**Goals of today’s lecture**

- Read and understand the syntax description for Eiffel
- Write simple syntax descriptions

**An example (1) - Conditional**

```eiffel
fare (half, first: BOOLEAN; base_price: REAL): REAL is
  -- Ticket price
  -- Base_price: 2nd class, no halbtax
  do
    Result := base_price
    if half then
      Result := Result / 2
    end
    if first then
      Result := Result * 2
    end
  end
```

**An example (2) - Conditional**

```eiffel
fare (half, first: BOOLEAN; base_price: REAL): REAL is
  -- Ticket price
  -- Base_price: 2nd class, no halbtax
  do
    Result := base_price
    if not half and first then
      Result := base_price / 2
    elsif half and not first then
      Result := base_price * 2
    else
      Result := base_price
    end
  end
```

**An example (3) – Nested conditional**

```eiffel
ga_price (first: BOOLEAN; age: INTEGER): REAL is
  -- Price for GA depending on age
  require
    old_enough: age >= 16
  do
    Result := 3450.0
    if age <= 25 then
      Result = 3450.0
    elseif age >= 64 then
      Result = 3550.0
    else
      Result = 4700.0
    end
  end
```

**Intro. to Programming, lecture 8: Describing the syntax**
Syntax: Conditional

A conditional instruction consists, in order, of:
An "If part", of the form if condition.
A "Then part" of the form then compound.
Zero or more "Else if parts", each of the form elseif condition then compound.
Zero or one "Else part" of the form else compound
The keyword end.

Here each condition is a boolean expression, and each compound is a compound instruction.

The power of human mind

I cdnoult blèveew taht I cluod aulacity uesdnatnrd waht I was rdgnieg. The Paomnnehal Pweor of the Hmuan Mnid Aaccðñig to a rscheearch at Cmabrigde Umervtisy, is deosn’t mttaer in wtah oreder the ltteers in a wrod ar. the olny iprmoatnt thng is tghat the frist nd lsit ltteer be in the rghit pclae. The rst can b a taa lnes nd you cn stll rd it woutht porbelm. Thts b cuseae the huamn mnid deos not rd every ltteer by istlef, but ths wrod s a wlohe. Ptrety Amanig Huh?

Why describe the syntax formally?

We know syntax descriptions for human languages:
> e.g. grammar for German, French, ...
> Expressed in natural language
> Good enough for human use
> Ambiguous, like human languages themselves

Why describe the syntax formally?

Programming languages need better descriptions:
> More precise: must tell us unambiguously whether given program text is legal or not
> Use formalism similar to mathematics
> Can be fed into compilers for automatic processing of programs

Why describe the syntax formally?

Compilers use algorithms to
Determine if input is correct program text (parser)
Analyze program text to extract specimens
Translate program text to machine instructions

Compilers need strict formal definition of programming language

Formal Description of Syntax

Use formal language to describe programming languages.

Languages used to describe other languages are called Meta-Languages

Meta-Language used to describe Eiffel:
BNF-E (Variant of the Backus-Naur-Form, BNF)
History

1954 FORTRAN: First widely recognized programming language (developed by John Backus et al.)
1958 ALGOL 58: Joint work of European and American groups
1960 ALGOL 60: Preparation showed a need for a formal description → John Backus (member of ALGOL team) proposed Backus-Normal-Form (BNF)
1964: Donald Knuth suggested acknowledging Peter Naur for his contribution → Backus-Naur-Form
Many variants since then, e.g. graphical variant by Niklaus Wirth

Formal description of a language

BNF lets us describe syntactical properties of a language
Remember: Description of a programming language also includes lexical and semantic properties → other tools

Formal Description of Syntax

A language is a set of phrases
A phrase is a finite sequence of tokens from a certain “vocabulary”
Not every possible sequence is a phrase of the language
A grammar specifies which sequences are phrases and which are not
BNF is used to define a grammar for a programming language

Example of phrases

```
class PERSON
feature
  age: INTEGER
  -- Age
end
end PERSON

feature
  age: INTEGER
  -- Age
end
```

Grammar

Definition

A Grammar for a language is a finite set of rules for producing phrases, such that:

1. Any sequence obtained by a finite number of applications of rules from the grammar is a phrase of the language.
2. Any phrase of the language can be obtained by a finite number of applications of rules from the grammar.

