Grammar Maturity Model

Dr. Vadim Zaytsev, Universiteit van Amsterdam @ GEMOC’14
Grammars in a Broad Sense

- Definitions of languages
- Finite definitions
- of infinite languages
- Focus
  - SE: sets of programs (analytic sense, “parsing”)
  - FL: sets of words (generative sense)
- Also, rewriting systems
Arithmetic Expressions, Boolean Expressions, and Expressions

\[
\begin{align*}
\text{factor} & \equiv \text{number} \lor \text{function} \lor \text{variable} \lor \text{subscr var} \\
& \lor (\text{ar exp}) \lor (\text{factor} \uparrow \text{ar exp} \downarrow) \\
term & \equiv \text{factor} \lor \text{term} \times \text{factor} \lor \text{factor} / \text{term} \\
ar exp & \equiv \text{term} \lor \text{factor} + \text{term} \lor \text{term} - \text{term} \lor \text{factor} + \text{term} \\
\text{ar exp A} & \equiv \text{ar exp} \\
\text{relation} & \equiv \text{or} \lor \text{lesseq} \lor \text{greatereq} \lor \text{rel} \equiv \text{or} \neq \\
\text{rel exp} & \equiv (\text{ar exp} \text{relation} \text{ar exp A}) \\
\text{bool term} & \equiv 0 \lor 1 \lor \text{rel exp} \lor \text{function} \lor \text{variable} \lor \text{subscr var} \lor (\text{bool exp}) \lor \\
& \lor \neg \text{bool term} \\
\text{bool exp} & \equiv \text{bool term} \lor \text{bool exp} \lor \text{bool exp} \lor \text{bool term} \lor \text{bool exp} \lor \text{bool exp} \\
& \lor \text{bool exp} \equiv \text{bool term} \\
\text{exp} & \equiv \text{ar exp} \lor \text{bool exp}
\end{align*}
\]
1.5 Greibach normal form

Problem 5.
Is it true that for every Boolean grammar there exists a Boolean grammar in Greibach normal form that generates the same language?

For a small special case the question has been answered affirmatively:

Theorem 5 (Okhotin, Reitwießner [7]).
For every conjunctive grammar over a unary alphabet there exists and can be effectively constructed a conjunctive grammar in Greibach normal form generating the same language.

One previously given advice must be revoked. It was suggested to try the following language:

The question whether the language \( \{ a^n b^{2n} | n \in \mathbb{N} \} \) can be represented by a Boolean grammar in Greibach normal form might be a good starting point in approaching Problem 5. The answer to this question is likely negative, and a negative solution to the problem can be thus obtained.

Surprisingly, there exists a conjunctive grammar for this language:

Example 3.
The following conjunctive grammar generates the language \( \{ a^n b^{2n} | n \in \mathbb{N} \} \):

\[
S \rightarrow aSB \land AB \mid b \\
A \rightarrow aA \mid \varepsilon \\
B \rightarrow B_1 \mid B_2 \\
B_1 \rightarrow B_1 B_3 \land B_2 B_2 \mid b \\
B_2 \rightarrow B_1 B_1 \land B_2 B_6 \mid bb \\
B_3 \rightarrow B_1 B_2 \land B_6 B_6 \mid bbb \\
B_6 \rightarrow B_1 B_2 \land B_3 B_3
\]

Alexander Okhotin, Nine open problems on conjunctive and Boolean grammars
Grammars in a Broad Sense

- Grammars as
  - contracts
  - protocols
  - schemata
  - domain models
  - format definitions
  - ...

metamodels
Grammar Zoo

- Collect ALL the grammars
- Atlantic, ANTLR, TXL, OMG, ASF+SDF, RELAX, W3C, ...
Information technology -- Programming languages, their environments and system software interfaces -- Programming language COBOL

Abstract

ISO/IEC 1989:2014 specifies the syntax and semantics of COBOL. Its purpose is to promote a high degree of machine independence to permit the use of COBOL on a variety of data processing systems.

ISO/IEC 1989:2014 specifies:
- the form of a compilation group written in COBOL;
- the effect of compiling a compilation group;
- the effect of executing run units;
- the elements of the language for which a conforming implementation is required to supply a definition;
- the elements of the language for which meaning is explicitly undefined;
- the elements of the language that are dependent on the capabilities of the processor.

