Theory of GF and Parsing, Hints for Efficient Grammars

Krasimir Angelov

Chalmers University of Technology

August 21, 2009

Introduction

The GF language is:

- domain specific - Grammars first of all
- declarative - What vs How

The GF system is:

- optimizing compiler and interpreter
- not as smart as you

Although the language is declarative, the compiler needs some help to produce efficient grammars

Parsing vs Linearization

There are two major operations that someone could do with a grammar

- Linearization
  - **Efficient** - mapping from tree to string
- Parsing
  - **Search Problem** - find the tree(s) that produce a given string

Speaking about the efficiency I mean efficient parsing and compact grammar

Inside the Black Box

- The GF language is too complex to be used directly for efficient parsing
- The compiler transforms it into simpler **GF Core** language
- The efficiency of the grammar depends on the GF Core
- Small Core $=$ Efficient Grammar

$$\text{GF} \Rightarrow \text{GF Core}$$

$$\text{GF Core} \equiv \text{PMCFG}$$

Parallel Multiple Context-Free Grammar (PMCFG)

- Well known grammar formalism (Seki at al., 1991)
- Natural extension of CFG that produces tuples of strings instead of simple strings
- It is trivial to implement classical context-sensitive languages - $\{a^n b^n c^n | n > 0\}$:

$$\begin{aligned}
\textbf{fun}\ z \quad &= < "", "", "" > \\
s\ x \quad &= < "a" +\!\!+\ x.p1, "b" +\!\!+\ x.p2, "c" +\!\!+\ x.p3 > \\
c\ x \quad &= < x.p1 +\!\!+\ x.p2 +\!\!+\ x.p3 >
\end{aligned}$$

Mildly Context-Sensitive Languages and GF

Joshi Aravind. 1991. Tree Adjoining Grammars: How much context-sensitivity is required to provide reasonable structural descriptions?

- The language must be parsable in polynomial time.
- The language must have constant growth.
- The language should admit limited cross-serial dependencies.

Example language: $\{a^n b^n c^n | n > 0\}$ and all MCFG

More than Mildly Context-Sensitive Languages

The exponential language $\{a^{2^n} | n > 0\}$:

$$\begin{aligned} \textbf{fun } z &= <\text{"a"}> \\ s\ x &= < x.p1 \mathbin{+\!\!+} x.p1 > \end{aligned}$$

is not Mildly Context-Sensitive (**reduplication**).

Four non-MCS Natural Languages

- Mandarin Chinese numeral names
- Mandarin Chinese yes/no questions
- Old Georgian case system
- Lindenmayer system

Mandarin Chinese numeral names

- Concatenation of ten thousands

	wan	10 000
wan wan	yi	100 000 000
wan yi	zhao	1 000 000 000 000

- Composite numbers

wu	zhao	zhao	wu	zhao
five	trillion	trillion	five	trillion

5 000 000 000 005 000 000 000 000

- Formally

$$\{\text{wu zhao}^{k_1}\ \text{wu zhao}^{k_2}\ \ldots\ \text{wu zhao}^{k_n} \mid k_1 > k_2 > \ldots > k_n\}$$

Mandarin Chinese yes/no questions

Zhangsan	like	play	basketball	not	like	play	basketball
Zhangsan	ai	da	lanqiu	bu	ai	da	lanqiu

Does Zhangsan like to play basketball?

Zhangsan	like	play	basketball	not	like	play	volleyball
Zhangsan	ai	da	lanqiu	bu	ai	da	paiqiu

Zhangsan likes to play basketball, not to play volleyball

Old Georgian case system

- Example:

tkuenda	micemul	ars	cnob-ad	saidumlo-j
to you	given	is	knowing-Adv	mistery-Nom

igi	sasupevel-isa	m-is	ymrt-isa-jsa-j
Art=Nom	kingdom-Gen	Art-Gen	god-Gen-Gen-Nom

Onto You it is given to know the mistery of the kingdom of God

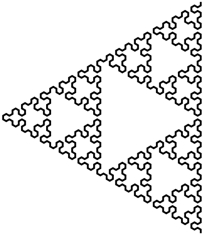
Mark 4:11

- Formal

$$N_1\text{-Nom} \quad N_2\text{-Gen-Nom} \quad N_3\text{-Gen}^2\text{-Nom} \ldots N_k\text{-Gen}^{k-1}\text{-Nom}$$

Lindenmayer system

- Mathematical objects
- The structure of some plants
- The growth of some cristals
- Symmetry in music



$$A \longrightarrow a$$
$$B \longrightarrow b$$
$$A \longrightarrow BRARB$$
$$B \longrightarrow ALBLA$$

The reduplication is a norm!

