

Declarative Syntax Definition Grammars and Trees

Guido Wachsmuth

Assessment

last lecture

Assessment

last lecture

Identify three examples of polymorphism in the following Java expression: "1" + ((2 + 4) + 3.5)

Assessment

last lecture

Identify three examples of polymorphism in the following Java expression: "1" + ((2 + 4) + 3.5)

- 1st and 2nd `+` symbolise different operations on different types
- 2nd and 3rd `+` symbolise same operation on different types
- `(2 + 4)` is of type `int`, but 3rd `+` requires a `float`
- `((2 + 4) + 3.5)` is of type `float`, but 1st `+` requires a `String`

Assessment

last lecture

Identify three examples of polymorphism in the following Java expression: "1" + ((2 + 4) + 3.5)

- 1st and 2nd `+` symbolise different operations on different types
- 2nd and 3rd `+` symbolise same operation on different types
- `(2 + 4)` is of type `int`, but 3rd `+` requires a `float`
- `((2 + 4) + 3.5)` is of type `float`, but 1st `+` requires a `String`

What kinds of polymorphism do they represent?

Assessment

last lecture

Identify three examples of polymorphism in the following Java expression: "1" + ((2 + 4) + 3.5)

- 1st and 2nd `+` symbolise different operations on different types
- 2nd and 3rd `+` symbolise same operation on different types
- `(2 + 4)` is of type `int`, but 3rd `+` requires a `float`
- `((2 + 4) + 3.5)` is of type `float`, but 1st `+` requires a `String`

What kinds of polymorphism do they represent?

- ad-hoc polymorphism, overloading
- ad-hoc polymorphism, type coercion, implicit conversion

Assessment

last lecture

Identify three examples of polymorphism in the following Java expression: "1" + ((2 + 4) + 3.5)

- 1st and 2nd `+` symbolise different operations on different types
- 2nd and 3rd `+` symbolise same operation on different types
- `(2 + 4)` is of type `int`, but 3rd `+` requires a `float`
- `((2 + 4) + 3.5)` is of type `float`, but 1st `+` requires a `String`

What kinds of polymorphism do they represent?

- ad-hoc polymorphism, overloading
- ad-hoc polymorphism, type coercion, implicit conversion

What are the differences?

Assessment

last lecture

```
public class A {}

public class B extends A {}

public class C {
    public int m() { return 1 + 2; }
}

public class D {
    public String m() { return "1" + "2"; }
    public A m(A a1, A a2) { return a1; }
}

public class E extends D {
    public A m(B b1, B b2) { return b2; }
}
```

Type Systems

overriding

methods

- parameter types
- return type

Type Systems

overriding

methods

- parameter types
- return type

covariance

- method in subclass
- return type: subtype of original return type

Type Systems

overriding

methods

- parameter types
- return type

covariance

- method in subclass
- return type: subtype of original return type

contravariance

- method in subclass
- parameter types: supertypes of original parameter types

Example: Java overloading vs. overriding

```
public class F {  
    public A m(B b) { System.out.println("F.m(B b)"); return b; }  
}  
  
public class G extends F {  
    public A m(A a) { System.out.println("G.m(A a)"); return a; }  
}  
  
public class H extends F {  
    public B m(B b) { System.out.println("H.m(B b)"); return b; }  
}  
  
A a = new A(); B b = new B(); F f = new F(); G g = new G(); F h = new H();  
A b1 = b;  
  
f.m(b);  
g.m(a); g.m(b); g.m(b1);  
h.m(b);
```

Example: Java invariance

```
public class X {  
    public A a;  
    public A getA() { return a; }  
    public void setA(A a) { this.a = a; }  
}  
  
public class Y extends X {  
    public B a;  
    public B getA() { return a; }  
    public void setA(B a) { this.a = a; }  
}  
  
A a = new A(); B b = new B(); X y = new Y();  
  
y.getA(); y.setA(b); y.setA(a);  
  
String[] s = new String[3]; Object[] o = s; o[1] = new A();
```

Overview

today's lecture

P A R E N T A L

ADVISORY

THEORETICAL CONTENT

Overview

today's lecture

Overview today's lecture

theory

- formal languages
- formal grammars
- decidability & complexity
- syntax trees

Overview today's lecture

theory

- formal languages
- formal grammars
- decidability & complexity
- syntax trees

practice

- syntax definition for MiniJava

Overview today's lecture

theory

- formal languages
- formal grammars
- decidability & complexity
- syntax trees

practice

- syntax definition for MiniJava

first assignment

I

generative grammar



infinite productivity

EE A TOU
K-BRENTAN

LA DEUXIÈME ANNÉE

DÉ LATIN

finite models

Théorie 250 pages.

227 Exercices 100 pages.

Lexiques

24 pages blanches pour notes et 4 cartes

Study of Language

linguistics and friends

philosophy

linguistics

- lexicology
- grammar
- morphology
- syntax
- phonology
- semantics

interdisciplinary

- applied linguistics
- computational linguistics
- historical linguistics
- neurolinguistics
- psycholinguistics
- sociolinguistics

computer science

- grammar
- semantics

A Theory of Language

formal languages



A Theory of Language

formal languages

vocabulary Σ

finite, nonempty set of elements (words, letters)
alphabet



A Theory of Language

formal languages

vocabulary Σ

finite, nonempty set of elements (words, letters)

alphabet

string over Σ

finite sequence of elements chosen from Σ

word, sentence, utterance



A Theory of Language

formal languages

vocabulary Σ

finite, nonempty set of elements (words, letters)
alphabet

string over Σ

finite sequence of elements chosen from Σ
word, sentence, utterance

formal language λ

set of strings over a vocabulary Σ
 $\lambda \subseteq \Sigma^*$



A Theory of Language

formal grammars



A Theory of Language

formal grammars

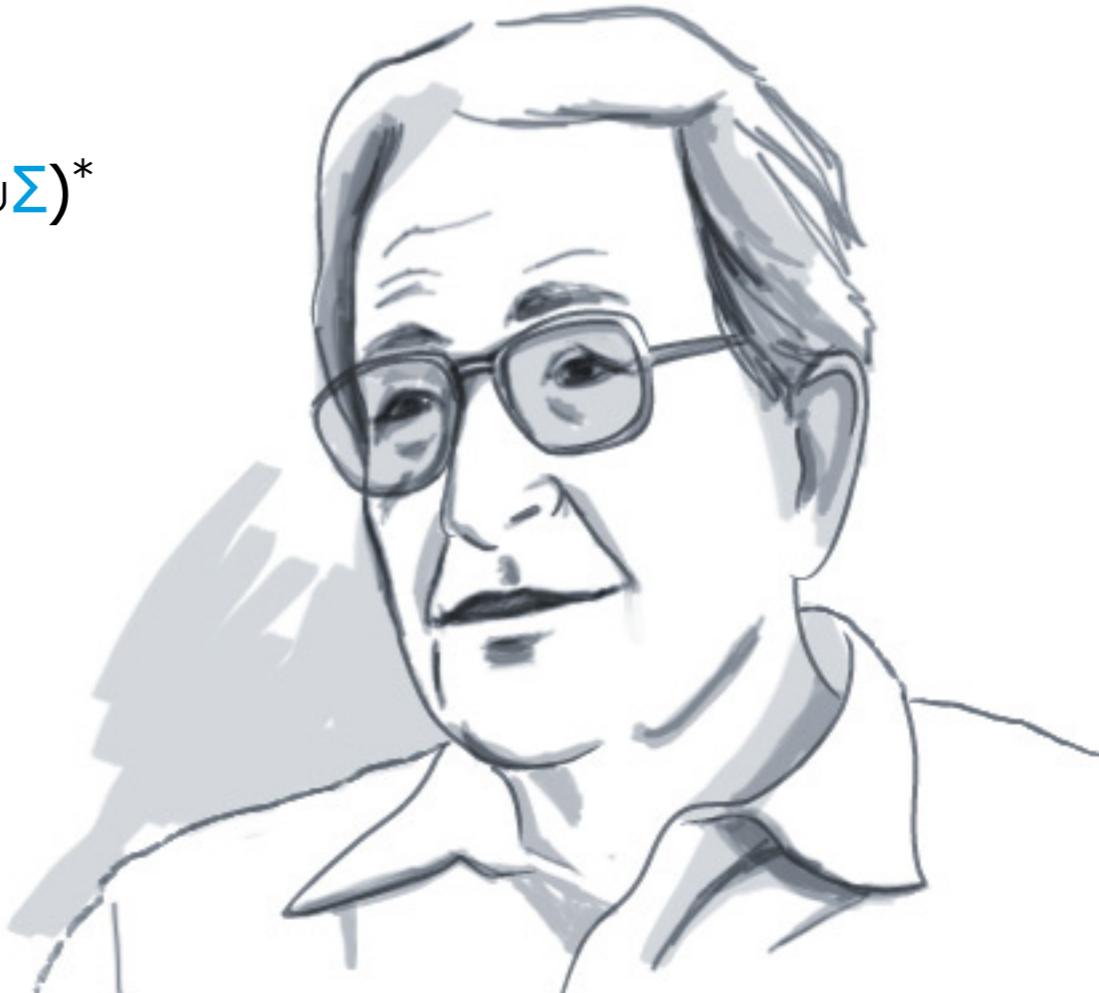
formal grammar $G = (N, \Sigma, P, S)$

nonterminal symbols N

terminal symbols Σ

production rules $P \subseteq (N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$

start symbol $S \in N$



A Theory of Language

formal grammars

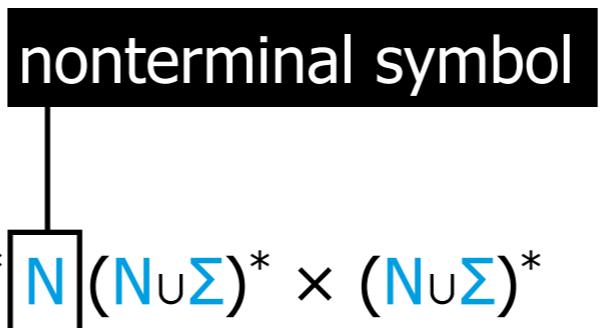
formal grammar $G = (N, \Sigma, P, S)$

nonterminal symbols N

terminal symbols Σ

production rules $P \subseteq (N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$

start symbol $S \in N$



A Theory of Language

formal grammars

formal grammar $G = (N, \Sigma, P, S)$

nonterminal symbols N

terminal symbols Σ

production rules $P \subseteq (N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$

start symbol $S \in N$

context



A Theory of Language

formal grammars

formal grammar $G = (N, \Sigma, P, S)$

nonterminal symbols N

terminal symbols Σ

production rules $P \subseteq (N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$

start symbol $S \in N$

replacement



A Theory of Language formal grammars

formal grammar $G = (N, \Sigma, P, S)$

nonterminal symbols N

terminal symbols Σ

production rules $P \subseteq (N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$

start symbol $S \in N$

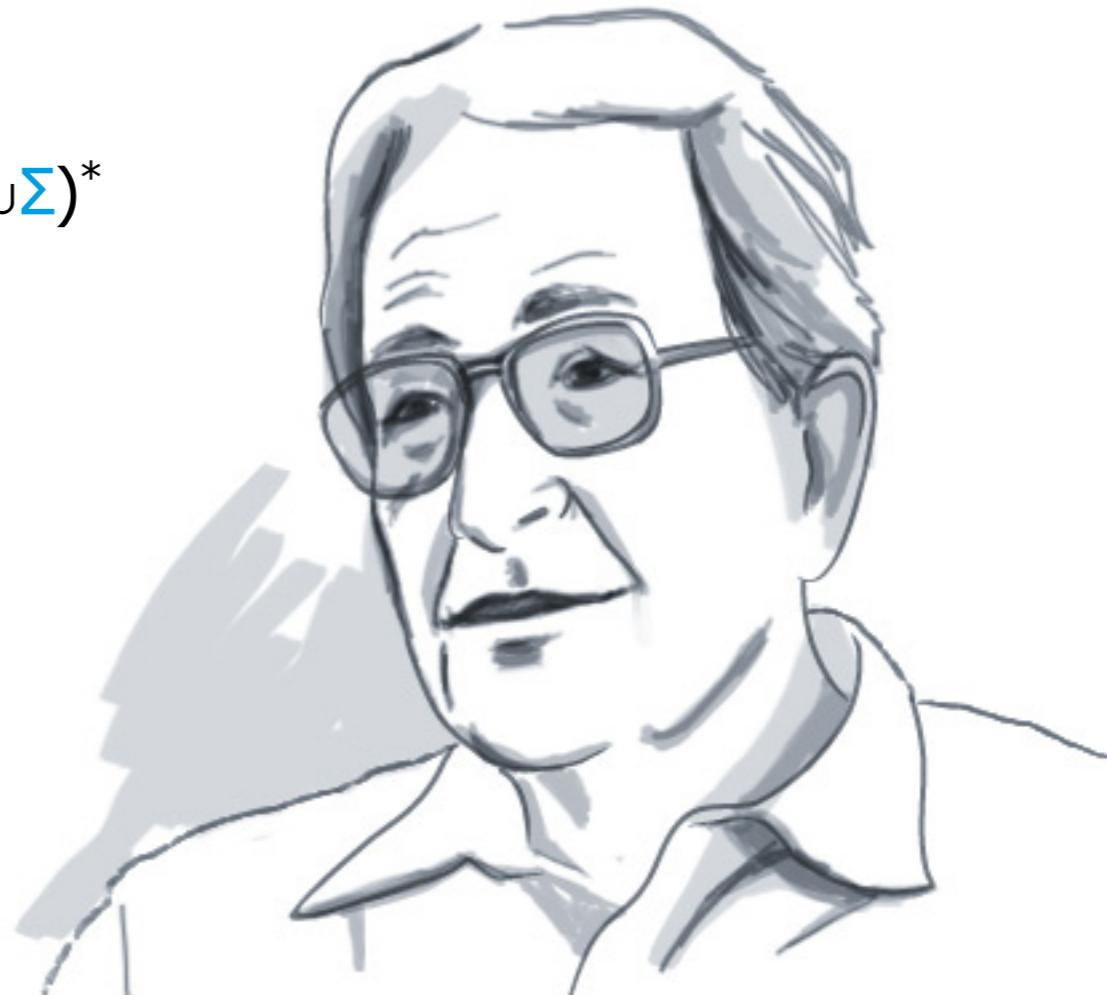
grammar classes

type-0, unrestricted

type-1, context-sensitive: $(a A c, a b c)$

type-2, context-free: $P \subseteq N \times (N \cup \Sigma)^*$

type-3, regular: (A, x) or (A, xB)



Decimal Numbers

morphology

$$G = (N, \Sigma, P, S)$$

Num → Digit Num

Num → Digit

Digit → "0"

Digit → "1"

Digit → "2"

Digit → "3"

Digit → "4"

Digit → "5"

Digit → "6"

Digit → "7"

Digit → "8"

Digit → "9"



Decimal Numbers

morphology

$$G = (N, \Sigma, P, S)$$

Num → Digit Num

Num → Digit

Digit → "0"

Digit → "1"

Digit → "2"

Digit → "3"

Digit → "4"

