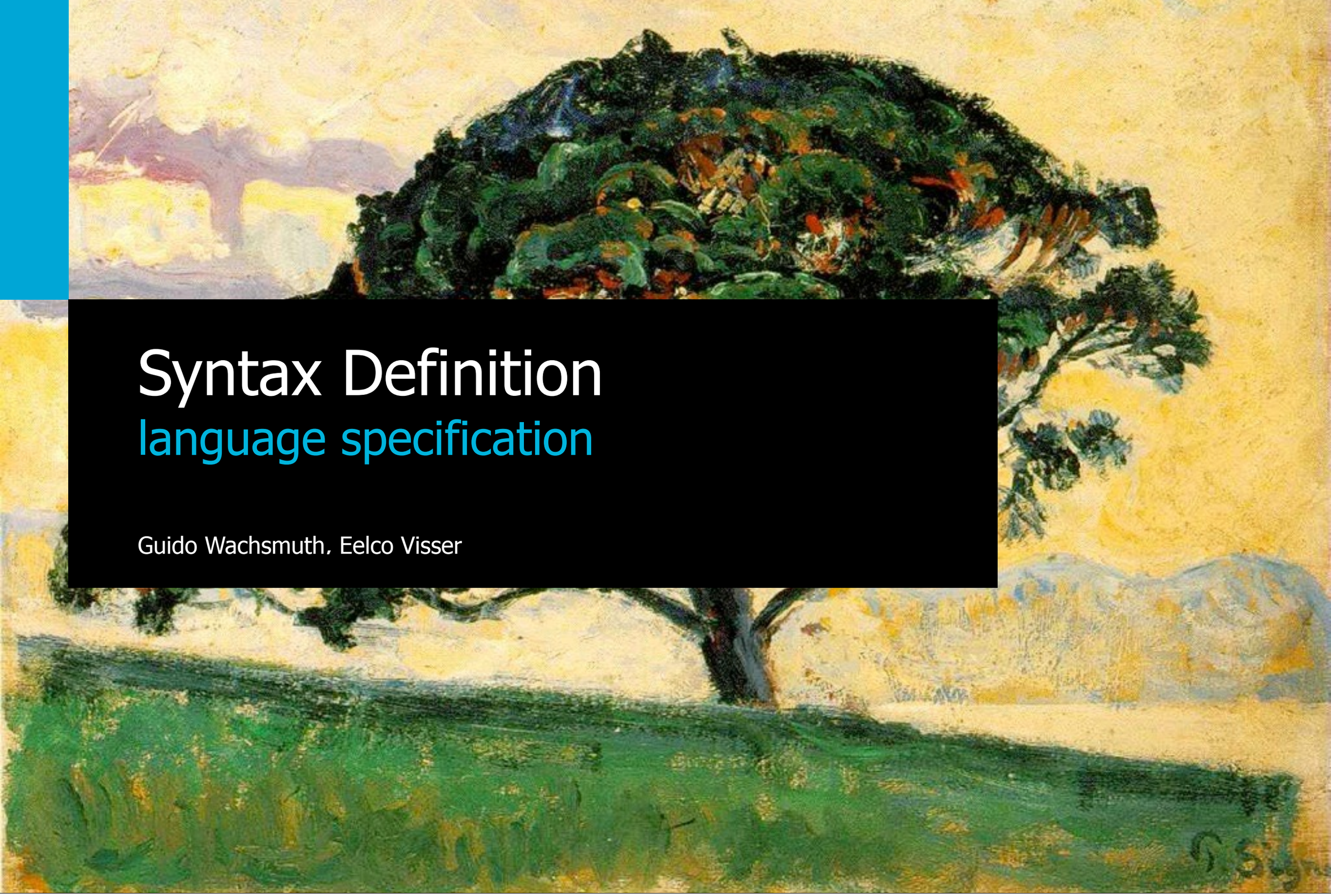
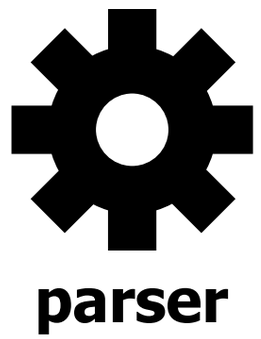
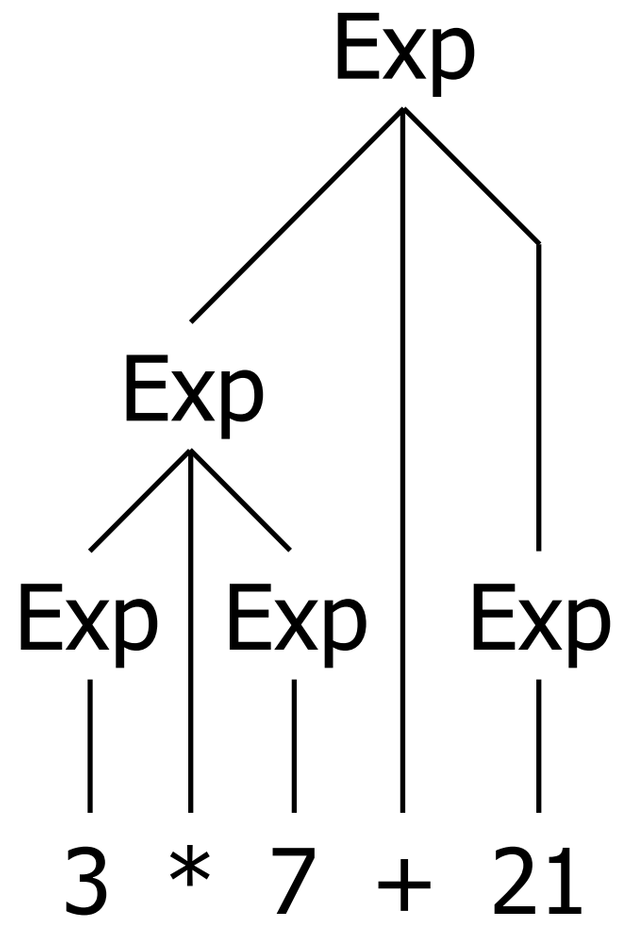


