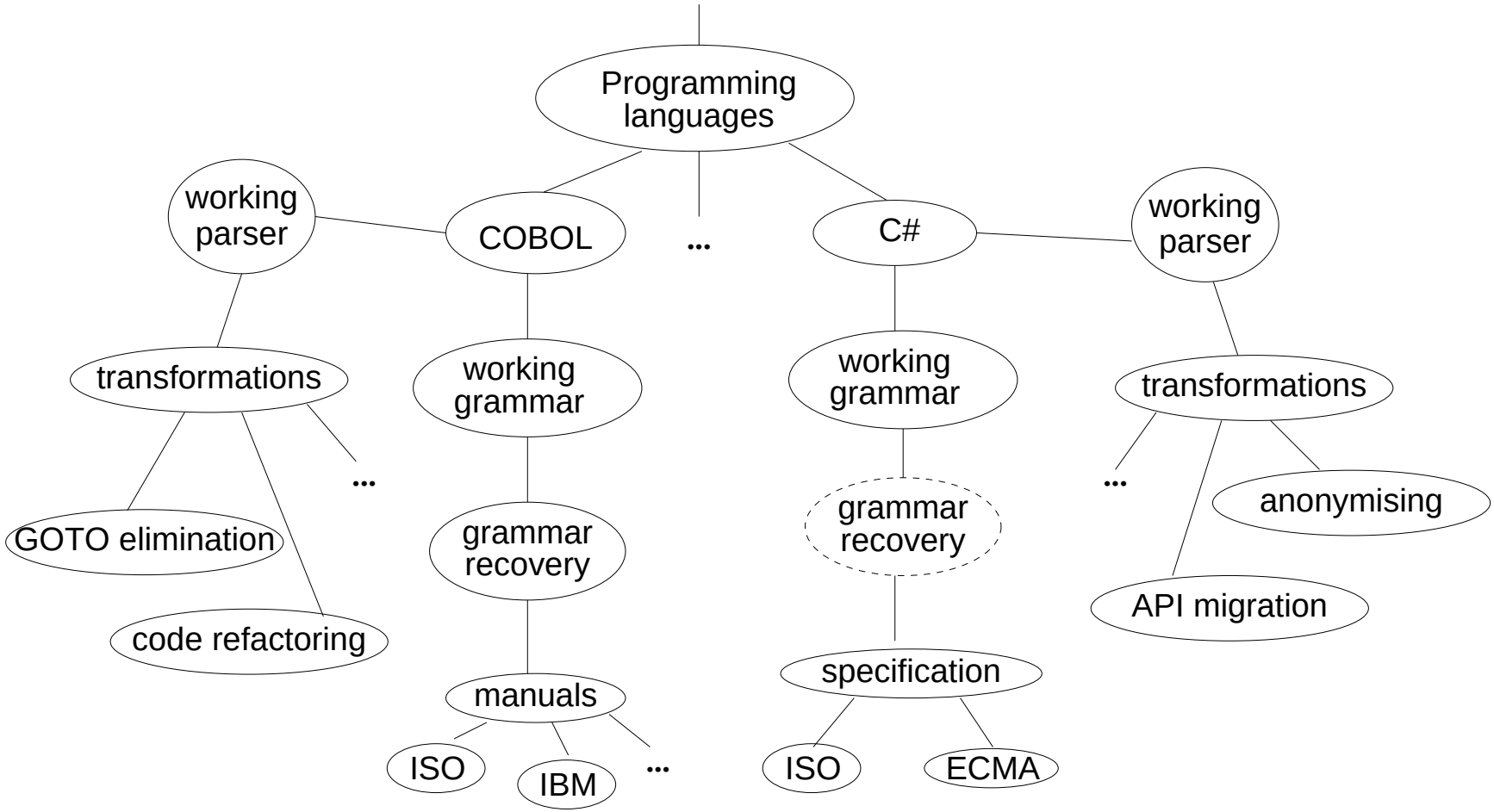
Correct C# Grammar
too Sharp for ISO

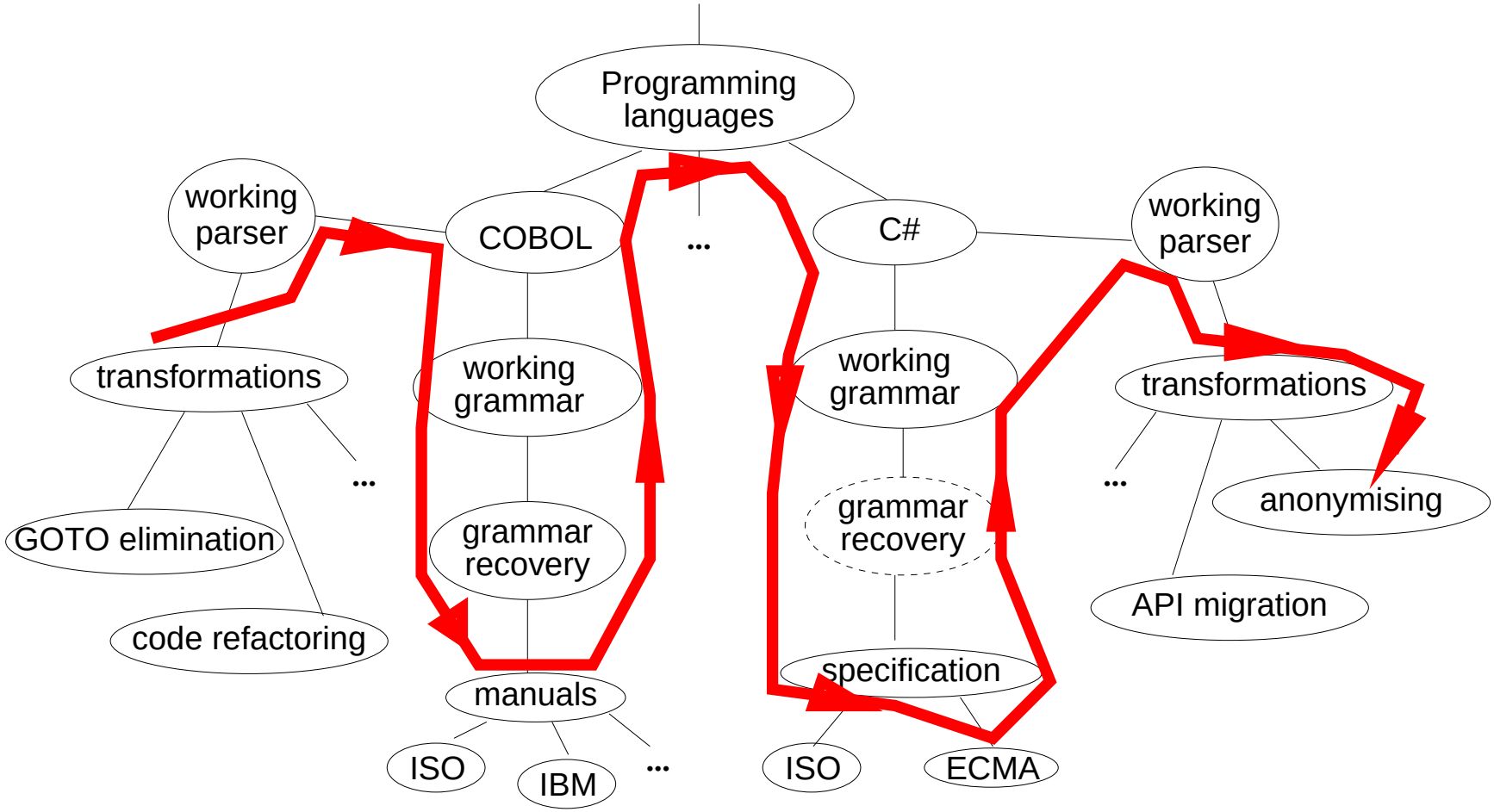
Drs.ir. Vadim V. Zaytsev,
vrije Universiteit *amsterdam*, The Netherlands,
vadim@cs.vu.nl