Digit → "5"

Digit → "6"

Digit → "7"

Digit → "8"

Digit → "9"

Σ : finite set of terminal symbols



Decimal Numbers

morphology

$$G = (N, \Sigma, P, S)$$

Num → Digit Num

Num → Digit

Digit → "0"

Digit → "1"

Digit → "2"

Digit → "3"

Digit → "4"

Digit → "5"

Digit → "6"

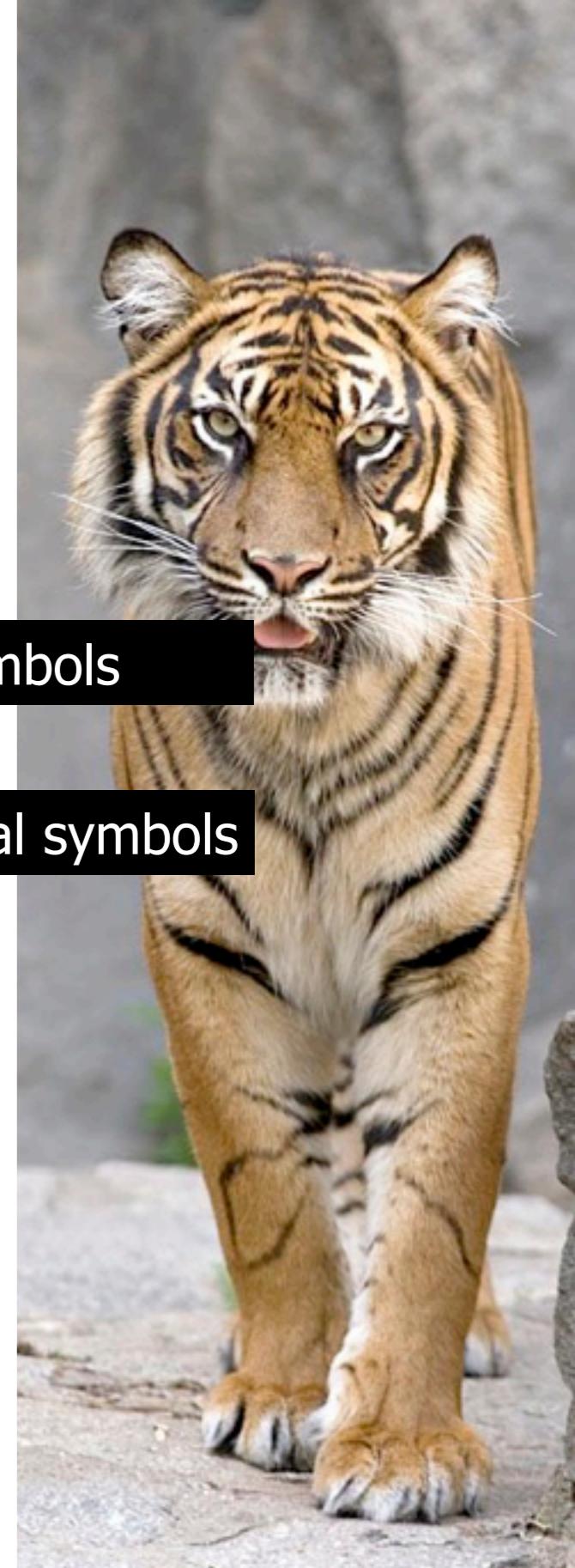
Digit → "7"

Digit → "8"

Digit → "9"

Σ : finite set of terminal symbols

N: finite set of non-terminal symbols



Decimal Numbers

morphology

$$G = (N, \Sigma, P, S)$$

Num → Digit Num

Num → Digit

Digit → "0"

Digit → "1"

Digit → "2"

Digit → "3"

Digit → "4"

Digit → "5"

Digit → "6"

Digit → "7"

Digit → "8"

Digit → "9"

Σ : finite set of terminal symbols

N: finite set of non-terminal symbols

S ∈ N: start symbol



Decimal Numbers

morphology

$$G = (N, \Sigma, P, S)$$

Num → Digit Num

Num → Digit

Digit → "0"

Digit → "1"

Digit → "2"

Digit → "3"

Digit → "4"

Digit → "5"

Digit → "6"

Digit → "7"

Digit → "8"

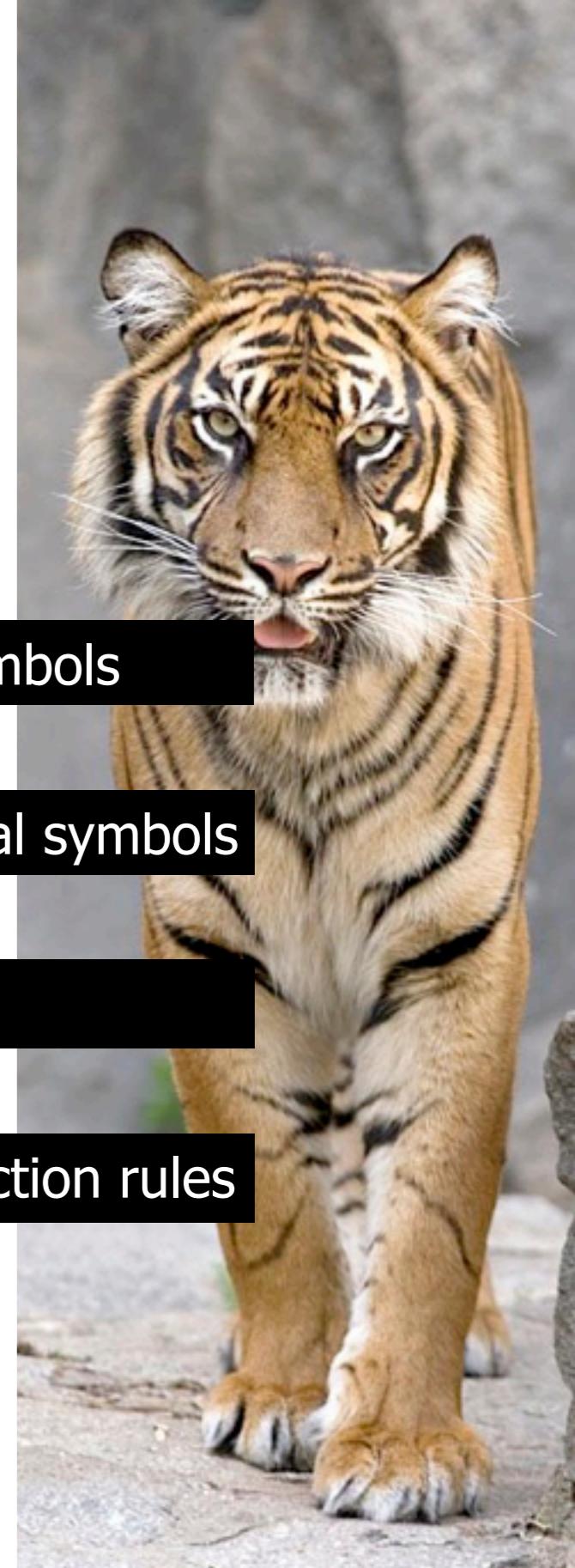
Digit → "9"

Σ : finite set of terminal symbols

N: finite set of non-terminal symbols

S ∈ N: start symbol

P ⊆ (N × N ∪ Σ)*: set of production rules



Decimal Numbers

production

Num

Decimal Numbers

production

Num

Digit Num

Num → Digit Num

Decimal Numbers

production

Num

Digit Num

Digit Digit Num

Num → Digit Num

Num → Digit Num

Decimal Numbers

production

Num

Digit Num

Digit Digit Num

Digit Digit Digit Num

Num → Digit Num

Num → Digit Num

Num → Digit Num

Decimal Numbers

production

Num

Digit Num

Digit Digit Num

Digit Digit Digit Num

Digit Digit Digit Digit

Num → Digit Num

Num → Digit Num

Num → Digit Num

Num → Digit

Decimal Numbers

production

Num

Digit Num

Digit Digit Num

Digit Digit Digit Num

Digit Digit Digit Digit

Digit Digit Digit 3

Num → Digit Num

Num → Digit Num

Num → Digit Num

Num → Digit

Digit → "3"

Decimal Numbers

production

Num

Digit Num

Digit Digit Num

Digit Digit Digit Num

Digit Digit Digit Digit

Digit Digit Digit 3

Digit Digit 0 3

Num → Digit Num

Num → Digit Num

Num → Digit Num

Num → Digit

Digit → "3"

Digit → "0"

Decimal Numbers

production

Num

Digit Num

Digit Digit Num

Digit Digit Digit Num

Digit Digit Digit Digit

Digit Digit Digit 3

Digit Digit 0 3

Digit 3 0 3

Num → Digit Num

Num → Digit Num

Num → Digit Num

Num → Digit

Digit → "3"

Digit → "0"

Digit → "3"

Decimal Numbers

production

Num

Digit Num

Digit Digit Num

Digit Digit Digit Num

Digit Digit Digit Digit

Digit Digit Digit 3

Digit Digit 0 3

Digit 3 0 3

4 3 0 3

Num → Digit Num

Num → Digit Num

Num → Digit Num

Num → Digit

Digit → "3"

Digit → "0"

Digit → "3"

Digit → "4"

rightmost derivation

Decimal Numbers

production

Num

Decimal Numbers

production

Num

Digit Num

Num → Digit Num

Decimal Numbers

production

Num

Digit Num

4 Num

Num → Digit Num

Digit → "4"

Decimal Numbers

production

Num

Digit Num

4 Num

4 Digit Num

Num → Digit Num

Digit → "4"

Num → Digit Num

Decimal Numbers

production

Num

Digit Num

4 Num

4 Digit Num

4 3 Num

Num → Digit Num

Digit → "4"

Num → Digit Num

Digit → "3"

Decimal Numbers

production

Num

Digit Num

4 Num

4 Digit Num

4 3 Num

4 3 Digit Num

Num → Digit Num

Digit → "4"

Num → Digit Num

Digit → "3"

Num → Digit Num

Decimal Numbers

production

Num

Digit Num

4 Num

4 Digit Num

4 3 Num

4 3 Digit Num

4 3 0 Num

Num → Digit Num

Digit → "4"

Num → Digit Num

Digit → "3"

Num → Digit Num

Digit → "0"

Decimal Numbers

production

Num

Digit Num

4 Num

4 Digit Num

4 3 Num

4 3 Digit Num

4 3 0 Num

4 3 0 Digit

Num → Digit Num

Digit → "4"

Num → Digit Num

Digit → "3"

Num → Digit Num

Digit → "0"

Num → Digit

Decimal Numbers

production

Num

Digit Num

4 Num

4 Digit Num

4 3 Num

4 3 Digit Num

4 3 0 Num

4 3 0 Digit

4 3 0 3

Num → Digit Num

Digit → "4"

Num → Digit Num

Digit → "3"

Num → Digit Num

Digit → "0"

Num → Digit

Digit → "3"

leftmost derivation

Binary Expressions

syntax

$$G = (N, \Sigma, P, S)$$

Exp → Num
Exp → Exp "+" Exp
Exp → Exp "-" Exp
Exp → Exp "*" Exp
Exp → Exp "/" Exp
Exp → "(" Exp ")"



Binary Expressions

syntax

$G = (N, \Sigma, P, S)$

$\text{Exp} \rightarrow \text{Num}$

$\text{Exp} \rightarrow \text{Exp} "+"$ Exp

$\text{Exp} \rightarrow \text{Exp} "-"$ Exp

$\text{Exp} \rightarrow \text{Exp} "*"$ Exp

$\text{Exp} \rightarrow \text{Exp} "/"$ Exp

$\text{Exp} \rightarrow "(" \text{Exp} ")"$

Σ : finite set of terminal symbols



Binary Expressions

syntax

$$G = (N, \Sigma, P, S)$$

Exp → Num

Exp → Exp "+" Exp

Exp → Exp "-" Exp

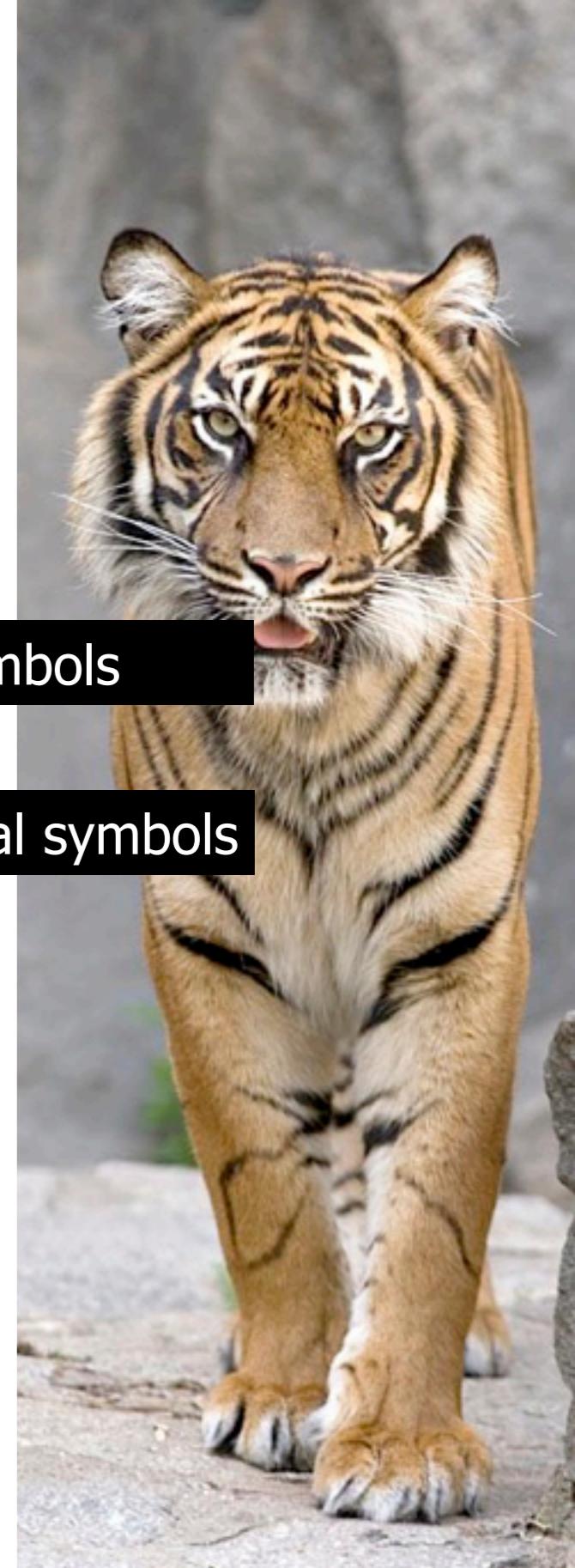
Exp → Exp "*" Exp

Exp → Exp "/" Exp

Exp → "(" Exp ")"