Syntax Definition
language specification

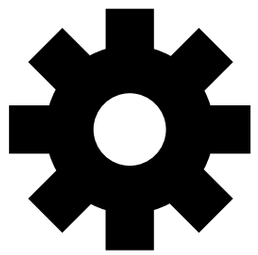
Guido Wachsmuth, Eelco Visser

3 * 7 + 21

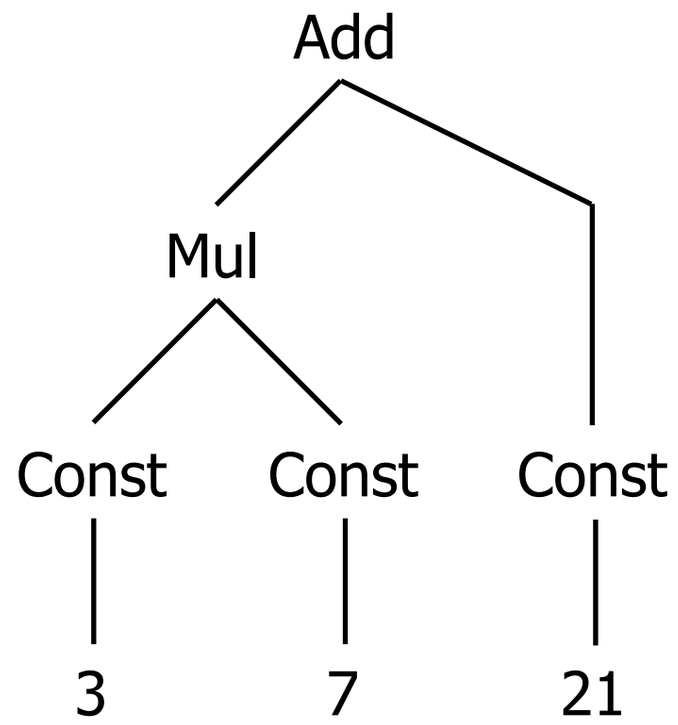




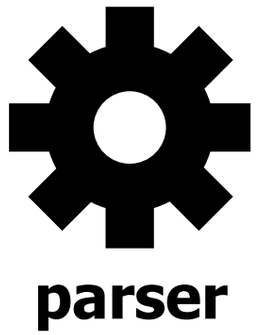
3 * 7 + 21



parser



```
3 * 7 + 21
```




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

SPT

tests

syntax definition

concrete syntax

abstract syntax

SDF3

static semantics

name binding

type system

NaBL

TS

dynamic semantics

translation

interpretation

Stratego

ESV

editor

syntax definition

regular expressions

Backus-Naur Form

Extended BNF

Spoofax

architecture

testing

editors

SDF3

lexical syntax

context-free syntax

abstract syntax

disambiguation

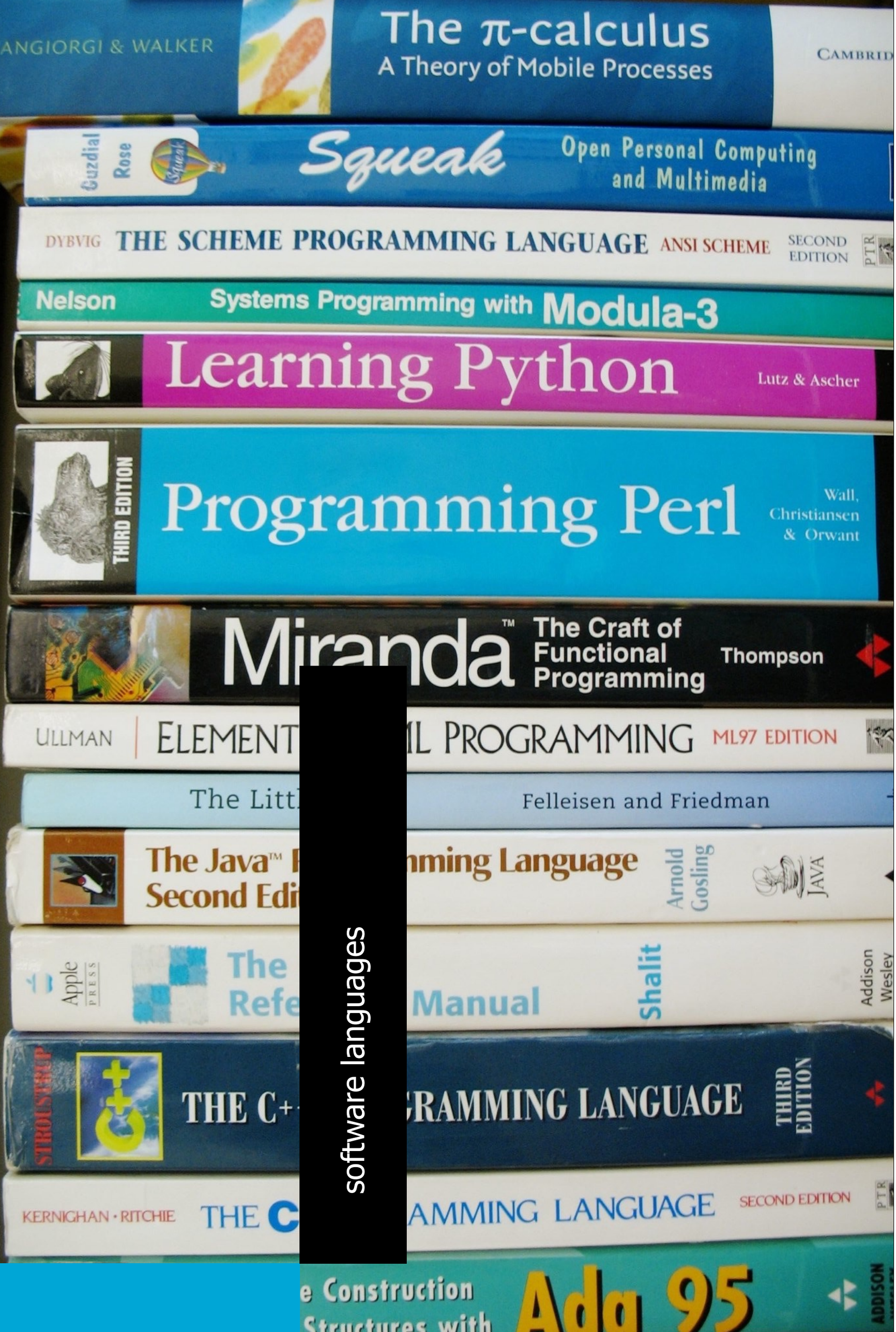
syntax processing

pretty-printing

discovery

more editor services

syntax definition



software languages

LA DEUXIÈME ANNÉE

DE LATIN

finite models

Théorie 250 pages.

227 Exercices 100 pages.

Lexiques

24 pages blanches pour notes et 4 cartes

Philosophy

Linguistics

lexicology

grammar

morphology

syntax

phonology

semantics

Interdisciplinary

Computer Science

syntax

semantics

computer science

syntax

lexical syntax

words made from letters

irrelevant structure

literals `42` `"foo"` `true`

identifiers `x` `foo`

keywords `if` `while`

operators `+` `*`

whitespace `// FIXME`

regular expressions

context-free syntax

sentences made from words

relevant structure

context-free grammars

Backus-Naur Form

Extended Backus-Naur Form

syntax definition

regular expressions

regular expressions

basics

- strings `"nil"`
- character classes `[a-zA-Z]`

combinators

- concatenation `E1 E2`
- option `E?`
- repetition (zero or more) `E*`
- repetition (one or more) `E+`
- alternative `E1 | E2`

Tiger

the lecture language

```
/* factorial function */

let

  var x := 0

  function fact(n : int) : int =
    if n < 1 then 1 else (n * fact(n - 1))

in

  for i := 1 to 3 do (
    x := x + fact(i);
    printint(x);
    print(" ")
  )

end
```

Tiger

lexical syntax

An identifier is a sequence of letters, digits, and under-scores, starting with a letter. Uppercase letters are distinguished from lowercase.

Tiger

lexical syntax

An identifier is a sequence of letters, digits, and under-scores, starting with a letter. Uppercase letters are distinguished from lowercase.

```
[a-zA-Z][a-zA-Z0-9\_]*
```

Tiger

lexical syntax

An identifier is a sequence of letters, digits, and under-scores, starting with a letter. Uppercase letters are distinguished from lowercase.

```
[a-zA-Z][a-zA-Z0-9\_]*
```

A comment may appear between any two tokens. Comments start with `/*` and end with `*/` and may be nested.

Tiger

lexical syntax

An identifier is a sequence of letters, digits, and under-scores, starting with a letter. Uppercase letters are distinguished from lowercase.

```
[a-zA-Z][a-zA-Z0-9\_]*
```

A comment may appear between any two tokens. Comments start with `/*` and end with `*/` and may be nested.

Pumping Lemma

syntax definition

Backus-Naur Form

Backus-Naur Form

basics

- strings “nil”
- symbols <s>

combinators

- concatenation E1 E2
- production <s> ::= E1 | … | En

```
<exp> ::=  <num>
         |  <exp> “+” <exp>
         |  <exp> “-“ <exp>
         |  <exp> “*” <exp>
         |  <exp> “/“ <exp>
         |  “(“ <exp> “)”
         |   <id> “(“ <args> “)”

<args> ::=
         | <argp>

<argp> ::= <exp>
         | <argp> “,” <exp>
```

Tiger

context-free syntax in BNF

```
<exp> ::=  <num>
       |   <exp> “+” <exp>
       |   <exp> “-“ <exp>
       |   <exp> “*” <exp>
       |   <exp> “/“ <exp>
       |   “(“ <exp> “)”
```

syntax definition

Extended Backus-Naur Form

Extended Backus-Naur Form

basics

- strings `"nil"`
- symbols `s`

combinators

- concatenation `E1,E2`
- option `[E]`
- repetition (zero or more) `{E}`
- alternative `E1 | E2`
- production `s = E1 | … | En ;`

```
<exp>  ::=  <num>
         |  <exp> “+” <exp>
         |  <exp> “-“ <exp>
         |  <exp> “*” <exp>
         |  <exp> “/“ <exp>
         |  “(“ <exp> “)”
         |  <id> “(“ [<args>] “)”

<args> ::= <exp> {“,” <exp>}
```

Tiger

context-free syntax

```
exp =   num
    |   exp , "+" , exp
    |   exp , "-" , exp
    |   exp , "*" , exp
    |   exp , "/" , exp
    |   "(" , exp , ")" ;
```

SDF3

Syntax Definition Formalism 3

SDF3

lexical syntax

SDF3

lexical syntax

basics

strings `"nil"`

character classes `[a-zA-Z]`

complements `~[a-zA-Z]`

sorts `S`

combinators

concatenation `E1 E2`

option `E?`

repetition (zero or more) `E*`

with separator `{E s}*`

repetition (one or more) `E+`

with separator `{E s}+`

alternative `E1 | E2`

production `S = E`

Tiger

lexical syntax

```
module Literals

lexical syntax

ID              = [a-zA-Z] [a-zA-Z0-9\_]*

INT             = "-"? [0-9]+

STRING          = "\"" StringChar* "\""
StringChar      = ~[\"\n]
StringChar      = "\\\""
StringChar      = "\\"
```

Tiger

lexical syntax

```
module Whitespace

lexical syntax

LAYOUT      = [\ \t\n\r]

LAYOUT      = "//" ~[\n\r]* NewLineEOF
NewLineEOF  = [\n\r] | EOF
EOF         =

LAYOUT      = "/*" CommentPart* "*/"
CommentPart = ~[\*]
CommentPart = [\*]
```

SDF3

context-free syntax

SDF3

context-free syntax

basics

strings `"nil"`

sorts `S`

combinators

concatenation `E1 E2`

option `E?`

repetition (zero or more) `E*`

with separator `{E s}*`

repetition (one or more) `E+`

with separator `{E s}+`

production `S = E`

```
<exp> ::=  <num>
<exp> ::=  <exp> “+” <exp>
<exp> ::=  <exp> “-“ <exp>
<exp> ::=  <exp> “*” <exp>
<exp> ::=  <exp> “/“ <exp>
<exp> ::=  “(“ <exp> “)”
<exp> ::=  <id> “(“ {<exp> “,”}* “)”
```

Tiger

context-free syntax

```
module Expressions

context-free syntax

  Exp        = LValue
  LValue     = ID
  LValue     = LValue "." ID
  LValue     = LValue "[" Exp "]"

  Exp        = "nil"
  Exp        = INT
  Exp        = STRING
  Exp        = ID "(" {Exp ","}* ")"
  Exp        = TypeId "{" {InitField ","}* "}"
  Exp        = TypeId "[" Exp "]" "of" Exp

  InitField = ID "=" Exp
```

```
x
x.f1
x[i]

nil
42
"foo"
sqr(x)
aty [x] of 42
rty {f1 = "foo", f2 = 42}
```

Tiger

context-free syntax

```
context-free syntax

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

  Exp = Exp "&"  Exp
  Exp = Exp "|"  Exp
```

SDF3

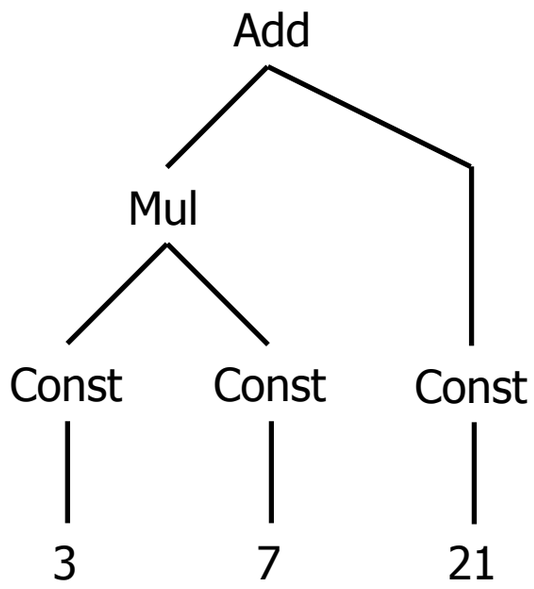
abstract syntax

ATerms

trees - terms

tree

- leaf
- parent node
  children are trees



term

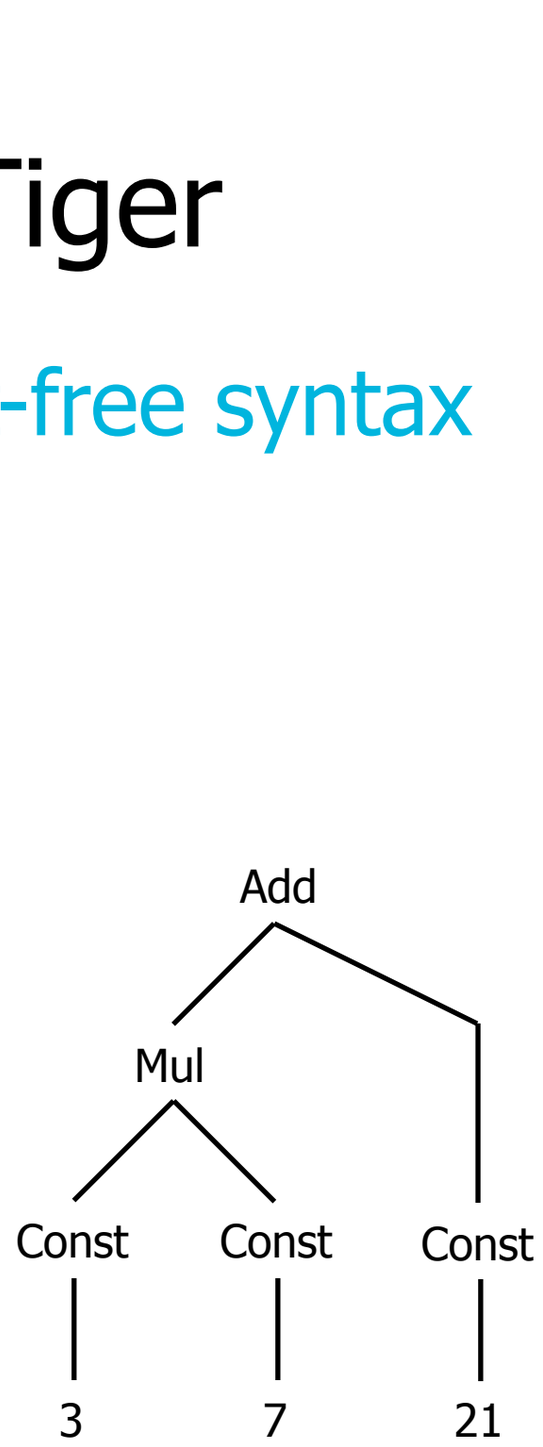
- constant
- constructor
  arguments are terms

