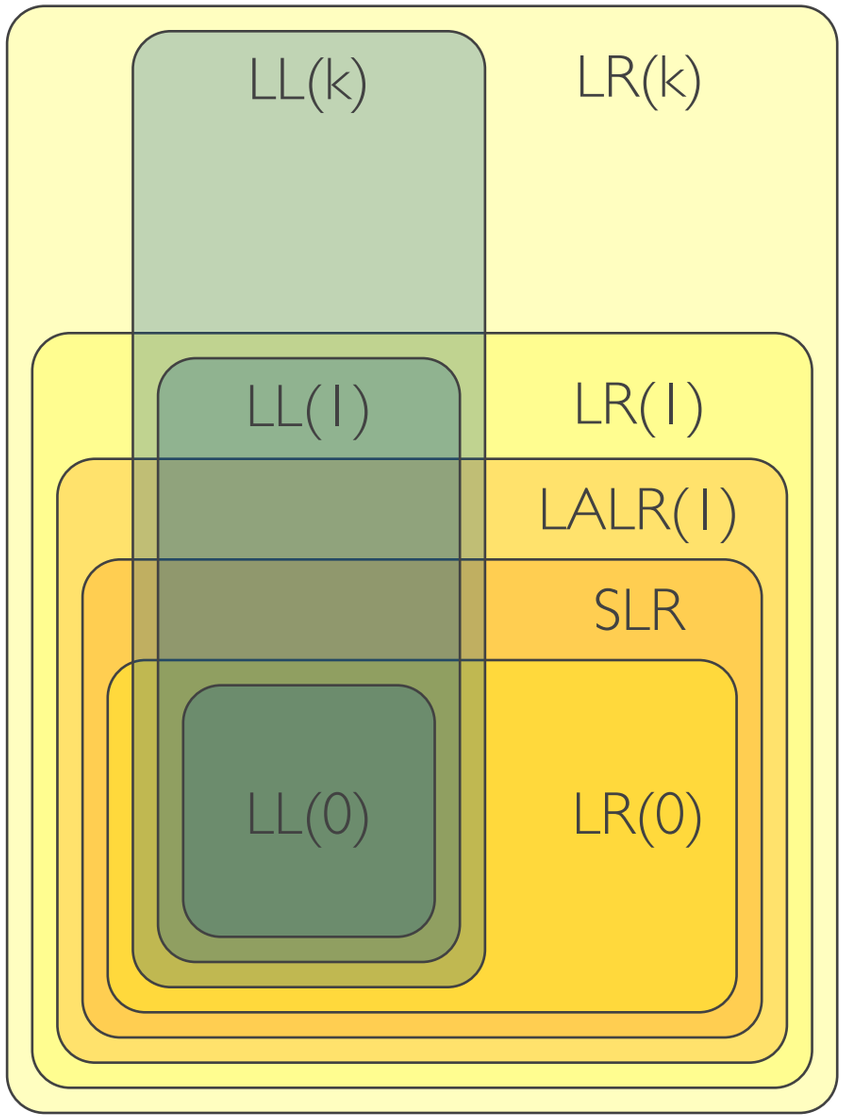
grammar classes

THE SIXTH PLAGUE WAS

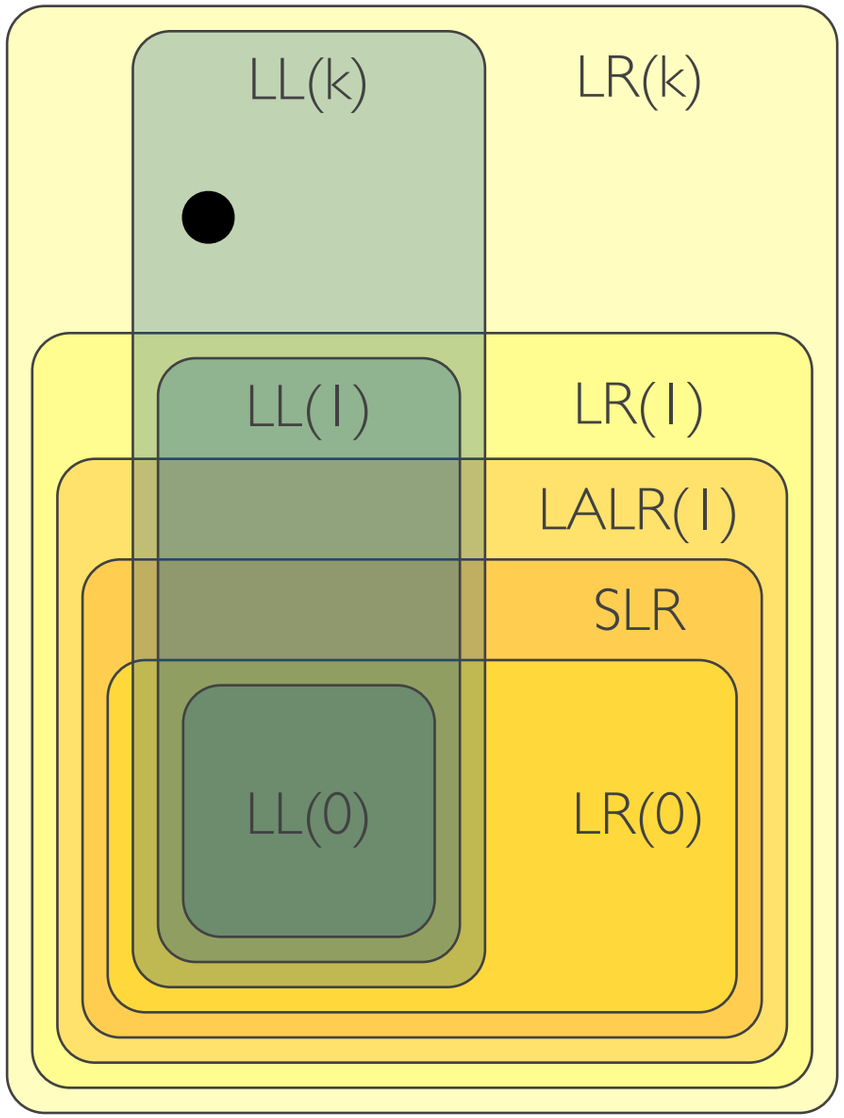
COMPOSITION

parsers

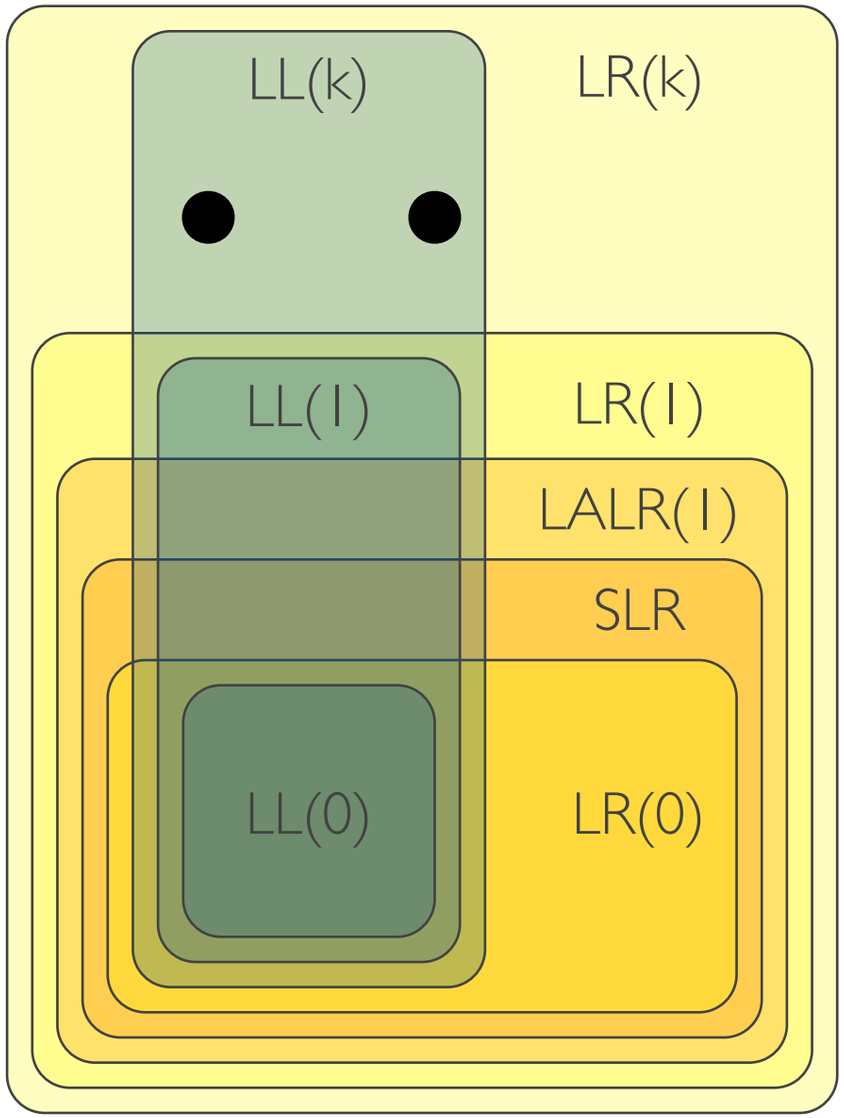
context-free grammars



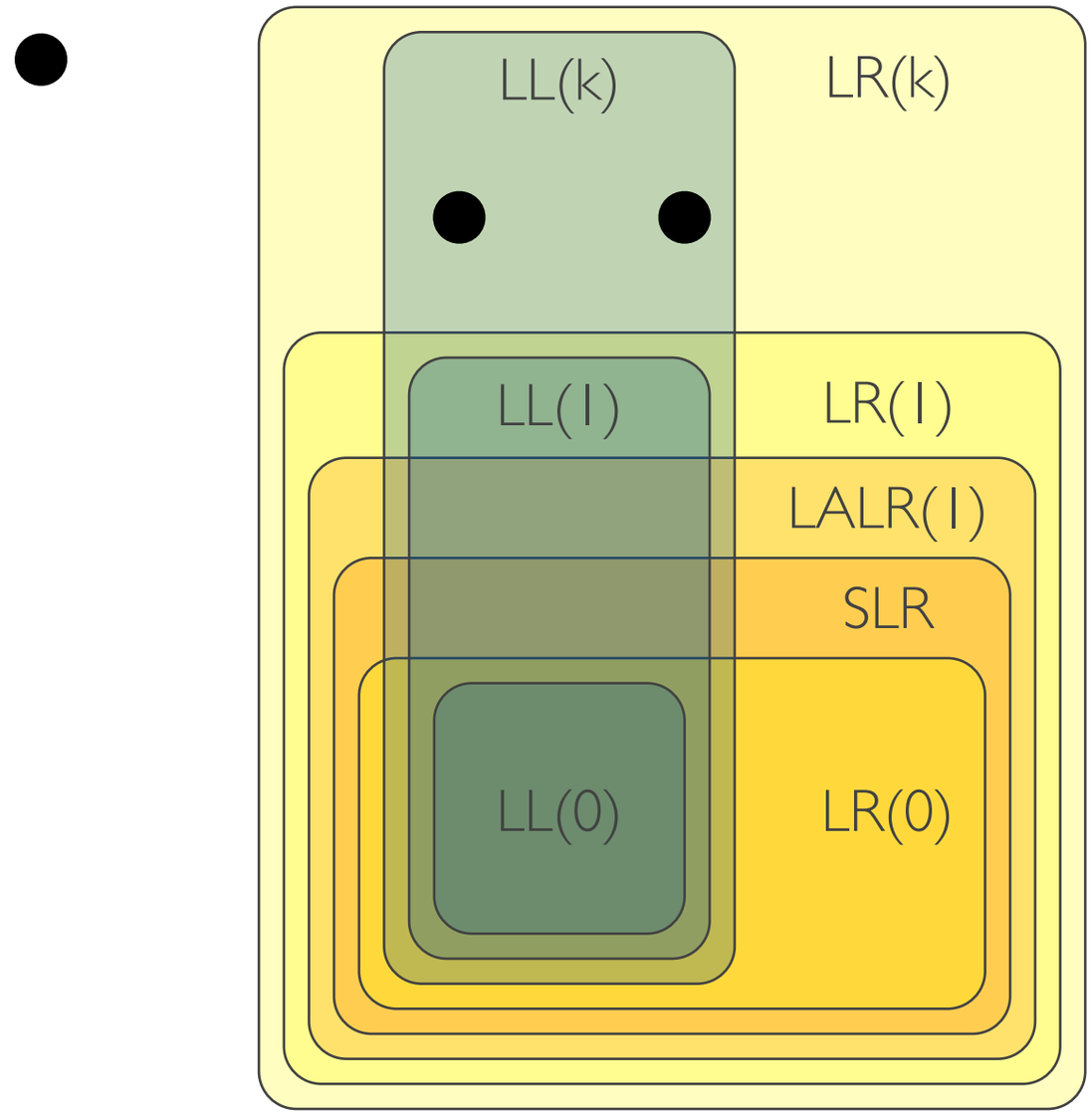
context-free grammars



context-free grammars



context-free grammars



scanners

```
public boolean
  authenticate(String user, String pw) {

    SQL stm = <| SELECT id FROM Users
                 WHERE name = ${user}
                 AND   password = ${pw} |>;

    return executeQuery(stm).size() != 0;
}
```