Σ : finite set of terminal symbols

N: finite set of non-terminal symbols



Binary Expressions

syntax

$$G = (N, \Sigma, P, S)$$

$Exp \rightarrow Num$

$Exp \rightarrow Exp "+" Exp$

$Exp \rightarrow Exp "-" Exp$

$Exp \rightarrow Exp "*" Exp$

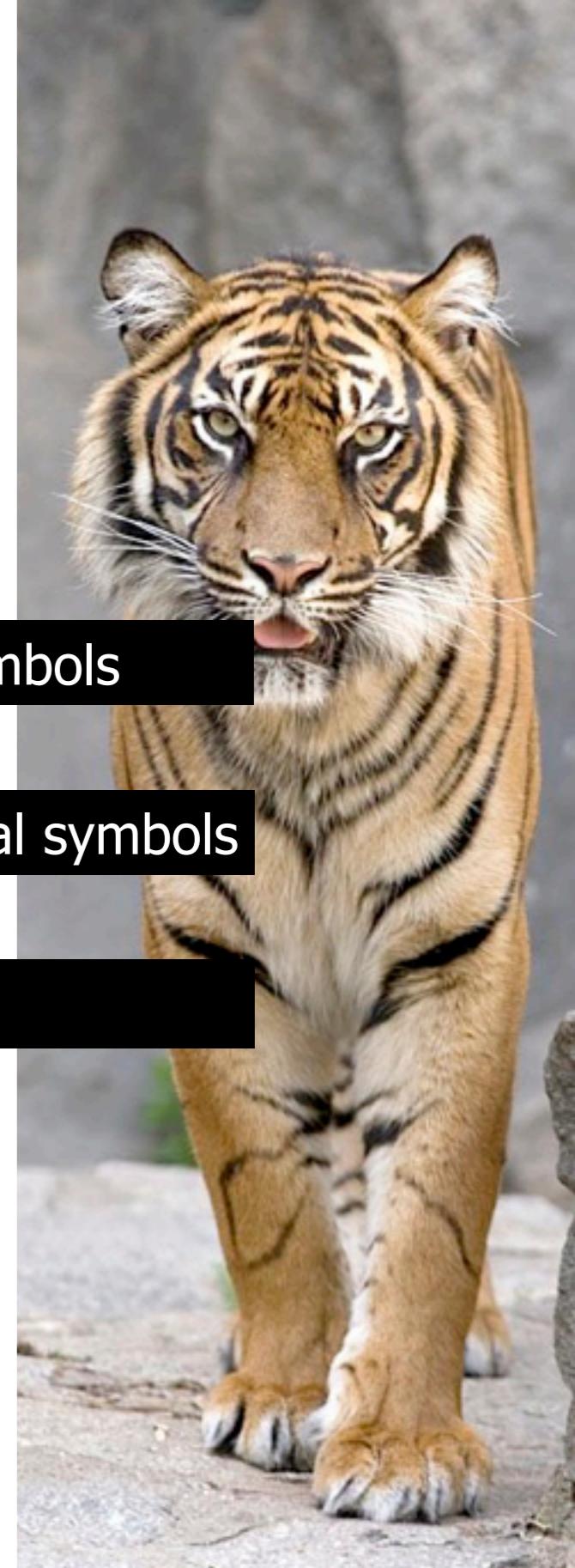
$Exp \rightarrow Exp "/" Exp$

$Exp \rightarrow "(" Exp ")"$

Σ : finite set of terminal symbols

N : finite set of non-terminal symbols

$S \in N$: start symbol



Binary Expressions

syntax

$G = (N, \Sigma, P, S)$

$Exp \rightarrow Num$

$Exp \rightarrow Exp "+" Exp$

$Exp \rightarrow Exp "-" Exp$

$Exp \rightarrow Exp "*" Exp$

$Exp \rightarrow Exp "/" Exp$

$Exp \rightarrow "(" Exp ")"$

Σ : finite set of terminal symbols

N : finite set of non-terminal symbols

$S \in N$: start symbol

$P \subseteq (N \times N \cup \Sigma)^*$: set of production rules



Binary Expressions

production

Exp

Binary Expressions

production

Exp

Exp + Exp

Exp → Exp “+” Exp

Binary Expressions

production

Exp

Exp + Exp

Exp + Exp * Exp

Exp → Exp "+" Exp

Exp → Exp "*" Exp

Binary Expressions

production

Exp

Exp + Exp

Exp + Exp * Exp

3 + Exp * Exp

Exp → Exp "+" Exp

Exp → Exp "*" Exp

Exp → Num

Binary Expressions

production

Exp	$\text{Exp} \rightarrow \text{Exp} \text{ "+" Exp}$
Exp + Exp	$\text{Exp} \rightarrow \text{Exp} \text{ "*" Exp}$
Exp + Exp * Exp	$\text{Exp} \rightarrow \text{Num}$
3 + Exp * Exp	$\text{Exp} \rightarrow \text{Num}$
3 + 4 * Exp	

Binary Expressions

production

Exp

Exp + Exp

Exp + Exp * Exp

3 + Exp * Exp

3 + 4 * Exp

3 + 4 * 5

Exp → Exp "+" Exp

Exp → Exp "*" Exp

Exp → Num

Exp → Num

Exp → Num

A Theory of Language

formal languages



A Theory of Language

formal languages

formal grammar $G = (N, \Sigma, P, S)$



A Theory of Language

formal languages

formal grammar $G = (N, \Sigma, P, S)$

derivation relation $\Rightarrow_G \subseteq (N \cup \Sigma)^* \times (N \cup \Sigma)^*$

$w \Rightarrow_G w' \Leftrightarrow$

$\exists (p, q) \in P : \exists u, v \in (N \cup \Sigma)^* :$

$w = u p v \wedge w' = u q v$



A Theory of Language

formal languages

formal grammar $G = (N, \Sigma, P, S)$

derivation relation $\Rightarrow_G \subseteq (N \cup \Sigma)^* \times (N \cup \Sigma)^*$

$w \Rightarrow_G w' \Leftrightarrow$

$\exists (p, q) \in P: \exists u, v \in (N \cup \Sigma)^*:$

$w = u p v \wedge w' = u q v$

formal language $L(G) \subseteq \Sigma^*$

$L(G) = \{w \in \Sigma^* \mid S \xrightarrow{G}^* w\}$



A Theory of Language

formal languages

formal grammar $G = (N, \Sigma, P, S)$

derivation relation $\Rightarrow_G \subseteq (N \cup \Sigma)^* \times (N \cup \Sigma)^*$

$w \Rightarrow_G w' \Leftrightarrow$

$\exists (p, q) \in P : \exists u, v \in (N \cup \Sigma)^* :$

$w = u p v \wedge w' = u q v$

formal language $L(G) \subseteq \Sigma^*$

$L(G) = \{w \in \Sigma^* \mid S \xrightarrow{G}^* w\}$

classes of formal languages



II

word problem

Theoretical Computer Science

decidability & complexity

Theoretical Computer Science

decidability & complexity

word problem $x_L: \Sigma^* \rightarrow \{0,1\}$

$w \rightarrow 1$, if $w \in L$

$w \rightarrow 0$, else

Theoretical Computer Science

decidability & complexity

word problem $x_L: \Sigma^* \rightarrow \{0,1\}$

$w \rightarrow 1$, if $w \in L$

$w \rightarrow 0$, else

decidability

type-0: semi-decidable

type-1, type-2, type-3: decidable

Theoretical Computer Science

decidability & complexity

word problem $x_L: \Sigma^* \rightarrow \{0,1\}$

$w \rightarrow 1$, if $w \in L$

$w \rightarrow 0$, else

decidability

type-0: semi-decidable

type-1, type-2, type-3: decidable

complexity

type-1: PSPACE-complete

type-2, type-3: P

Theoretical Computer Science

decidability & complexity

word problem $x_L: \Sigma^* \rightarrow \{0,1\}$

$w \rightarrow 1$, if $w \in L$

$w \rightarrow 0$, else

decidability

type-0: semi-decidable

type-1, type-2, type-3: decidable

complexity

type-1: PSPACE-complete

PSPACE \supseteq NP \supseteq P

type-2, type-3: P

Context-free Grammars

production vs. reduction rules

$\text{Exp} \rightarrow \text{Num}$

$\text{Exp} \rightarrow \text{Exp} \text{ "+" Exp}$

$\text{Exp} \rightarrow \text{Exp} \text{ "-" Exp}$

$\text{Exp} \rightarrow \text{Exp} \text{ "*" Exp}$

$\text{Exp} \rightarrow \text{Exp} \text{ "/" Exp}$

$\text{Exp} \rightarrow (" \text{Exp} ")$

Context-free Grammars

production vs. reduction rules

$\text{Exp} \rightarrow \text{Num}$

$\text{Exp} \rightarrow \text{Exp} "+"$ Exp

$\text{Exp} \rightarrow \text{Exp} "-"$ Exp

$\text{Exp} \rightarrow \text{Exp} "*"$ Exp

$\text{Exp} \rightarrow \text{Exp} "/"$ Exp

$\text{Exp} \rightarrow "(" \text{Exp} ")"$

productive form

Context-free Grammars

production vs. reduction rules

$\text{Exp} \rightarrow \text{Num}$

$\text{Exp} \rightarrow \text{Exp} "+" \text{Exp}$

$\text{Exp} \rightarrow \text{Exp} "-" \text{Exp}$

$\text{Exp} \rightarrow \text{Exp} "*" \text{Exp}$

$\text{Exp} \rightarrow \text{Exp} "/" \text{Exp}$

$\text{Exp} \rightarrow "(" \text{Exp} ")"$

$\text{Num} \rightarrow \text{Exp}$

$\text{Exp} "+" \text{Exp} \rightarrow \text{Exp}$

$\text{Exp} "-" \text{Exp} \rightarrow \text{Exp}$

$\text{Exp} "*" \text{Exp} \rightarrow \text{Exp}$

$\text{Exp} "/" \text{Exp} \rightarrow \text{Exp}$

$"(" \text{Exp} ")" \rightarrow \text{Exp}$

productive form

Context-free Grammars

production vs. reduction rules

$\text{Exp} \rightarrow \text{Num}$

$\text{Exp} \rightarrow \text{Exp} "+" \text{Exp}$

$\text{Exp} \rightarrow \text{Exp} "-" \text{Exp}$

$\text{Exp} \rightarrow \text{Exp} "*" \text{Exp}$

$\text{Exp} \rightarrow \text{Exp} "/" \text{Exp}$

$\text{Exp} \rightarrow "(" \text{Exp} ")"$

$\text{Num} \rightarrow \text{Exp}$

$\text{Exp} "+" \text{Exp} \rightarrow \text{Exp}$

$\text{Exp} "-" \text{Exp} \rightarrow \text{Exp}$

$\text{Exp} "*" \text{Exp} \rightarrow \text{Exp}$

$\text{Exp} "/" \text{Exp} \rightarrow \text{Exp}$

$"(" \text{Exp} ")" \rightarrow \text{Exp}$

productive form

reductive form

Binary Expressions

reduction

$3 + 4 * 5$

Binary Expressions

reduction

3 + 4 * 5

Num → Exp

Exp + 4 * 5

Binary Expressions

reduction

$3 + 4 * 5$

Num → Exp

Exp + $4 * 5$

Num → Exp

Exp + Exp * 5

Binary Expressions

reduction

$3 + 4 * 5$

Num → Exp

Exp + $4 * 5$

Num → Exp

Exp + Exp * 5

Num → Exp

Exp + Exp * Exp

Binary Expressions

reduction

3 + 4 * 5

Num → Exp

Exp + 4 * 5

Num → Exp

Exp + Exp * 5

Num → Exp

Exp + Exp * Exp

Exp "*" Exp → Exp

Exp + Exp

Binary Expressions

reduction

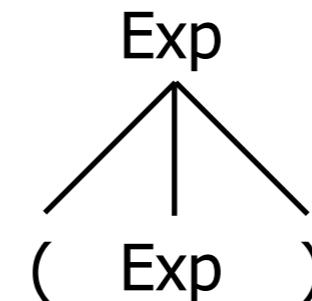
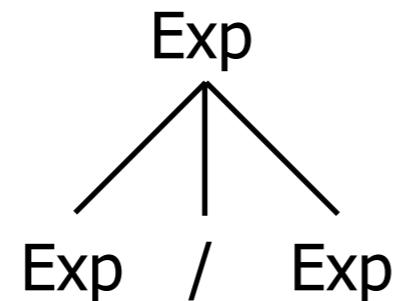
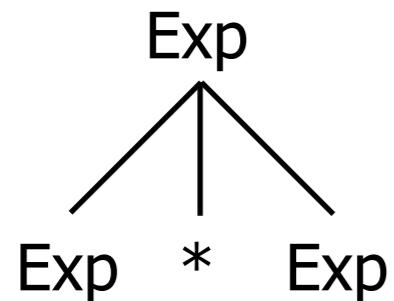
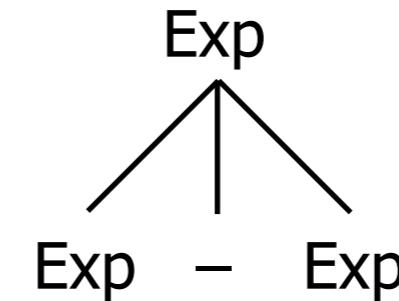
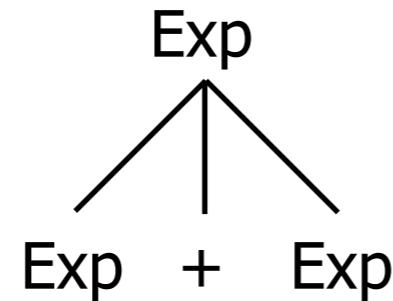
3 + 4 * 5	Num → Exp
Exp + 4 * 5	Num → Exp
Exp + Exp * 5	Num → Exp
Exp + Exp * Exp	Exp "*" Exp → Exp
Exp + Exp	Exp "+" Exp → Exp
Exp	