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

Tiger

context-free syntax

```
context-free syntax

  Exp.Add = Exp "+" Exp
  Exp.Sub = Exp "-" Exp
  Exp.Mul = Exp "*" Exp
  Exp.Div = Exp "/" Exp

  Exp.Eq  = Exp "="  Exp
  Exp.Neq = Exp "<>" Exp
  Exp.Gt  = Exp ">"  Exp
  Exp.Lt  = Exp "<"  Exp
  Exp.Gte = Exp ">=" Exp
  Exp.Lte = Exp "<=" Exp

  Exp.And = Exp "&"  Exp
  Exp.Or  = Exp "|"  Exp
```




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

Tiger*

abstract syntax : algebraic signature

```
signature
  constructors
    Uminus     : Exp -> Exp
    Power      : Exp * Exp -> Exp
    Times      : Exp * Exp -> Exp
    Divide     : Exp * Exp -> Exp
    Plus       : Exp * Exp -> Exp
    Minus      : Exp * Exp -> Exp
    CPlus      : Exp * Exp -> Exp
    CMinus     : Exp * Exp -> Exp
    Eq         : Exp * Exp -> Exp
    Neq        : Exp * Exp -> Exp
    Gt         : Exp * Exp -> Exp
    Lt         : Exp * Exp -> Exp
    Geq        : Exp * Exp -> Exp
    Leq        : Exp * Exp -> Exp
    True       : Exp
    False      : Exp
    And        : Exp * Exp -> Exp
    Or         : Exp * Exp -> Exp
```

SDF3 disambiguation

Tiger

lexical disambiguation

```
lexical syntax

  ID  = [a-zA-Z] [a-zA-Z0-9\_]*

  ID  = "nil" {reject}

  INT = "-"? [0-9]+

lexical restrictions

  ID  -/- [a-zA-Z0-9\_]
  INT -/- [0-9]
```

Tiger

lexical disambiguation

```
lexical syntax

  LAYOUT      = "//" ~[\n\r]* NewLineEOF
  NewLineEOF  = [\n\r] | EOF
  EOF         =

  LAYOUT      = "/*" CommentPart* "*/"
  CommentPart = ~[\*]
  CommentPart = "*"

lexical restrictions

  EOF -/- ~[]      "*" -/- [\/]
```

Tiger

lexical disambiguation

```
lexical syntax

  LAYOUT      = "//" ~[\n\r]* NewLineEOF
  NewLineEOF  = [\n\r] | EOF
  EOF         =

  LAYOUT          = "/*" CommentPart* "*/"
  CommentPart     = ~[\*]
  CommentPart     = CommentEndChar
  CommentEndChar = "*"

lexical restrictions
  EOF -/- ~[]       CommentEndChar -/- [\/]
```

Tiger

context-free disambiguation

```
lexical syntax

  LAYOUT = [\ \t\n\r]

  LAYOUT = "//" ~[\n\r]* NewLineEOF

  LAYOUT = "/*" CommentPart* "*/"

context-free restrictions

  LAYOUT? -/- [\ \t\n\r]
  LAYOUT? -/- [\/].[\/]
  LAYOUT? -/- [\/].[\*]
```

Tiger

context-free syntax

```
context-free syntax

  Exp.Add = Exp "+" Exp
  Exp.Sub = Exp "-" Exp
  Exp.Mul = Exp "*" Exp
  Exp.Div = Exp "/" Exp

  Exp.Eq  = Exp "="  Exp
  Exp.Neq = Exp "<>" Exp
  Exp.Gt  = Exp ">"  Exp
  Exp.Lt  = Exp "<"  Exp
  Exp.Gte = Exp ">=" Exp
  Exp.Lte = Exp "<=" Exp

  Exp.And = Exp "&"  Exp
  Exp.Or  = Exp "|"  Exp
```

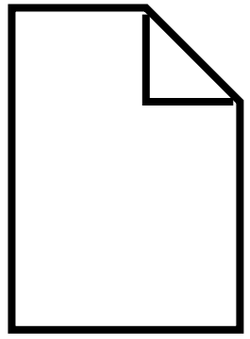
Tiger

context-free disambiguation

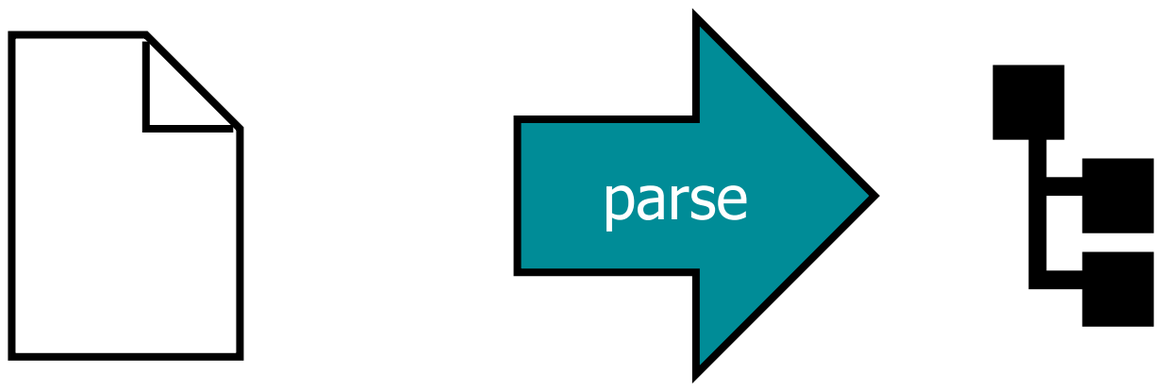
```
context-free syntax

  Exp.Add = Exp "+" Exp {left}
  Exp.Sub = Exp "-" Exp {left}
  Exp.Mul = Exp "*" Exp {left}
  Exp.Div = Exp "/" Exp {left}

  Exp.Eq  = Exp "="  Exp {non-assoc}
  Exp.Neq = Exp "<>" Exp {non-assoc}
  Exp.Gt  = Exp ">"  Exp {non-assoc}
  Exp.Lt  = Exp "<"  Exp {non-assoc}
  Exp.Gte = Exp ">=" Exp {non-assoc}
  Exp.Lte = Exp "<=" Exp {non-assoc}

  Exp.And = Exp "&"  Exp {left}
  Exp.Or  = Exp "|"  Exp {left}
```

```
context-free priorities
  { left:
      Exp.Mul
      Exp.Div
   } > { left:
      Exp.Add
      Exp.Sub
  } > { non-assoc:
      Exp.Eq
      Exp.Neq
      Exp.Gt
      Exp.Lt
      Exp.Gte
      Exp.Lte
  } > Exp.And
    > Exp.Or
```

