Dynamic Generalized Parsing and Natural Mathematical Language

Defensio

Mag. Kevin Kofler, Bakk.
Supervised by: Univ.-Prof. Dr. Arnold Neumaier

University of Vienna, Faculty of Mathematics

October 13, 2017
# Contents

- Acknowledgements
- Introduction
  - Problem Statement
  - Goals
- The Dynamic Generalized Parser DynGenPar
  - The DynGenPar Algorithm
  - Analysis
  - Additional Features
- Applications
  - DynGenPar and Concise
  - Applications of DynGenPar to Formal Languages
  - DynGenPar and the Grammatical Framework (GF)
  - Applications of DynGenPar to Natural Language
- Results
  - Efficiency Results
  - Test Results on Real-World \LaTeX\ Formulas
- Outlook – Future Extensions

---

Mag. Kevin Kofler, Bakk.

University of Vienna, Faculty of Mathematics

Dynamic Generalized Parsing and Natural Mathematical Language – Defensio
Acknowledgements

Support by the Austrian Science Fund (FWF) under contract numbers P20631, P23554 and P22239 is gratefully acknowledged. In addition, I would like to thank:

- my supervisor: Arnold Neumaier,
- my work group: Hermann Schichl, Peter Schodl, Ferenc Domes, and Tiago de Morais Montanher,
- many more people (too many to list here).
Problem Statement

What is Parsing?

- automatic input of text into a form understandable to a computer
- Input: text (sequence of characters from an alphabet)
- Output: syntax tree (tree representation of the text structure)
- often in 2 steps:
  - lexing (scanning):
    - Input: text
    - Output: sequence of tokens
  - parsing (in a strict sense):
    - Input: sequence of tokens
    - Output: syntax tree
- scannerless parsing: tokens = characters
Grammars

- fundamental information for parsing
- describe matching between text and syntax tree
- **grammar** = set of rules
  - variables in rules: **categories**
  - **rule**: how to generate text from a category
    - start from **start category**
    - parsing: reversed reading
- most frequent case: **context-free grammars**
  - rules of the form $\text{Cat} \rightarrow \alpha \beta \gamma \ldots$
    - where $\alpha, \beta, \gamma, \ldots$ categories or tokens
    - e.g., $S \rightarrow A \ b, A \rightarrow A \ a, A \rightarrow a$
      (produces ab, aab, aaab, etc.)
  - formal definition in the next section
Types of Mathematical Text

- **natural language text:**
  The cost is the sum of the area of the square and half of its perimeter. Minimize the cost.

- **formulas:** \( \min x^2 + 2x \)
  - programming language style: \( \min(x^2+2x) \)
  - \( \text{\LaTeX} \) style: \( \min x^2+2x \)

- **mixed:** Minimize \( x^2 + 2x \).
Primary Target: FMathL (Formal Mathematical Language)

A **modeling language** is an artificial language for the user friendly specification of mathematical problems, with interfaces to the corresponding solvers.

**FMathL** is intended to be a modeling and documentation language for the working mathematician that

- is based on traditional mathematical syntax,
- allows to express arbitrary mathematics,
- decides automatically which tools to use.
FMathL Goals

- modeling language for optimization problems (short term)
- build computer-oriented library of math. knowledge
  - formalized for the computer
  - from input as informal as possible
    - textbooks and papers (e.g., arXiv papers)
- reasoning
  - checking the correctness of proofs
  - solving computational (e.g., optimization) problems
  - semantic classification and retrieval
    - search engine for mathematics
- (mostly) automatic translation of mathematical texts
- vision: MathResS –
  automatic mathematical research system
Goals

Requirements

- allow efficient incremental addition of new rules
  - without recompiling the whole grammar
  - e.g. for a mathematical definition
- more general grammars than LR(1) or LALR(1)
  - natural language is usually not LR(1)
  - parallel multiple context-free grammars (PMCFGs)
- exhaustively produce all possible parse trees
  - packed representation
- predictive parsing
  - incremental processing
  - predicting the next token
- both scanner-driven (natural math. language) and scannerless (most other parsing tasks)
Dynamic Generalized Parser