III

syntax trees

Context-free Grammars

tree construction rules



Binary Expressions

tree construction

3 + 4 * 5

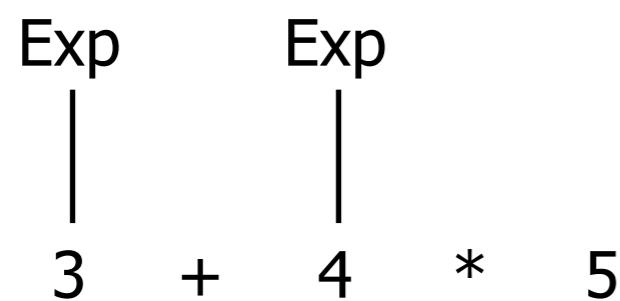
Binary Expressions

tree construction



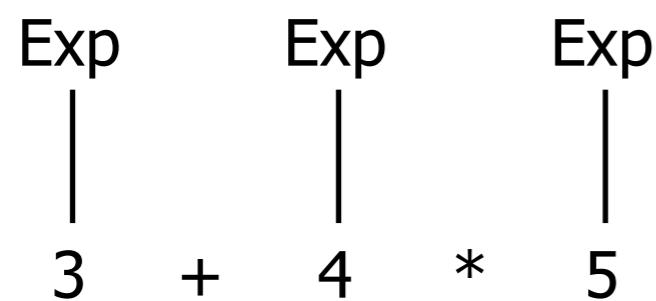
Binary Expressions

tree construction



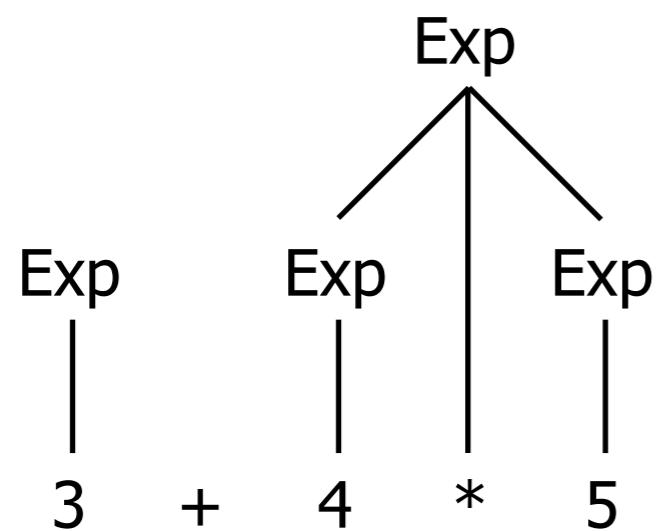
Binary Expressions

tree construction



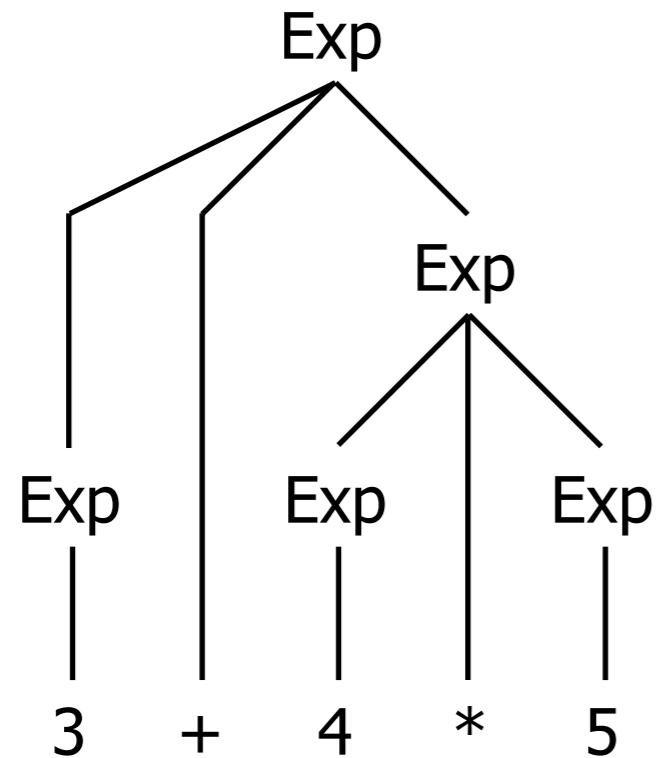
Binary Expressions

tree construction



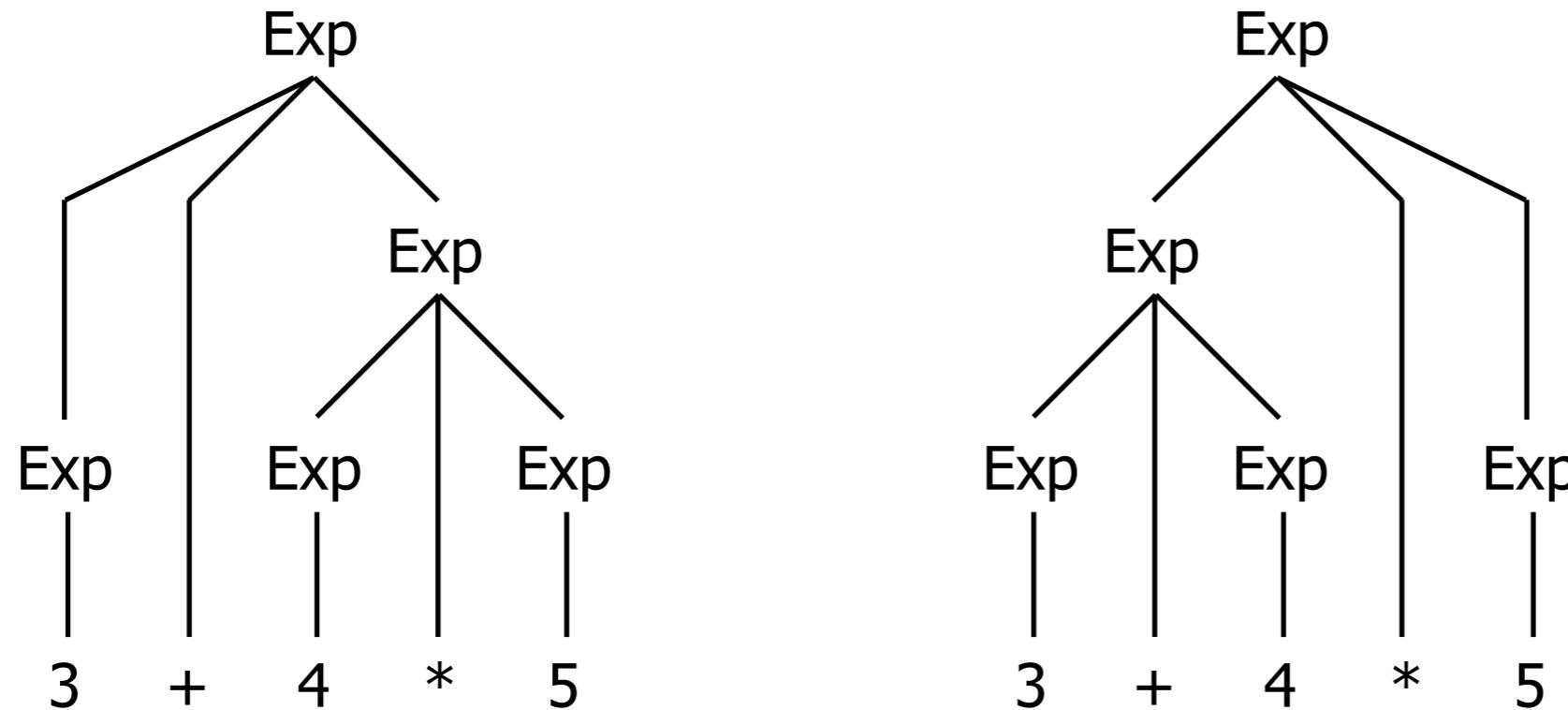
Binary Expressions

tree construction



Context-free Grammars

ambiguities



Abstract Syntax Trees

parse trees

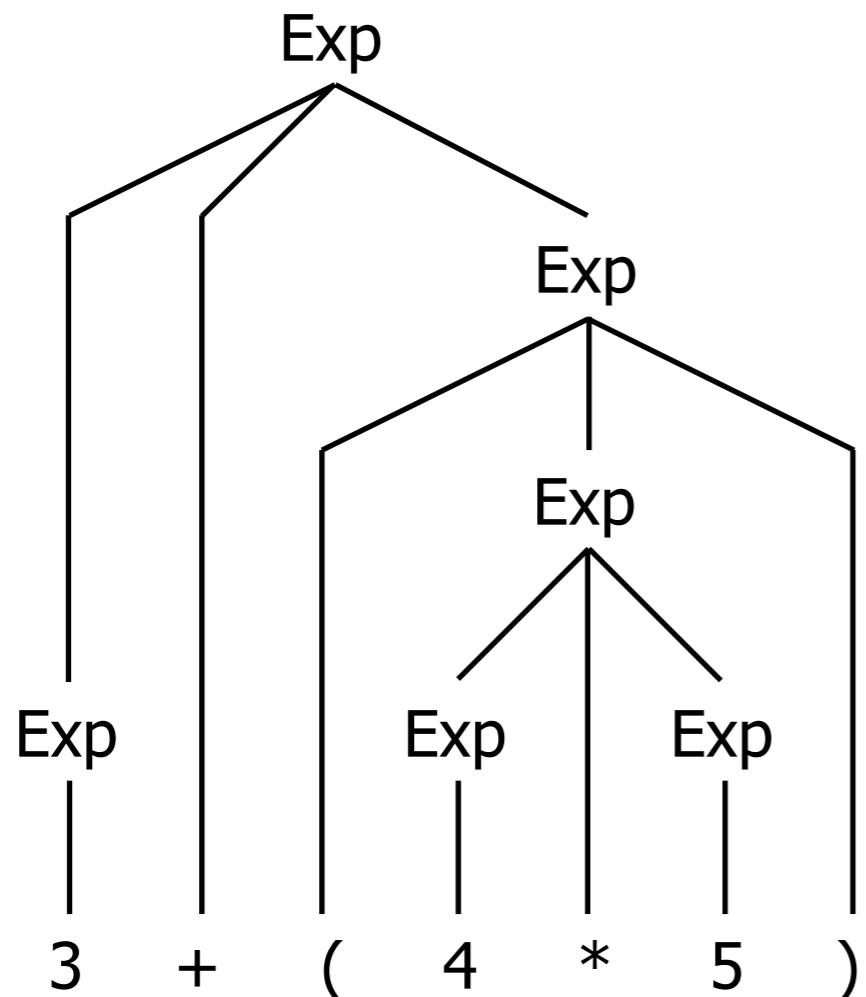
- parent node: nonterminal symbol
- children nodes: nonterminal and terminal symbols

abstract syntax trees

- abstract over terminal symbols
- convey information at parent nodes
- abstract over injective production rules

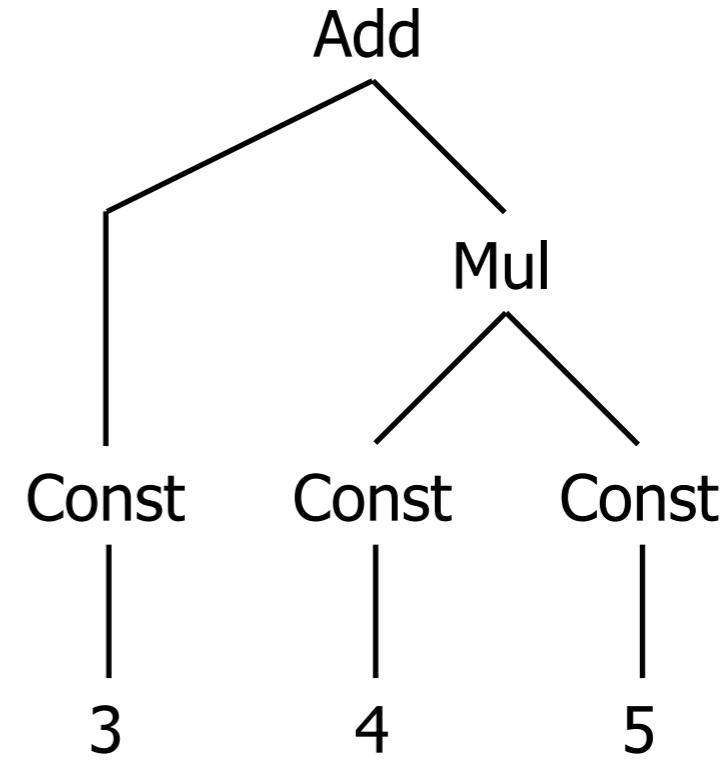
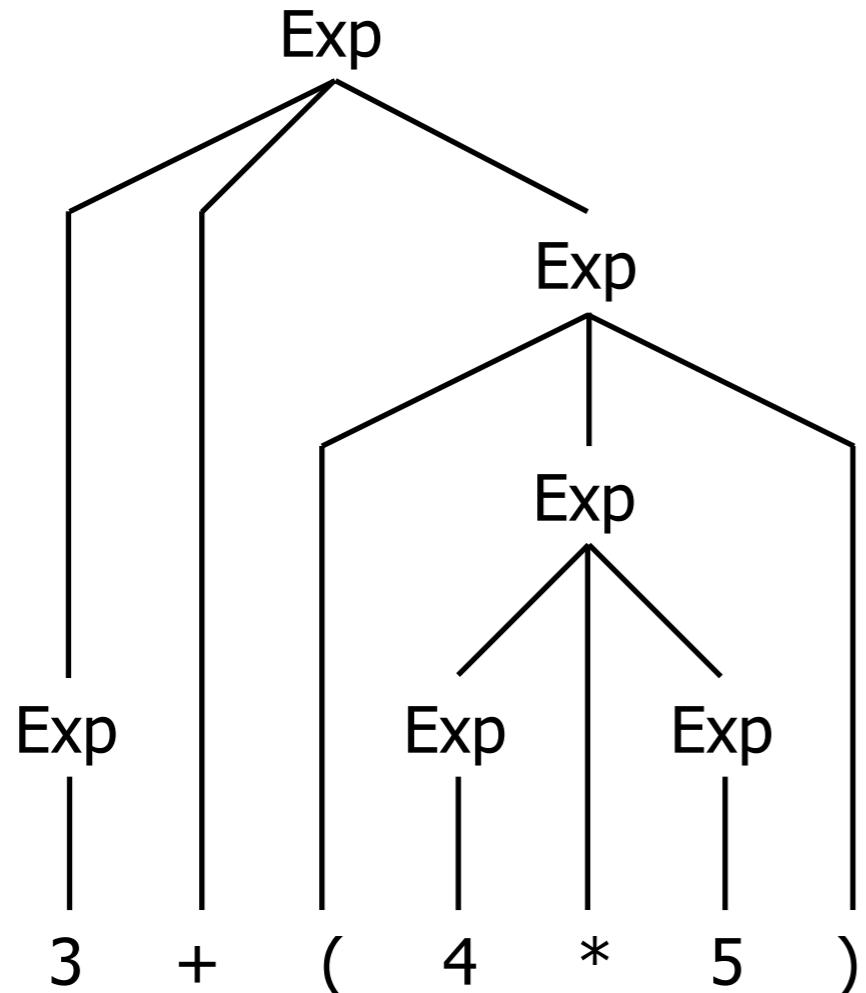
Binary Expressions