THE SEVENTH PLAGUE WAS

RESTRICTION

TO

PARSERS

PRETTY PRINTERS

SENTENCE GENERATORS

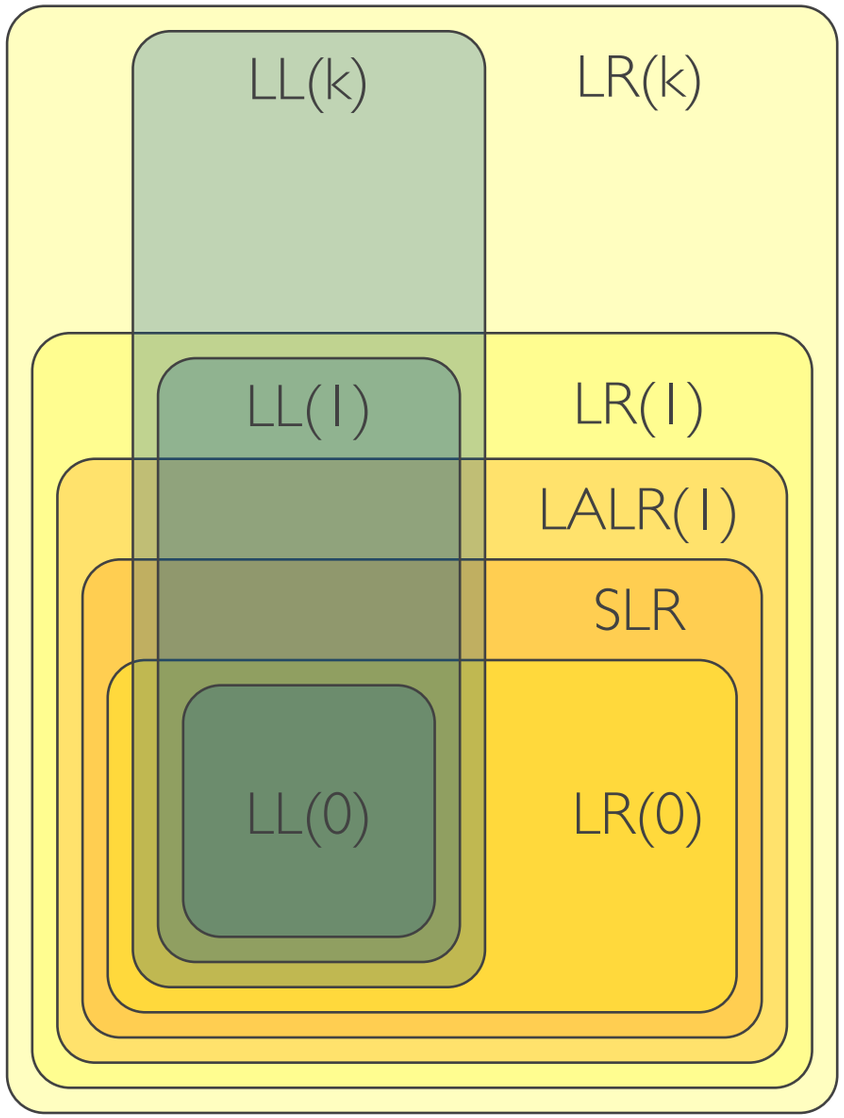
AST ACCESS

IDE SUPPORT

PARADISE REGAINED

GENERALISED
PARSING

context-free grammars



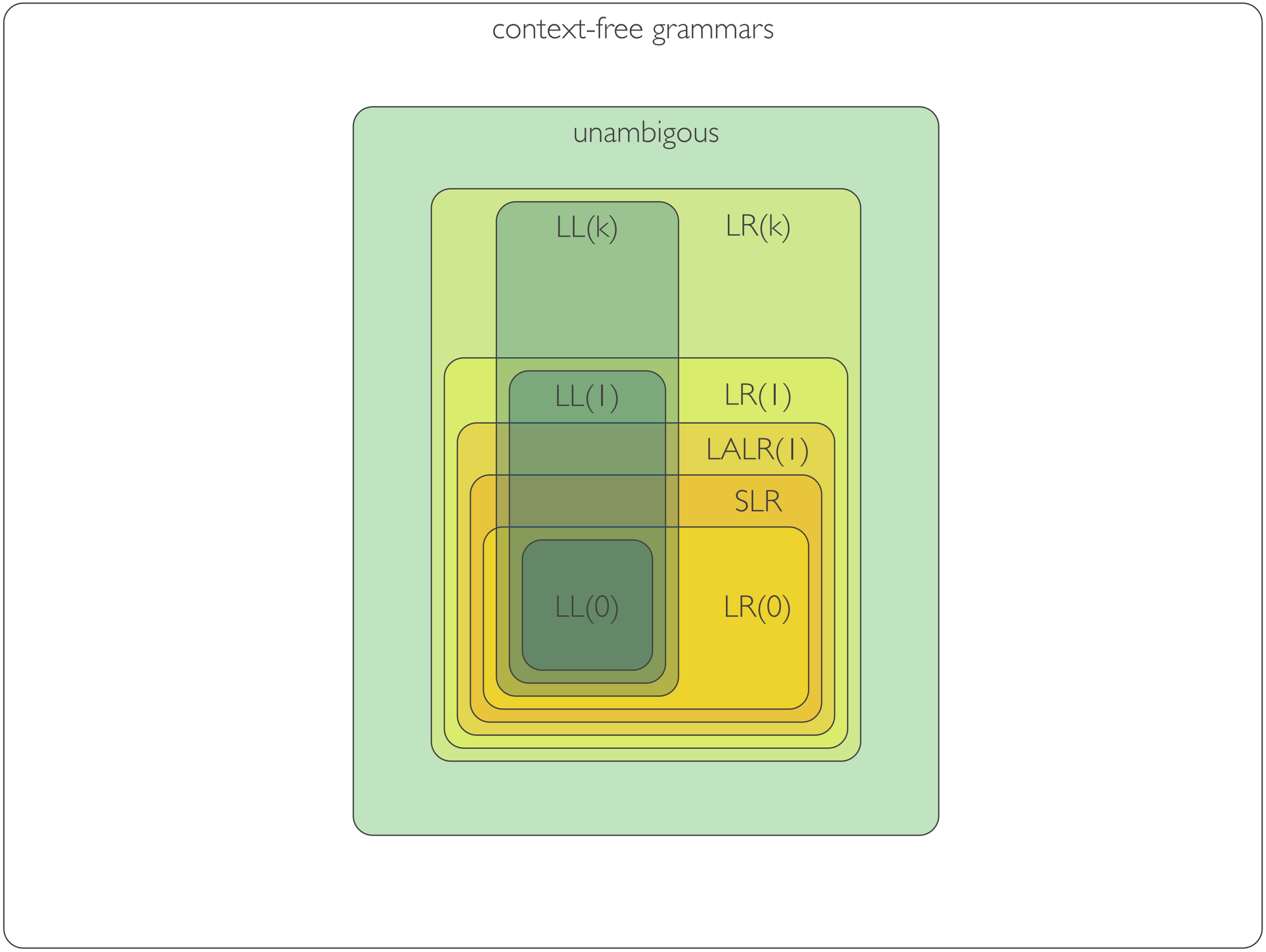
context-free grammars

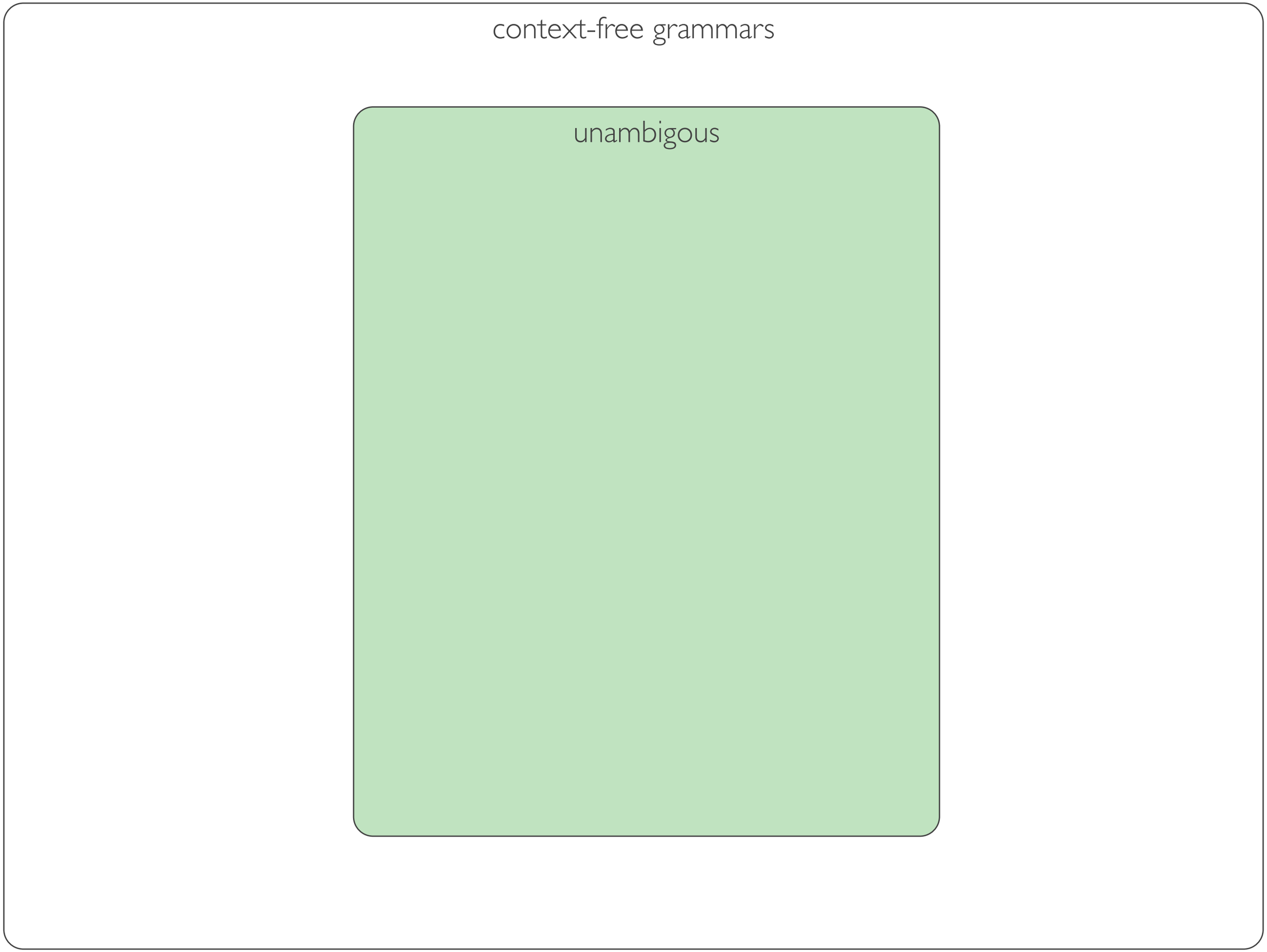
Exp "+" Exp → Exp
Exp "*" Exp → Exp
Num → Exp

```
context-free syntax

  Exp "+" Exp -> Exp
  Exp "*" Exp -> Exp
  NUM         -> Exp
```