Elements of a grammar: Terminals

Terminals

Tokens of the language that are not defined by a production of the grammar.
E.g. keywords from Eiffel such as `if`, `then`, `end` or symbols such as the semicolon `;` or the assignment `:=`
Elements of a grammar: Nonterminals

Nonterminals
Names of syntactical structures or substructures used to build phrases.

Elements of a grammar: Productions

Productions
Rules that define nonterminals of the grammar using a combination of terminals and (other) nonterminals.

An example production

Graphical representation:

Conditional:

```
if
  condition
then
  instruction
else
  instruction
end
```

BNF Elements: Concatenation

Graphical representation:

```
BNF: A B
```

Meaning: A followed by B

BNF Elements: Optional

Graphical representation:

```
BNF: [ A ]
```

Meaning: A or nothing

BNF Elements: Choice

Graphical representation:

```
BNF: A | B
```

Meaning: either A or B
**BNF Elements: Repetition**

- **Graphical representation:**
  - A

- **BNF:** \( \{ A \}^* \)

- **Meaning:** sequence of zero or more \( A \)

**BNF Elements: Repetition, once or more**

- **Graphical representation:**
  - A

- **BNF:** \( \{ A \}^+ \)

- **Meaning:** sequence of one or more \( A \)

---

**BNF elements: Overview**

- **Concatenation:** \( A B \)

- **Optional:** \([ A ]\)

- **Choice:** \( A \mid B \)

- **Repetition (zero or more):** \( \{ A \}^* \)

- **Repetition (at least once):** \( \{ A \}^+ \)

---

**A simple example**

- **Example phrases:**
  - .76
  - -.76
  - 1.56
  - 12.845
  - -1.34
  - 13.0

---

**BNF Elements Combined**

- **Conditional:** \( if \) \( then \) \( instruction \) \( else\) \( instruction \) \( end \)

- **written in BNF:**
  - \( if \) \( condition \) \( then \) \( instruction \) \( else\) \( instruction \) \( end \)

---

**BNF: Conditional with ** **elseif**

- **Conditional:** \( if \) \( Then\_part\_list\) \( [elseif] \) \( Then\_part\) \( \) \( end \)

- **Then\_part:** \( Then\_part\) \( (elseif) \) \( Then\_part\) \( \) \( \) \( * \)

- **Then\_part:** \( Boolean\_expression\) \( then\) \( Compound\)

- **Else\_part:** \( else\) \( Compound\)
Different Grammar for Conditional

Conditional \( \rightarrow \) If_part Then_part Else_list end

If_part \( \rightarrow \) if Boolean_expression

Then_part \( \rightarrow \) then Compound

Else_list \( \rightarrow \) { Elseif_part }*[ else Compound ]

Elseif_part \( \rightarrow \) elseif Boolean_expression Then_part

Simple BNF example

Sentence \( \rightarrow \) I [ don't ] Verb Names Quant
Names \( \rightarrow \) Name (and Name)*
Name \( \rightarrow \) tomatoes | shoes | books | football
Verb \( \rightarrow \) like | hate
Quant \( \rightarrow \) a lot | a little

Which of the following phrases are correct?
- I like tomatoes and football
- I don't like tomatoes a little
- I hate football a lot
- I like shoes and tomatoes a little
- I don't hate tomatoes, football and books a lot

Rewrite the BNF to include the incorrect phrases

Simple BNF example (Solution)

Which of the following phrases are correct?
- I like tomatoes and football
- I don't like tomatoes a little
- I hate football a lot
- I like shoes and tomatoes a little
- I don't hate tomatoes, football and books a lot

Rewrite the BNF to include the incorrect phrases

Sentence \( \rightarrow \) I [ don't ] Verb Names [ Quant ]
Names \( \rightarrow \) Name (and Name)*
Name \( \rightarrow \) tomatoes | shoes | books | football
Verb \( \rightarrow \) like | hate
Quant \( \rightarrow \) a lot | a little

BNF-E

Used in official description of Eiffel.
Every Production is one of Concatenation, Choice or Repetition

Concatenation

A \( \rightarrow \) B C [ D ]