ISO/IEC 1989:2014 does not specify:
- the means whereby a compilation group written in COBOL is compiled into code executable by a processor;
- the time at which method, function, or program runtime modules are linked or bound to an activating statement, except that runtime binding occurs of necessity when the identification of the appropriate program or method is not known at compile time;
- the time at which parameterized classes and interfaces are expanded;
- the mechanism by which locales are defined and made available on a processor;
Welcome to the official home of

ISO/IEC JTC1/SC22/WG21 - The C++ Standardization Committee

2014-07-10: standards | projects | papers | mailings | internals | meetings | contacts

News 2014-07-09: The deadline for the next mailing is 2014-10-1
News 2014-07-09: The 2014-07-post-Rapperswil mailing is available
News 2014-07-09: The C++ Standard Core Language Issues List (Revision 90) is available (.zip)
News 2014-05-29: The C++ Standard Library Issues List (Revision 88) is available (.zip)
News 2013-05-24: The CD for the new C++ standard is released
News 2013-05-24: New ub reflector and archive, for undeclared behaviour study group.
News 2011-09-11: The new C++ standard - C++11 - is published!

ISO/IEC JTC1/SC22/WG21 is the international standardization working group for the programming language C++.

Published standards and technical reports include:

- ISO/IEC 14882:2011 Programming Language C++ - draft
- ISO/IEC TR 18015:2006 C++ Performance - draft TR

Work on projects and their milestones include:

- ISO/IEC 14882: Programming Language C++ - latest publicly available draft
- ISO/IEC TR 24773: C++ decimal floating point arithmetic extensions - draft
- ISO/IEC 29124: C++ Special Math Functions - draft

Other information:

- Some further information on C++ standardization - isocpp.org
- C++ Standard Core Issues List
- C++ Standard Library Issues List
- WG papers
- WG mailings
- WG21 business plan and convener’s report 2011
- Information on past and future WG meetings
- Contacts: membership, liaisons and related work
- ISO/IEC TR 24772 Information technology -- Programming languages -- Guidance to avoiding vulnerabilities in programming languages through language selection and use
- WG internal information (For members of the C++ standardization committee)

If you want further information, or want to participate, please contact your national member body or one of the contact addresses of the WG.

2014-07-10: standards | projects | papers | mailings | internals | meetings | contacts
Last login: Sun Sep 28 09:19:54 on ttys003

~ zaytsev$ ghci

GHCi, version 7.6.3: http://www.haskell.org/ghc/ :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
Prelude>
NAME
awk - pattern-directed scanning and processing language

SYNOPSIS
awk [ -F fs ] [ -v var=value ] [ 'prog' | -f progfile ] [ file ...

DESCRIPTION
Awk scans each input file for lines that match any of a set of patterns specified literally in prog or in one or more files specified as -f progfile. With each pattern there can be an associated action that will be performed when a
Browsable Business Motivation Model (BMM) Grammar

Grammar extracted by Vadim Zaytsev, see the Grammar Zoo entry for details: dsl/business/motivation/v1.0/atlantic1/extracted
Source used for this grammar: OMG, The Business Motivation Model from the OMG, written from the OMG BMM specification, generated from the CMOF XMI for BMM 1.0, bmmOmg.ecore, January 2010

Summary

- Total 62 production rules with 62 top alternatives and 480 symbols.
- Vocabulary: 122 = 32 nonterminals + 0 terminals + 90 labels + 0 markers.
- Total 32 nonterminal symbols: 32 defined (see below), 0 root (—), 2 top (MotivationElement, String), 0 bottom (—).

Syntax

<table>
<thead>
<tr>
<th>Production Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assessment ::=</td>
</tr>
<tr>
<td>[motivatedDirective]: Directives+ [affectedEnd]: End+ [judgedInfluencer]: Influencers+ [assessingOrganizationUnit]: OrganizationUnit</td>
</tr>
<tr>
<td>MotivationElement ::=</td>
</tr>
<tr>
<td>Assessment</td>
</tr>
<tr>
<td>MotivationElement ::=</td>
</tr>
<tr>
<td>Means</td>
</tr>
<tr>
<td>MotivationElement ::=</td>
</tr>
<tr>
<td>OrganizationUnit</td>
</tr>
<tr>
<td>MotivationElement ::=</td>
</tr>
<tr>
<td>End</td>
</tr>
<tr>
<td>MotivationElement ::=</td>
</tr>
<tr>
<td>BusinessProcess</td>
</tr>
<tr>
<td>MotivationElement ::=</td>
</tr>
<tr>
<td>Asset</td>
</tr>
<tr>
<td>MotivationElement ::=</td>
</tr>
<tr>
<td>Liability</td>
</tr>
<tr>
<td>MotivationElement ::=</td>
</tr>
<tr>
<td>Influencer</td>
</tr>
</tbody>
</table>
Browsable Business Motivation Model (BMM) Grammar