paradise **SDF**

DECLARATIVE
DISAMBIGUATION


context-free grammars
unambigous
LL(k)
LR(k)
LL(1)
LR(1)
LALR(1)
SLR
LL(0)
LR(0)


context-free grammars
unambigous

context-free grammars

precedence	operators	associativity
1	( ), [ ]	non-associative
2	new	non-associative
3	.	left-associative
4	++, --	non-associative
5	-, +, !, ~, ++, --, (type)	right-associative
6	*, /, %	left-associative
7	+, -	left-associative
…	…	…

text books

context-free priorities

```
  Exp "*" Exp -> Exp {left}
> Exp "+" Exp -> Exp {left}
```

```
context-free syntax

  "if" E "then" S                -> S {prefer}
  "if" E "then" S "else" S -> S
```

SCANNERLESS
PARSING

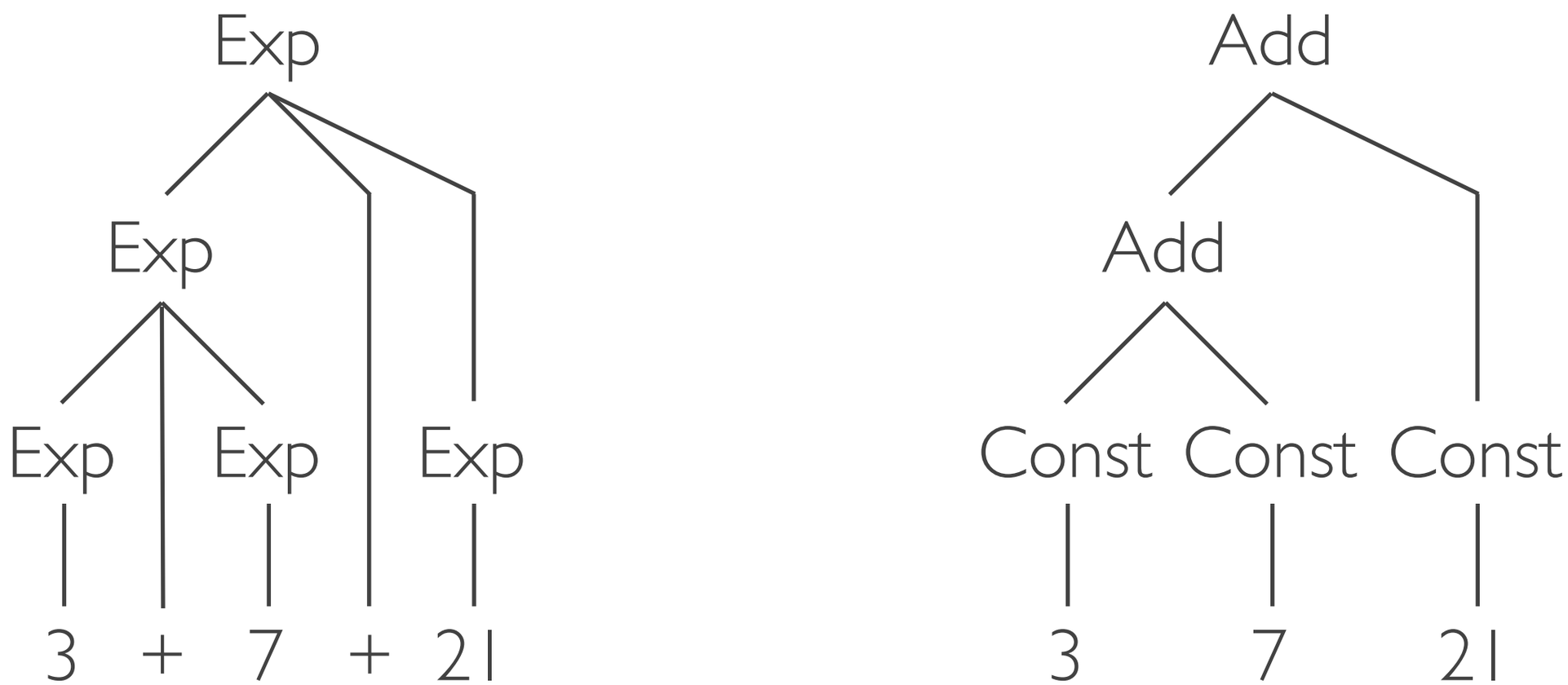
morphology & syntax

```
lexical syntax

  [0-9]+ -> NUM

  [\ \t\n]              -> LAYOUT
  "//" ~[\n]* [\n] -> LAYOUT
```