Choice

A \( \rightarrow \) B | C | D

Repetition

A \( \rightarrow \) { B terminal ... }*[ also with + ]

BNF-E Rules

- Every nonterminal must appear on the left-hand side of exactly one production, called its defining production
- Every production must be of one kind: Concatenation, Choice or Repetition

Conditional with elseif (BNF)

Conditional \( \rightarrow \) if Then_part_list [ Else_part ] end

Then_part_list \( \rightarrow \) Then_part { elseif Then_part }*

Then_part \( \rightarrow \) Boolean_expression then Compound

Else_part \( \rightarrow \) else Compound
BNF-E: Conditional

```
Conditional \( \rightarrow [ \text{Then} \_ \text{part} \ [ \text{Else} \_ \text{part} \ ] \ \text{end} ] \)
Then_part_list \( \rightarrow [ \text{Then} \_ \text{part} \ [ \text{elseif} \ \ ... ] \ ]^+ \)
Then_part \( \rightarrow \text{Boolean expression} \ [ \text{then} \ \text{Compound} ] \)
Else_part \( \rightarrow [ \text{else} \ \text{Compound} ] \)
```

Recursive grammars

```
Conditional \( \rightarrow [ \text{Then} \_ \text{part} \ [ \text{Else} \_ \text{part} \ ] \ \text{end} ] \)
Then_part_list \( \rightarrow [ \text{Then} \_ \text{part} \ [ \text{elseif} \ \ ... ] \ ]^+ \)
Then_part \( \rightarrow \text{Boolean expression} \ [ \text{then} \ \text{Compound} ] \)
Else_part \( \rightarrow [ \text{else} \ \text{Compound} ] \)
```

Recursive grammars

```
Constructs may be nested

Express this in BNF with recursive grammars

Recursion: circular dependency of productions
```

Recursive grammars

```
Conditionals can be nested within conditionals:

Else_part \( \rightarrow [ \text{else} \ \text{Compound} ] \)

Compound \( \rightarrow [ \text{Instruction} \ [ \ ... ]^* \]
Instruction \( \rightarrow \text{Conditional} \ [ \text{Loop} \ [ \text{Call} \ [ \ ... ] \]
```

Recursive grammars

```
Production name can be used in its own definition

Definition of Then_part_list with repetition:
Then_part_list \( \rightarrow [ \text{Then} \_ \text{part} \ [ \text{elseif} \ \ ... ] \ ]^* \)
Recursive definition of Then_part_list:
Then_part_list \( \rightarrow [ \text{Then} \_ \text{part} \ [ \text{elseif} \ \text{Then} \_ \text{part} \ ] \)
```

Recursive grammars

```
BNF for simple arithmetic expressions

```
Expr \( \rightarrow \text{Term} \ ( \text{Add} \_ \text{op} \ \text{Term} )^* \)
Term \( \rightarrow \text{Factor} \ ( \text{Mult} \_ \text{op} \ \text{Factor} )^* \)
Factor \( \rightarrow \text{Number} \ | \ \text{Variable} \ | \ \text{Nested} \)
Nested \( \rightarrow ( \text{Expr} ) \)
Add_op \( \rightarrow \text{+} \ | \text{-} \)
Mult_op \( \rightarrow \text{*} \ | \text{/} \)
```

Conditional

```
if a = b then
 a := a - 1
 b := b + 1
elsif a > b then
 a := a + 1
else
 b := b + 1
end
```

BNF for simple arithmetic expressions

```
Is this a recursive grammar?
How would the same grammar in BNF-E look like?
Which of the following phrases are correct?
a
 a + b
 -a + b
a * 7 + b
7 / (3 * 12) - 7
(3 * 7)
(5 + a (7 + b))
```

BNF for simple arithmetic expressions

```
Expr \( \rightarrow \text{Term} \ ( \text{Add} \_ \text{op} \ \text{Term} )^* \)
Term \( \rightarrow \text{Factor} \ ( \text{Mult} \_ \text{op} \ \text{Factor} )^* \)
Factor \( \rightarrow \text{Number} \ | \ \text{Variable} \ | \ \text{Nested} \)
Nested \( \rightarrow ( \text{Expr} ) \)
Add_op \( \rightarrow \text{+} \ | \text{-} \)
Mult_op \( \rightarrow \text{*} \ | \text{/} \)
```

Recursive grammars