July 6, 2005

Parse tree of the research



Traversal for this presentation



COBOL

- The most used programming language
- Large systems (10000000+ lines of code)
- Standardised by ISO, dialects by vendors
- **Legacy**: systems not understood, hardware outdated, manuals incomplete
- **Experience**: *IBM VS COBOL II Reference Summary* transformed; cooperation with ISO

“New stuff”

- Mainstream yet new language:
  - Visual Basic?
  - Java?
  - C#?
- Made by *the* big corporation, approved by ISO and ECMA International

C# specification

- Three diffently formatted versions:
  - ECMA 334
  - ISO/IEC 23270:2003 (free)
  - Microsoft-ECMA
- 500 pages of English (conditionally normative) text
- Formal (BNF-like) appendix with a grammar (informative text)

Example: not quite BNF

```
letter-character::
  A Unicode character of classes Lu, Ll, Lt, Lm, Lo, or Nl
  A unicode-character-escape-sequence representing a character of
    classes Lu, Ll, Lt, Lm, Lo, or Nl

decimal-digit:: one of
  0 1 2 3 4 5 6 7 8 9

integer-type-suffix:: one of
  U u L l UL Ul uL ul LU Lu lU lu
```

- `unicode-character-escape-sequence` is a non-terminal
- Unicode classes are defined elsewhere

Example: duplicates

```
A.2.4 Expressions
...
constant-expression:
  expression
boolean-expression:
  expression

A.2.5 Statements
...
if-statement:
  "if" "(" boolean-expression ")" embedded-statement
  "if" "(" boolean-expression ")" embedded-statement
       "else" embedded-statement
boolean-expression:
  expression
```

Example: semantics & ambiguities

```
A.2.1 Basic concepts
namespace-name:
  namespace-or-type-name
type-name:
  namespace-or-type-name
namespace-or-type-name:
  identifier
  namespace-or-type-name "." identifier

A.2.2 Types
type: value-type | reference-type
value-type: struct-type | enum-type
struct-type: type-name | simple-type
enum-type: type-name
reference-type:
  class-type | interface-type | array-type | delegate-type
class-type: type-name | "object" | "string"
interface-type: type-name
delegate-type: type-name
```

Example: needless complications

```
block:
  "{" statement-list? "}"

statement-list:
  statement
  statement-list statement
```

- `block:: "{" statement* "}"`

- More straightforward, less non-terminals, left recursion is not preferred.

Example: obvious ambiguities

```
A.2.6 Classes
static-constructor-modifiers:
  "extern"? "static"
  "static" "extern"?

25.1 Unsafe constructs
static-constructor-modifiers:
  "extern"? "unsafe"? "static"
  "unsafe"? "extern"? "static"
  "extern"? "static" "unsafe"?
  "unsafe"? "static" "extern"?
  "static" "extern"? "unsafe"?
  "static" "unsafe"? "extern"?
```

Example: `yaccification`

```
expression:
  conditional-expression
  assignment
conditional-expression:
  conditional-or-expression
  conditional-or-expression "?" expression ":" expression
conditional-or-expression:
  conditional-and-expression
  conditional-or-expression "||" conditional-and-expression
conditional-and-expression:
  inclusive-or-expression
  conditional-and-expression "&&" inclusive-or-expression
inclusive-or-expression:
  exclusive-or-expression
  inclusive-or-expression "|" exclusive-or-expression
exclusive-or-expression:
  and-expression
  exclusive-or-expression "^" and-expression
...
```

Example: `yacc`ified parse tree

Example: yaccified parse tree

Example: redundancy

```
method-body:
  block
  ";"
accessor-body:
  block
  ";"
operator-body:
  block
  ";"
constructor-body:
  block
  ";"
static-constructor-body:
  block
  ";"
destructor-body:
  block
  ";"
```

