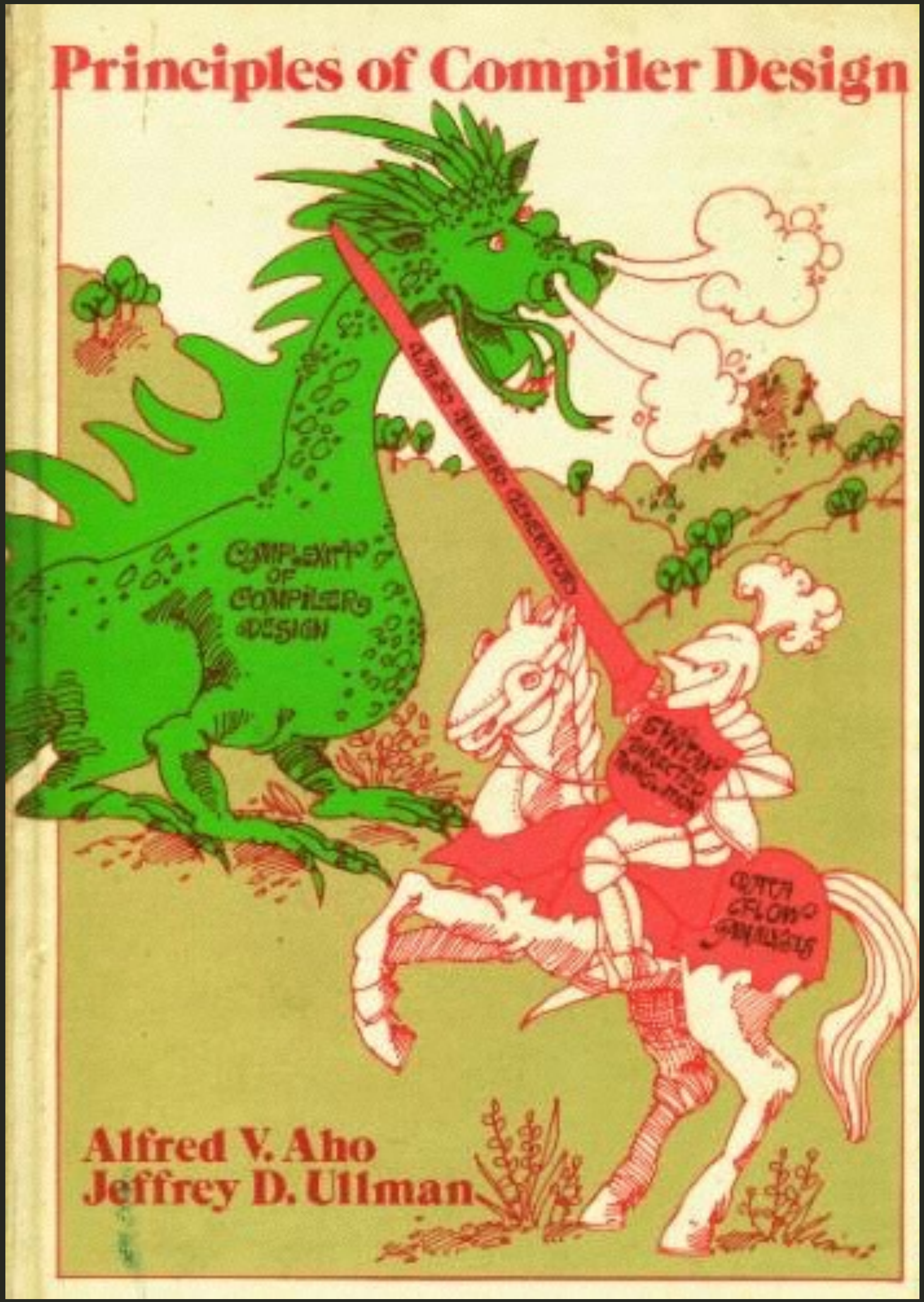
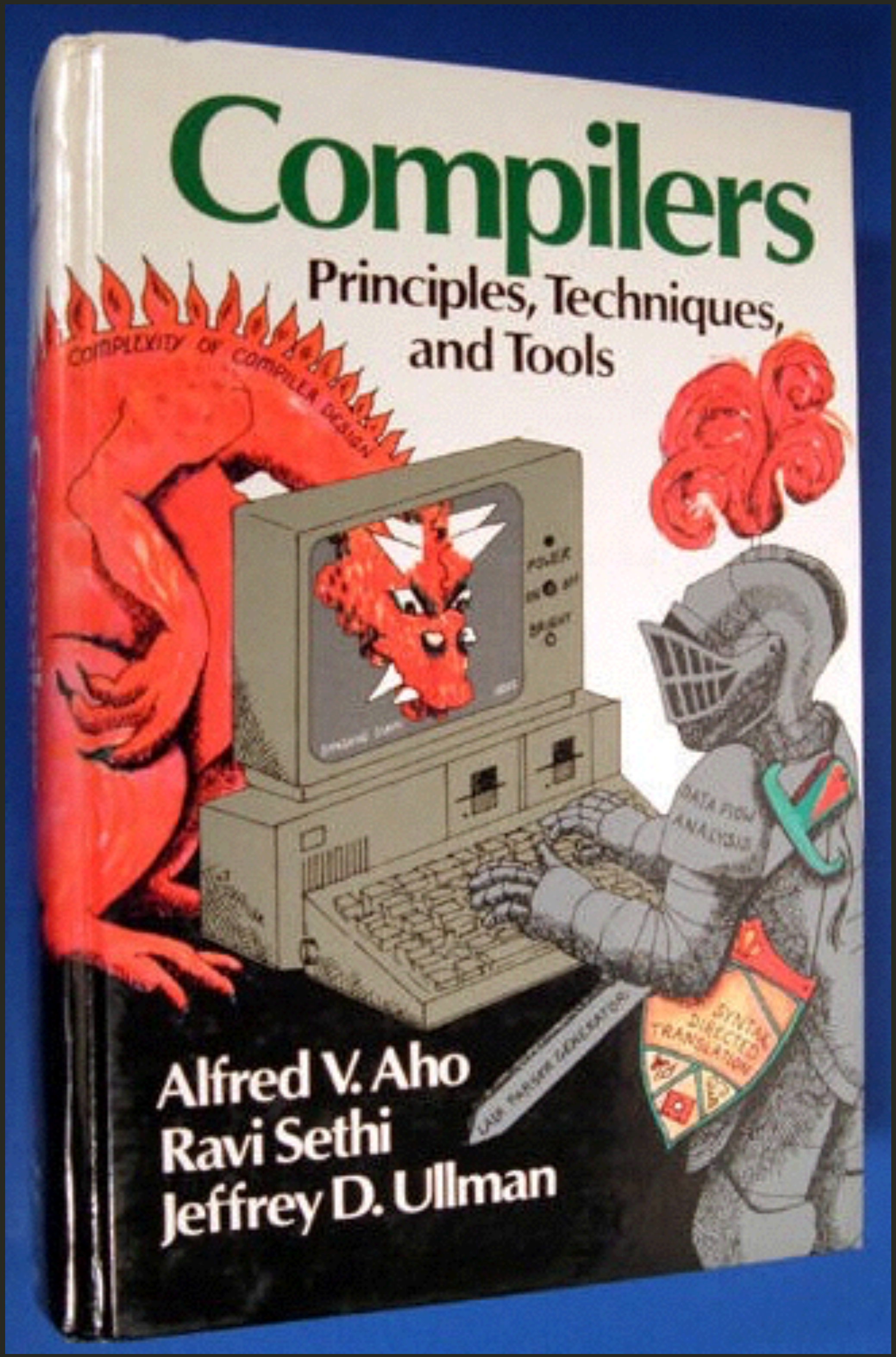
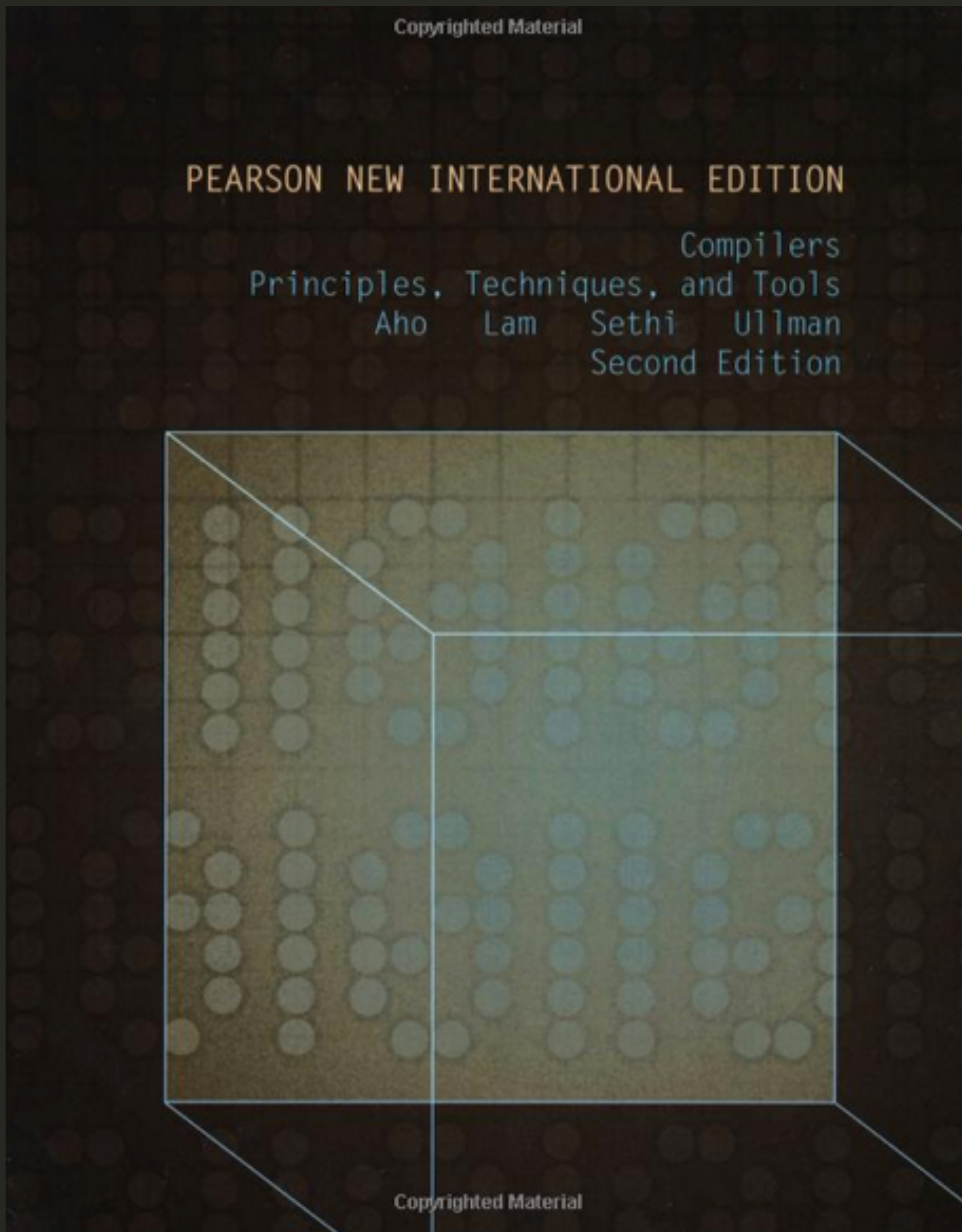
Grammars and Trees