- dynamic
  - grammar not fixed in advance
  - allows adding rules at any time
    - even during parsing (mathematical definitions)
  - avoids precompiled tables
- generalized
  - general context-free grammars
  - produces all parse trees for ambiguous grammars
  - additional generalization: PMCFGs (Seki et al.)
    - Parallel Multiple Context-Free Grammars
    - interoperability with GF (Grammatical Framework)
- simplicity is important
  - formal verifiability (in the future)
Goals

Problems with Existing Approaches

▶ top-down parsing
  ▶ chokes on left recursion
  ▶ e.g. $\text{Expr} \rightarrow \text{Expr} + \text{Term} \mid \text{Term}$
  ▶ requires grammar transformation or complex workarounds
    ▶ fails simplicity criterion

▶ bottom-up parsing
  ▶ problem: what operation (shift, reduce) to do when
  ▶ requires parse states and stacks, lookahead
  ▶ requires precomputed tables
    ▶ have to recompute the whole table when the grammar changes
Definition: Context-free Grammar (CFG)

- $G = (N, T, P, S)$
- $N$ ... (finite) set of nonterminals (categories)
- $T$ ... (finite) set of tokens
  - called alphabet of the grammar
  - disjoint from $N$
- $P$ ... (finite) set of productions (rules)
  - of the form $n \rightarrow \alpha_1 \ldots \alpha_k$, $n \in N$, $\alpha_i \in N \cup T \ \forall i$
- $S \in N$ ... start symbol (start category)
The DynGenPar Algorithm

Initial Graph

- replaces precompiled tables
  - dynamically extensible for new rules
- directed, labeled multigraph on $\Gamma = N \cup T$
- tokens $T$ are sources
- edge from symbol $s \in \Gamma$ to category $n \in N$ iff
  - $\exists$ rule $n \rightarrow n_1 n_2 \ldots n_k s \ldots$ with $n_i \in N_0 \ \forall i$
  - where $N_0 \subseteq N$ the set of all nonterminals from which $\varepsilon$ can be derived
- label of the edge
  - that rule
  - number $k$ of $n_i$ set to $\varepsilon$
- more than one possible label ... multi-edge
The DynGenPar Algorithm

Example Initial Graph (1/2)

- Example grammar
  - \( N = \{ S, \text{Task}, \text{Expr}, \text{Term}, \text{Factor} \} \)
  - \( T = \{ \text{min}, \text{max}, +, *, x, \text{NUMBER} \} \)
  - \( S \rightarrow \text{Task} \text{Expr} \)
  - \( \text{Task} \rightarrow \text{min} | \text{max} \)
  - \( \text{Expr} \rightarrow \text{Expr} + \text{Term} | \text{Term} \)
  - \( \text{Term} \rightarrow \text{Term} * \text{Factor} | \text{Factor} \)
  - \( \text{Factor} \rightarrow x | \text{NUMBER} \)

- \( N_0 = \emptyset \)
  - because there is no rule \( n \rightarrow \varepsilon \) in the grammar
  - thus consider only rules of the form \( n \rightarrow s \ldots \)
  - \( k \) (number of skipped \( \varepsilon \) categories) = 0 everywhere

Mag. Kevin Kofler, Bakk.
University of Vienna, Faculty of Mathematics
Dynamic Generalized Parsing and Natural Mathematical Language – Defensio
The DynGenPar Algorithm

Example Initial Graph (2/2)

- we obtain the graph

\[
\begin{align*}
S &\rightarrow \text{Task} \text{ Expr} \\
\text{Task} &\rightarrow \text{min} \\
\text{Task} &\rightarrow \text{max} \\
\text{Expr} &\rightarrow \text{Expr} + \text{Term} \\
\text{Term} &\rightarrow \text{Term} \times \text{Factor} \\
\text{Factor} &\rightarrow x \\
\text{Factor} &\rightarrow \text{NUMBER} \\
\text{min} \quad &\quad \text{max} \quad + \quad * \\
\end{align*}
\]
Neighborhoods