abstract syntax trees



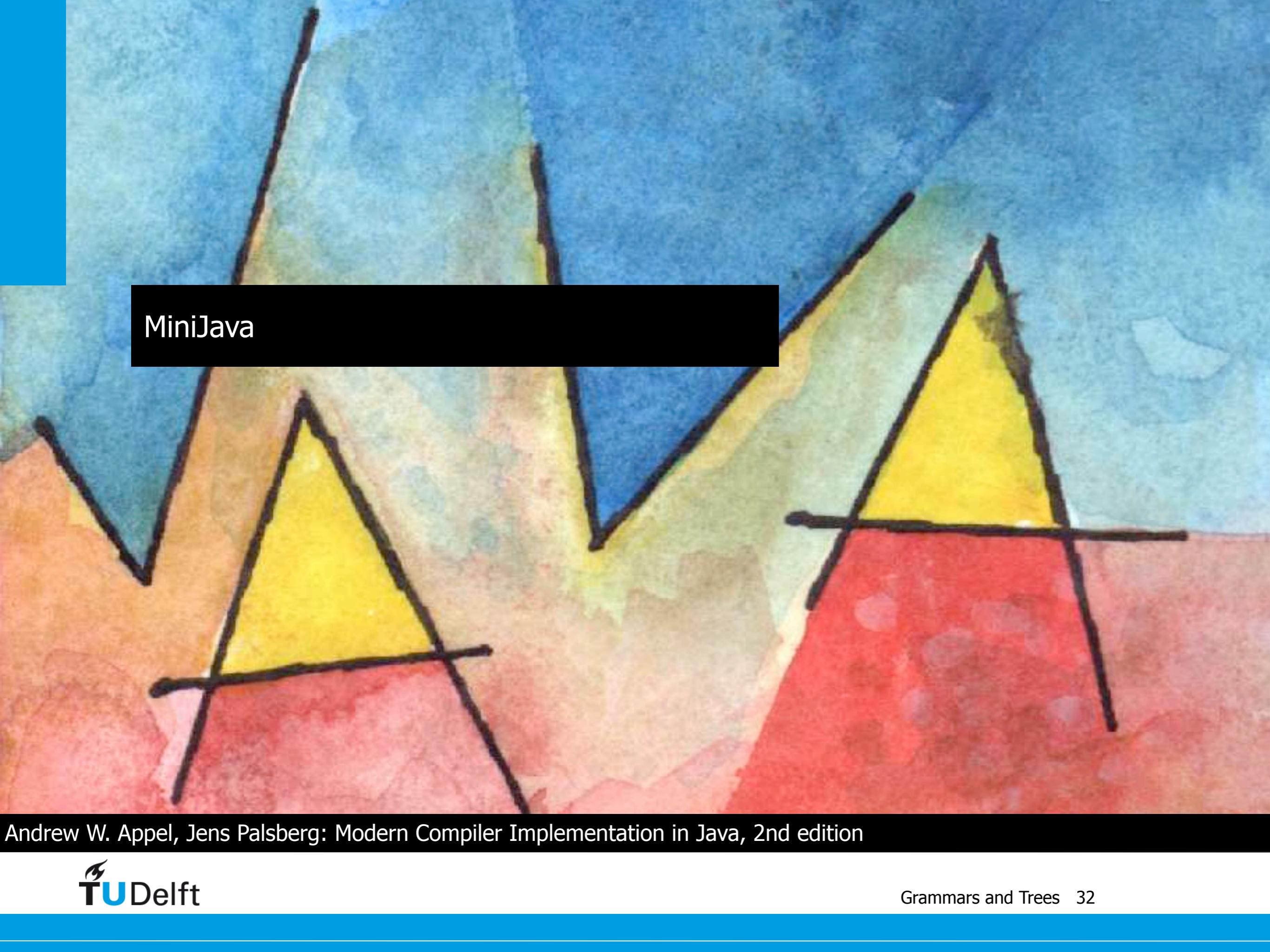
Binary Expressions

abstract syntax trees



coffee break



The background of the slide is an abstract painting featuring a repeating pattern of triangles in various colors like blue, green, yellow, and orange. A thick black horizontal bar is positioned across the middle of the slide, containing the text "MiniJava".

MiniJava

Andrew W. Appel, Jens Palsberg: Modern Compiler Implementation in Java, 2nd edition

MiniJava

lexical syntax

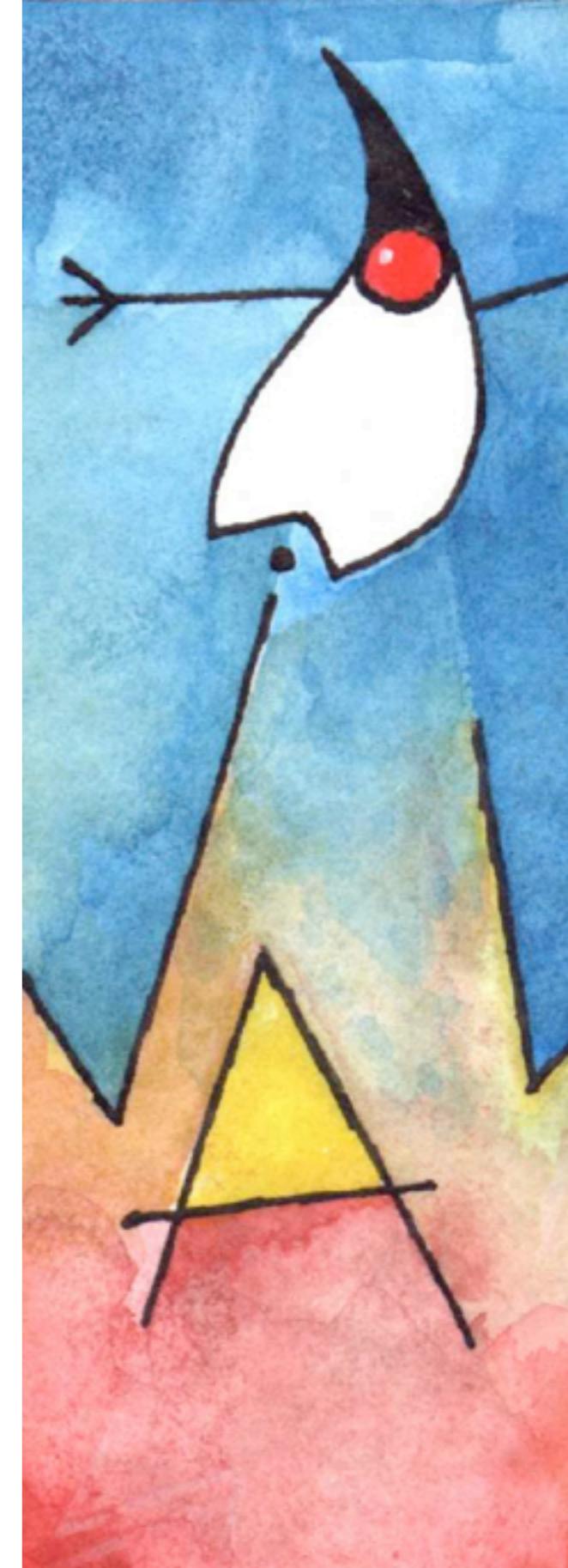
An **identifier** is a sequence of letters, digits, and underscores, starting with a letter. Uppercase letters are distinguished from lowercase.

A sequence of decimal digits is an **integer constant** that denotes the corresponding integer value.

A **binary operator** is one of

&& < + - *

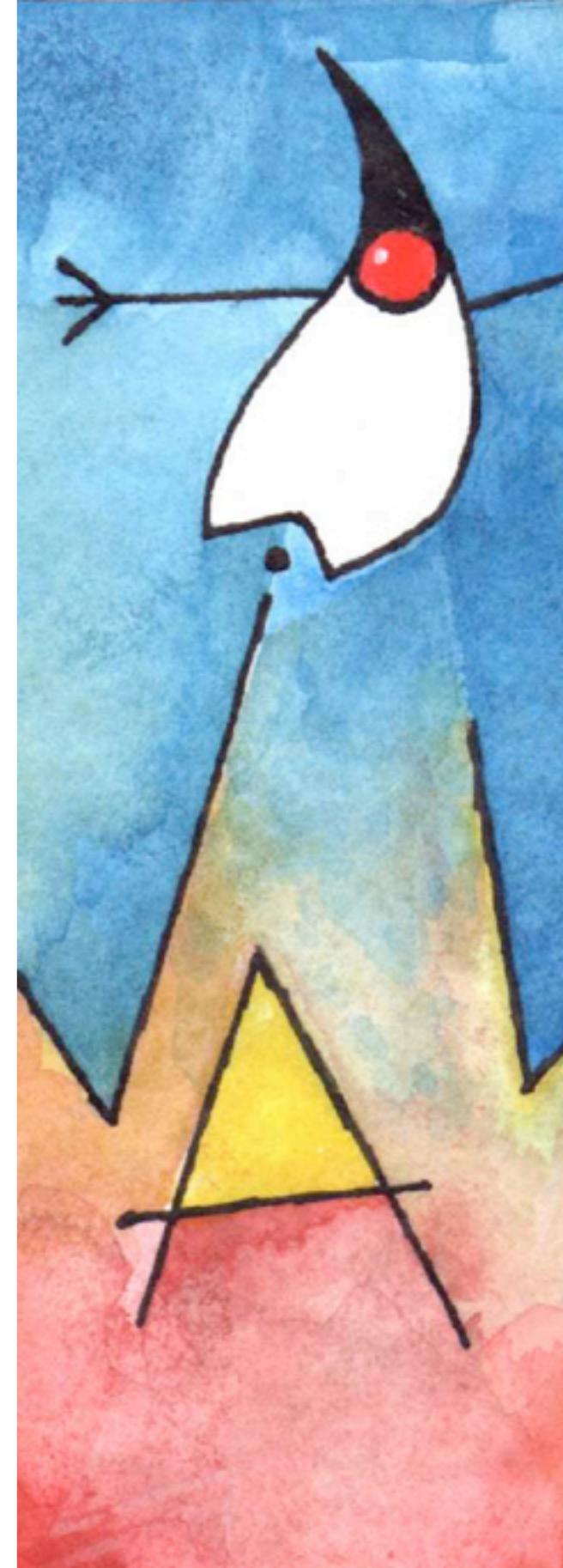
A **comment** may appear between any two tokens. There are two forms of comments: One starts with `/*`, ends with `*/`, and may be nested; another begins with `//` and goes to the end of the line.



MiniJava Expressions

42

Exp → Int

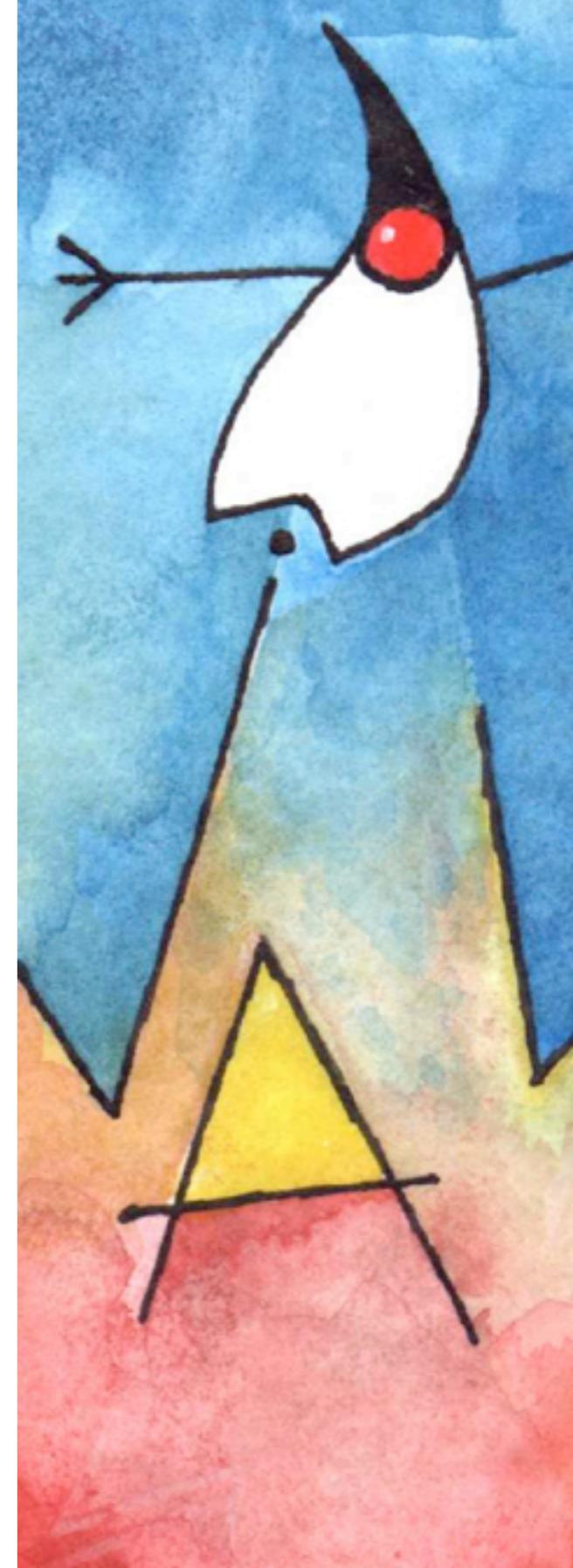


MiniJava Expressions

42

true

Exp → Int
→ "true"



MiniJava Expressions

42

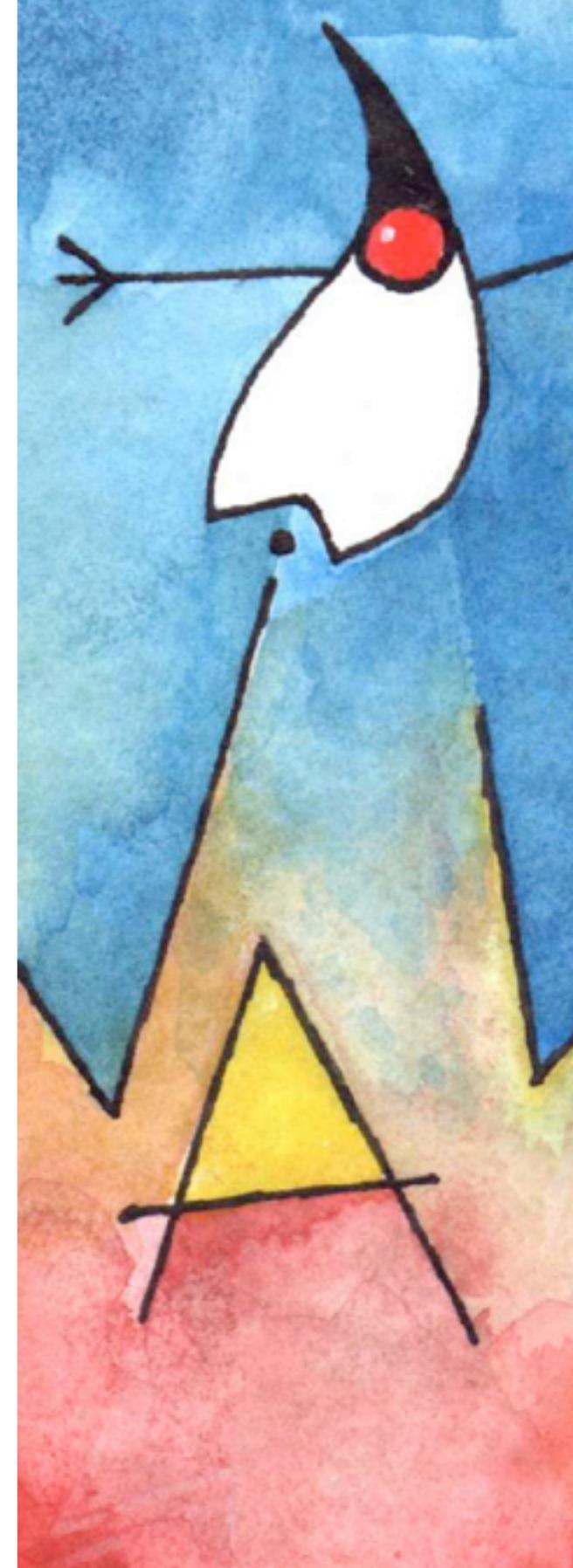
true

false

Exp → Int

→ "true"