The current status of parsing with GF

Angelov. 2009. Incremental Parsing with PMCFG

Features:

- Very Efficient (**polynomial** - close to **linear**)
- Supports PMCFG for free
- PMCFG allows more compact grammars
- It is incremental !!!

Things to consider:

- the GF $\Rightarrow$ GF Core conversion is often exponential
- The grammar should be carefully written to avoid combinatorial explosion
- In practice careful means linguistically motivated

The Rules of The Game

Initial Predict

$$\frac{S \rightarrow f[\vec{B}]}{[^0_0 S \rightarrow f[\vec{B}]; 1 : \bullet\alpha]} \quad S \text{ - start category}, \alpha = \text{rhs}(f, 1)$$

Predict

$$\frac{B_d \rightarrow g[\vec{C}] \qquad [^k_j A \rightarrow f[\vec{B}]; l : \alpha \bullet \langle d; r \rangle\ \beta]}{[^k_k B_d \rightarrow g[\vec{C}]; r : \bullet\gamma]} \quad \gamma = \text{rhs}(g, r)$$

Scan

$$\frac{[^k_j A \rightarrow f[\vec{B}]; l : \alpha \bullet s\ \beta]}{[^{k+1}_j A \rightarrow f[\vec{B}]; l : \alpha\ s \bullet \beta]} \quad s = w_{k+1}$$

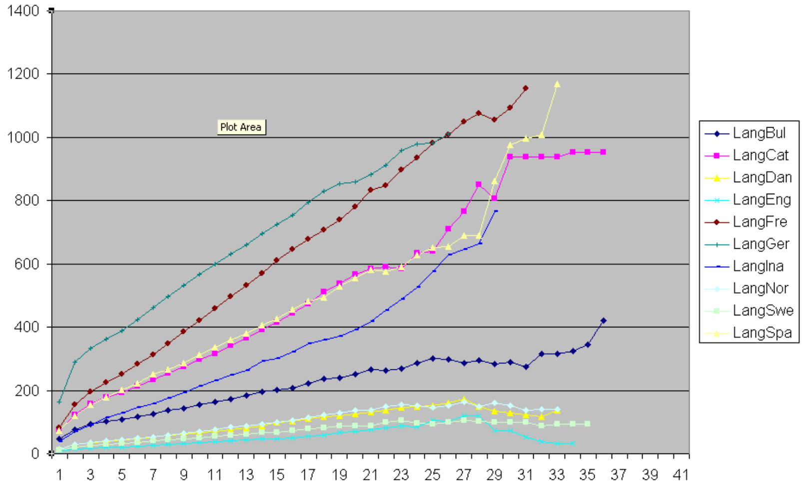
Complete

$$\frac{[^k_j A \rightarrow f[\vec{B}]; l : \alpha\bullet]}{N \rightarrow f[\vec{B}] \qquad [^k_j A; l; N]} \quad N = (A, l, j, k)$$