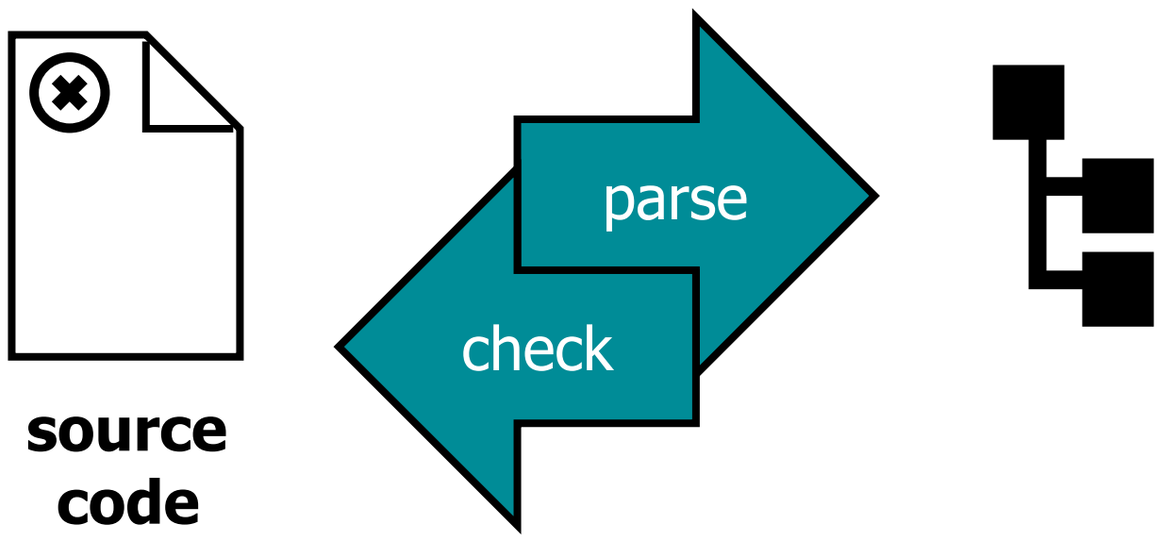
Spoofax architecture

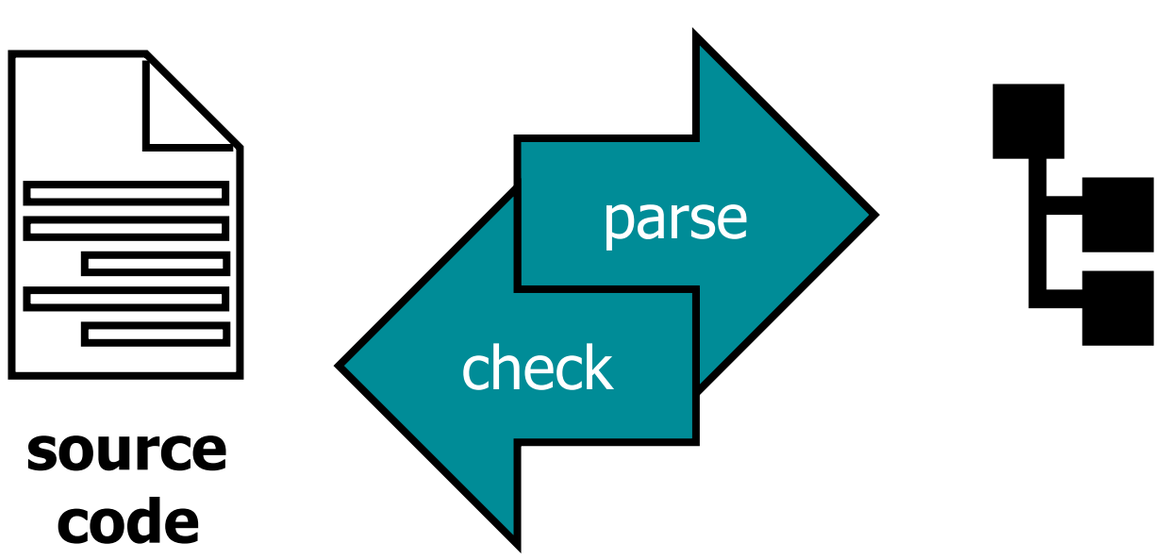


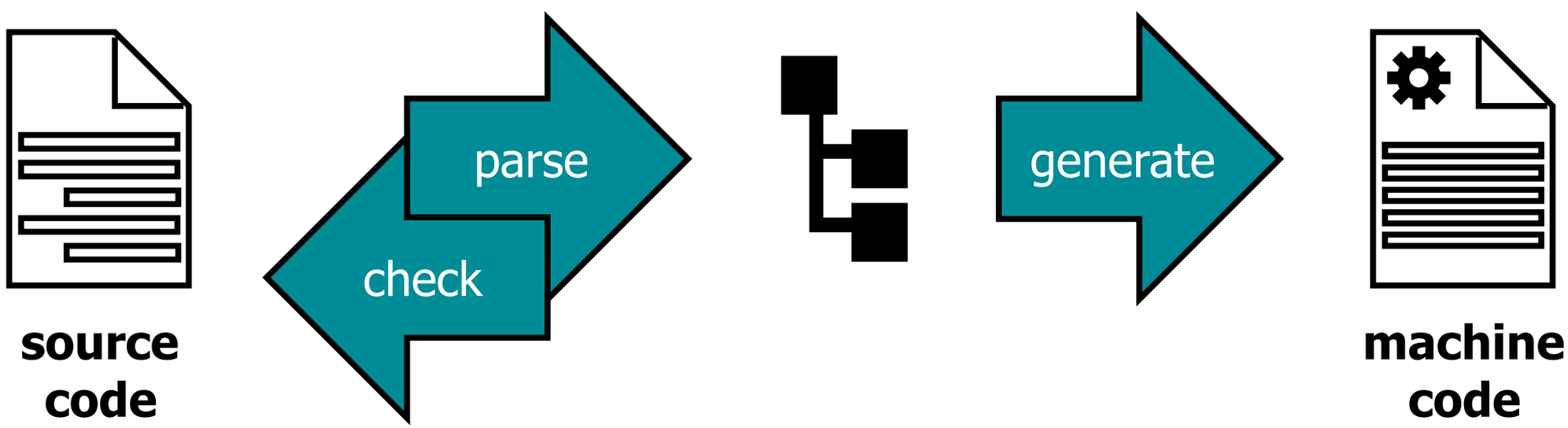
source code



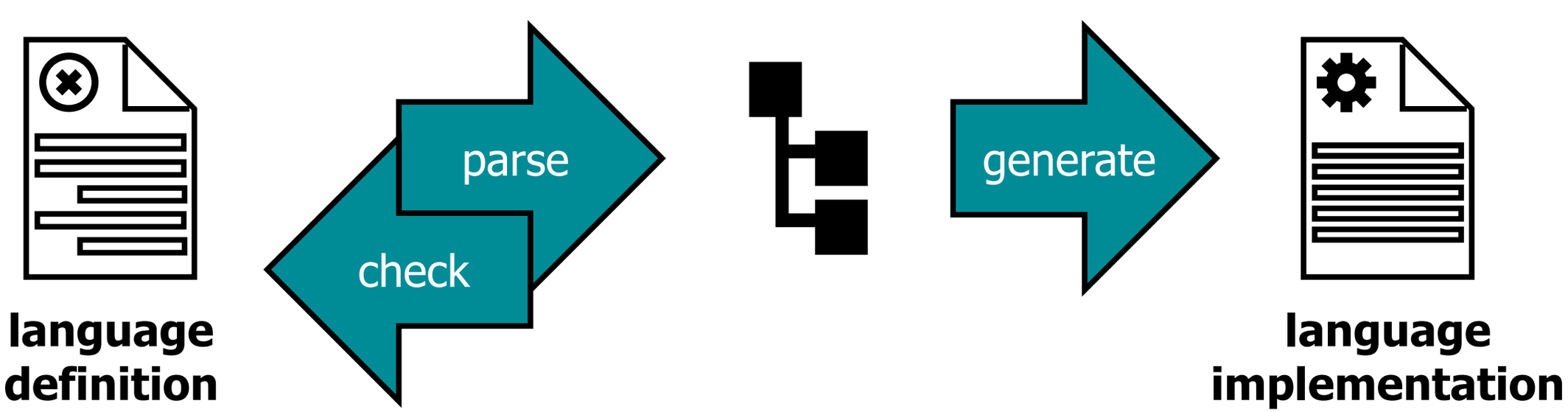
source code

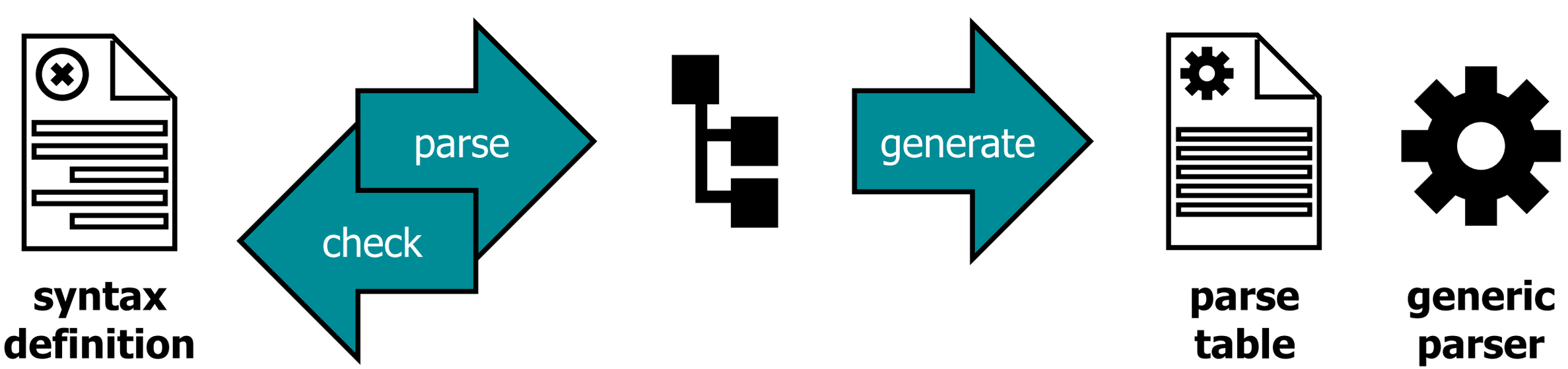

parse
check
source
code


parse
check
source
code


parse
check
generate
source code
machine code

Language	Eclipse	Java
Programs	Fact.java	Java
Data	42	int


parse
check
generate
language definition
language implementation


parse
check
generate
syntax definition
parse table
generic parser

Language	**Spoofax**	**SDF3**
Language	**Java**	**SDF3**
Programs	**Fact.java**	**Java**
Data	**42**	**int**

context-free syntax

SDF3 in SDF3

```
context-free syntax

  Grammar.Lexical = <
   lexical syntax

       <Production*>
   >
  Grammar.Contextfree = <
   context-free syntax

       <Production*>
   >

  Production.SdfProduction          = <<Symbol> = <Symbol*> <Attributes>>
  Production.SdfProductionWithCons = <<SortCons> = <Symbol*> <Attributes>>

  SortCons.SortCons = <<Symbol>.<Constructor>>
```

SPT

tests

syntax definition

concrete syntax

abstract syntax

SDF3

static semantics

name binding

type system

NaBL

TS

dynamic semantics

translation

interpretation

Stratego

ESV

editor

SPT

tests

syntax definition

concrete syntax

abstract syntax

SDF3

static semantics

name binding

type system

NaBL

TS

dynamic semantics

translation

interpretation

Stratego

ESV

editor

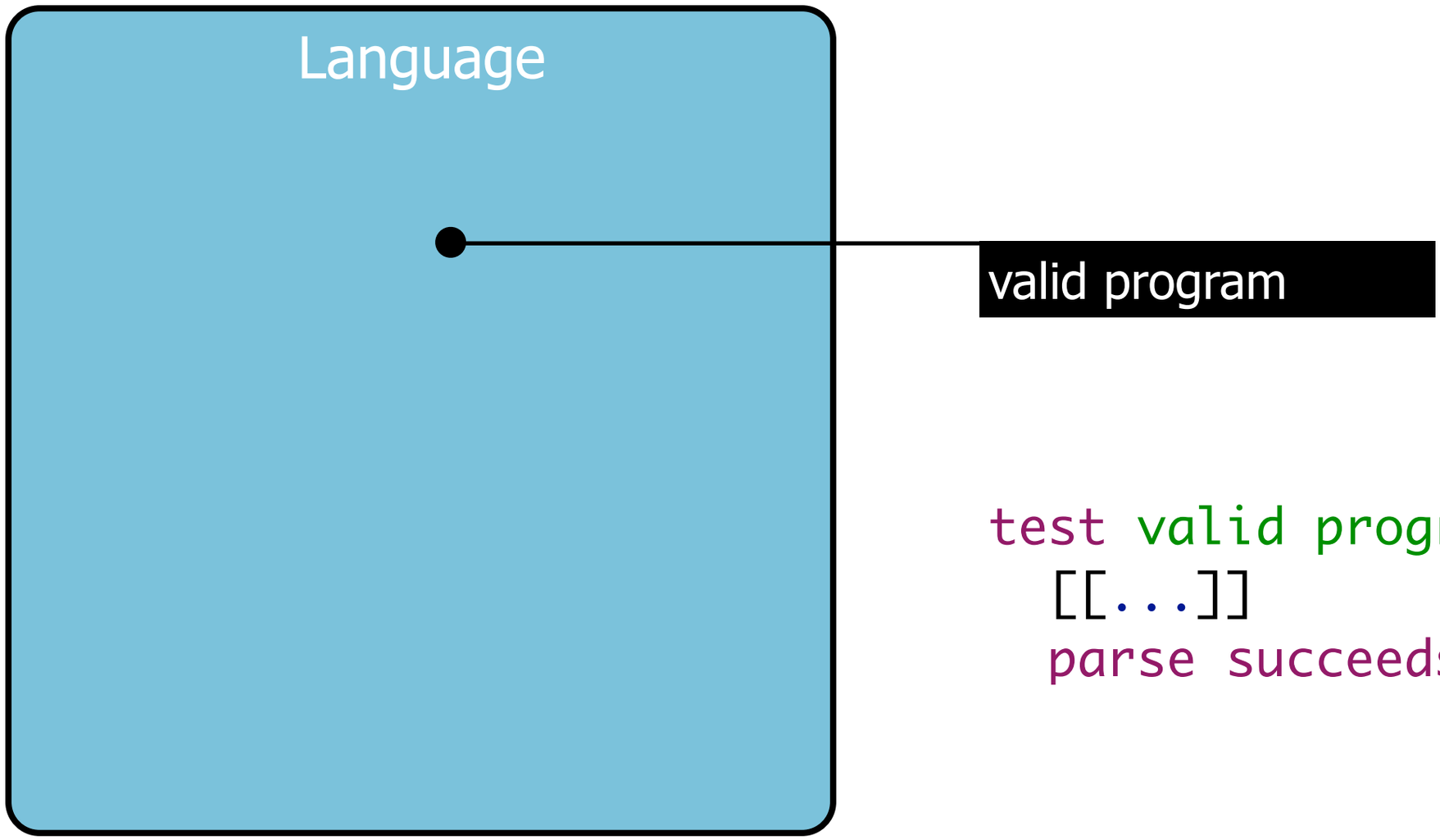
Spoofax testing

test suites

```
module example-suite

language Tiger
start symbol Start

test name
  [[...]]
  parse succeeds

test another name
  [[...]]
  parse fails
```