→ "false"



MiniJava Expressions

42

true

false

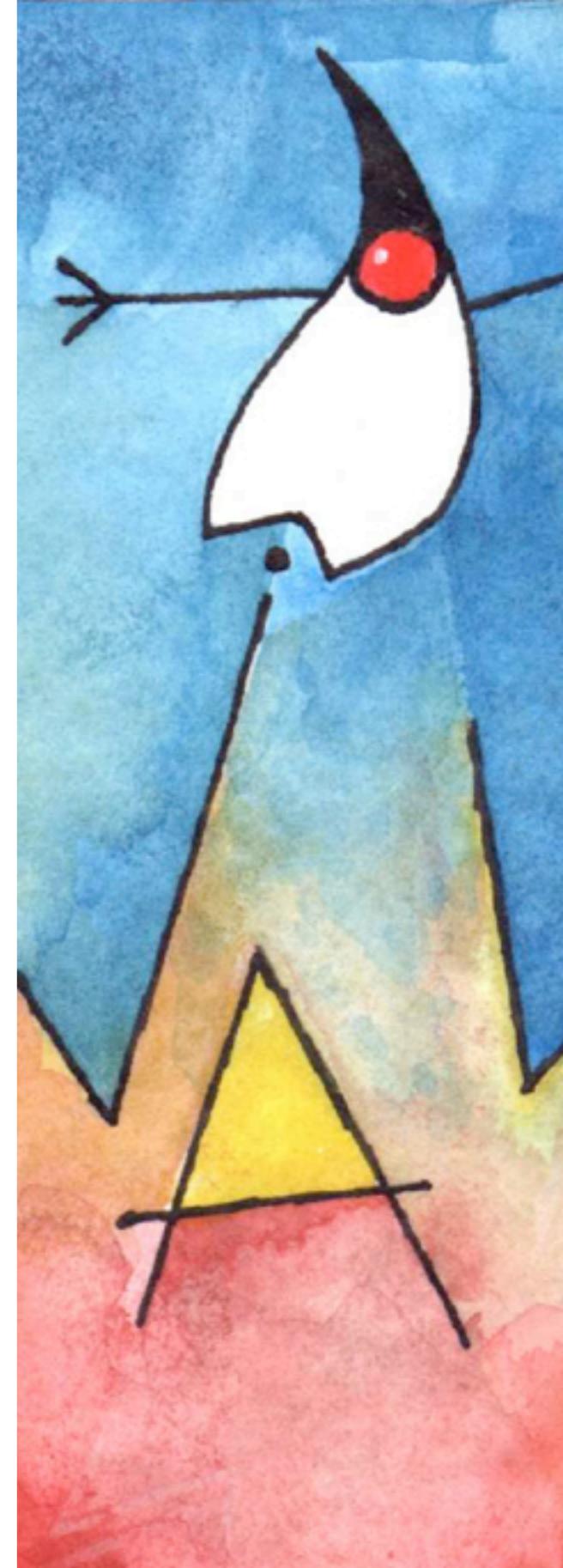
x

Exp → Int

→ "true"

→ "false"

→ Id



MiniJava Expressions

42

true

false

x

x*y

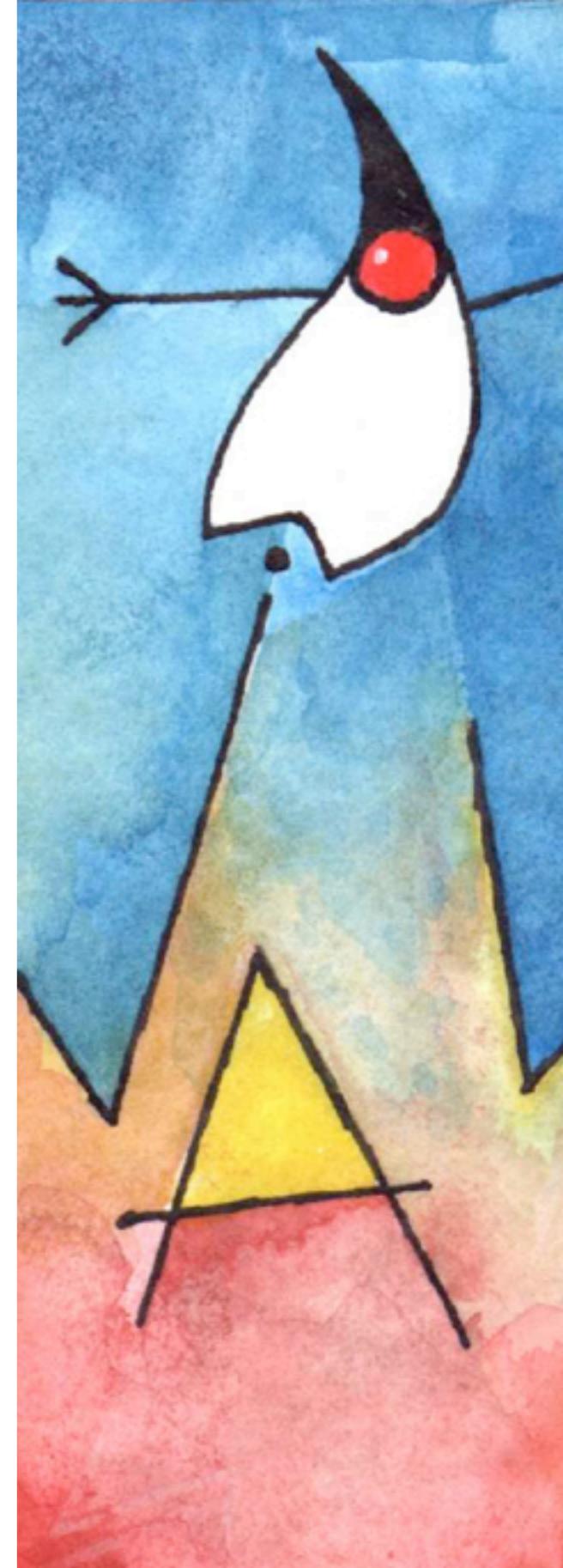
Exp → Int

→ "true"

→ "false"

→ Id

→ Exp Op Exp



MiniJava Expressions

42

true

false

x

x*y

!x

Exp → Int

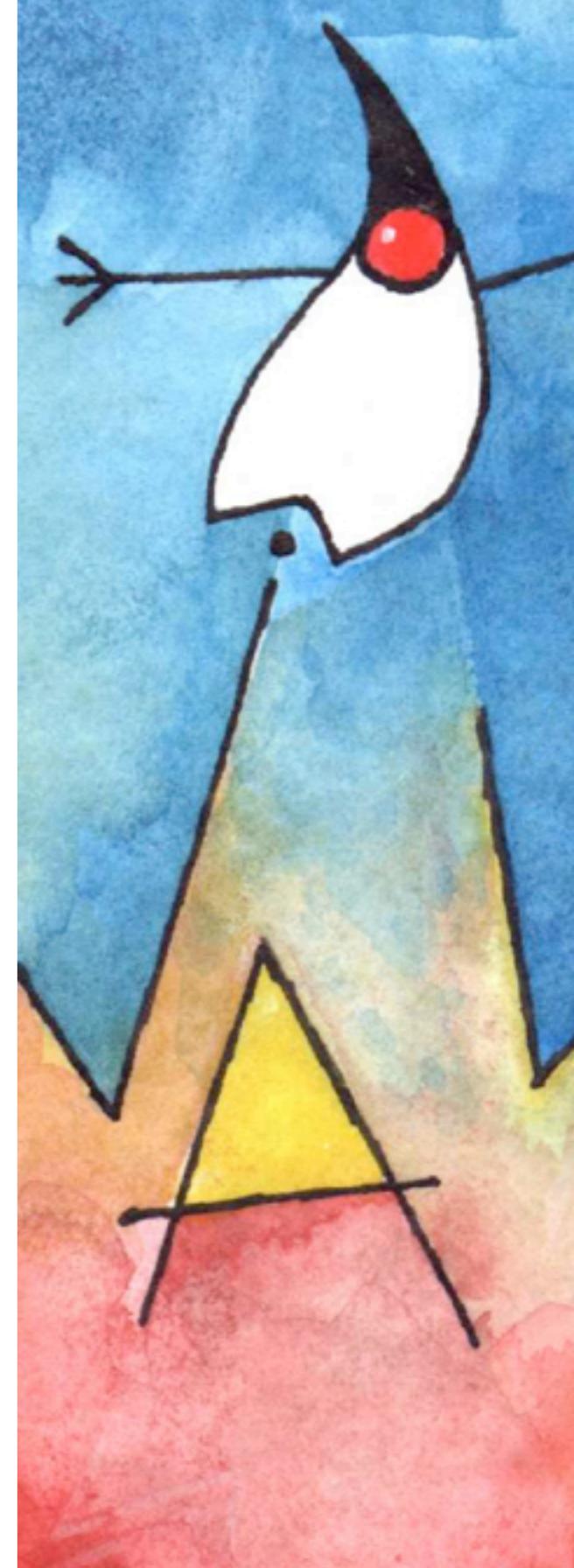
→ "true"

→ "false"

→ Id

→ Exp Op Exp

→ "!" Exp



MiniJava Expressions

42

true

false

x

x*y

!x

(x)

Exp → Int

→ "true"

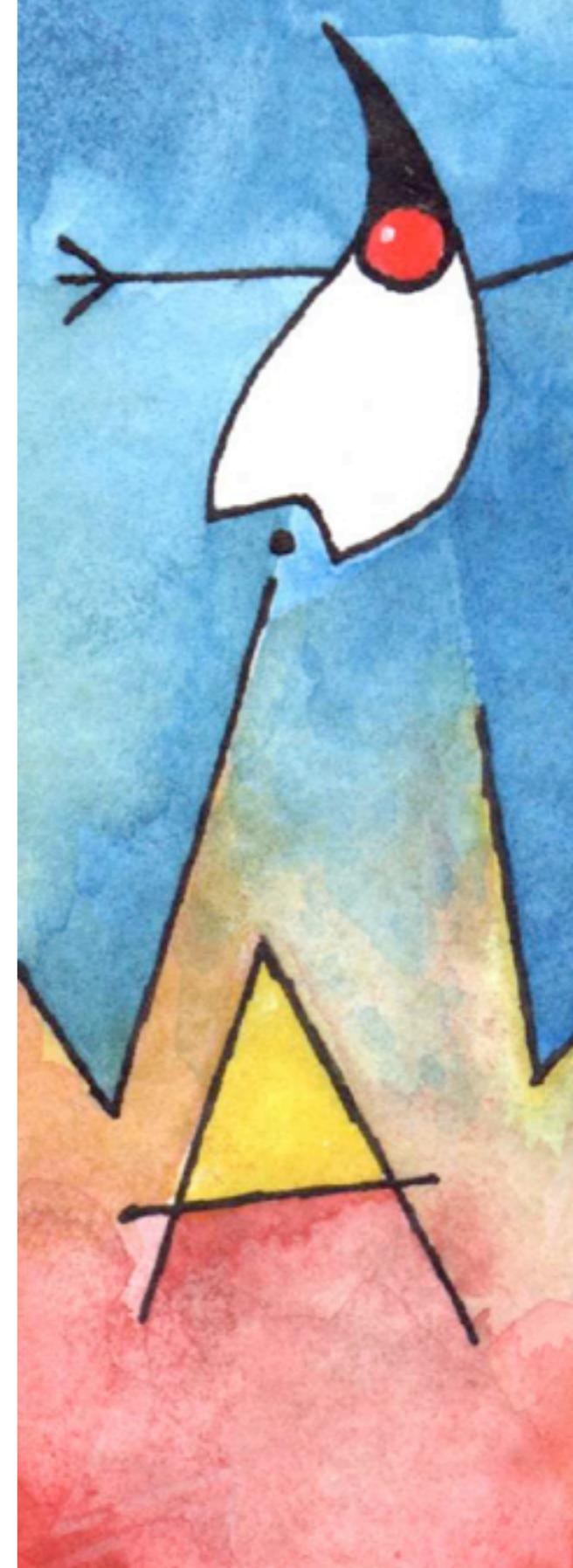
→ "false"

→ Id

→ Exp Op Exp

→ "!" Exp

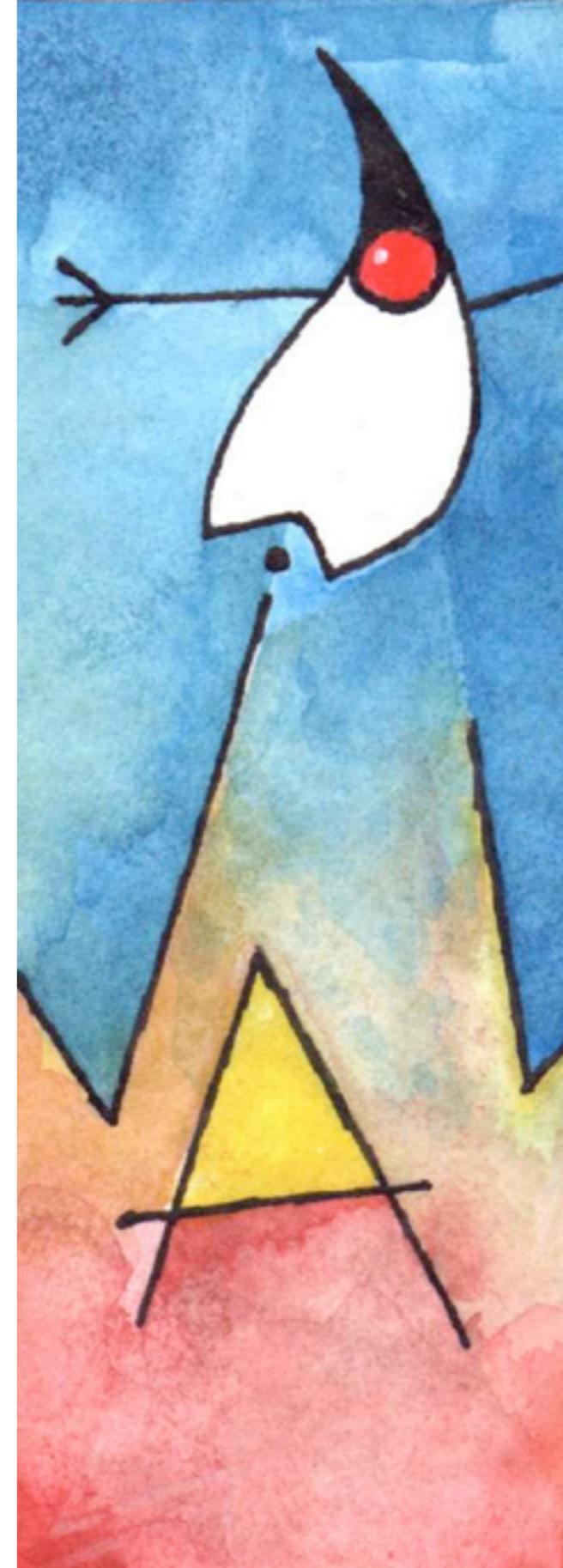
→ "(" Exp ")"