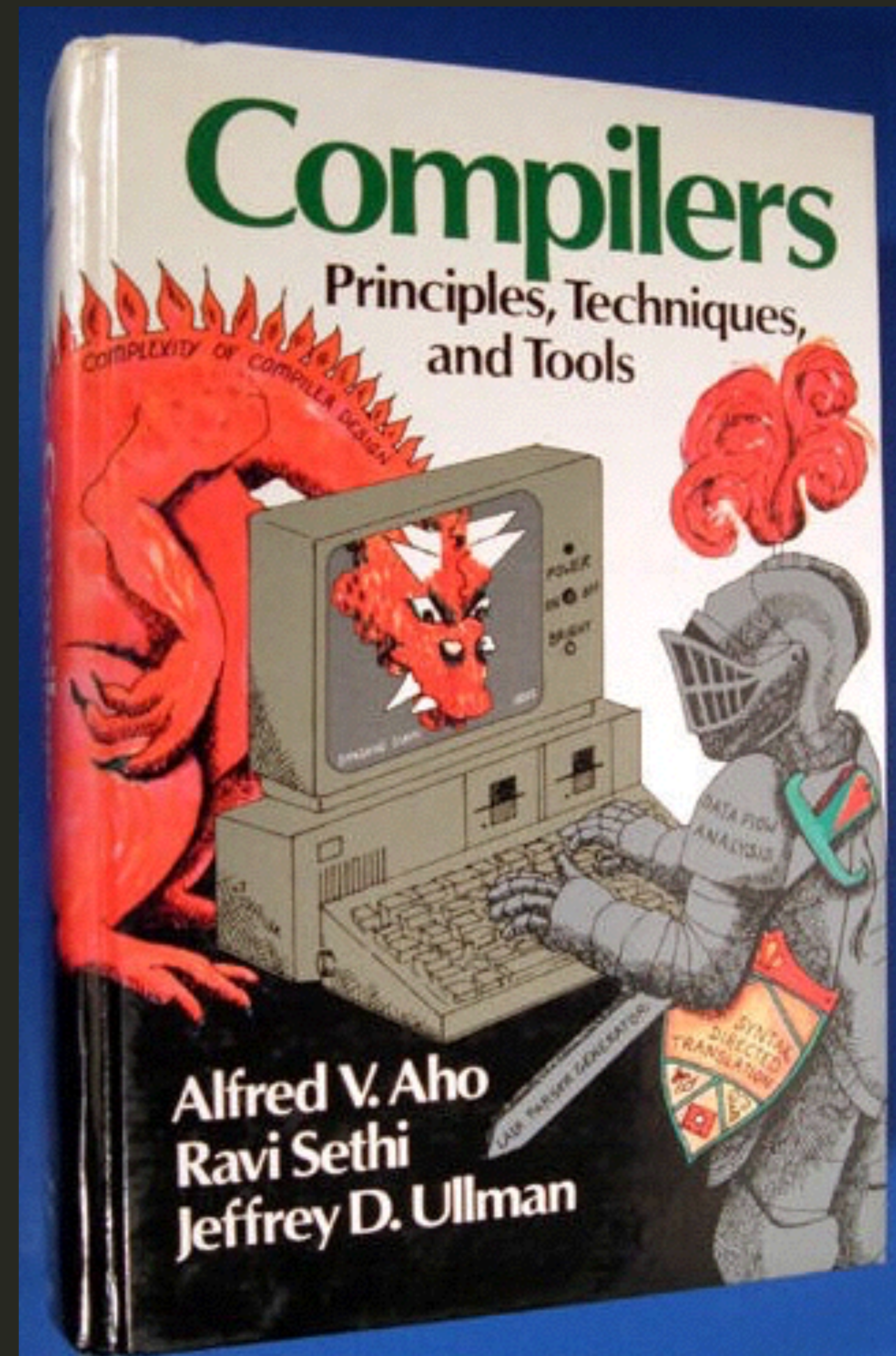
Dr. Vadim Zaytsev aka @grammarware

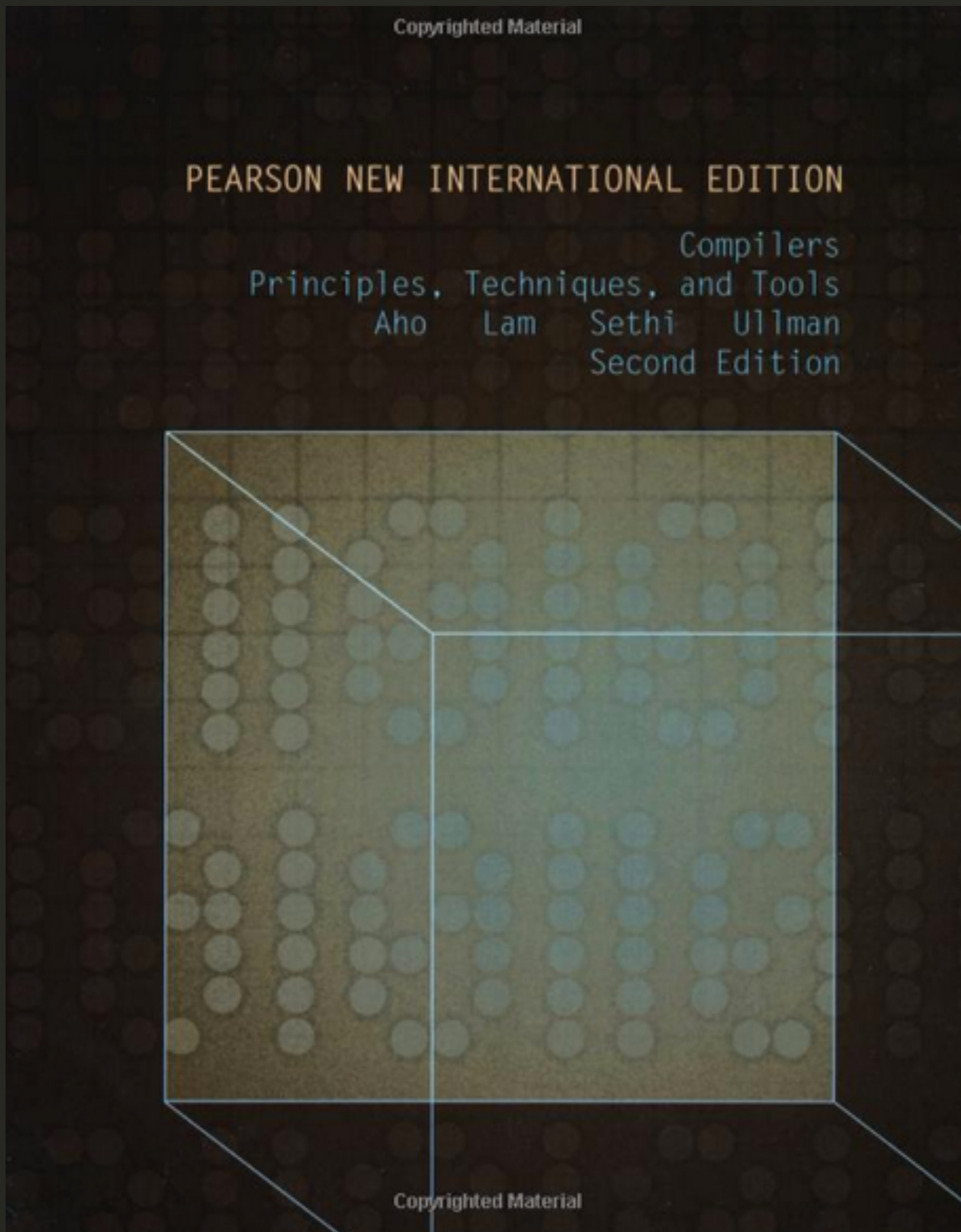
2015


Principles of Compiler Design
COMPLEXITY OF COMPILER DESIGN
Alfred V. Aho
Jeffrey D. Ullman