test cases

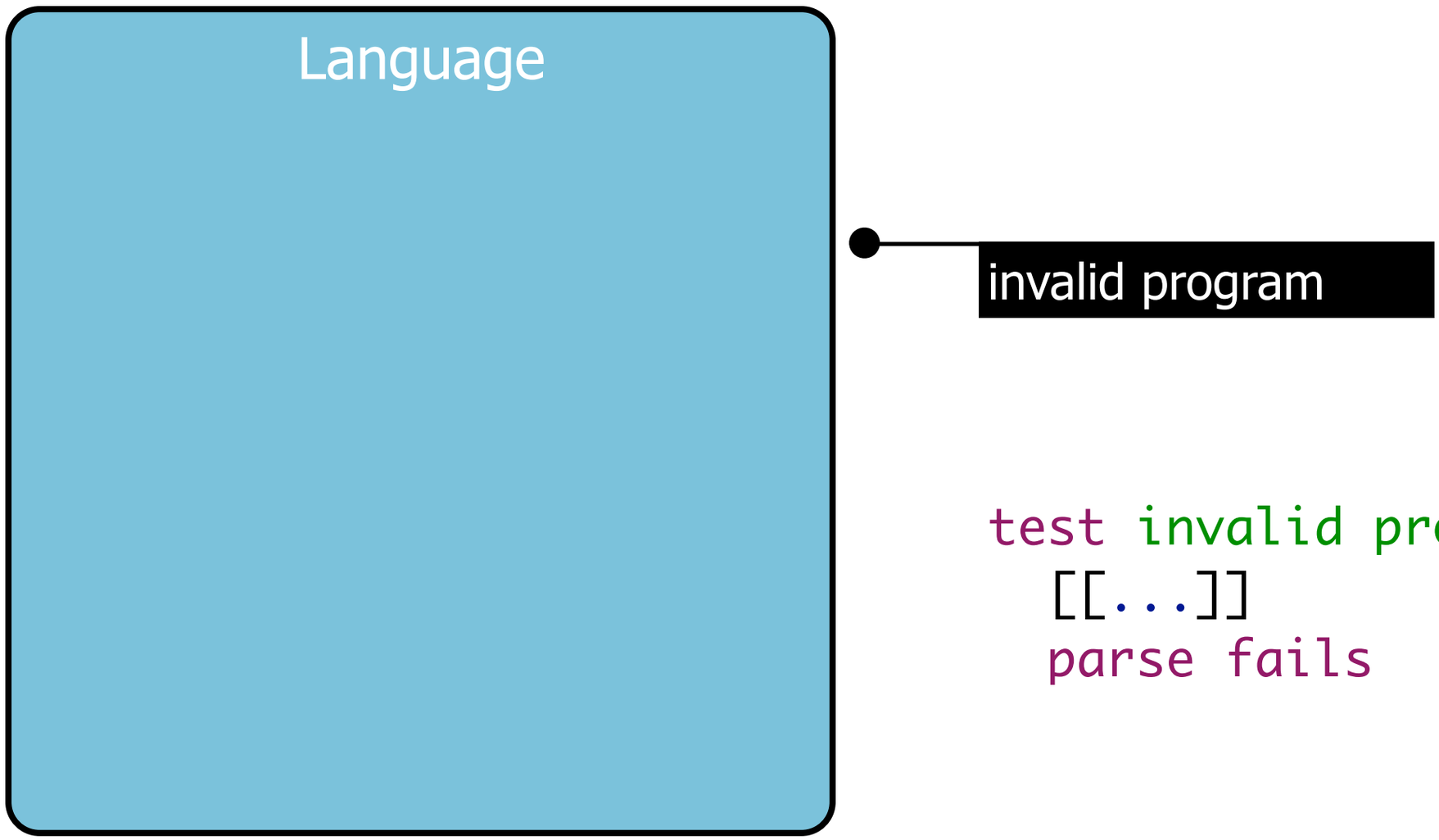
syntax




```
test valid program
  [[...]]
  parse succeeds
```

test cases

syntax




```
test invalid program
  [[...]]
  parse fails
```

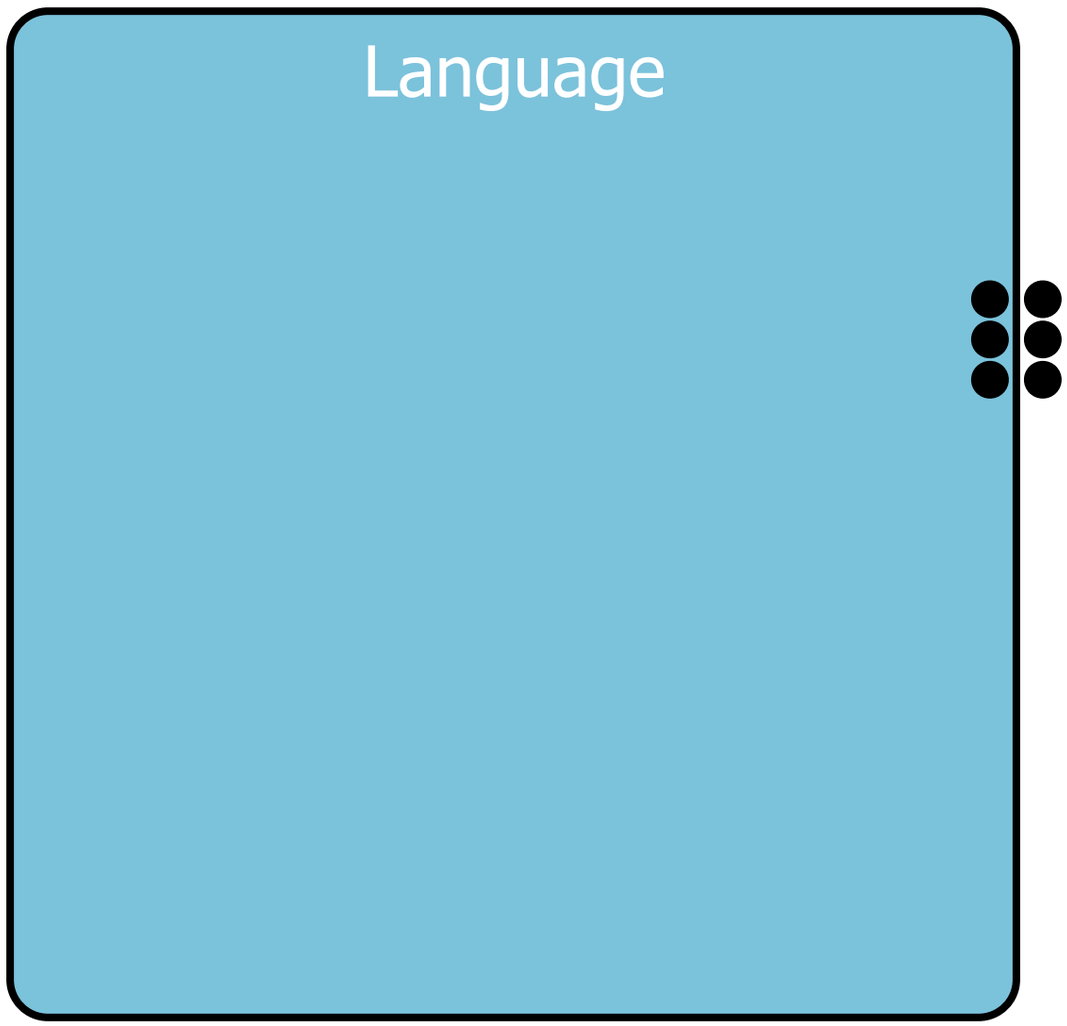
test cases

syntax

```
module syntax/identifiers

language Tiger start symbol Id

test single lower case [[x]] parse succeeds
test single upper case [[X]] parse succeeds
test single digit      [[1]] parse fails

test single lc digit   [[x1]] parse succeeds
test single digit lc   [[1x]] parse fails
test single uc digit   [[X1]] parse succeeds
test single digit uc   [[1X]] parse fails
test double digit      [[11]] parse fails
```

test cases

syntax



test cases

ambiguities

```
module precedence

language Tiger start symbol Exp

test parentheses
  [[(42)]] parse to [[42]]

test left-associative addition
  [[21 + 14 + 7]] parse to [[(21 + 14) + 7]]

test precedence multiplication
  [[3 * 7 + 21]] parse to [[(3 * 7) + 21]]
```

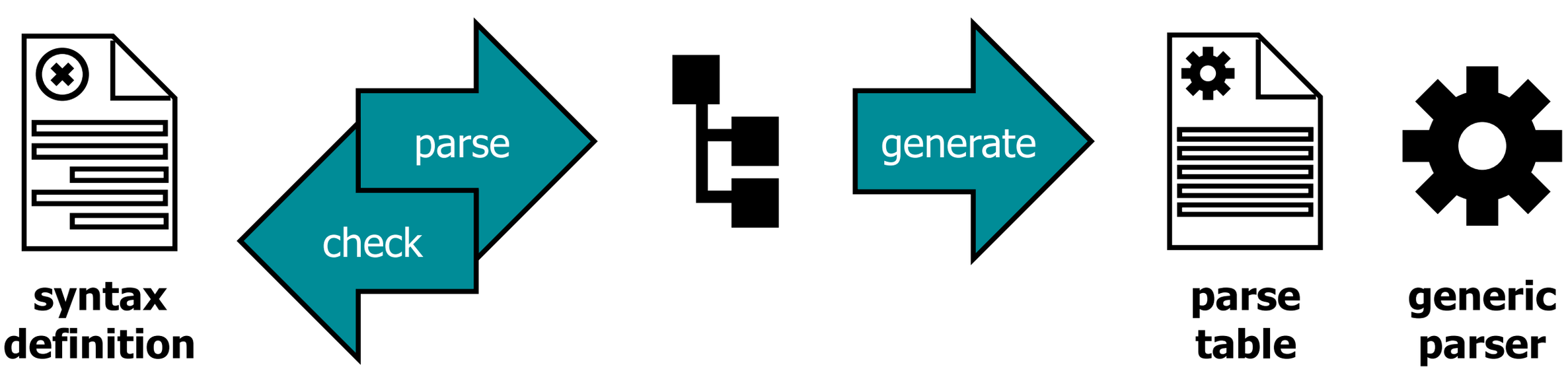
Note : this does not actually work in Tiger, since () is a sequencing construct; but works fine in Mini-Java