parser

parser

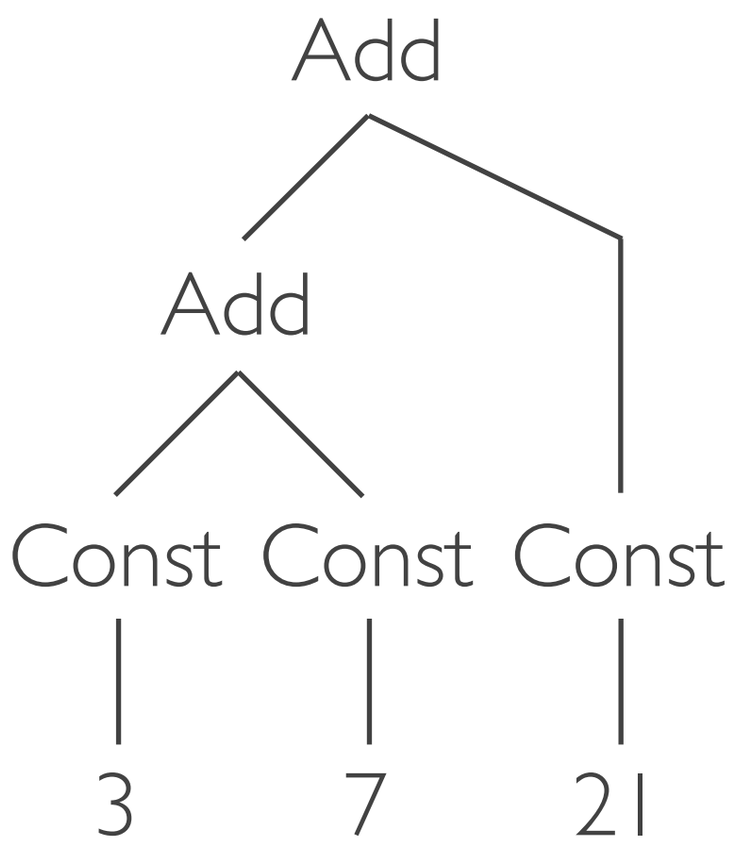
DECLARATIVE
TREE CONSTRUCTION



paradise

```
context-free syntax

  Exp "+" Exp -> Exp {cons("Add")}
  Exp "*" Exp -> Exp {cons("Mul")}
  NUM         -> Exp {cons("Const")}
```

SDF




```
Add(
  Add(
    Const("3"),
    Const("7")
  ),
  Const("21")
)
```

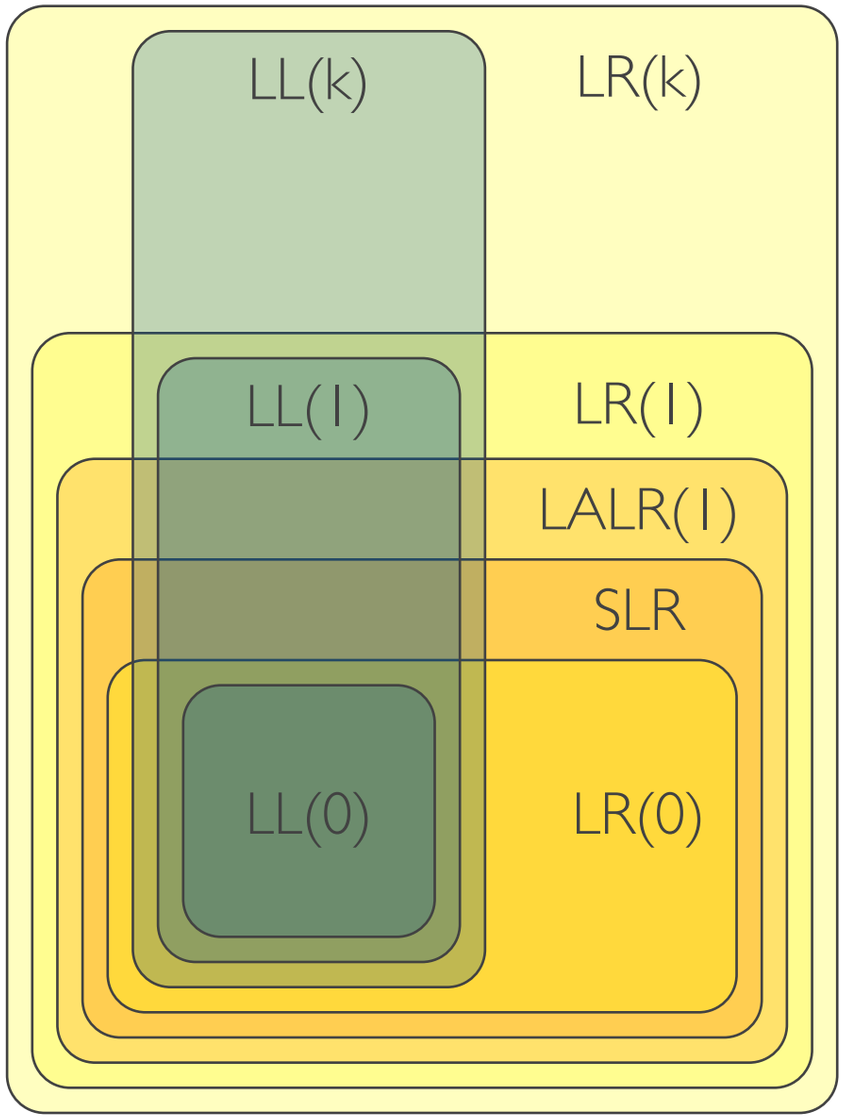
paradise **SDF**

SEAMLESS
EVOLUTION

```
context-free syntax

  Exp "+" Exp -> Exp {cons("Add")}
  Exp "*" Exp -> Exp {cons("Mul")}
  NUM         -> Exp {cons("Const")}

  Exp "=" Exp -> Exp {cons("Eq")}
  Exp ">" Exp -> Exp {cons("Gt")}
  Exp "<" Exp -> Exp {cons("Lt")}
```

MODULAR
COMPOSITION

context-free grammars



context-free grammars

```
public boolean
  authenticate(String user, String pw) {

    SQL stm = <| SELECT id FROM Users
                 WHERE name = ${user}
                 AND   password = ${pw} |>;

    return executeQuery(stm).size() != 0;
}
```

```
module Java-SQL
  imports
    Java SQL

  exports
    context-free syntax

    "<|" Query "|>" -> Exp {cons("ToSQL")}
    "${" Exp "}" -> SqlExp {cons("FromSQL")}
```

BEYOND
PARSERS

PRETTY PRINTERS

SENTENCE GENERATORS

AST ACCESS

Java – MiniJava/test/example.mjv – Eclipse SDK – /Users/guwac/Documents/EclipseWo...

Transform

Team Synchr...

example.mjv

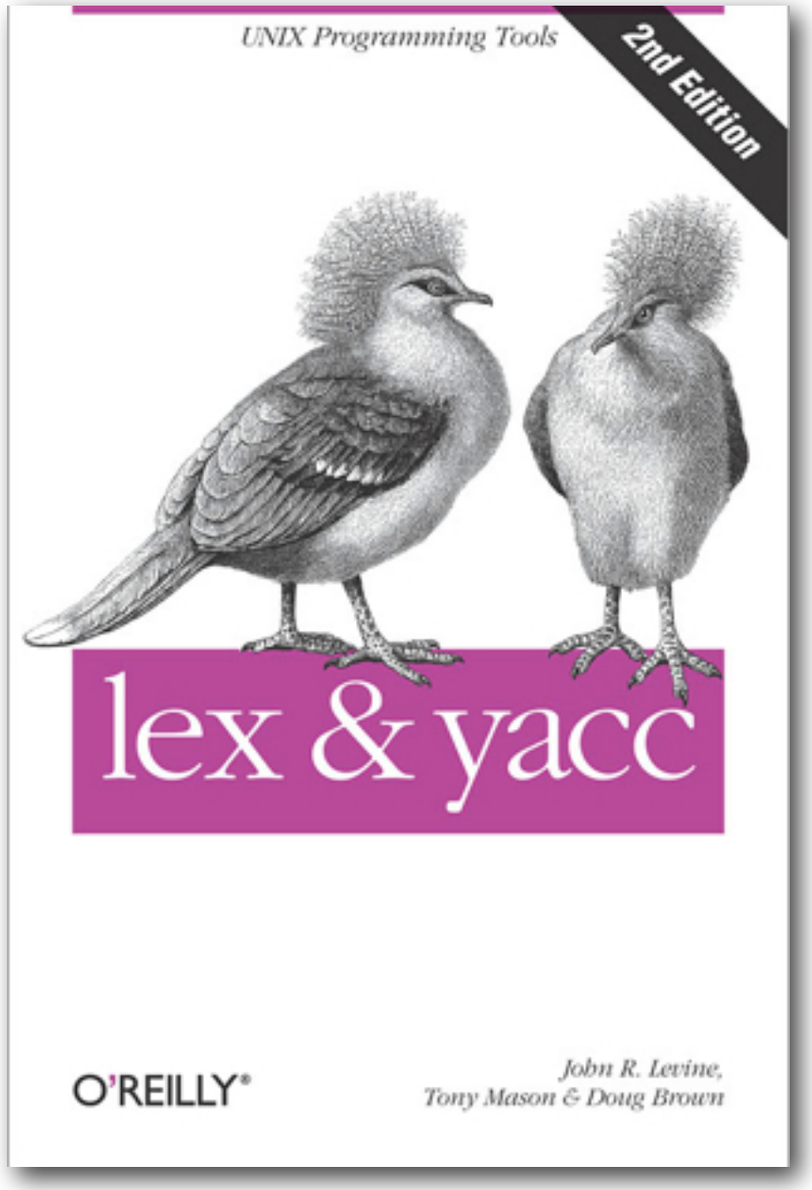
```
class Main {

    public static void main(String[] args) {
        System.out.println(0);
    }
}

class Fac {

    public int fac(int[] num) {
        int num_aux ;
        if (num < 1)
            num_aux = 1 ;
        else
            num_aux = num*this.fac(num-1) ;
        return num_aux ;
    }
}
```