MiniJava Expressions

```
new int[42]
```

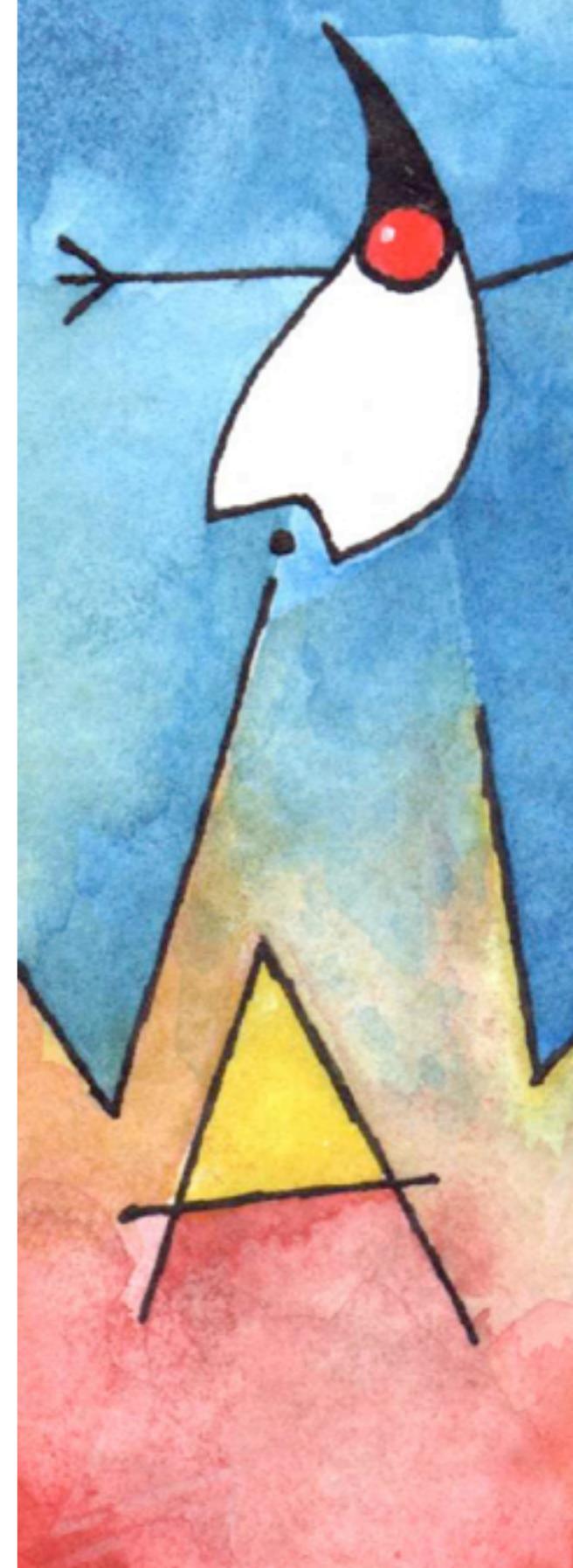
```
Exp → "new" "int" "[" Exp "]"
```



MiniJava Expressions

```
new int[42]  
x[42]
```

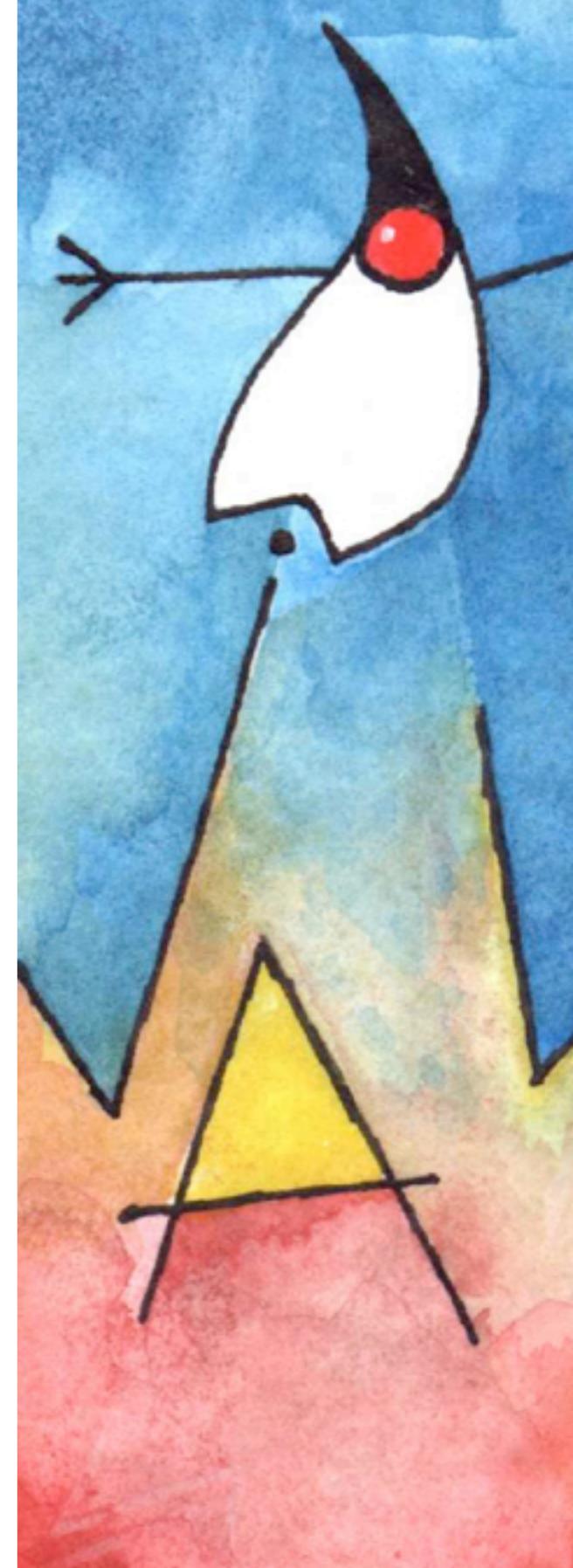
```
Exp → "new" "int" "[" Exp "]"  
      → Exp "[" Exp "]"
```



MiniJava Expressions

```
new int[42]  
x[42]  
x.length
```

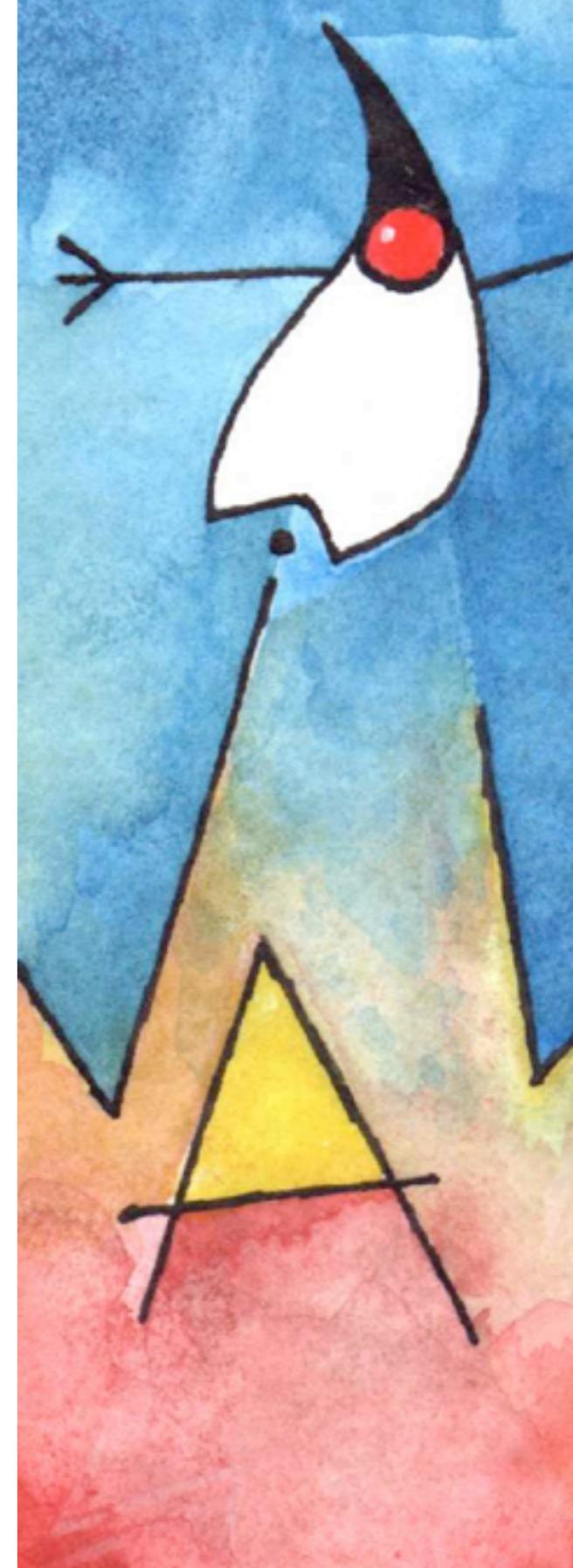
```
Exp → "new" "int" "[" Exp "]"  
      → Exp "[" Exp "]"  
      → Exp "." "length"
```



MiniJava Expressions

```
new int[42]
x[42]
x.length
new Class()
```

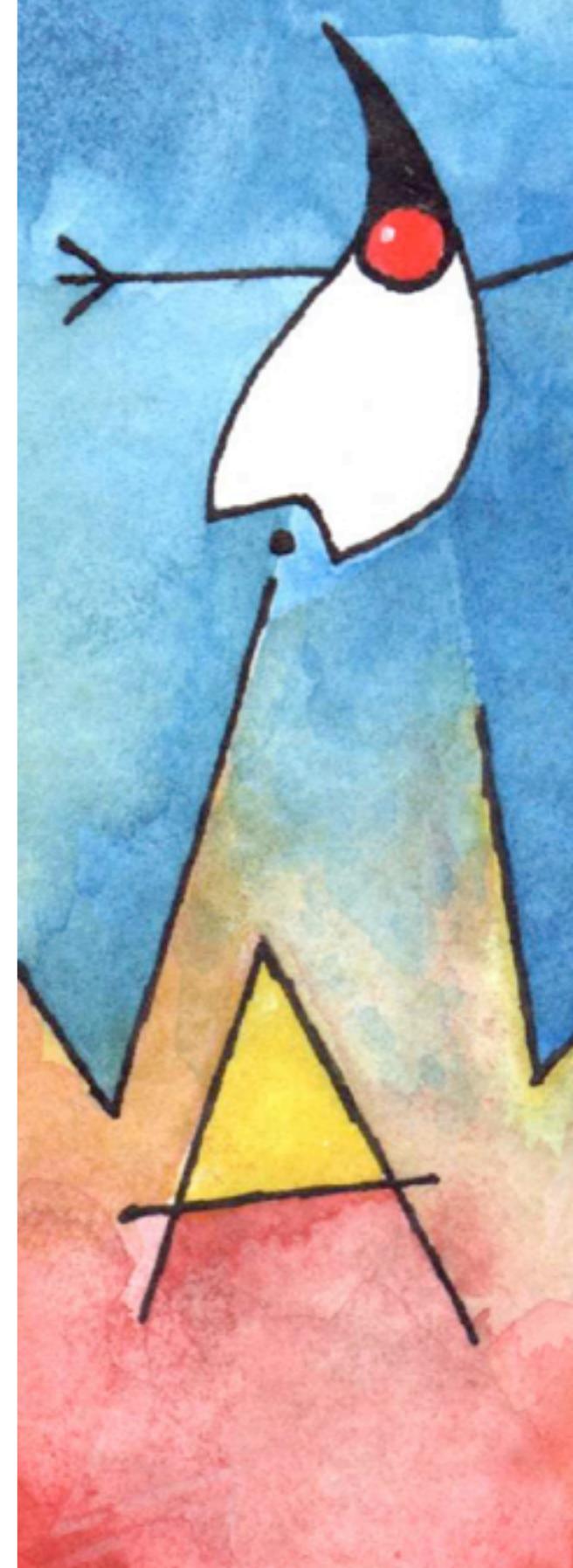
```
Exp → "new" "int" "[" Exp "]"
      → Exp "[" Exp "]"
      → Exp "." "length"
      → "new" Id "(" ")"
```



MiniJava Expressions

```
new int[42]
x[42]
x.length
new Class()
this
```

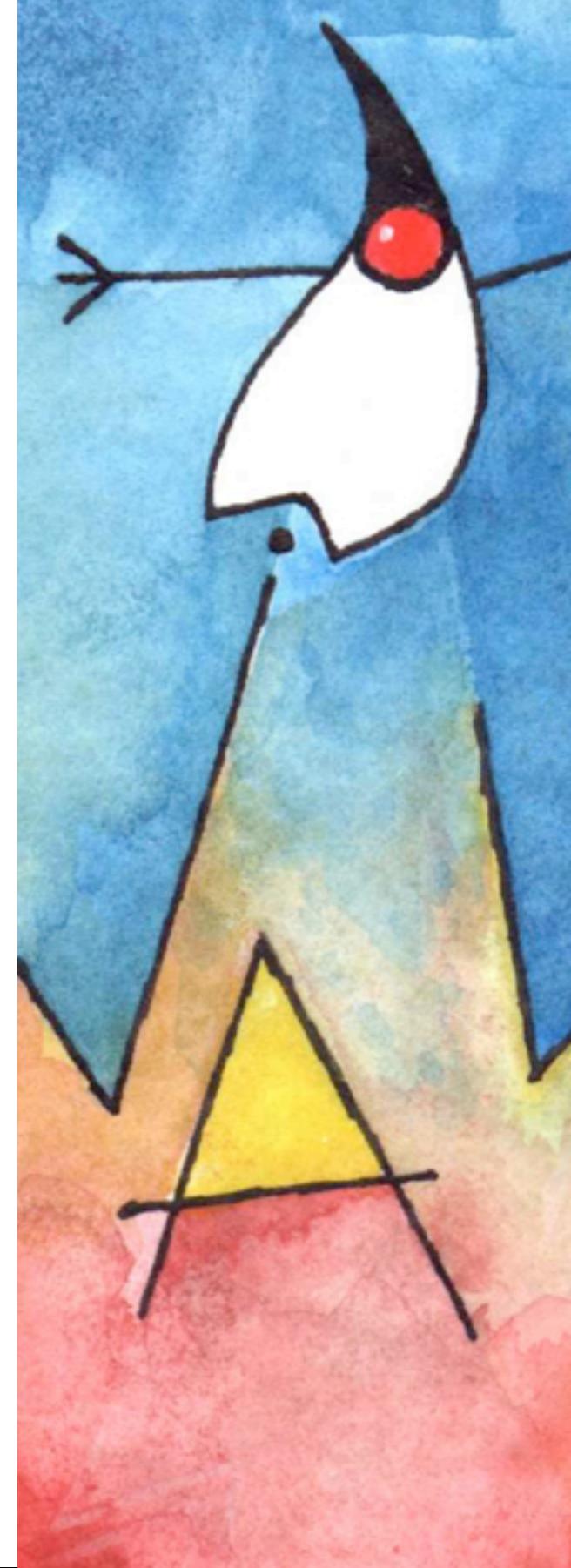
```
Exp → "new" "int" "[" Exp "]"
      → Exp "[" Exp "]"
      → Exp "." "length"
      → "new" Id "(" ")"
      → "this"
```