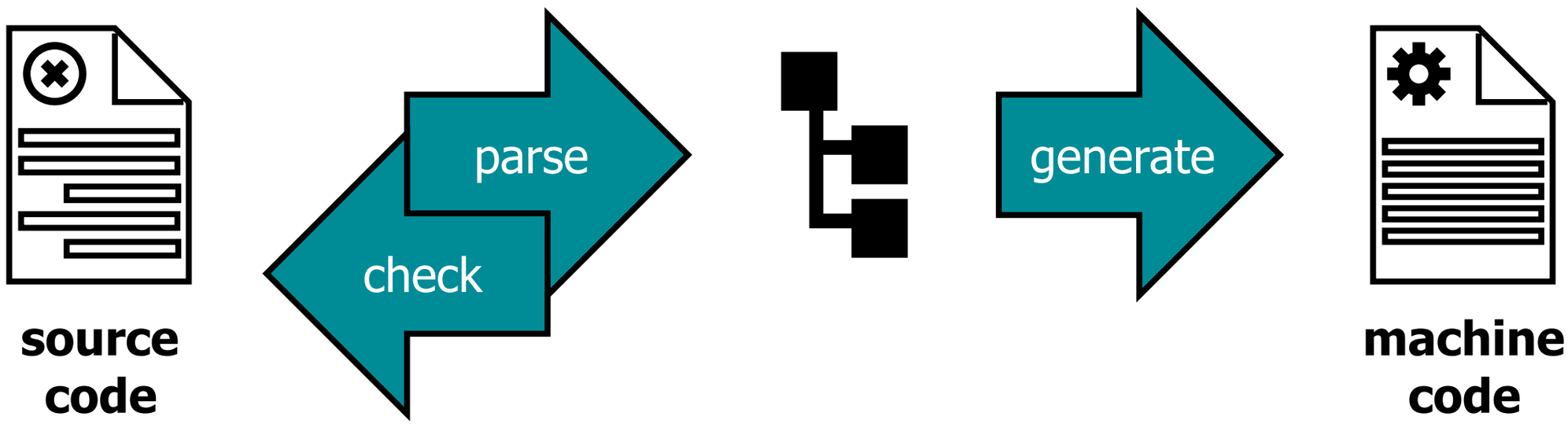
test cases

ambiguities

```
module precedence

language Tiger start symbol Exp

test plus/times priority [[
  x + 4 * 5
]] parse to Plus(Var("x"), Times(Int("4"), Int("5")))

test plus/times sequence [[
  (x + 4) * 5
]] parse to Times(
  Seq([Plus(Var("x"), Int("4"))])
, Int("5")
)
```

Spoofax editors


parse
check
generate
syntax definition
parse table
generic parser


parse
check
generate
source code
machine code

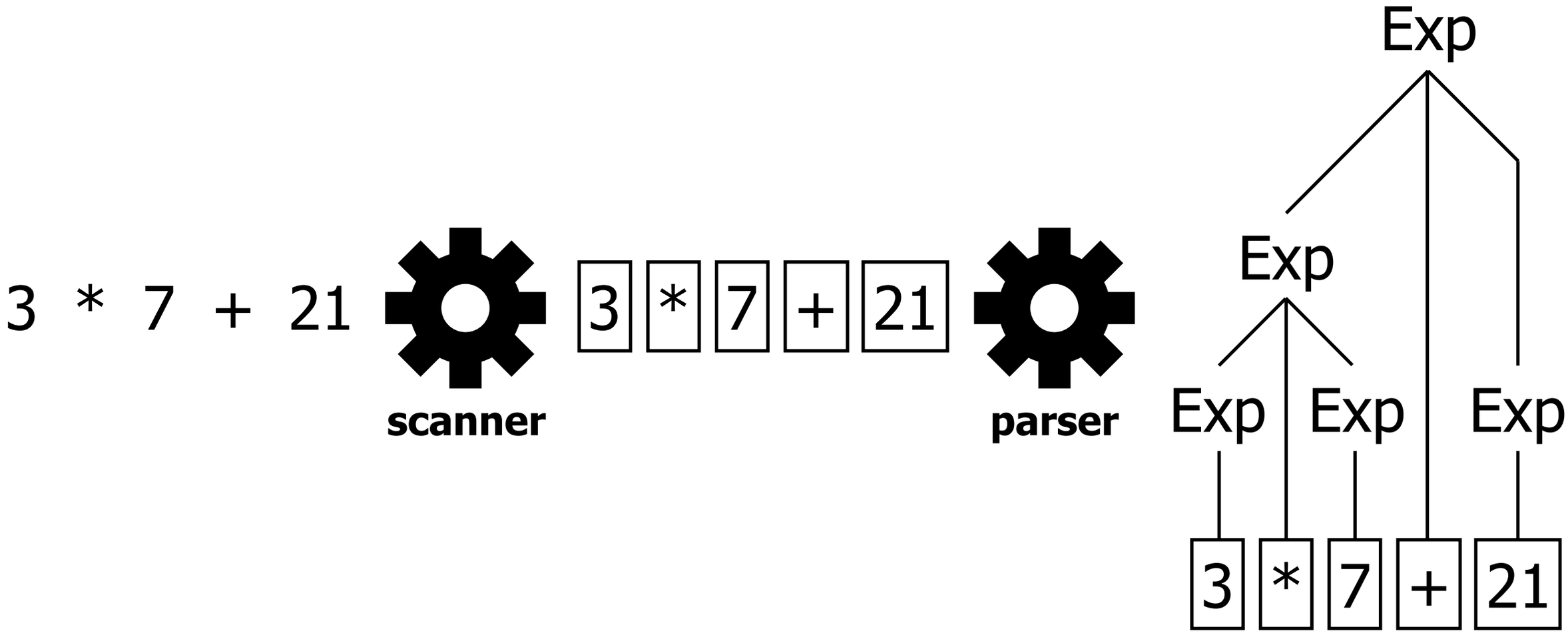
editor specification

```
module Tiger.main imports ...

language General properties

  name:           Tiger
  id:             org.metaborg.cube.tiger
  extends:        Root
  description:    "Spoofax-generated editor for the Tiger language"
  url:            http://metaborg.org

  extensions:     tig
  table:          include/Tiger.tbl
  start symbols:  Start

  provider:       include/Tiger.ctree
  observer:       editor-analyze (multifile)
  on save:        editor-save
```