Outline

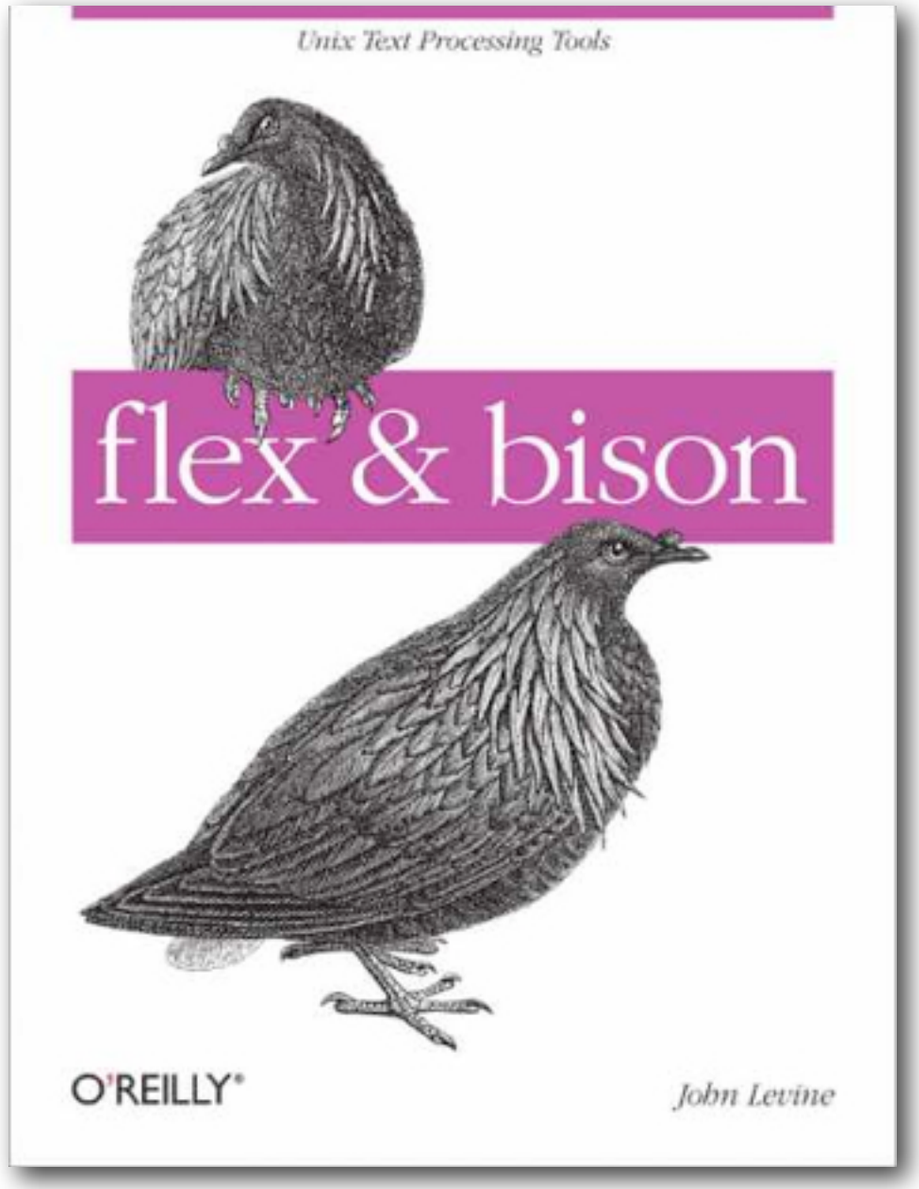
- Main
  - main
    - Void
    - args
      - StringArray
- Fac
  - fac
    - Int
    - num
      - Array
    - num_aux
      - Int