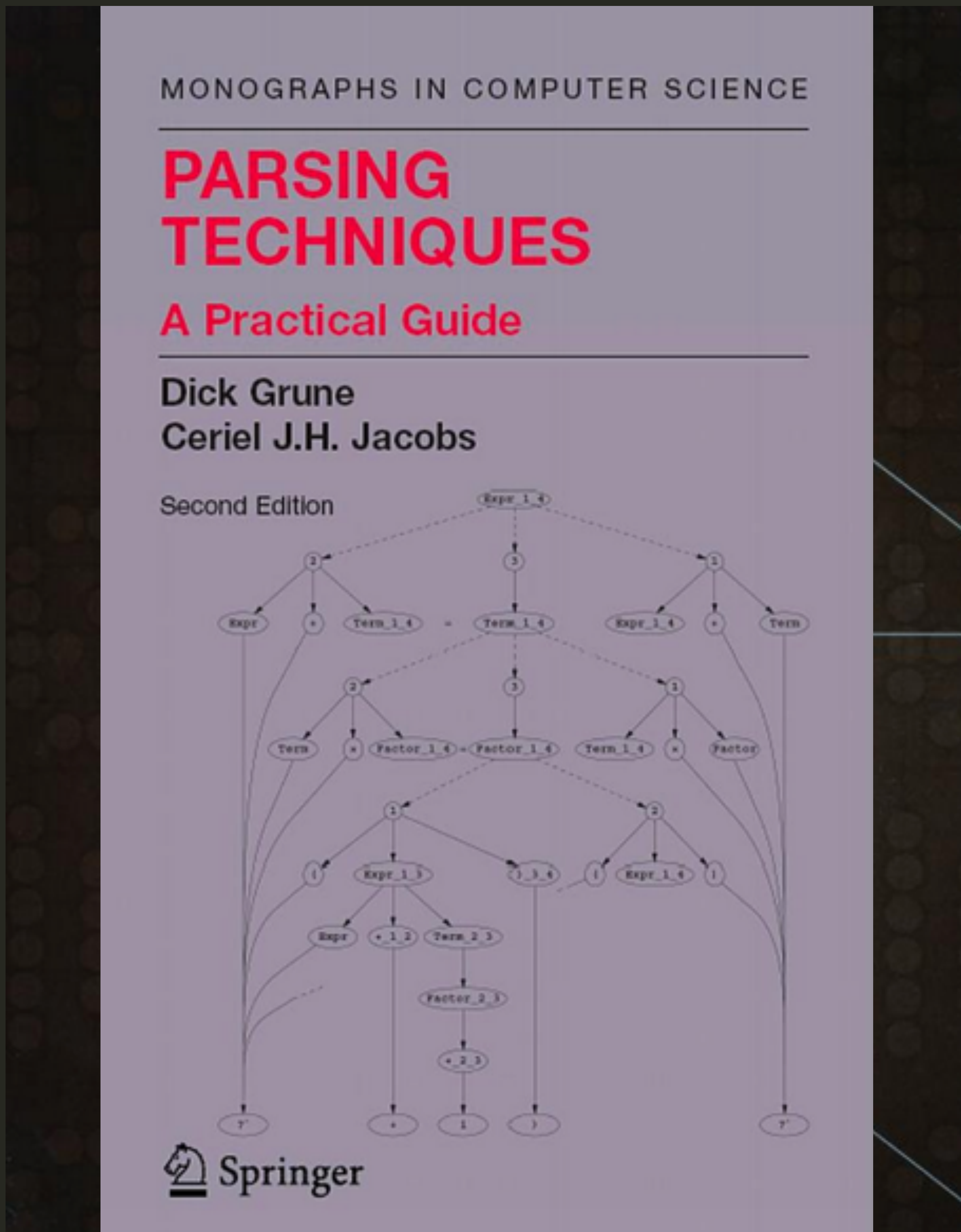

Compilers
Principles, Techniques, and Tools
COMPLEXITY OF COMPILER DESIGN
DATA FLOW ANALYSIS
SYNTAX DIRECTED TRANSLATION
Alfred V. Aho
Ravi Sethi
Jeffrey D. Ullman


Copyrighted Material
PEARSON NEW INTERNATIONAL EDITION
Compilers
Principles, Techniques, and Tools
Aho Lam Sethi Ullman
Second Edition
Copyrighted Material


Compilers
Principles, Techniques, and Tools
Alfred V. Aho
Ravi Sethi
Jeffrey D. Ullman


Copyrighted Material
PEARSON NEW INTERNATIONAL EDITION
Compilers
Principles, Techniques, and Tools
Aho Lam Sethi Ullman
Second Edition
Copyrighted Material


Copyrighted Material
Dick Grune · Kees van Reeuwijk
Henri E. Bal · Ceriel J.H. Jacobs
Koen Langendoen
Modern Compiler Design
Second Edition
EXTRA MATERIALS
extras.springer.com
Springer
Copyrighted Material

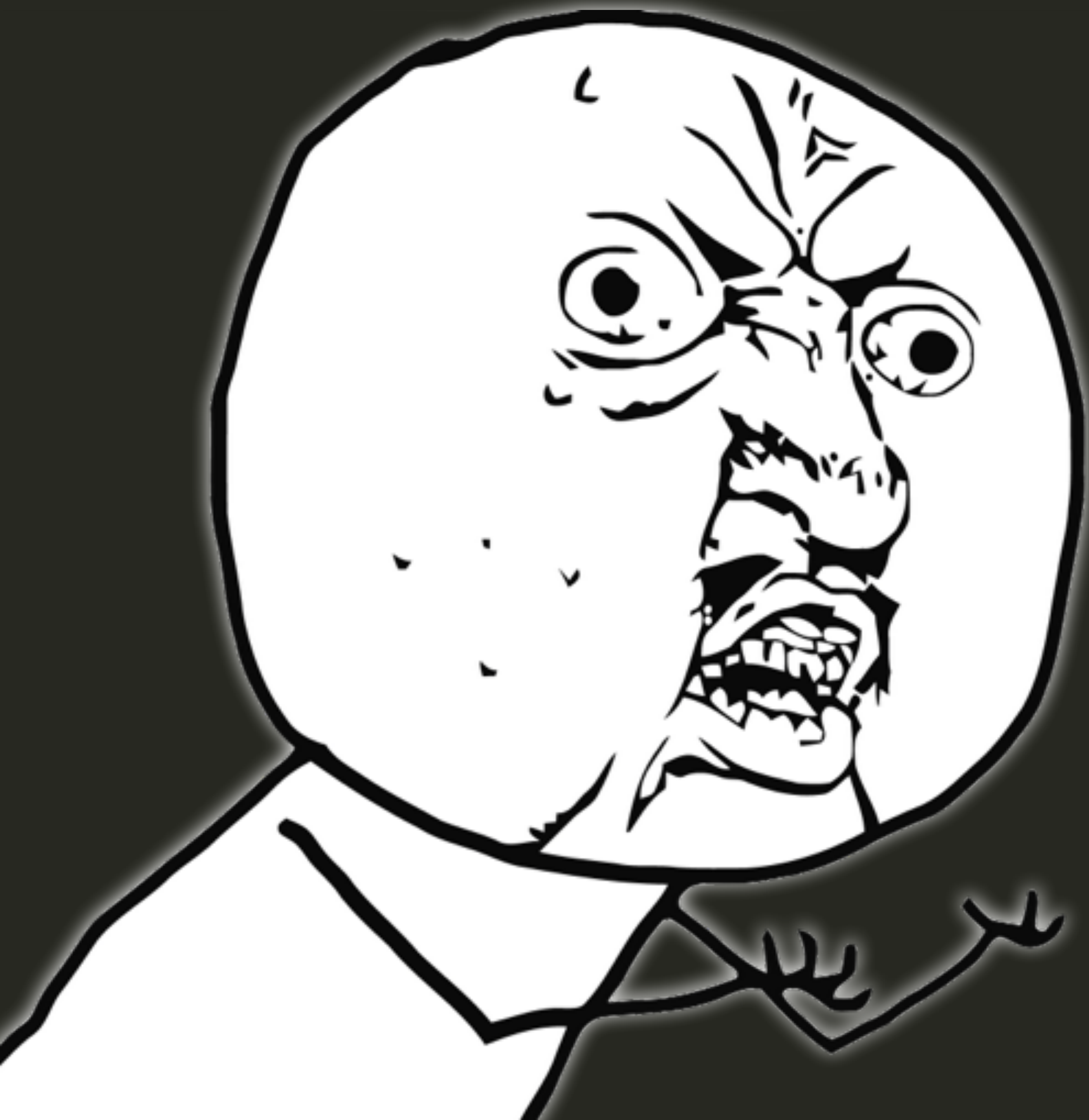

MONOGRAPHS IN COMPUTER SCIENCE
PARSING TECHNIQUES
A Practical Guide
Dick Grune
Ceriel J.H. Jacobs
Second Edition
Springer


Copyrighted Material
Dick Grune · Kees van Reeuwijk
Henri E. Bal · Ceriel J.H. Jacobs
Koen Langendoen
Modern Compiler Design
Second Edition
EXTRA MATERIALS extras.springer.com
Springer
Copyrighted Material

Recap

- ✓ Lexical analysis
- ✓ Syntactic analysis
- ✓ Semantic analysis
- ✓ Intermediate representation
- ✓ Code generation
- ✓ Optimisation
- ✓ . . .