Grammar extracted by Vadim Zaytsev, see the Grammar Zoo entry for details: dsl/business/motivation/v1.0/atlantic1/extracted
Source used for this grammar: OMG, The Business Motivation Model from the OMG, written from the OMG BMM specification, generated from the CMOF XMI for BMM 1.0, bmmOmg.ecore, January 2010

Summary

• Total 62 production rules with 62 top alternatives and 480 symbols.
• Vocabulary: 122 = 32 nonterminals + 0 terminals + 90 labels + 0 markers.
• Total 32 nonterminal symbols: 32 defined (see below), 0 root (→), 2 top (MotivationElement, String), 0 bottom (→).
• Total 90 labels: [motivatedDirective] 4, [affectedEnd], [judgedInfluencer], [assessingOrganizationUnit], [affectedMeans], [identifiedPotentialImpact], [usedAssessment], [usingAssessment], [categorizingAssessmentCategory], [supportedDesiredResult] 2, [governedCourseOfAction], [derivedCourseOfAction], [governedAsset], [motivatingPotentialImpact], [directiveRegulation], [motivatingAssessment], [establishingOrganizationUnit], [judgingAssessment] 3, [definedEnd], [managedBusinessProcess], [managedLiability], [managedAsset] 2, [determiningStrategy], [recognizedInfluencer], [internalInfluencingOrganization], [madeAssessment], [establishedMeans], [definingOrganizationUnit], [deliveredOffering], [realizedCourseOfAction], [governingBusinessPolicy], [guidingBusinessRule], [responsibleOrganizationUnit] 3, [definingCourseOfAction], [usedAsset], [requiredResource], [deliveringBusinessProcess], [deployingCourseOfAction], [governingDirective] 2, [managingBusinessProcess], [enabledCourseOfAction], [enablingCourseOfAction], [moreSpecificCourseOfAction], [broaderCourseOfAction], [definedOffering], [dischargedLiability], [realizingBusinessProcess], [baseDirective], [deployedAsset], [moreSpecificDesiredResult], [broaderDesiredResult], [supportingDirective], [supportingCourseOfAction], [claimedResource], [dischargingCourseOfAction], [providingFixedAsset], [requiringOffering], [claimingLiability], [usingOffering], [providedResource], [moreSpecificBusinessPolicy], [broaderBusinessPolicy], [derivedBusinessRule], [governingBusinessProcess], [effectingTactic], [guidedBusinessProcess], [baseBusinessPolicy], [implementedStrategy], [enforcedBusinessRule], [determinedOrganizationUnit], [plannedMission], [implementingTactic], [operativeVision], [componentStrategy], [amplifyingGoal], [deliveringMission], [quantifyingObjective], [amplifiedVision], [quantifiedGoal], [categorizingInfluencerCategory], [sourceInfluencingOrganization], [recognizingOrganizationUnit], [categorizedInfluencer], [categorizingOrganizationCategory], [influencingOrganizationCategory], [providedInfluencer], [categorizedInfluencingOrganization], [identifyingAssessment] 3, [regulatingDirective], [categorizedAssessment].

Syntax

Assessment ::= motivatedDirective::Directive+ affectedEnd::End+ judgedInfluencer::Influencer+ assessingOrganizationUnit::OrganizationUnit

MotivationElement ::= Assessment

MotivationElement ::= Means

MotivationElement ::= Means
Browsable Ada 83 Grammar

Grammar extracted by Vadim Zaytsev, see the Grammar Zoo entry for details: ada/ada83/ichbiah/extracted
Source used for this grammar: Jean D. Ichbiah, Preliminary Ada reference manual; Syntax Summary, ACM SIGPLAN Notices, Volume 14 Issue 6a, June 1979, pages E-1 to E-5 (142-146) [DOI]

Summary

- Total 134 production rules with 245 top alternatives and 1321 symbols.
- Vocabulary: 244 = 152 nonterminals + 92 terminals + 0 labels + 0 markers.
- Total 152 nonterminal symbols: 134 defined (see below), 0 root (—), 4 top (pragma, logical_operator, exponentiating_operator, compilation), 18 bottom (task_name, static_expression, character, digit, subtype_name, entry_name, constant_name, unit_name, type_name, exception_name, module_name, component_name, DQUOTE, character_literal, subprogram_name, underscore, lower_case_letter, upper_case_letter).
- Total 92 terminal symbols: 63 keywords ("pragma", "constant", "type", "is", "subtype", "new", "range", "digits", "delta", "array", "of", "choice", "others", "record", "end", "null", "case", "when", "access", "all", "and", "or", "xor", "nor", "in", "mod", "return", "if", "then", "else", "loop", "for", "reverse", "while", "exit", "goto", "assert", "function", "procedure", "out", "begin", "exception", "declare", "private", "package", "task", "body", "restricted", "use", "renames", "initiate", "entry", "accept", "do", "delay", "select", "abort", "separate", "raise", "generic", "packing", "at"), 0 numerics (—), 28 signs ("#", ".", "+", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",", ",
Browsable Ada 83 Grammar