IDE SUPPORT

PARADISE **DENIED**



still around

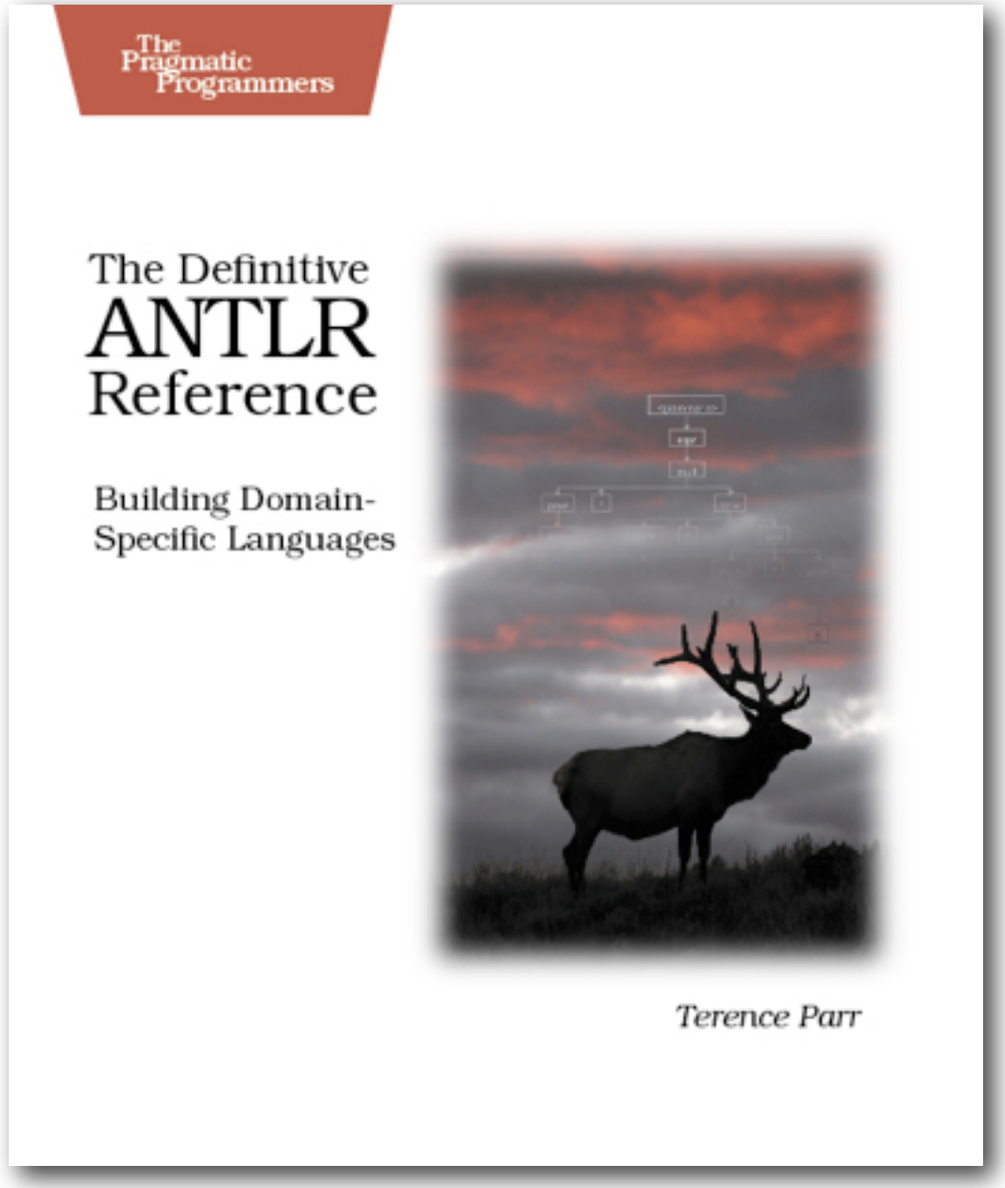


still around

still have to use it

still have to learn
LL, LR, SLR, LALR

still think using
parser generators is hard



modern parser generator

modern parser generator

PARADISE **OPEN**

coffee break

II

pretty-printing

Recap: Parsing & AST construction

```
let
   function fact(n : int): int = if n < 1 then 1 else n * fact(n - 1)
in
   printint(fact(10))
end
```

```
Let(
   [  FunDec(
         "fact"
      ,  [FArg("n", Tid("int"))]
      ,  Tid("int")
      ,  IfThenElse(
            Lt(Var("n"), Int("1"))
         ,  Int("1")
         ,  Times(Var("n"), Call("fact", [Minus(Var("n"), Int("1"))]))
         )
      )
   ]
,  [Call("printint", [Call("fact", [Int("10")])])]
)
```

Pretty Printing

from ASTs to text

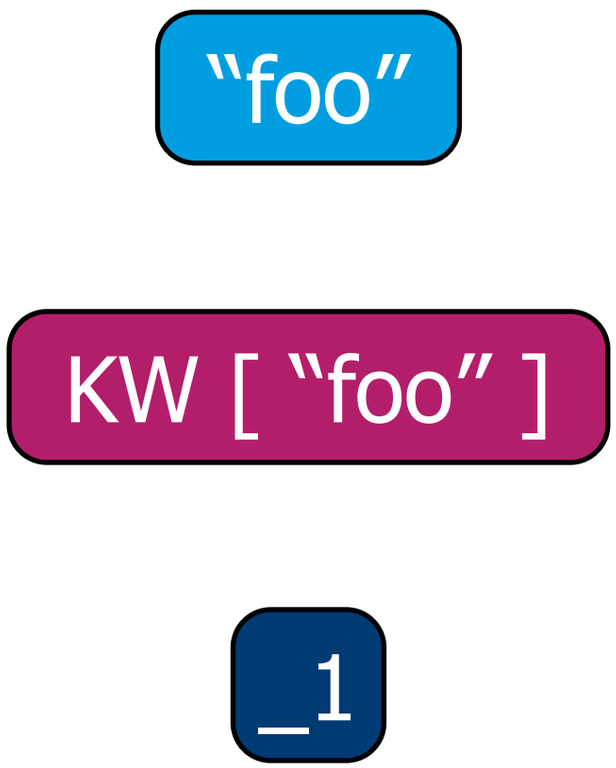
- keywords
- layout: spaces, line breaks, indentation