WHY



- ✓ Formats everywhere
- ✓ DSLs are easy
- ✓ SLs have many faces
- ✓ 90% automated,
  10% hard work

Models of Languages

- ✓ How can a language be defined?

Models of Languages

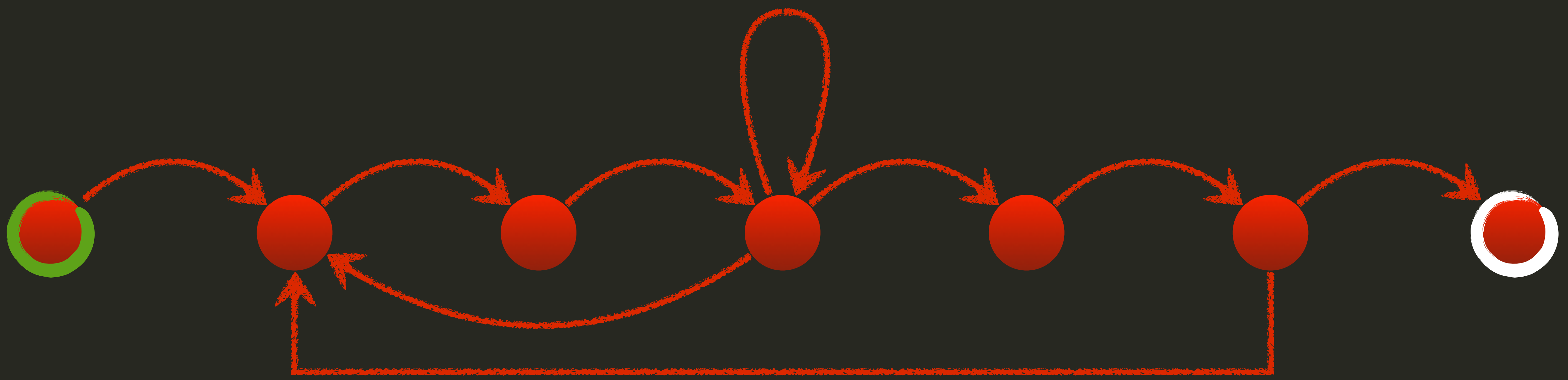
- ✓ Actual (in)finite set
  - ✓ {“a”, “b”, “c”}
  - ✓ $\{0^i 1^n \ldots\}$
  - ✓ English
  - ✓ set arithmetic works
    - ✓ concatenation, union, difference, intersection, complement, closure

Models of Languages

- ✓ Formal grammar
  - ✓ term rewriting system
  - ✓ “semi-Thue”
  - ✓ all about rewriting rules
    - ✓ $\alpha \rightarrow \beta$

Models of Languages

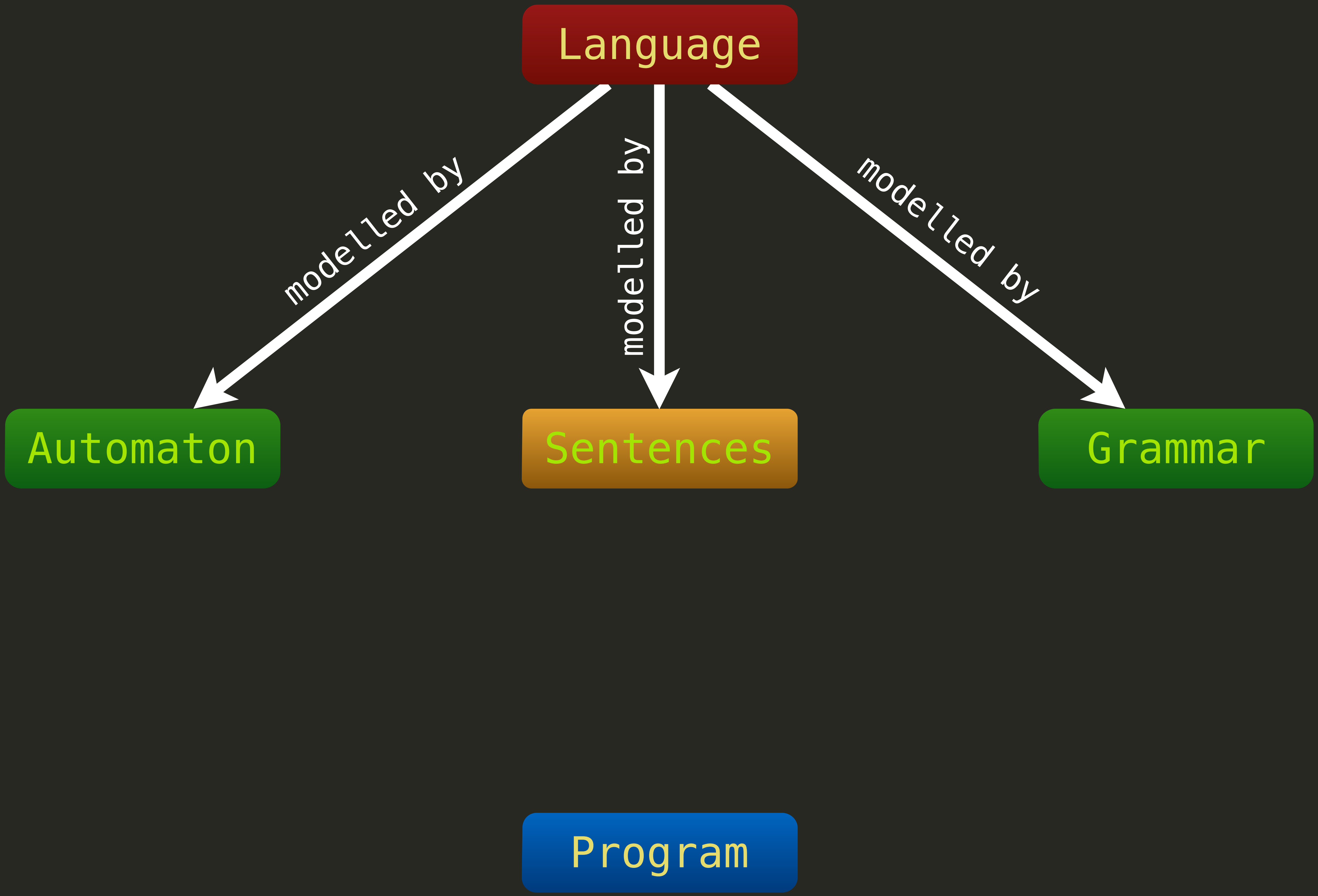
- ✓ Recognising automaton
  - ✓ states
  - ✓ transitions
  - ✓ extra stuff