- Let $s \in \Gamma = N \cup T$ (symbol), $z \in N$ (target)
- Neighborhood $\mathcal{N}(s, z)$...
  - Edges from $s$ to a category $c$ where $z$ reachable from $c$
  - in the example
    - $\mathcal{N}(\text{min}, S) = \{ \text{Task} \rightarrow \text{min} \}$
    - $\mathcal{N}(x, S) = \emptyset$
    - $\mathcal{N}(x, \text{Expr}) = \{ \text{Factor} \rightarrow x \}$
    - $\mathcal{N}(\text{Term}, \text{Expr}) = \{ \text{Expr} \rightarrow \text{Term}, \text{Term} \rightarrow \text{Term} \ast \text{Factor} \}$
- computed by graph walk
- can be cached
  - but must be recomputed if the grammar changes (at least locally)
The DynGenPar Algorithm

Operations

- \( \text{match}_\varepsilon(n), \ n \in N_0 \ldots \) derive \( \varepsilon \) from \( n \)
  - top-down
  - ignore recursion (would produce \( \infty \)ly many parse trees)
- \( \text{shift} \ldots \) read in the next token
- \( \text{reduce}(s, z), \ s \in \Gamma, \ z \in N \ldots \) reduce \( s \) to \( z \)
  - different from LR \( \text{reduce}! \)
  - already reduce after the first symbol
  - \( \text{reduce} \) must complete the match
- \( \text{match}(s), \ s \in \Gamma = N \cup T \)
  - if \( s \in N_0 \): \( \text{match}_\varepsilon(s) \), remember result
  - \( t = \text{shift} \)
  - if \( s \in T \): compare \( s \) with \( t \)
  - if \( s \in N \): \( \text{reduce}(t, s) \)
The DynGenPar Algorithm

Reduce(s,z) Step

- pick a rule \( c \rightarrow n_1 \ n_2 \ \ldots \ n_k \ s \ \alpha_1 \ \alpha_2 \ \ldots \ \alpha_\ell \) in \( \mathcal{N}(s, z) \)
- for each \( n_i \in \mathcal{N}_0 \):
  - \( \text{match}_\varepsilon(n_i) \)
- \( s \) already recognized ... parse tree \( S \)
- for each \( \alpha_j \in \Gamma = \mathcal{N} \cup \mathcal{T} \):
  - \( \text{match}(\alpha_j) \) (top-down step)
- parse tree:

\[
\begin{array}{c}
  c \\
  \text{match}_\varepsilon(n_1) \quad \ldots \\
  S \quad \text{match}(\alpha_1) \quad \ldots \\
\end{array}
\]

- if \( c \neq z \): continue reducing recursively
  - \( \text{reduce}(c, z) \)
- (also consider \( \text{reduce}(z, z) \) for left recursion)
Analysis

- more conflicts as for LR/GLR
  - reduce already after the 1st symbol
  - no lookahead
- but need neither states nor tables
  - initial graph can be dynamically added to
- implementation keeps efficiency even in case of conflicts
  - never execute the same match step more than once
Additional Features (1/2)

- **predictive parsing**
  - incremental operation driven by shift steps
  - keep explicit parse stacks
  - can predict next token from stacks
    - top-down expansion

- **efficient storage reduces effort**
  - DAG-structured stacks
    - match steps executed only once
  - syntax trees as DAGs (*packed forests*)
    - less storage and storing effort
    - avoids some case distinctions
  - allows efficient exhaustive parsing
Additional Features (2/2)

- constraints enforced during parsing
  - parallel multiple context-free grammars (PMCFGs, Seki et al.)
    - parsed as constrained CFGs
  - next token constraints
    - require (expect) / forbid (taboo) token(s) after a rule
    - special case: scannerless parsing (token = character), e.g. maximal matches