MiniJava Expressions

```
new int[42]
x[42]
x.length
new Class()
this
x.m(y, z)
```

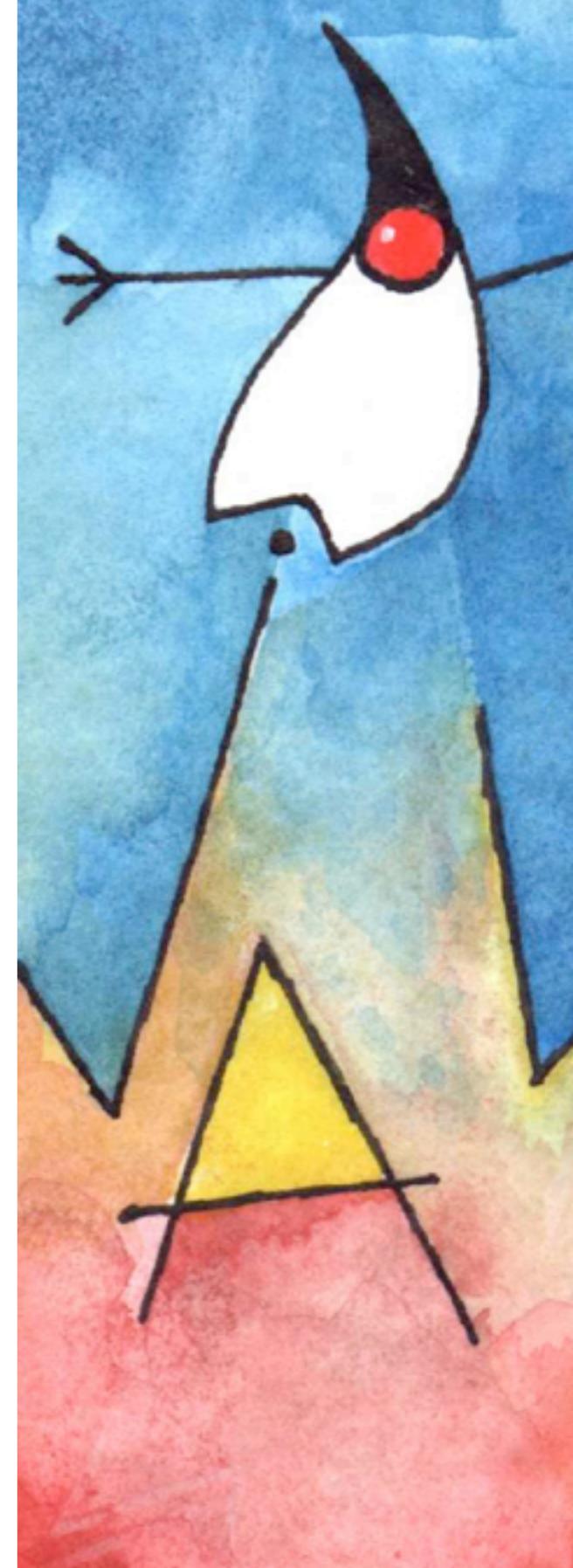
```
Exp → "new" "int" "[" Exp "]"
      → Exp "[" Exp "]"
      → Exp "." "length"
      → "new" Id "(" ")"
      → "this"
      → Exp "." Id "(" ExpList ")"
```



MiniJava Expressions

```
new int[42]
x[42]
x.length
new Class()
this
x.m(y, z)
```

```
ExpList → Exp ExpRest*
          →
ExpRest → "," Exp
```



MiniJava Statements

```
x = y ;
```

```
Statement → Id "=" Exp ";"
```

MiniJava Statements

```
x = y ;  
x[z] = y ;
```

```
Statement → Id "=" Exp ";"  
          → Id "[" Exp "]"" "=" Exp ";"
```

MiniJava Statements

```
x = y ;  
x[z] = y ;  
if (b) x = y ; else y = x ;
```

```
Statement → Id "=" Exp ";"  
          → Id "[" Exp "]""=" Exp ";"  
          → "if" "(" Exp ")" Statement  
            "else" Statement
```

MiniJava Statements

```
x = y ;  
x[z] = y ;  
if (b) x = y ; else y = x ;  
while (x < 42) x = x+1 ;
```

Statement → Id "=" Exp ";"
→ Id "[" Exp "]""=" Exp ";"
→ "if" "(" Exp ")" Statement
"else" Statement
→ "while" "(" Exp ")" Statement

MiniJava Statements

```
x = y ;  
x[z] = y ;  
if (b) x = y ; else y = x ;  
while (x < 42) x = x+1 ;  
System.out.println(x) ;
```

Statement → Id "=" Exp ";"
→ Id "[" Exp "]""=" Exp ";"
→ "if" "(" Exp ")" Statement
 "else" Statement
→ "while" "(" Exp ")" Statement
→ "System.out.println" "(" Exp ")" ";"

MiniJava Statements

```
x = y ;  
x[z] = y ;  
if (b) x = y ; else y = x ;  
while (x < 42) x = x+1 ;  
System.out.println(x) ;  
{ x = y ; y = z ; }
```

Statement → Id "=" Exp ";"
→ Id "[" Exp "]""=" Exp ";"
→ "if" "(" Exp ")" Statement
 "else" Statement
→ "while" "(" Exp ")" Statement
→ "System.out.println" "(" Exp ")" ";"
→ "{" Statement* "}"

MiniJava Class Declarations

```
class Main {  
    public static void main(String[] args) {  
        System.out.println(new Fac().fac(10));  
    }  
}  
  
class Fac { ... }
```

Program → MainClass ClassDecl*

MainClass → "class" Id "{" "public" "static" "void"
 "main" "(" "String" "[" "]" Id ")"
 "{" Statement "}" "}"

ClassDecl → "class" Id "{" VarDecl* MethodDecl* "}"
 → "class" Id "extends" Id
 "{" VarDecl* MethodDecl* "}"

MiniJava Declarations

```
int x ;
public int id(int p) {int r ; r = p ; return r ; }
int
boolean
int[]
Class
```

```
VarDecl      → Type Id ";"  
MethodDecl   → "public" Type Id "(" FormalList ")" "{"  
                           VarDecl* Statement* "return" Exp ";" "}"  
Type          → "int"  
                  → "boolean"  
                  → "int" "[" "]"  
                  → Id
```

MiniJava Declarations

```
int x ;
public int id(int p) {int r ; r = p ; return r ; }
int
boolean
int[]
Class
```

FormalList \rightarrow Type Id FormalRest*
 \rightarrow
FormalRest \rightarrow "," Type Id

VI

summary

Summary

lessons learned

Summary lessons learned

What is a formal language?

- set of strings over a vocabulary

Summary lessons learned

What is a formal language?

- set of strings over a vocabulary

What is a formal grammar?

- quadruple of sets of nonterminal and terminal symbols, a set of productions, and a start symbol

Summary lessons learned

What is a formal language?

- set of strings over a vocabulary

What is a formal grammar?

- quadruple of sets of nonterminal and terminal symbols, a set of productions, and a start symbol

What is a context-free grammar and how can you read it?

- formal grammar with a single nonterminal symbol at left-hand sides of production rules
- productive form, reductive form, tree construction rules

Summary lessons learned

What is a formal language?

- set of strings over a vocabulary

What is a formal grammar?

- quadruple of sets of nonterminal and terminal symbols, a set of productions, and a start symbol

What is a context-free grammar and how can you read it?

- formal grammar with a single nonterminal symbol at left-hand sides of production rules
- productive form, reductive form, tree construction rules

What is the syntax of MiniJava programs?

Literature

learn more

Literature

[learn more](#)

formal languages

Noam Chomsky: Three models for the description of language. 1956

Noam Chomsky: Syntactic Structures. 1957

J. E. Hopcroft, R. Motwani, J. D. Ullman: Introduction to Automata Theory, Languages, and Computation. 2006

Literature

[learn more](#)

formal languages

Noam Chomsky: Three models for the description of language. 1956

Noam Chomsky: Syntactic Structures. 1957

J. E. Hopcroft, R. Motwani, J. D. Ullman: Introduction to Automata Theory, Languages, and Computation. 2006

MiniJava

Andrew W. Appel, Jens Palsberg: Modern Compiler Implementation in Java, 2nd edition. 2002

Outlook

coming next

declarative syntax definition

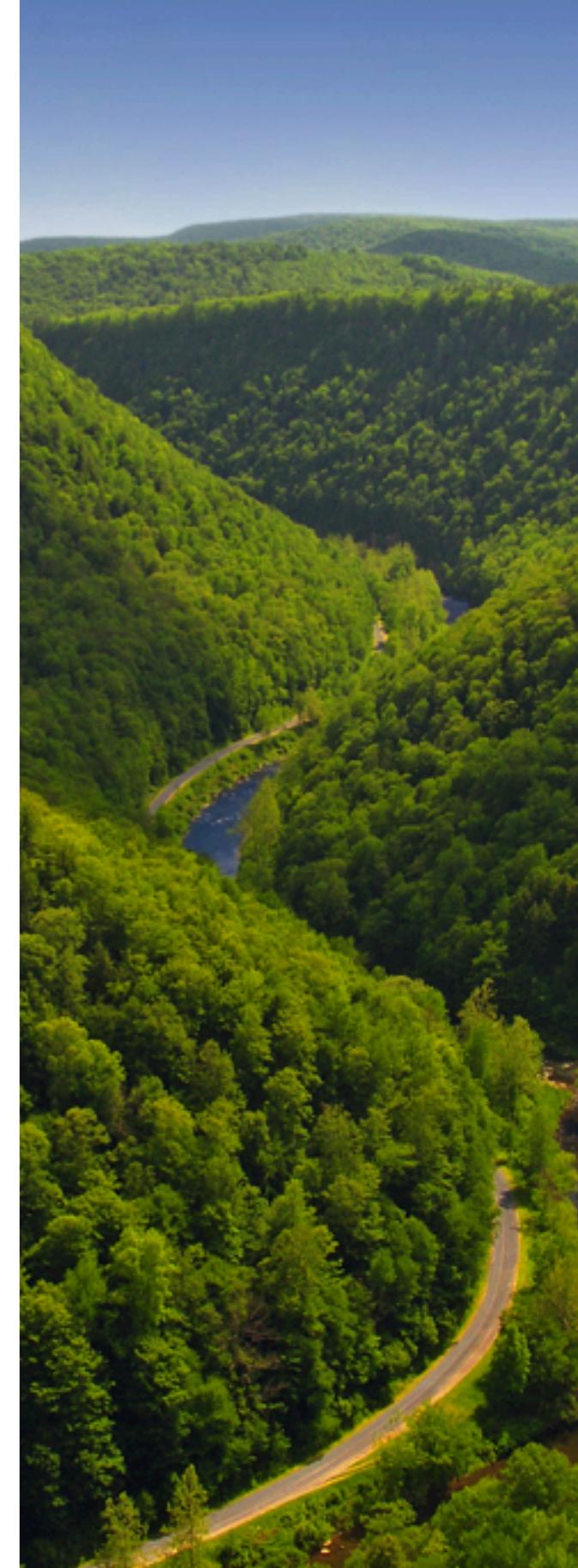
- Lecture 2: SDF and ATerms
- Lecture 3: Pretty Printing

declarative semantics definition

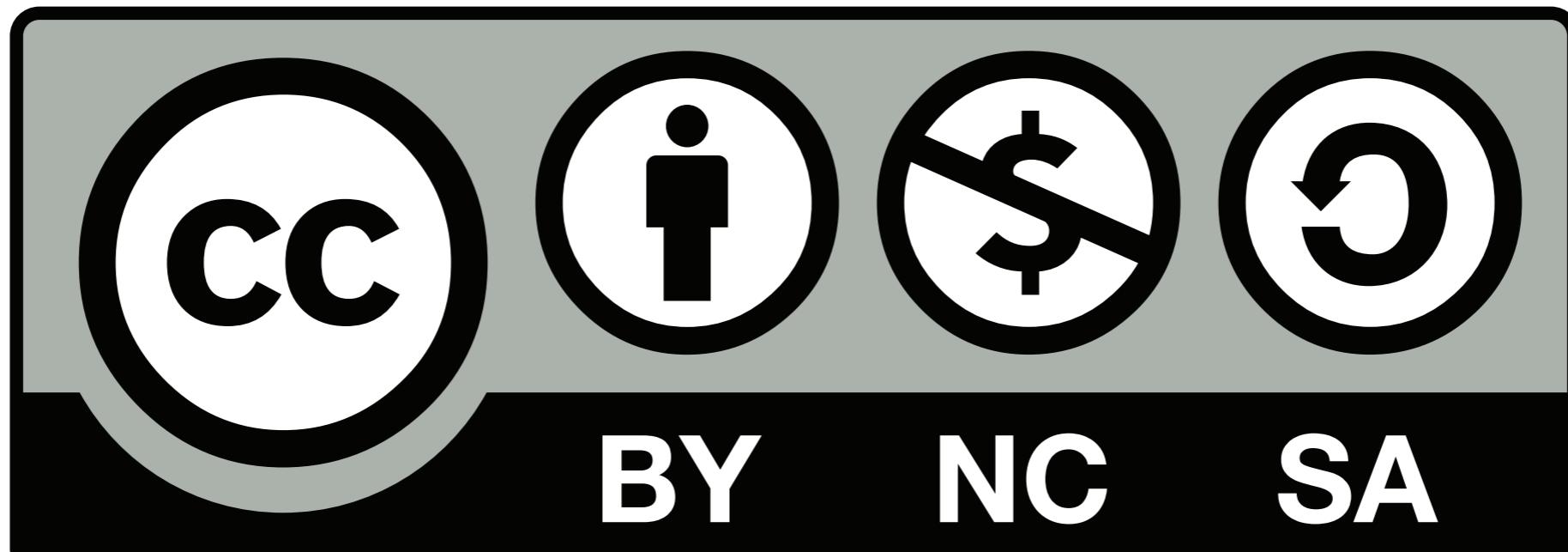
- Lecture 4: Term Rewriting
- Lecture 5: Static Analysis and Error Checking
- Lecture 6: Code Generation

Lab Sep 13

- install minimal MiniJava project
- write positive and negative test cases



copyrights



Pictures copyrights

Slide 1:

The Pine, Saint Tropez by Paul Signac, public domain

Slide 10:

Writing by Caitlin Regan, some rights reserved

Slide 11:

Latin Grammar by Anthony Nelzin, some rights reserved

Slides 13, 14, 20:

Noam Chomsky by Fellowsisters, some rights reserved

Slide 15, 18:

Tiger by Bernard Landgraf, some rights reserved

Slide 31:

Coffee Primo by Dominica Williamson, some rights reserved

Slides 32-36:

Italian Java book cover by unknown artist, some rights reserved

Slide 44:

Pine Creek by Nicholas, some rights reserved