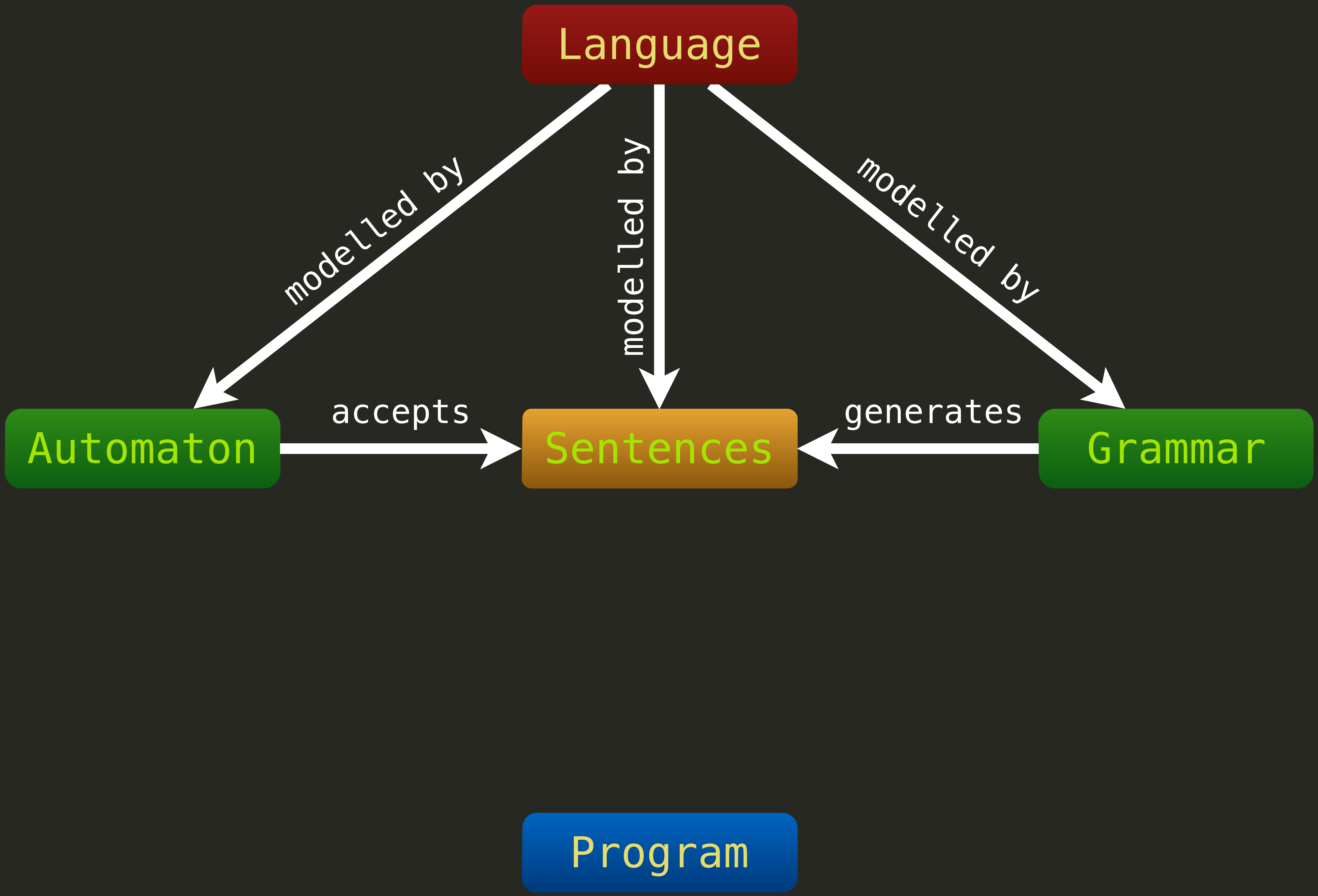


Models of Languages

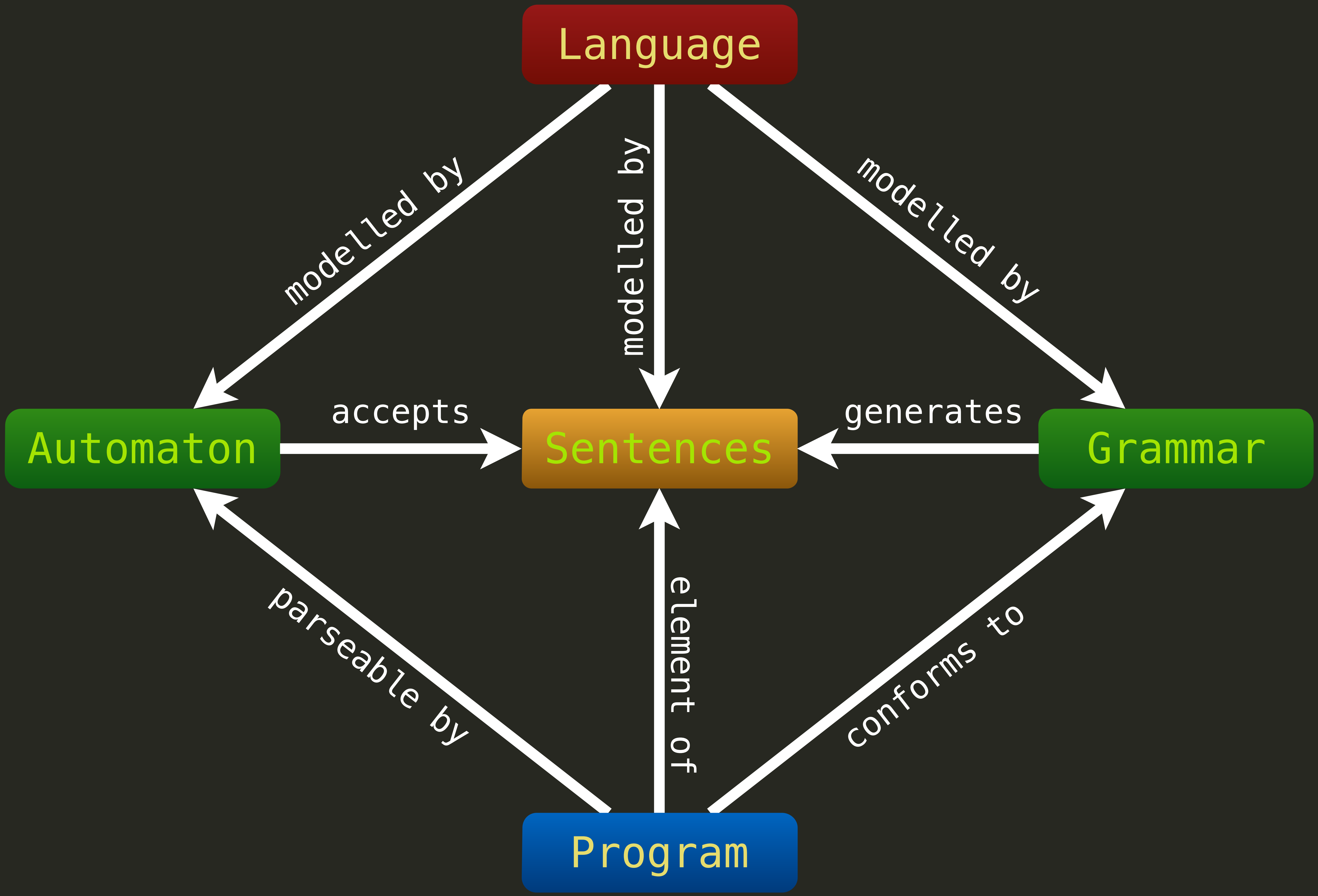
- ✓ Declarative
  - ✓ enumeration / description
  - ✓ characteristic function
- ✓ Analytic
  - ✓ recogniser / parser
  - ✓ analytic grammar
- ✓ Generative
  - ✓ term rewriting system
  - ✓ generative grammar