- CFG rules can have labels
  - reproduced in the parse tree

- custom parse actions
  - algorithm as presented generates only parse tree
  - want e.g., action for definition producing rule
Interoperability Features

- arbitrary token source (lexer, token buffer, ...)
  - other DynGenPar instance (hierarchical parsing)
    - or other arbitrary parser
  - can also return whole parse tree
  - Flex lexer (C++)
  - character token source (scannerless parsing)
- import of PGF files from Grammatical Framework (GF)
  - see Applications section
- Java bindings
  - built using Qt Jambi generator
  - used in the Concise GUI (see next slide)
Concise

- graphical user interface (GUI)
- integrated development environment (IDE) for FMathL
- implements and visualizes a **semantic memory**
  - graph-structured knowledge representation
- interactive graph editor
- written in Java
- DynGenPar integrated into Concise
  - through the Java bindings
  - used for almost all parsing tasks in Concise
DynGenPar and Concise

Screenshot of Concise (Graph View)
Embedding of DynGenPar into Concise

- **type sheet**: text file representing a **type system**
  - specification of types
    - required fields
    - optional fields
  - **usages** encoding grammar information
- **Concise Grammar Java class**
  - converts Concise type sheet to DynGenPar grammar
  - converts DynGenPar parse tree to Concise record
  - **record**: subgraph in the semantic memory
- **supported token sources:**
  - scannerless `TextByteTokenSource`
  - `ConciseTokenSource` producing tokens from a record
  - used to convert list-of-words record to semantic record
Concise Type Sheets

- representation of grammars in Concise
- themselves parsed using DynGenPar
- grammar is itself a type sheet
  - originally by Arnold Neumaier
- bootstrap parser
  - grammar manually converted to a C++ program
  - uses DynGenPar directly (without Concise)
  - produces Concise record as a text file
- can self host (passes bootstrap comparison)
  - can parse type sheets using the bootstrapped type sheet for type sheets
  - produced output matches bootstrap parser
Concise Code Sheets

- fundamental programming language in Concise
- textual representation of **elementary acts**
  - primitive operations on the semantic memory
  - 16 act types: Do, Return, Goto, Assign, Set, Get, GetType, IsSubtypeOf, Identical, Convert, Vcopy, ForAllFields, Call, Ask, Supervise, Resume
  - more structured than assembly: loops, etc.
- code sheet representation adds **declarations**
  - types, global and local variables, etc.
  - DynGenPar produces references by name
  - must be **resolved** when converting to elementary acts
Applications of DynGenPar to Formal Languages

Concise Record Transformation Sheets

- transform one record into another
  - usually of a different type
  - fixed source type and destination type
  - can be the same type (simplifier)
- structured like XSLT style sheets
- adapted to the Concise semantic memory
- added support for name resolution
- used, e.g., for the conversion from code sheets to elementary acts
Applications of DynGenPar to Formal Languages

Chemical Process Modeling (ChemProcMod)

- idea, input from Ali Baharev and Arnold Neumaier
  - optimization techniques for chemical process simulation
- attempted using Modelica
  - unsatisfactory syntax
  - issues with available implementations
    - bugs, performance
- specialized modeling language (Concise type sheet)
  - intuitive for a chemical engineer
    - Ali Baharev has chemical engineering background
  - declarative rather than imperative
  - vocabulary from the chemical application
    - not from the mathematical model or OOP
    - e.g., Modelica class $\rightarrow$ ChemProcMod unit
- comes with basic unit library
Extensible OptProbl Grammar

- context: COCONUT Project (Hermann Schichl et al.)
  - framework for global optimization
- need for extensible input format
  - e.g., adding Lie group notation
- PoC grammar for optimization problems
  - coded directly in C++ using DynGenPar (no Concise)
  - small subset of \LaTeX
    - only programming language notation (e.g., 2 * x)
    - e.g., no implied multiplication (e.g., 2x)
  - extensible with \texttt{\newcommand}
    - adds rules to the grammar at runtime
  - also features next token constraints
    - used to determine the end of a tag
Robust AMPL