specification

- partially defined in grammar
- missing layout

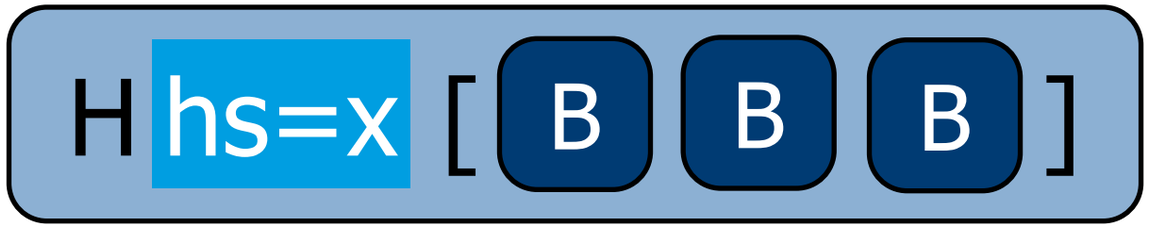
Box Layout

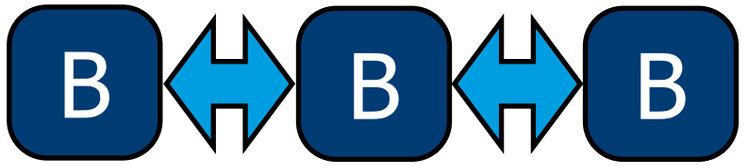
basic boxes



Box Layout

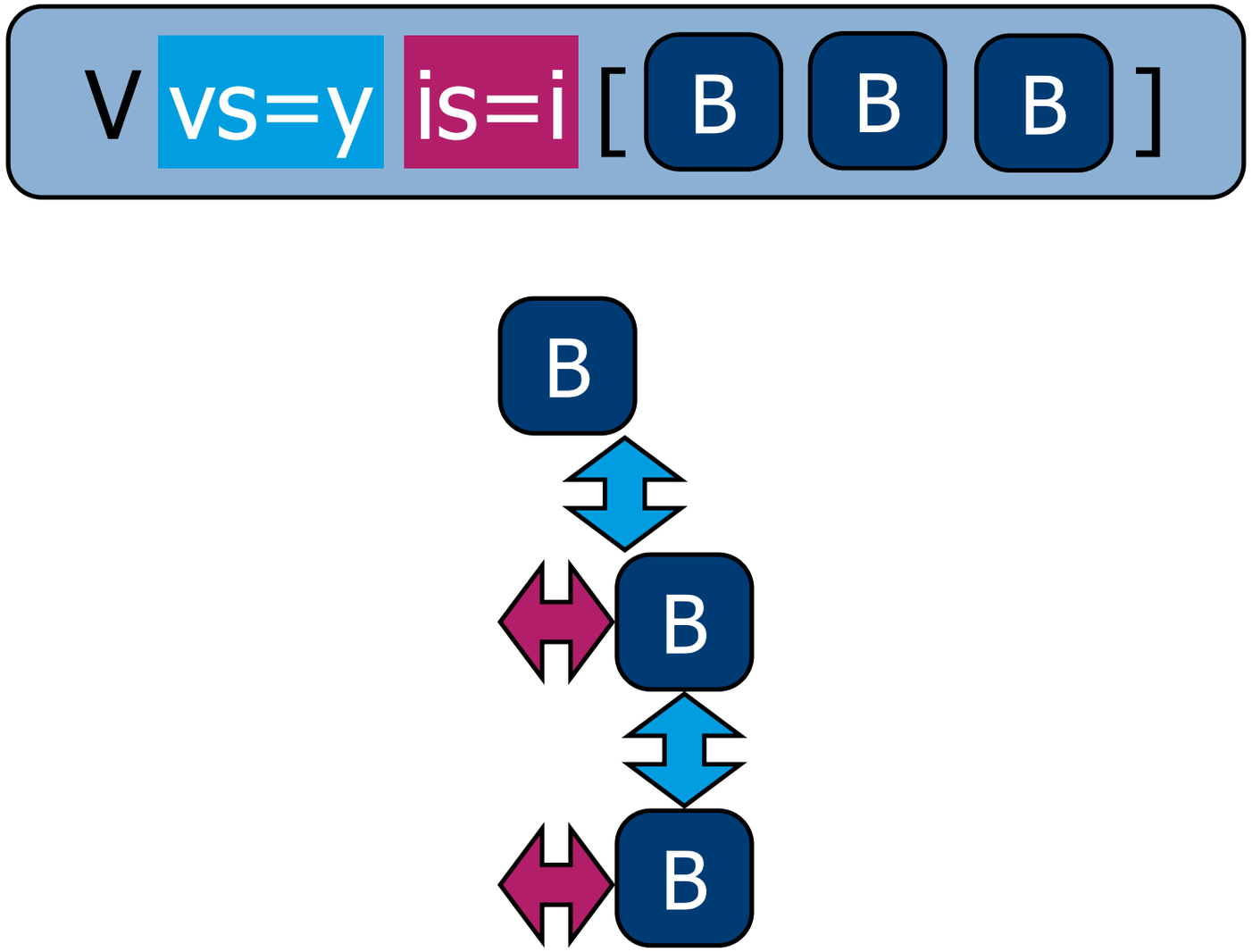
horizontal boxes





Box Layout

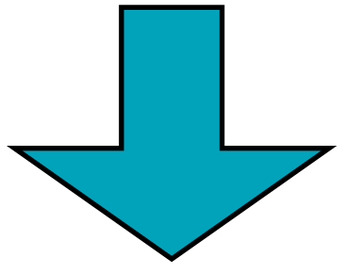
vertical boxes



Pretty Print Tables

generation

```
"if" Exp "then" Exp                    -> Exp {"IfThen"}
"let" Dec* "in" {Exp ";"}* "end" -> Exp {"Let"}
```



```
Exp.IfThen -- KW["if"] _1 KW["then"] _2,
Exp.Let    -- KW["let"] _1 KW["in"] _2 KW["end"],

Exp.Let.1:iter-star     -- _1,
Exp.Let.2:iter-star-sep -- _1 KW[";"]
```

Pretty Print Tables

manual override

```
"if" Exp "then" Exp                    -> Exp {"IfThen"}
"let" Dec* "in" {Exp ";"}* "end" -> Exp {"Let"}
```

```
Exp.Let -- V vs=1 is=0 [
             V vs=1 is=2 [KW["let"] _1]
             V vs=1 is=2 [KW["in"] _2]
             KW["end"]
           ]
```

III

syntax templates

Recap: SDF

```
sorts Exp

context-free syntax

   LValue ":=" Exp                          -> Exp

   "let" Dec* "in" {Exp ";"}* "end"         -> Exp

   "if" Exp "then" Exp                      -> Exp
   "if" Exp "then" Exp "else" Exp           -> Exp

   "while" Exp "do" Exp                     -> Exp
   "for" Id ":=" Exp "to" Exp "do" Exp -> Exp

   "break"                                  -> Exp

   "(" ")"                -> Exp
   "(" Exp ")"            -> Exp
   "(" {Exp ";"}+ ")" -> Exp
```

Pretty Print Tables

manual overrides

```
Exp.Let -- V vs=1 is=0 [
              V vs=1 is=2 [KW["let"] _1]
              V vs=1 is=2 [KW["in"] _2]
              KW["end"]
            ]
```

Recap: Code Completion

```
module Tiger-Completions

imports Tiger-Completions.generated

completions

   completion keyword  : "function"

   completion template Exp:

        "if " <e> " then\n\t" <s> (blank)

   completion proposer : editor-complete
```

Templates

explicit layout in syntax definitions

```
context-free syntax

   Exp "+"  Exp -> Exp {left, "Plus"}
   Exp "-"  Exp -> Exp {left, "Minus"}
```

```
context-free syntax

   Exp.Plus  = <<Exp> + <Exp>> {left}
   Exp.Minus = <<Exp> - <Exp>> {left}
```

Templates

explicit layout in syntax definitions

```
context-free syntax

   "let" Dec* "in" {Exp ";"}* "end" → Exp {"Let"}
```

```
context-free syntax

   Exp.Let = <
     let
       <Dec*; separator="\n">
     in
       <Exp*; separator=";\n">
     end>
```

IV

summary

Summary

lessons learned

Summary
lessons learned

Why are efficient parsing algorithms problematic?

- not longer pure, declarative, beautiful
- paradise lost: seven plagues
- paradise regained: scannerless generalised parsing

Summary
lessons learned

Why are efficient parsing algorithms problematic?

- not longer pure, declarative, beautiful
- paradise lost: seven plagues
- paradise regained: scannerless generalised parsing

How can we specify pretty-printers?

- generated from SDF
- manual overrides
- one stop shop: TemplateLanguage

Summary
lessons learned

Why are efficient parsing algorithms problematic?

- not longer pure, declarative, beautiful
- paradise lost: seven plagues
- paradise regained: scannerless generalised parsing

How can we specify pretty-printers?

- generated from SDF
- manual overrides
- one stop shop: TemplateLanguage

Literature

learn more

Literature

learn more

declarative syntax definition

Lennart C. L. Kats, Eelco Visser, Guido Wachsmuth: Pure and Declarative Syntax Definition - Paradise Lost and Regained. SPLASH 2010

Literature

learn more

declarative syntax definition

Lennart C. L. Kats, Eelco Visser, Guido Wachsmuth: Pure and Declarative Syntax Definition - Paradise Lost and Regained. SPLASH 2010

pretty-printing

Marc G.J. van den Brand, Eelco Visser: Generation of Formatters for Context-Free Languages. ACM TOSEM 5(1):1-41, 1996.

Merijn de Jonge. Pretty-Printing for Software Reengineering. ICSM 2002

Literature

learn more

declarative syntax definition

Lennart C. L. Kats, Eelco Visser, Guido Wachsmuth: Pure and Declarative Syntax Definition - Paradise Lost and Regained. SPLASH 2010

pretty-printing

Marc G.J. van den Brand, Eelco Visser: Generation of Formatters for Context-Free Languages. ACM TOSEM 5(1):1-41, 1996.

Merijn de Jonge. Pretty-Printing for Software Reengineering. ICSM 2002

template language

Tobi Vollebregt, Lennart C. L. Kats, Eelco Visser. Declarative Specification of Template-Based Textual Editors. LDTA 2012

Outlook
beyond IN4303

Model-driven Software Development

- domain-specific languages
- build your own language with Spoofax

Seminar Metaprogramming

- science behind the scenes

Master theses

- theory & practice
- WebDSL & mobl
- Spoofax

Outlook

coming next

declarative semantics definition

- Lecture 4: Term Rewriting
- Lecture 5: Static Analysis and Error Checking
- Lecture 6: Code Generation

Labs

- Sep 27 syntax definition
- Oct 4 code completion & pretty printing

copyrights

BY NC SA

Pictures

attribution & copyrights

Slide 1:
Latin Bible by Gerard Brils (photo by Adrian Pingstone), public domain

Slide 188:
West Cornwall Pasty Co. by Dominica Williamson, some rights reserved

Slide 199-200:
Ostsee by Mario Thiel, some rights reserved

Slides 5-180:
Pure and Declarative Syntax Definition: Paradise Lost and Regained, some rights reserved