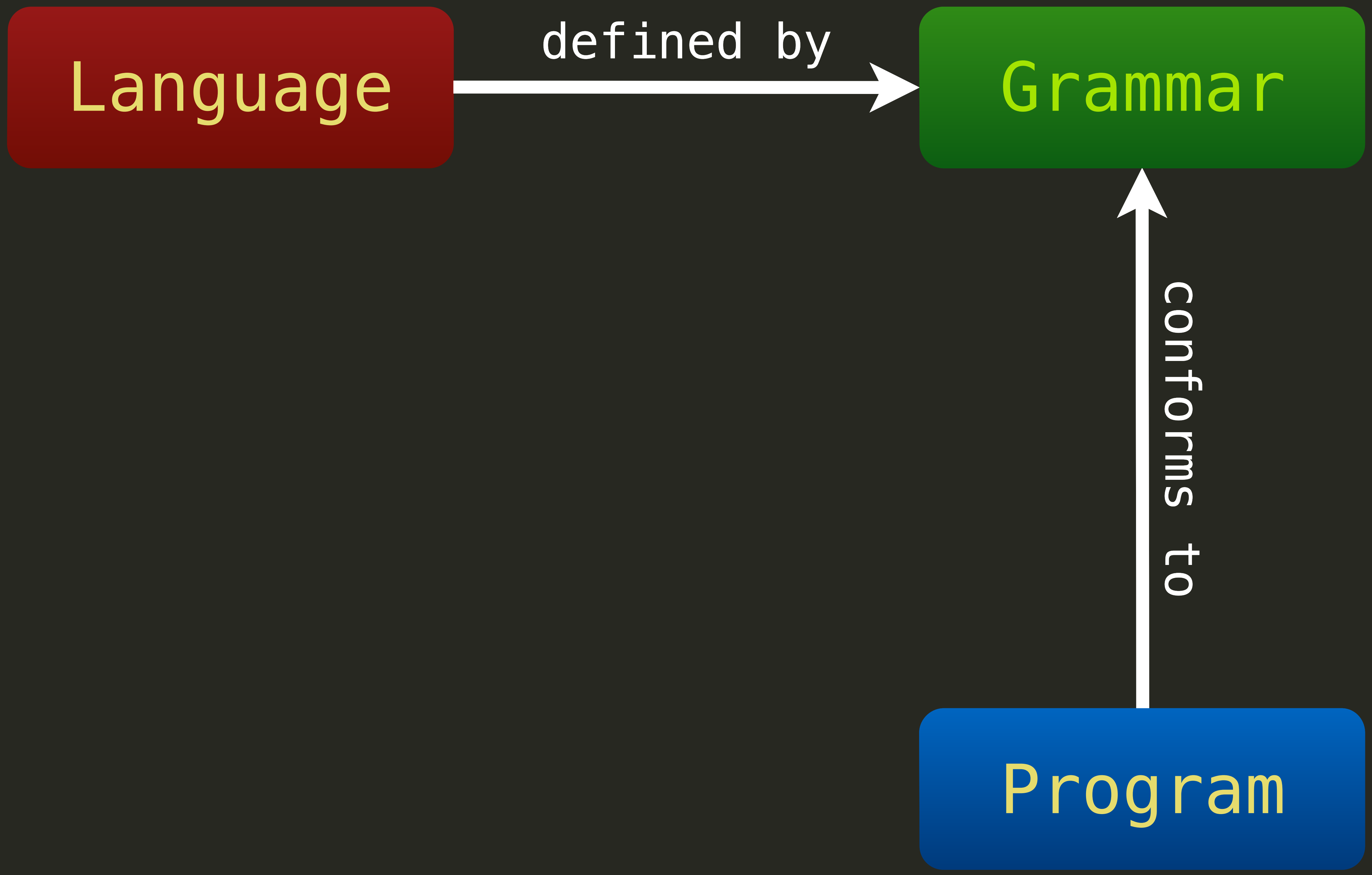

Language
instance of
Program

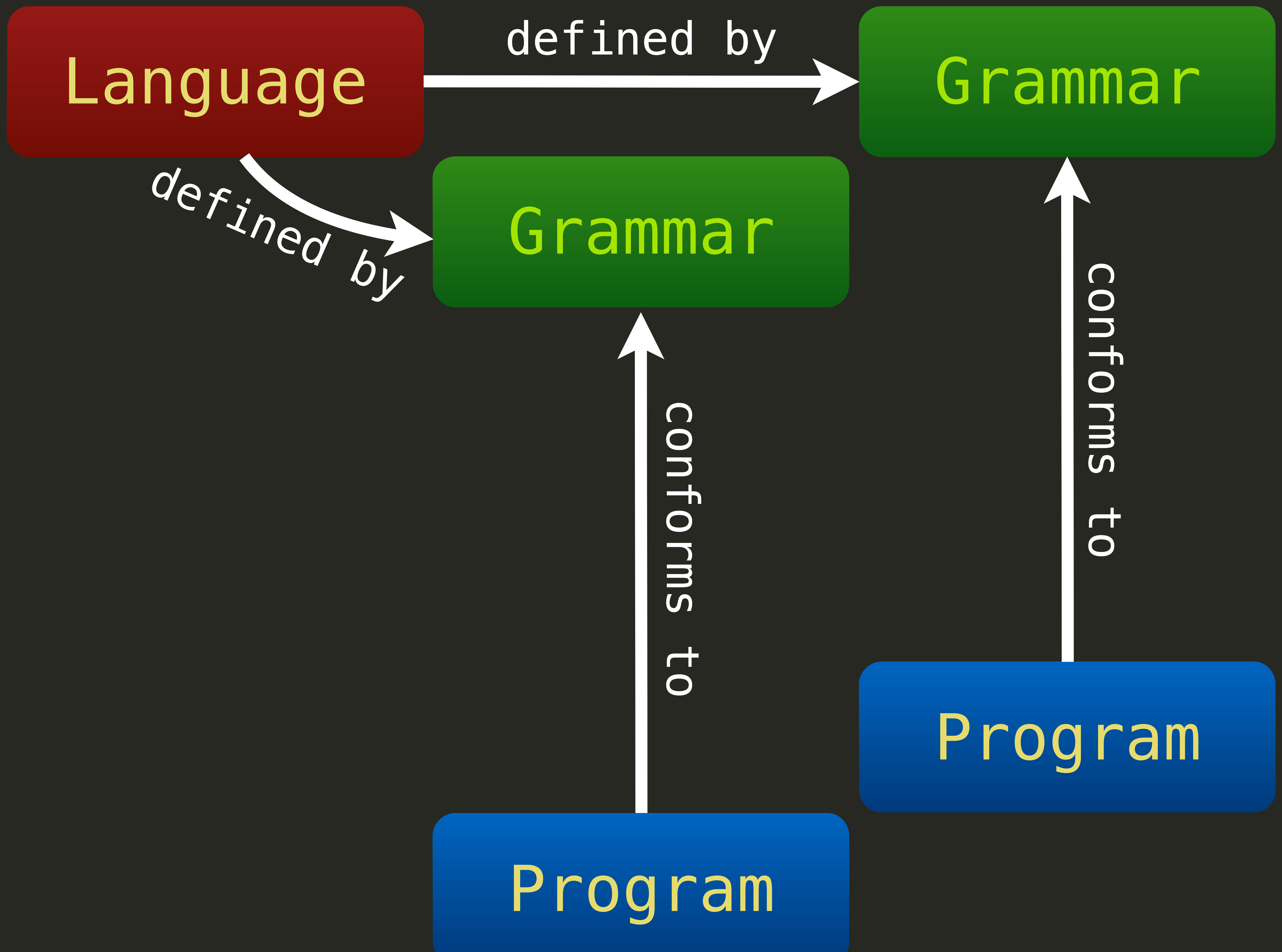

Language
modelled by
modelled by
modelled by
Automaton
Sentences
Grammar
Program



Language
modelled by
modelled by
modelled by
Automaton
accepts
Sentences
generates
Grammar
Program


Language
modelled by
modelled by
modelled by
Automaton
accepts
Sentences
generates
Grammar
parseable by
element of
conforms to
Program


Language
defined by
Grammar
conforms to
Program


Language
defined by
Grammar
defined by
Grammar
conforms to
conforms to
Program
Program

Example: XML

✓
```
X ::= ![<>]+
    | '<' ![>]+ '>' X* '<' '/' ![>]+ '>'
```

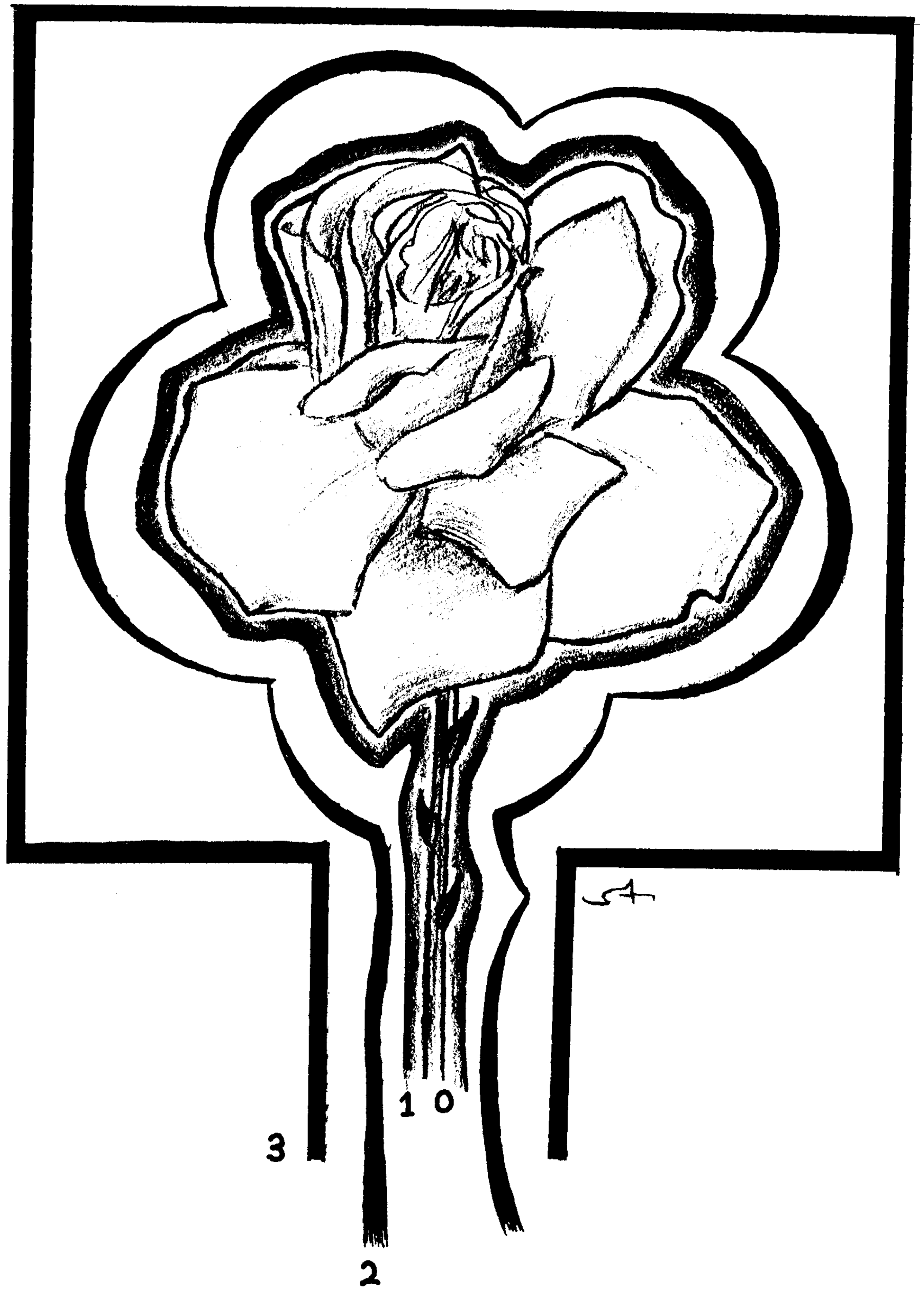
✓
```
X ::= D
    | '<' T A* '>' X* '<' '/' T '>'
```