Grammar connected by Vadim Zaytsev, see the Grammar Zoo entry for details: ada/ada83/ichbiah/connected
Source used for this grammar: Jean D. Ichbiah, Preliminary Ada reference manual; Syntax Summary, ACM SIGPLAN Notices, Volume 14 Issue 6a, June 1979, pages E-1 to E-5 (142-146) [DOI]

Summary

- Total 134 production rules with 243 top alternatives and 1335 symbols.
- Vocabulary: 243 = 141 nonterminals + 92 terminals + 0 labels + 10 markers.
- Total 141 nonterminal symbols: 134 defined (see below), 1 root (compilation), 1 top (pragma), 7 bottom (character, digit, DQUOTE, underscore, lower_case_letter, upper_case_letter).
- Total 92 terminal symbols: 63 keywords (pragma, constant, type, is, subtype, new, range, digits, delta, array, of, choice, others, record, end, null, case, when, access, all, and, or, xor, not, in, mod, return, if, then, elseif, else, loop, for, reverse, while, exit, goto, assert, function, procedure, out, begin, exception, declare, private, package, task, body, restricted, use, renames, initiate, entry, accept, do, delay, select, abort, separate, raise, generic, packing, at, 0 numerics (--), 28 signs (#, ., +, -, =, <, <=, >, >=, !, &), 47 .: =: :.
- Total 10 markers: (type_name), (subtype_name), (constant_name), (component_name), (subprogram_name), (unit_name), (module_name), (task_name), (entry_name), (exception_name).

Syntax

```
identifier ::= 
    letter (underscore? letter_or_digit)*

letter_or_digit ::= 
    letter 
    digit
```
fold exponentiating_operator everywhere.

eliminate logical_operator.

// As described in §4.8, a static_expression can be detected at compile time and must only contain
// (a) literals
// (b) aggregates whose components are static expressions
// (c) constants initialised by static expressions
// (d) predefined operators, functions and attributes
// (e) qualified static expressions
// (f) indexed and selected components of constants
// There are two ways to solve this: either duplicate a grammar fragment and make these changes or use the easy way:

define static_expression ::= expression ; .

// This is a convention they used (italics)_(name) to mean (name) with some annotation

replace task_name with <task_name>:name everywhere.
replace subtype_name with <subtype_name>:name everywhere.
replace entry_name with <entry_name>:name everywhere.
replace constant_name with <constant_name>:name everywhere.
replace unit_name with <unit_name>:name everywhere.
replace type_name with <type_name>:name everywhere.
replace exception_name with <exception_name>:name everywhere.
replace module_name with <module_name>:name everywhere.
replace component_name with <component_name>:name everywhere.
replace subprogram_name with <subprogram_name>:name everywhere.
Maturity Model

- fetched
- extracted
- connected
- adapted
- exported
fetched
extracted
connected
adapted
exported
GrammarZoo

The objective of the Grammar Zoo is to accumulate grammars of various software languages, extracted and recovered from language documentation, parser specifications and other artefacts and make them available in a range of formats.

483 entries and counting
772 grammars: 483 fetched + 260 extracted + 20 connected + 7 corrected + 1 recovered + 1 imported

Bulk download of the whole corpus:


(●●) Ada 83 [git]

Ada 83 — Ada 95 — Ada 2005

(●●) Ichbiah [git] [ReadMe]

Source: Jean D. Ichbiah, Preliminary Ada reference manual; Syntax Summary, ACM SIGPLAN Notices, Volume 14 Issue 6a, June 1979, pages E-1 to E-5 (142-146) [DOI]

(●) The fetched grammar is [src.syntax.summary.txt]

dir: fetched  level: 0  method: copy

(●) The extracted grammar is [grammar.bgf] [Browse now!]

Files used: [connect.glue]


(●) The connected grammar is [grammar.bgf] [Browse now!]

Files used: [connect.glue]


(●) LNCS 0106 [git] [ReadMe]


(●) The fetched grammar is [src.syntax.summary.txt]

dir: fetched  level: 0  method: copy

(●) LNCS 0155 [git] [ReadMe]


(●) The fetched grammar is [src.syntax.summary.txt]
Conclusion

- More details?
- read the paper (ME’14, pages 42–51)
- read the other paper (“Grammar Zoo”, SCP, 2014)
- Grammar Zoo
- new version SOON
- Sources
  - All photos taken from PEXELS, CCO Universal license.
- Questions?