Combine

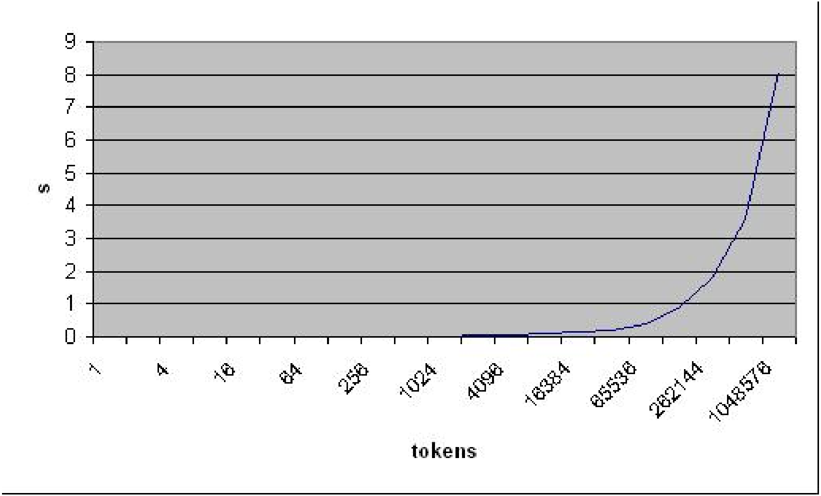
$$\frac{[^u_j A \rightarrow f[\vec{B}]; l : \alpha \bullet \langle d; r \rangle\ \beta] \qquad [^k_u B_d; r; N]}{[^k_j A \rightarrow f[\vec{B}\{d := N\}]; l : \alpha\ \langle d; r \rangle \bullet \beta]}$$

Parsing with the resource library



Note: Much faster with application grammars

Parsing of $\{a^{2^n} | n > 0\} - O(n \log n)$



GF Core Language

The parser uses language which is simplified version of GF.

- The linearization types are flat tuples of strings:

$$\textbf{lincat}\ C = Str * Str * \ldots * Str;$$

- The linearizations are simple concatenations:

$$\textbf{lin}\ f\ x\ y = < x.p1, x.p2 +\!\!+\ y.p3 >;$$

- No operations are allowed
- No variants are allowed
- No parameters and tables
- No pattern matching
- No gluing is allowed (i.e. $+\!\!+$ but not $+$)

GF $\Rightarrow$ GF Core

- Operations elimination
- Variants elimination
- Parameter types elimination
- Linearization rules transformations
- Common subexpressions optimization

Operations elimination

The operations are **NONRECURSIVE** functions. They are evaluated at compile time. *(macroses)*

GF

$$
\begin{aligned}
&\textbf{oper } mkN\ noun = \textbf{case } noun \textbf{ of } \{\\
&\qquad \_ + \text{"s"} \Rightarrow < noun, noun + \text{"es"} >;\\
&\qquad \_ \qquad\quad \Rightarrow < noun, noun + \text{"s"} >\\
&\qquad \};\\
&\quad \textbf{lin } apple\_N = mkN\ \text{"apple"};\\
&\qquad plus\_N = mkN\ \text{"plus"};
\end{aligned}
$$

GF Core

$$
\begin{aligned}
&\textbf{lin } apple\_N = < \text{"apple"}, \text{"apples"} >;\\
&\qquad plus\_N = < \text{"plus"}, \text{"pluses"} >;
\end{aligned}
$$

Note: the pattern matching in mkN was eliminated

Hints for Operations

Since the operations are computed at compile time this doesn't affect the runtime efficiency. However they affect the compilation speed *(slightly)*.

Variants elimination

The variants are just expanded:

GF

$$\textbf{lin}\ \mathit{girl\_N} = \mathit{mkN}\ (\textbf{variants}\ \{\text{"tjej"};\ \text{"flicka"}\});$$

GF Core

$$\begin{aligned}\textbf{lin}\ \mathit{girl\_N}_1 &= \mathit{mkN}\ \text{"tjej"};\\ \mathit{girl\_N}_2 &= \mathit{mkN}\ \text{"flicka"};\end{aligned}$$

Note: Appropriate for application specific grammars. Should be avoided in resource grammars.

Variants are not always what you want

GF

$$
\begin{aligned}
&\textbf{lin } Answer\ pol\ verb = \text{"I"} +\!\!+ pol +\!\!+ verb;\\
&\quad eat = \text{"eat"};\\
&\quad like = \text{"like"};\\
&\quad Pos = \text{""};\\
&\quad Neg = \textbf{variants}\{\text{"do not"};\text{"don't"}\};\\
&\\
&\quad Comp\ s1\ s2 = s1\ +\!\!+\ \text{";"}\ +\!\!+\ s2;
\end{aligned}
$$

$$Comp\ (Answer\ Neg\ like)\ (Answer\ Neg\ eat)$$

I don't like; I don't eat
I don't like; I do not eat
I do not like; I don't eat
I do not like; I do not eat