- **AMPL**: A Mathematical Programming Language
  - well-known modeling language for optimization problems
- subset reimplemented as Concise type sheet
  - official AMPL software not needed
- **robust**: added full support for intervals
  - official AMPL: intervals allowed only as sets
  - robust AMPL: intervals allowed wherever numbers are
- record transformation to internal representation
  - to RobustOptProb type sheet (Ferenc Domes)
  - record transformation can be **rigorous** (all numbers become intervals with outward rounding) or not (output numbers are of double type)
DynGenPar and the Grammatical Framework (GF)

**PGF (Portable Grammar Format) File Import**

- GF = Grammatical Framework (Ranta et al.)
- PGF (Angelov et al.): format produced by GF
  - based on parallel multiple context-free grammars (PMCFGs, Seki et al.)
  - binary serialization of Haskell data structures
- import implemented in DynGenPar
  - Haskell-compatible deserialization
  - conversion to standard PMCFG format
    - some needed extensions (context-free categories as tokens, next token constraints, tokens with values) supported by DynGenPar
- parsing requires GF-compatible lexer
  - implemented as PgfTokenSource
DynGenPar and the Grammatical Framework (GF)

PGF GUI – Graphical Demo Application

demonstrates DynGenPar PGF import and prediction
GF Application Grammar for Mathematical Language

- joint work with Peter Schodl
- early research on grammars for natural mathematical language in the FMathL project
- focus on linearization (the opposite of parsing)
  - using GF Haskell runtime (no DynGenPar)
- only for text, formulas as verbatim strings
- 2 versions
  1. handwritten GF grammar
     - in lockstep with the Concise type system
  2. automatically generated GF grammar
     - generated from the Concise type system
- abandoned in favor of Concise type sheets
Applications of DynGenPar to Natural Language

Naproche (Cramer et al.)

- controlled natural language for mathematical logic
- grammar described in Kühlwein’s diploma thesis
- implemented by me
  1. in Bison, using Generalized LR (GLR)
  2. in DynGenPar (in C++), for comparison
- hierarchical grammar
  - formulas parsed by separate grammar
- all 4 grammars use a Flex lexer
  - Bison: Flex C mode
  - DynGenPar: Flex C++ mode (C++ classes)
Applications of DynGenPar to Natural Language

TextDocument Toolchain

- imports \LaTeX\ document as Concise record
  - in the TextDocument type system

1. LaTeXML (3\textsuperscript{rd} party): converts \LaTeX\ to XML
2. processxml: transforms LaTeXML XML to record XML
3. xmltocnr: converts record XML to Concise record sheet

- central tool: processxml
  - transforms \LaTeX\ structure to TextDocument structure

- resulting TextDocument records are unparsed
  - only document structure represented
  - paragraph = list of words
  - can be parsed in a later step
Basic Definitions

- Concise type sheet operating on paragraphs
  - unparsed TextDocument representation (list of words)
  - uses the ConciseTokenSource
- by Arnold Neumaier, Ferenc Domes, Kevin Kofler
- PoC for the handling of mathematical definitions
- main feature: definitions automatically trigger a hook that dynamically adds a rule at runtime
- static part covers just enough idiomatic mathematical English to represent basic mathematical definitions
- formulas kept as unparsed strings
  - can be parsed with separate formula grammar
BasicReasoning

- Concise type sheet operating on paragraphs
- by Arnold Neumaier and Kevin Kofler
- based on MathNat by Muhammad Humayoun
- idiomatic mathematical English needed to represent basic mathematical reasoning
  - significantly broader scope than BasicDefinitions
- work in progress
  - grammar not yet complete
Applications of DynGenPar to Natural Language

\textbf{\LaTeX{} Formulas}

- Concise type sheet (by me) operating on strings
- Grammar for \LaTeX{} formulas in typical notation
  - e.g., implied multiplication allowed
- By design, not all ambiguities resolved during parsing
  - Some resolvable by semantic analysis (planned)
  - Remaining ones must be resolved interactively
- Scannerless grammar (character tokens)
- Based on the expression grammar pattern
- Tested on 2 university text books in German
  - > 70\% success rate (see Results section)
Efficiency Results