Example: inconsistency

§22.1 Delegate declarations, page 297
(lines 15–16 in Msft version)

```
delegate-declaration:
  attributes? delegate-modifiers? "delegate" return-type identifier
     "(" formal-parameter-list? ")" ";"
```

Appendix A.2.11 Delegates, page 357
(lines 34–35 in Msft version)

```
delegate-declaration:
  attributes? delegate-modifiers? "delegate" type identifier
     "(" formal-parameter-list? ")" ";"
```

- (No) `void` allowed for delegates

Other examples

- §25 Unsafe code (pages 317–334)
- Appendix A.3 Grammar extensions for unsafe code (pages 359–360)
- Inane ambiguities: `+ +x` vs `++x` (and `-+x`)

*G*rammar *D*eployment *K*it

- LLL: EBNF-based grammar format
- Grammar transformations:
  - `%rename sort %to sort`
  - `%redefine rule %to rule`
  - `%include rule` and `%exclude rule`
  - `%eliminate sort`
  - `%introduce rule`

ASF+SDF Meta-Environment

- SDF — Syntax Definition Formalism
  - Parsing technology: SGLR (Scannerless Generalised Left-to-right with Rightmost derivation)
  - All non-circular context-free grammars allowed, modular
- ASF — Algebraic Specification Formalism
  - Rewriting rules
  - Traversal functions
  - It compiles to C
- Meta-Environment: the connecting GUI
- And the infrastructure that ties it all together

Grammar transformations

- Grammar transformations are a bit different from source code transformations
- Grammar $\longrightarrow$ specification or documentation
- Grammar $\longrightarrow$ dialect or implementation
- Grammar $\longrightarrow$ next version of a grammar
- http://www.cs.vu.nl/grammars/browsable/

Conclusion: methods

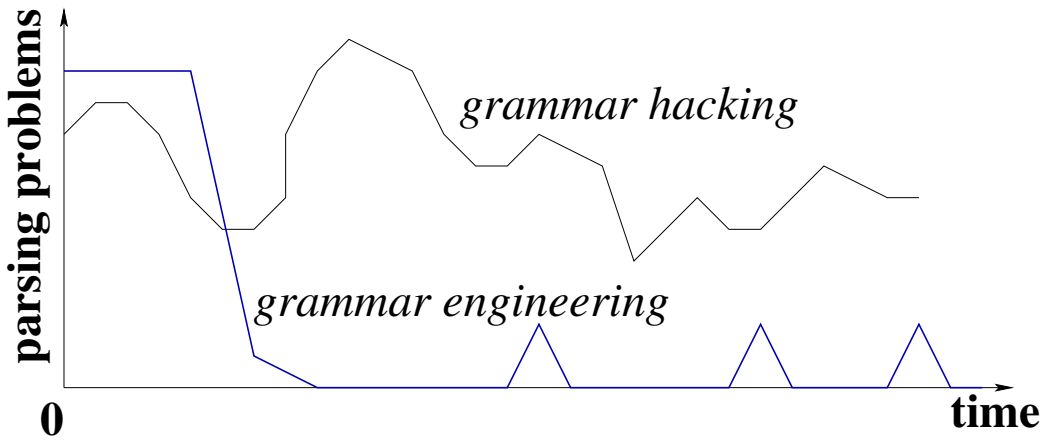
- **Grammar recovery** is a technique for extracting a complete grammar out of:
  - an existing programming language's manual
  - **a standardised specification**
  - compiler's source code,
- assessing it, correcting, testing, etc
- PLEX (1998), VS COBOL II (1999–2005), PL/I (1999), Ada 95 (2000), Fortran, C, C#

Conclusion: statements

- Grammar recovery is needed also for new languages
- Specifications should be (but are not) free from:
  - technical details
  - misprints
  - inconsistencies
  - invalid code
- Specification Deployment Kit. . .

Stay tuned!

Grammar engineering



- "Grammar engineering" approach works fine with specs, too.