syntax processing

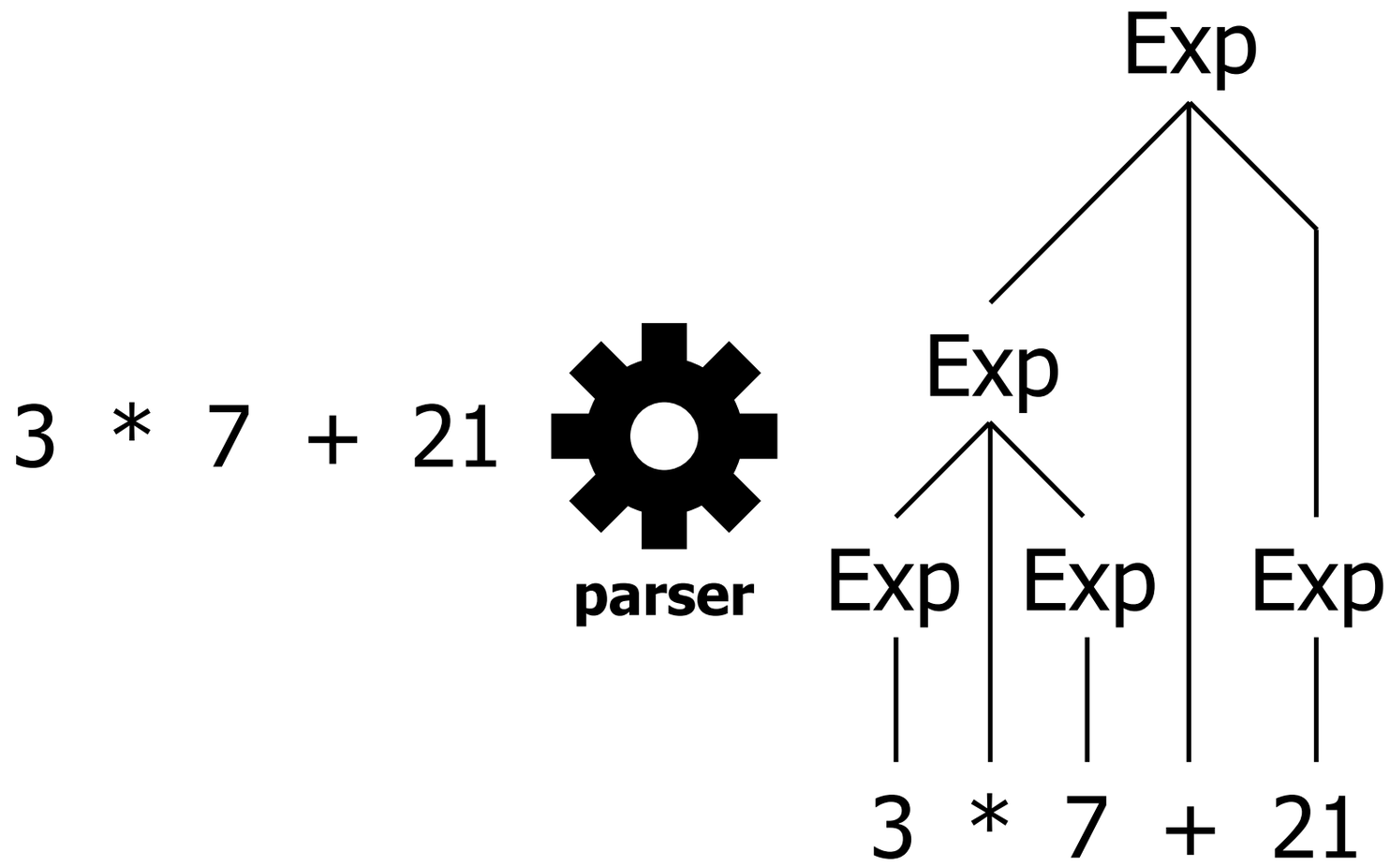
language processors

scanners & parsers



language processors

scannerless parsers

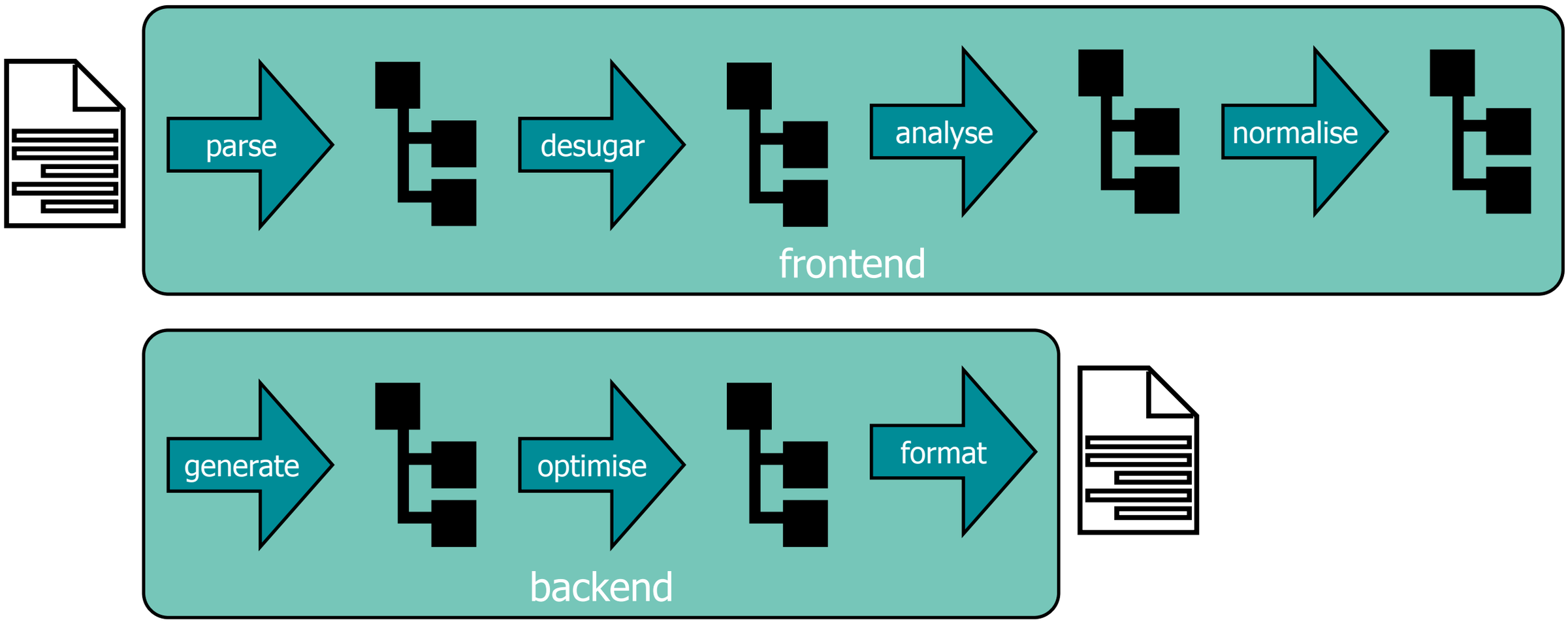


syntax processing

pretty-printing

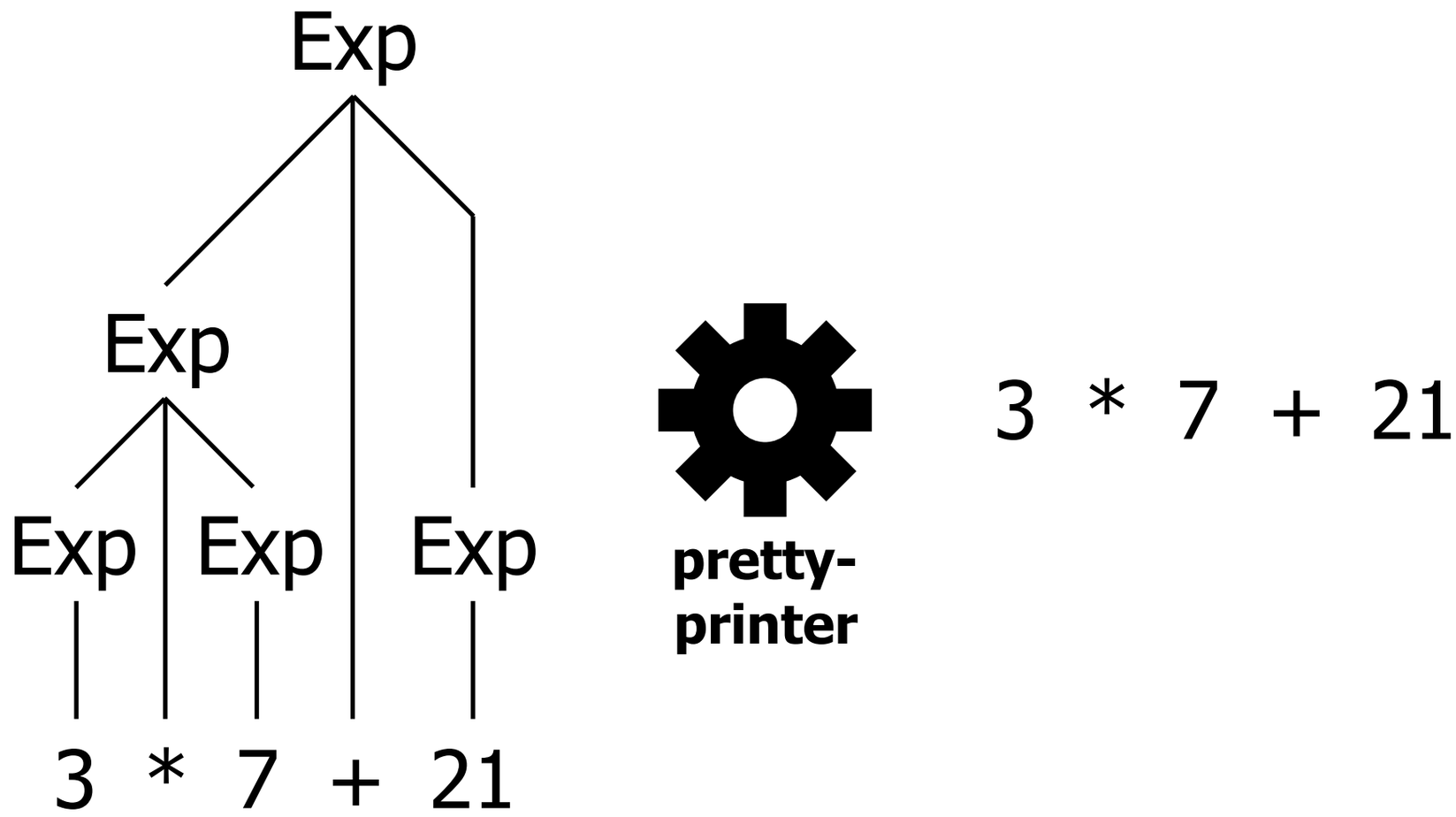
language processors

compilation by transformation



language processors

pretty-printers



language processors

pretty-printing

from ASTs to text

- keywords
- layout: spaces, line breaks, indentation

specification

- partially defined in grammar
- missing layout

SDF3

production rules

```
context-free syntax

  Exp.Nil           = "nil"
  Exp.IntC          = INT
  Exp.StringC       = STRING

  Exp.Call          = ID "(" {Exp ","}* ")"

  Exp.ArrayI        = TypeId "[" Exp "]" "of" Exp

  Exp.RecordI       = TypeId "{" {InitField ","}* "}"

  InitField.FieldI = ID "=" Exp
```

SDF3

templates

```
context-free syntax

  Exp.Nil           = <nil>
  Exp.IntC          = INT
  Exp.StringC       = STRING

  Exp.Call          = <<ID> ( <{Exp ","}*> )>

  Exp.ArrayI        = <<TypeId> [ <Exp> ] of <Exp>>

  Exp.RecordI       = <<TypeId> { <{InitField ","}*> }>

  InitField.FieldI = <<ID> = <Exp>>
```

SDF3

formatted templates

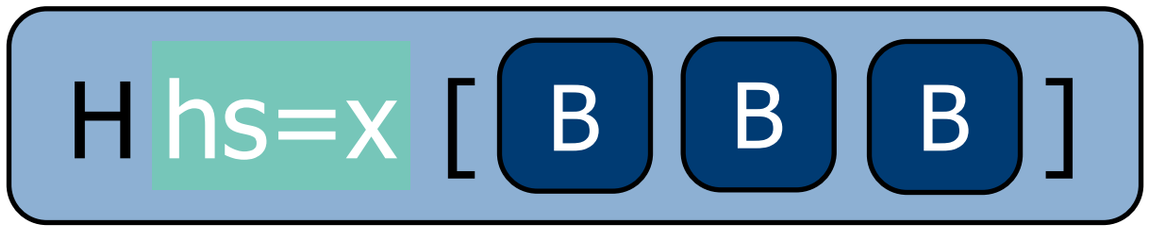
```
context-free syntax

  Exp.Nil           = <nil>
  Exp.IntC          = INT
  Exp.StringC       = STRING

Exp.Call            = <<ID>(<{Exp ", "}*>)>

Exp.ArrayI          = <<TypeId>[<Exp>] of <Exp>>

Exp.RecordI         = <
  <TypeId> {
    <{InitField ",\n"}*>
  }>

InitField.FieldI = <<ID> = <Exp>>
```