Competitive with Bison

- benchmarking results on the Naproche grammar

<table>
<thead>
<tr>
<th></th>
<th>compilation</th>
<th>grammar conversion</th>
<th>parsing (Burali-Forti)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bison</td>
<td>1089 ms</td>
<td>153 ms*</td>
<td>1.60 ms</td>
</tr>
<tr>
<td>DynGenPar</td>
<td>8851 ms</td>
<td>5.34 ms</td>
<td>9.37 ms**</td>
</tr>
</tbody>
</table>

* ... at compile time, thus requires recompilation
** ... total execution time of 14.71 ms minus grammar conversion time

- only 6 times slower than Bison at pure parsing
  - i.e. exactly what Bison is optimized for
- 29 times faster than Bison at grammar conversion
  - no recompilation required
  - ⇒ effectively over 200 times faster!
**Competitive with GF (Grammatical Framework)**

- benchmarking results on the GF *Phrasebook* grammar
  - testcase:
    
    *See you in the best Italian restaurant tomorrow!*

<table>
<thead>
<tr>
<th></th>
<th>Parsing time</th>
</tr>
</thead>
<tbody>
<tr>
<td>GF Haskell runtime</td>
<td>43.4 ms</td>
</tr>
<tr>
<td>GF C runtime</td>
<td>17.8 ms</td>
</tr>
<tr>
<td>DynGenPar</td>
<td>121.8 ms</td>
</tr>
</tbody>
</table>

- parsing time comparable to both GF runtimes
  - on practical application grammars
- DynGenPar honors next token constraints
  - both GF runtimes incorrectly accepted

*Where is an restaurant?*
### Performance of Dynamic Rule Addition

- **test results on BasicDefinitions grammar**
  - paragraphs containing 1 definition each
  - | number of definitions | parsing time |
  - |------------------------|-------------|
  - | 0                      | 7 ms        |
  - | 1                      | 9 ms        |
  - | 10                     | 31 ms       |
  - | 100                    | 228 ms      |
  - | 1000                   | 2298 ms     |
  - **time to parse a paragraph and add a rule to the grammar:** only 2.3 ms altogether
Test Results on Real-World LaTeX Formulas

LaTeX Formula Parser Success Rates

- unique formulas from 2 university textbooks
  - ALA (Arnold Neumaier: Analysis und lineare Algebra)
  - Einf (Hermann Schichl and Roland Steinbauer: Einführung in das mathematische Arbeiten)
  - multiple instances of the exact same formula deleted

**ALA**
- unambiguous: 42.4%
- ambiguous: 28.6%
- failed: 29.0%

**Einf**
- unambiguous: 52.0%
- ambiguous: 19.8%
- failed: 28.2%
Possible Future Extensions

- context-sensitive constraints on rules
  - generalize PMCFG and next token constraint support
  - main objective: figure out the needed class of constraints
- stateful parse actions (more state information)
- runtime parser for rules (directly in DynGenPar)
  - read rules into parser from user-writable format
  - allows dynamic extension by the user at runtime
  - now only possible through Concise
- scalability to larger PMCFGs
  - generalize optimizations that assume no constraints
- error correction (long-term research goal)
  - have only basic error detection and reporting
  - goal: suggest corrections to the user
References

- https://www.tigen.org/kevin.kofler/fmathl/dyngenpar/
- http://www.mat.univie.ac.at/~neum/FMathL.html
- http://www.mat.univie.ac.at/~dferi/concise.html
- http://www.mat.univie.ac.at/~neum/glopt/coconut/
- http://www.grammaticalframework.org/
- https://korpora-exp.zim.uni-duisburg-essen.de/naproche/
- https://www.gnu.org/software/bison/
- https://github.com/westes/flex
- http://dlmf.nist.gov/LaTeXML/
- more: see bibliography in the thesis