✓
```
<!ELEMENT dir (#PCDATA)>
<!ATTLIST dir xml:space (def|preserve) 'preserve'>
```

✓
```
<xsd:element name="tag">
    <xsd:complexType>
          . . .
```

Conclusion

- ✓ “Language” is intangible
- ✓ Grammars hide in:
  - ✓ data types
  - ✓ API and libraries
  - ✓ protocols and formats
  - ✓ structural commitments
  - ✓ . . .
- ✓ Not all grammars are equally “good”



Rose by Arwen Grune; p.58 of Grune/Jacobs' “Parsing Techniques”, 2008

Unrestricted grammars

$\alpha \rightarrow \beta$

Context-sensitive grammars

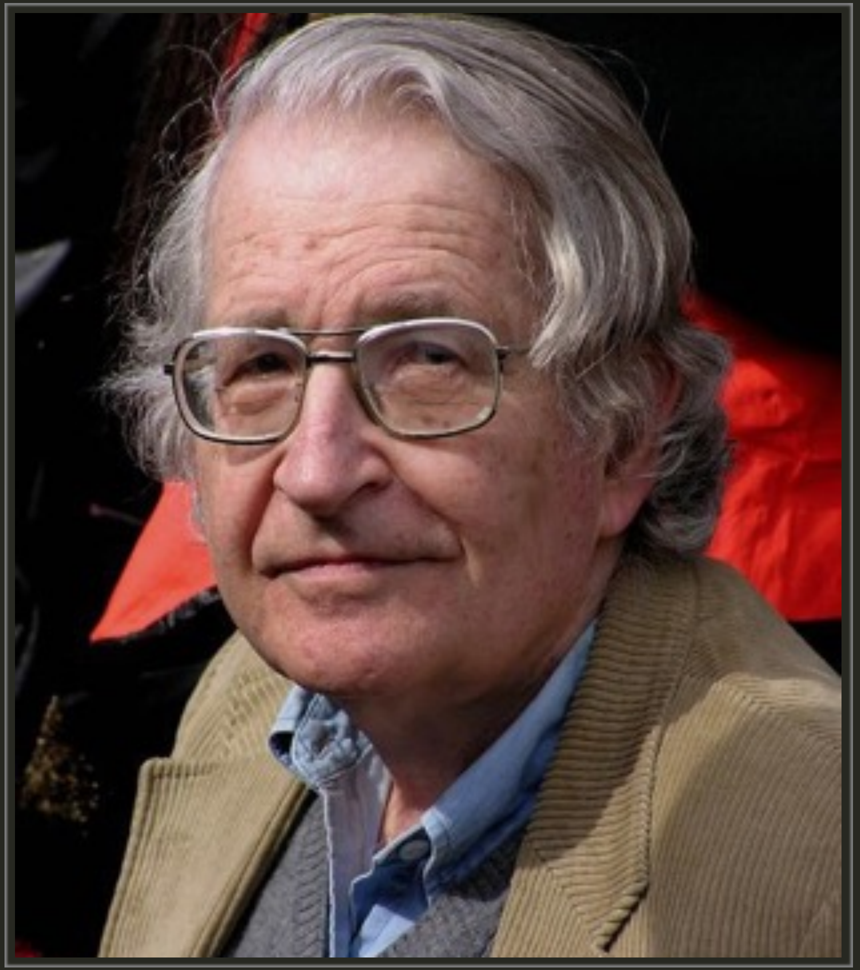
$\alpha X \beta \rightarrow \alpha \gamma \beta$

Context-free grammars

$X \rightarrow \gamma$

Regular grammars

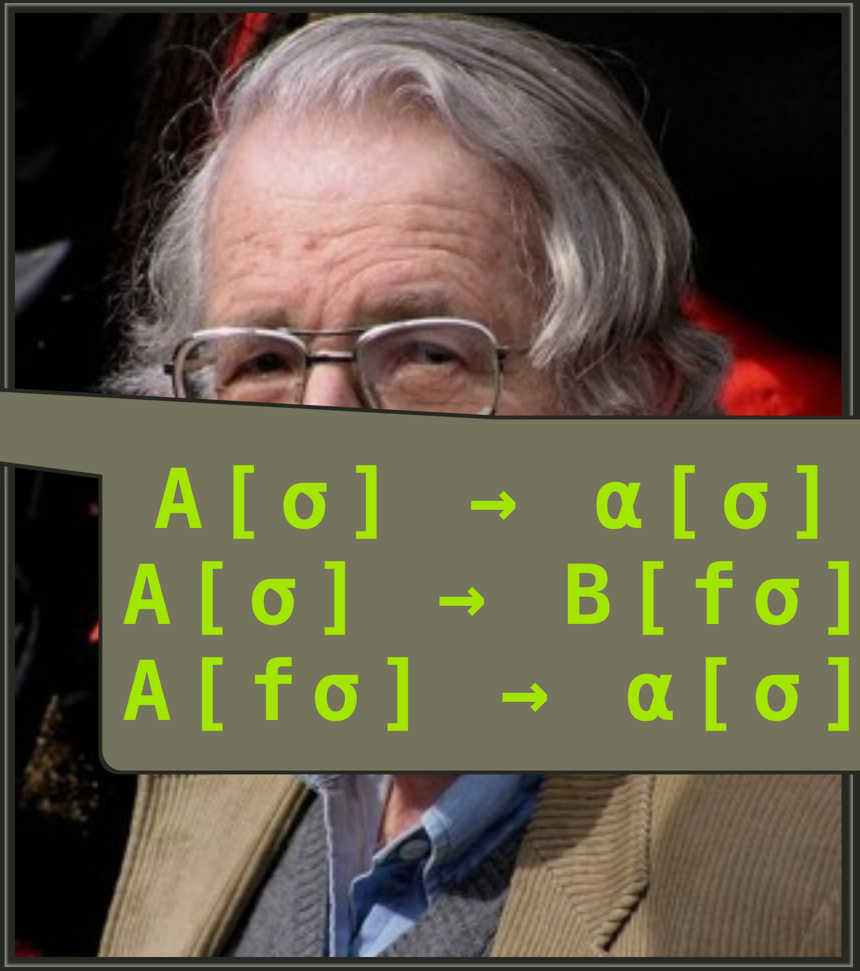
$X \rightarrow a$
$X \rightarrow aB$



Noam Chomsky (b.1928)

Noam Chomsky. *On Certain Formal Properties of Grammars*, Information & Control 2(2):137–167, 1959.

- Unrestricted grammars: $\alpha \to \beta$
- Decidable grammars
- Context-sensitive grammars: $\alpha X \beta \to \alpha \gamma \beta$
- Indexed grammars: $A[\sigma] \to \alpha[\sigma]$, $A[\sigma] \to B[f\sigma]$, $A[f\sigma] \to \alpha[\sigma]$
- Context-free grammars: $X \to \gamma$
- Deterministic CFG
- Nested word
- Regular grammars: $X \to a$, $X \to aB$
- Non-recursive grammars



Noam Chomsky (b.1928)

Noam Chomsky. *On Certain Formal Properties of Grammars*, Information & Control 2(2):137–167, 1959.

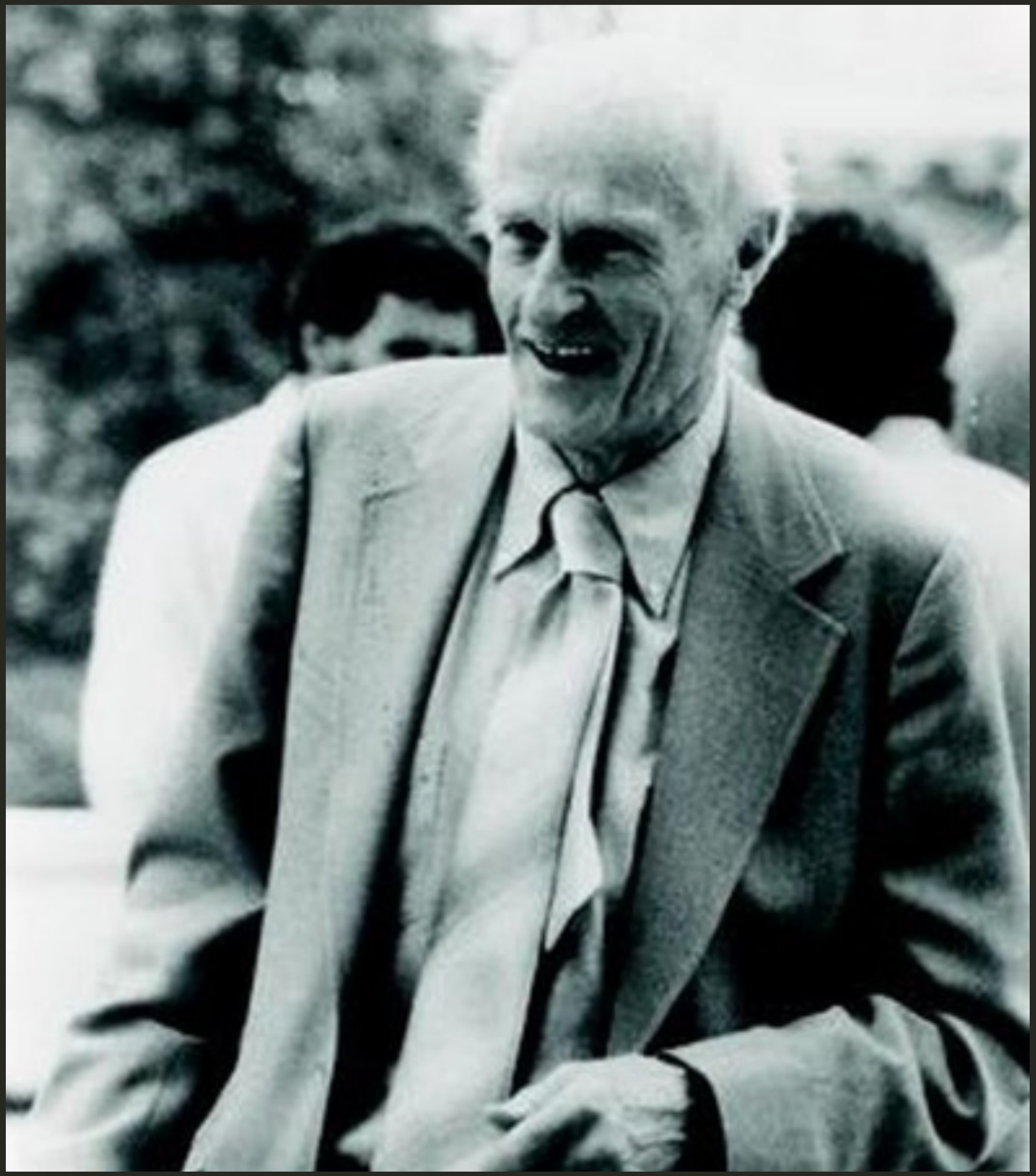
Unrestricted grammars	Recursively enumerable languages	Turing machine
Decidable grammars	Recursive languages	Terminating automata
Context-sensitive grammars	Context-sensitive languages	Linear-bounded automata
Indexed grammars	Languages with macros	Nested stack automata
Context-free grammars	Context-free languages	Pushdown automata
Deterministic CFG	Deterministic CFL	Deterministic PDA
Nested word	Nested word	Visibly PDA
Regular grammars	Regular languages	FSMs
Non-recursive grammars	Finite languages	FSMs without cycles