```
Branches may be nested

Express this in BNF with recursive grammars

Recursion: circular dependency of productions
```

Recursive grammars

```
Recursive grammars

```
Production name can be used in its own definition

Definition of Then_part_list with repetition:
Then_part_list \( \rightarrow [ \text{Then} \_ \text{part} \ [ \text{elseif} \ \ ... ] \ ]^* \)
Recursive definition of Then_part_list:
Then_part_list \( \rightarrow [ \text{Then} \_ \text{part} \ [ \text{elseif} \ \text{Then} \_ \text{part} \ ] \)
```

Recursive grammars

```
```

BNF for simple arithmetic expressions

```
Expr \( \rightarrow \text{Term} \ ( \text{Add} \_ \text{op} \ \text{Term} )^* \)
Term \( \rightarrow \text{Factor} \ ( \text{Mult} \_ \text{op} \ \text{Factor} )^* \)
Factor \( \rightarrow \text{Number} \ | \ \text{Variable} \ | \ \text{Nested} \)
Nested \( \rightarrow ( \text{Expr} ) \)
Add_op \( \rightarrow \text{+} \ | \text{-} \)
Mult_op \( \rightarrow \text{*} \ | \text{/} \)
```

Is this a recursive grammar?
How would the same grammar in BNF-E look like?
Which of the following phrases are correct?
a
 a + b
 -a + b
a * 7 + b
7 / (3 * 12) - 7
(3 * 7)
(5 + a (7 + b))

BNF for simple arithmetic expressions

```
Expr \( \rightarrow \text{Term} \ ( \text{Add} \_ \text{op} \ \text{Term} )^* \)
Term \( \rightarrow \text{Factor} \ ( \text{Mult} \_ \text{op} \ \text{Factor} )^* \)
Factor \( \rightarrow \text{Number} \ | \ \text{Variable} \ | \ \text{Nested} \)
Nested \( \rightarrow ( \text{Expr} ) \)
Add_op \( \rightarrow \text{+} \ | \text{-} \)
Mult_op \( \rightarrow \text{*} \ | \text{/} \)
```

Is this a recursive grammar?
How would the same grammar in BNF-E look like?
Which of the following phrases are correct?
a
 a + b
 -a + b
a * 7 + b
7 / (3 * 12) - 7
(3 * 7)
(5 + a (7 + b))

BNF for simple arithmetic expressions

```
Is this a recursive grammar?
How would the same grammar in BNF-E look like?
Which of the following phrases are correct?
a
 a + b
 -a + b
a * 7 + b
7 / (3 * 12) - 7
(3 * 7)
(5 + a (7 + b))
```
BNF for simple arithmetic expressions
(Solution)

Is this a recursive grammar? Yes (see Nested)
How would the same grammar in BNF-E look like?
(see yellow box below)
Which of the following phrases are correct?
a + b
\(-a + b\)
a \(* b\)
\(7/ (3 * 12)\)
\((3 * 7)\)
\((5 + a (7 * b))\)

Guidelines for Grammars

Keep productions short.
easier to read
better assessment of language size

Guidelines for Grammars

Treat lexical constructs like terminals
Identifiers
Constant values

Guidelines for Grammars

Use unambiguous productions.
Applicable production can be found by looking at one lexical element at a time

Guidelines for Grammars

Writing a Parser

One feature per Production
Concatenation:
Sequence of feature calls for Nonterminals, checks for Terminals
Choice:
Conditional with Compound per alternative
Repetition:
Loop

Guidelines for Grammars

Automatic generation of abstract syntax tree for phrase
Based on BNF-E
One class per production
Classes inherit from predefined classes AGGREGATE,
CHOICE, REPETITION, TERMINAL
Feature production defines Production
Writing a Parser: Tools

Yoct
Translates BNF-E to EiffelParse classes

Yacc / Bison
Translates BNF to C parser

BNF similar syntax descriptions

Unix/Linux: Synopsis of commands
SYNOPSIS

What we have seen

A way to describe syntax: BNF
Three variants: BNF, BNF-E, graphical
A glimpse into recursion
A preview of tomorrow: conditional

End of lecture 8