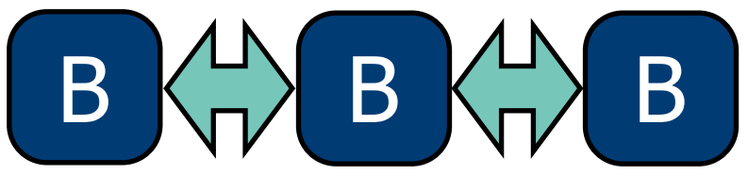
box layout

basic boxes

box layout

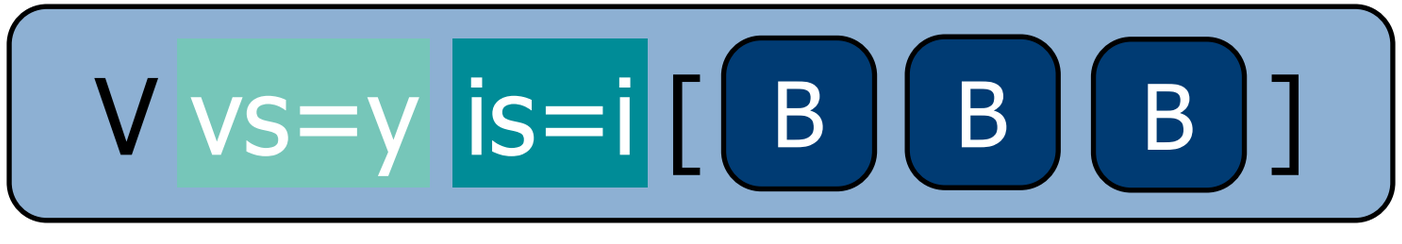
horizontal boxes

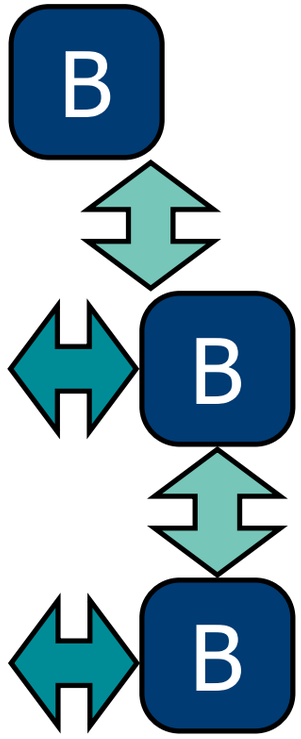




box layout

vertical boxes



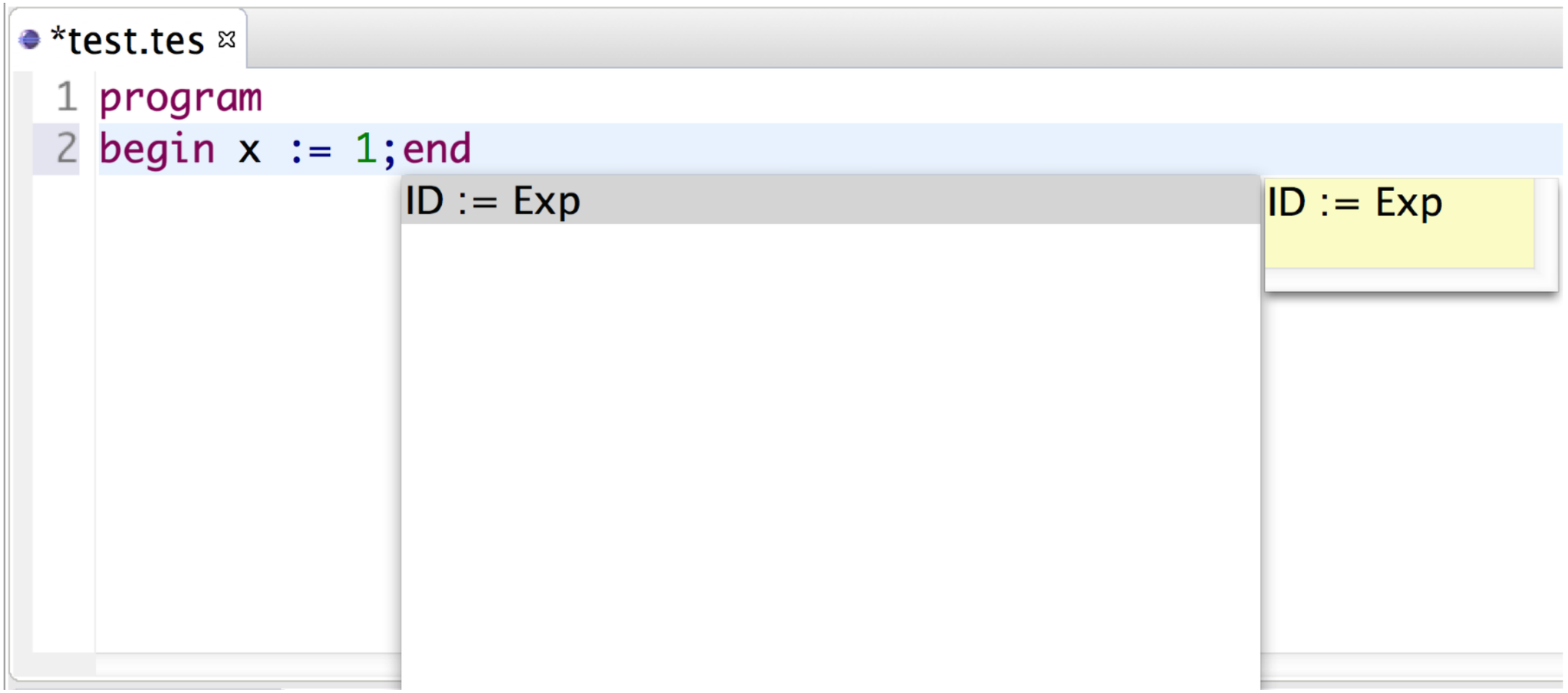


syntax processing

discovery

discovery

syntactic code completion



SDF3

formatted templates

```
context-free syntax

  Exp.Nil           = <nil>
  Exp.IntC          = INT
  Exp.StringC       = STRING

  Exp.Call          = <<ID>(<{Exp ", "}*>)>

Exp.ArrayI          = <<TypeId>[<Exp>] of <Exp>>

Exp.RecordI         = <
  <TypeId> {
    <{InitField ",\n"}*>
  }>

InitField.FieldI = <<ID> = <Exp>>
```

SDF3

generated completion templates

```
module src-gen/completions/Expressions-esv

completions
  completion template Exp : "nil" =
    "nil"

completion template Exp : "ID()" =
  <ID:ID> "(" <:Exp> ")"

completion template Exp : "TypeId[Exp] of Exp" =
  <TypeId:TypeId> "[" <Exp:Exp> "] of " <Exp:Exp>

completion template Exp : "TypeId { }" =
  <TypeId:TypeId> " {\n\t" (cursor) "\n}" (blank)

completion template InitField : "ID = Exp" =
  <ID:ID> " = " <Exp:Exp>
```

SDF3

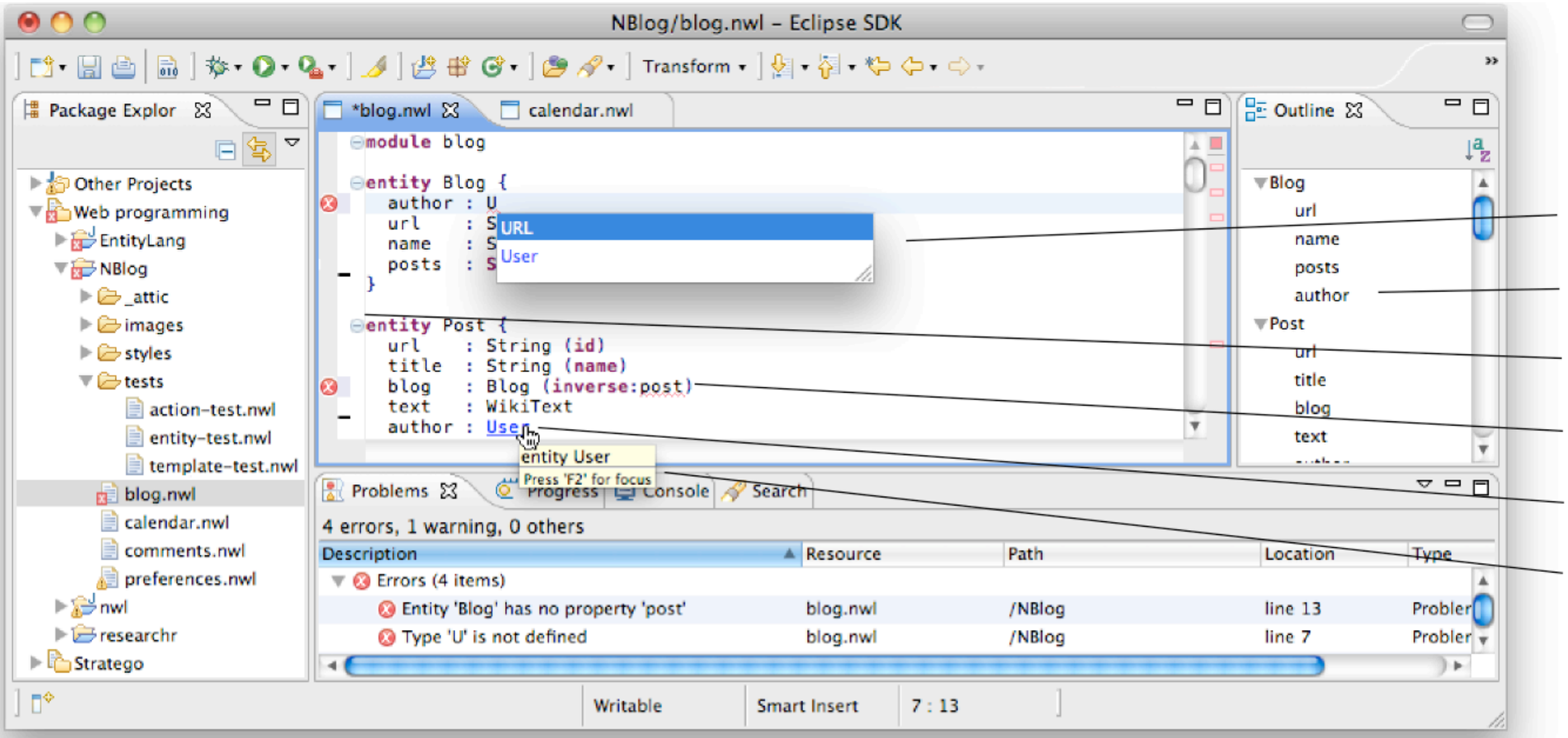
improved templates

```
context-free syntax

  Exp.Nil           = <nil>
  Exp.IntC          = INT
  Exp.StringC       = STRING

  Exp.Call          = <<ID; text="m">(<{Exp ", "}*>)>

Exp.ArrayI          = <<TypeId; text="type">[<Exp; text="size">] of <Exp; text="value">>

Exp.RecordI         = <
  <TypeId; text="type"> {
    <{InitField ",\n"}*>
  }>

InitField.FieldI = <<ID; text="var"> = <Exp; text="value">>
```

syntax processing

more editor services

editor services



Spoofax

generated highlighting rules

```
module libspoofax/color/default

imports

  libspoofax/color/colors

colorer // Default, token-based highlighting

  keyword    : 127 0 85 bold
  identifier : default
  string     : blue
  number     : darkgreen
  var        : 139 69 19 italic
  operator   : 0 0 128
  layout     : 63 127 95 italic
```

Spoofax

customised highlighting rules

```
module Tiger-Colorer

imports Tiger-Colorer.generated

colorer TU Delft colours

TUDlavender = 123 160 201

colorer token-based highlighting

   layout    : TUDlavender
   StrConst  : darkgreen
   TypeId    : blue
```

attribution

slide	title	author	license
1	The Pine, Saint Tropez	Paul Signac	public domain
2-4, 43, 45, 46, 59, 60, 63, 64, 66, 67	PICOL icons	Melih Bilgil	CC BY 3.0
8	Programming language textbooks	K.lee	public domain
9	Latin Grammar	Anthony Nelzin	
14, 15, 18, 21	Tiger	Bernard Landgraf	CC BY-SA 3.0