Finite languages

- Examples:
  - Boolean values
  - languages
  - countries
  - cities
  - postcodes

Regular languages

- ✓ Regular sets by Stephen Kleene in 1956
- ✓ ∅, ε, letters from Σ
- ✓ concatenation
- ✓ iteration
- ✓ alternation
- ✓ Precisely fit the regular class



Stephen Cole Kleene (1909–1994)

S. C. Kleene, *Representation of Events in Nerve Nets and Finite Automata*. In *Automata Studies*, pp. 3-42, 1956.
photo from: Konrad Jacobs, S. C. Kleene, 1978, MFO.

Regular languages

- ✓ PCRE
  - ✓ “Perl-compatible regular expressions”
  - ✓ (not compatible with Perl)
  - ✓ (not regular)
  - ✓ C library
  - ✓ (backrefs, recursion, assertions…)

Context-free

- ✓ FSM + memory (stack)
- ✓ Modular composition
  - ✓ A ::= “[” B “]” ;
  - ✓ B ::= A? ;

- ✓ Forget intersection & diff
- ✓ Closed under substitution

John Backus
(1924-2007)

Context-sensitive

- ✓ Explainable only in context
  - ✓ Sentence → List End
  - ✓ List → Name;
  - ✓ List → List “,” Name;
  - ✓ “,” Name End → “and” Name
- ✓ Parsing in exponential time

Unbounded

- ✓ (almost) anything
- ✓ recognising is impossible
- ✓ parsing is impossible

Which is which?

- ✓ Substring search
  - ✓ grep, contains(), find(), substring(), …
- ✓ Substring replacement
  - ✓ sed, awk, perl, vim, replace(), replaceAll(), …
- ✓ Pretty-printing
  - ✓ VS.NET, Sublime, TextMate, …

Which is which?

- ✓ Counting [non-empty] lines in a file
  - ✓ wc -l, grep -c “”
  - ✓ grep -v “^$”, sed -n /./p | wc -l
- ✓ Parsing HTML
  - ✓ <BODY><TABLE><P><A HREF=…
- ✓ Parsing a postcode
  - ✓ 1098 XG, …

Popular languages

- ✓ $\{a^i b^n \ldots\}$
  - ✓ 0 counters
  - ✓ 1 counter
  - ✓ n counters
  - ✓ ∞ counters
- ✓ Dyck language
  - ✓ parentheses
  - ✓ named parentheses

Walther von Dyck
(1856-1934)

Popular parsers

- ✓ Bottom-up
  - ✓ Reduce the input back to the start symbol
  - ✓ Recognise terminals
  - ✓ Replace terminals by nonterminals
  - ✓ Replace terminals and nonterminals by left-hand side of rule
  - ✓ LR, LR(0), LR(1), LR(k), LALR, SLR, GLR, SGLR, CYK, …
- ✓ Top-down
  - ✓ Imitate the production process by rederivation
  - ✓ Each nonterminal is a goal
  - ✓ Replace each goal by subgoals (= elements of its rule)
  - ✓ Parse tree is built from top to bottom
  - ✓ LL, LL(1), LL(k), LL(*), GLL, DCG, RD, Packrat, Earley

Popular parsers

✓ Bottom-up

- YACC / bison
- Beaver
- SableCC
- GDK
- Tom
- ASF+SDF
- Spoofax

✓ Top-down

- JavaCC
- ANTLR
- ModelCC
- Rascal
- TXL
- Rats!
- PetitParser

Popular data structures

- ✓ Lists (of tokens)
- ✓ Trees (hierarchy!)
- ✓ Forests (many trees)
- ✓ Graphs (loops!)
- ✓ Relations (tables)

Conclusion

- ✓ Parsing recognises structure
- ✓ Can be many models of a language
- ✓ Hierarchy of classes
- ✓ 90% automated, 10% hard work

Lexical syntax

- ✓ Terminal symbols
  - ✓ finite sublanguage
  - ✓ regular sublanguage
- ✓ Keywords
- ✓ Layout
  - ✓ whitespace
  - ✓ comments

Lexical syntax

```
lexical Boolean = "True" | "False";
```

```
lexical Id = [a-z]+ !>> [a-z];
```

```
keyword Reserved = "if" | "while";
lexical Id = [a-z]+ \ Reserved !>> [a-z];
```

```
lexical WS = [\ \t\n\r];
```

```
lexical Cm = "--" ... $;
```

```
layout L = (WS|Cm)*
!>> [\ \t\n\r] !>> "--";
```

- ✓ Termi
- ✓ K
- ✓ Layo
  - ✓ whi espa
  - ✓ comments

Lexical syntax

XML

```
layout L = [\ \t\n\r]* !>> [\ \t\n\r];
lexical D = ![\<\>]* !>> ![\<\>];
lexical T = [a-z][a-z0-9]* !>> [a-z0-9];
lexical A = [a-z]+ [=] [\"] ![\"]* [\"];
lexical X = D
       | "\<" T A* "\>" X+ "\<" "/" T "\>";
```

Beyond lexical

XML

```
layout L = [\ \t\n\r]* !>> [\ \t\n\r];
lexical D = ![\<\>]* !>> ![\<\>];
lexical T = [a-z][a-z0-9]* !>> [a-z0-9];
lexical A = [a-z]+ [=] [\"] ![\"
lexical X = D
  | "\<" T L {A L}* "\>" X+ "\<"
```

Beyond lexical

XML

```
layout L = [\ \t\n\r]* !>> [\ \t\n\r];
lex                       >> ![\<\>];
lex                       -9]* !>> [a-z0-9];
lexical A = [a-z]+ [=] [\"] ![\"
lexical X = D
  | "\<" T L {A L}* "\>" X+ "\<"
```

lexical → syntax

Beyond lexical

XML

```
layout L = [\ \t\n\r]* !>> [\ \t\n\r];
syntax D = W+;
lexical W = ![\ \t\n\r\<\>]+
        !>> ![\ \t\n\r\<\>];
lexical T = [a-z][a-z0-9]* !>> [a-z0-9];
lexical A = [a-z]+ [=] [\"] ![\"]* [\"];
syntax X = D
        | "\<" T A* "\>" X* "\<" "/" T "\>";
```

Recap: lexical

- ✓ Terminal: "if"
- ✓ Character class: [a-z]
- ✓ Inverse: ![a-z]
- ✓ Kleene closures: [a-z]+, [a-z]*
- ✓ Optionals: [a-z]?
- ✓ Reserve: [a-z]+ \ Keywords
- ✓ Follow: [a-z]+ !>> [a-z]

Beyond lexical

- ✓ Choice: |
- ✓ Priority: >
- ✓ Associativity: left, right, non-assoc
- ✓ Named alternatives: foo: x
- ✓ Named symbols: E left "+" E right
- ✓ Regular combinators: X*, X+, X?

Useful

- ✓ parse(#N, s)
- ✓ try parse(#N, s) catch: . . .
- ✓ vis::ParseTree::renderParsetree(t)
- ✓ /amb(_) !:= t
- ✓ t is foo
- ✓ t.x
- ✓ if (pattern := tree) . . .
- ✓ (E)`<E e1> + <E e2>`
- ✓ /regexp/