Variants are not always what you want

GF

$$
\begin{array}{l}
\textbf{lin}\ \mathit{Answer}\ \mathit{pol}\ \mathit{verb} = \backslash\backslash \mathit{style} \Rightarrow \text{"I"} +\!\!+ (\mathit{pol}!\mathit{style})\ +\!\!+\ \mathit{verb};\\
\quad \mathit{eat} = \text{"eat"};\\
\quad \mathit{like} = \text{"like"};\\
\quad \mathit{Pos} = \textbf{table}\{\mathit{Official} \Rightarrow \text{""}; \mathit{Spoken} \Rightarrow \text{""}\};\\
\quad \mathit{Neg} = \textbf{table}\{\mathit{Official} \Rightarrow \text{"do not"}; \mathit{Spoken} \Rightarrow \text{"don't"}\};\\
\\
\quad \mathit{Comp}\ \mathit{ids}1\ \mathit{ids}2 = \textbf{variants}\{\mathit{comp}\ \mathit{Official}; \mathit{comp}\ \mathit{Spoken}\};\\
\\
\textbf{oper}\ \mathit{comp}\ \mathit{style} = \mathit{s1}!\mathit{style}\ +\!\!+\ \text{";"} +\!\!+\ \mathit{s2}!\mathit{style};
\end{array}
$$

$$\mathit{Comp}\ (\mathit{Answer}\ \mathit{Neg}\ \mathit{like})\ (\mathit{Answer}\ \mathit{Neg}\ \mathit{eat})$$

I don't like; I don't eat
I do not like; I do not eat

The variants could blow up

When many variants are used in parallel the number of core rules grows exponentially.

GF

$$\textbf{lin}\ \textit{start\_word} = \textbf{variants}\{\text{"open"}; \text{"start"}\} +\!\!+\ \textbf{variants}\{\text{"Word"}; \text{"Microsoft Word"}\};$$

GF Core

$$\begin{aligned}
\textbf{lin}\ \textit{start_word}_1 &= \text{"open"} +\!\!+ \text{"Word"};\\
\textit{start_word}_2 &= \text{"open"} +\!\!+ \text{"Microsoft Word"};\\
\textit{start_word}_3 &= \text{"start"} +\!\!+ \text{"Word"};\\
\textit{start_word}_4 &= \text{"start"} +\!\!+ \text{"Microsoft Word"};
\end{aligned}$$

Variants explosion with tables

GF

lin *close_word* = *tense* ⇒ *close_V*!*tense* ++ **variants**{"Word";"Microsoft Word"};
oper *close_V* = **table** *Tense* {"close";"closed";"have closed";...};

Desugared GF

lin *close_word* = **table** *Tense* {
"close" ++ **variants**{"Word";"Microsoft Word"};
"closed" ++ **variants**{"Word";"Microsoft Word"};
"have closed" ++ **variants**{"Word";"Microsoft Word"}
...
};

Note: Leads to $2^3 = 8$ *possible combinations although there is only one variant in the original code*

Variants explosion with tables - CORE

GF Core

$$
\begin{aligned}
\textbf{lin}\ \mathit{close\_word}_1 &= <\text{"close" ++ "Word"};\\
&\quad \text{"closed" ++ "Word"};\\
&\quad \text{"have closed" ++ "Word"} >;\\
\mathit{close\_word}_2 &= <\text{"close" ++ "Microsoft Word"};\\
&\quad \text{"closed" ++ "Word"};\\
&\quad \text{"have closed" ++ "Word"} >;\\
\mathit{close\_word}_3 &= <\text{"close" ++ "Word"};\\
&\quad \text{"closed" ++ "Microsoft Word"};\\
&\quad \text{"have closed" ++ "Word"} >;\\
\mathit{close\_word}_4 &= <\text{"close" ++ "Microsoft Word"};\\
&\quad \text{"closed" ++ "Microsoft Word"};\\
&\quad \text{"have closed" ++ "Word"} >;\\
&\dots
\end{aligned}
$$

Variants explosion with tables - SOLUTION

GF

$$\textbf{lin}\ \mathit{close\_word} = \textbf{variants}\{\mathit{closeIt}\ \text{"Word"};\quad \mathit{closeIt}\ \text{"Microsoft Word"}\};$$

$$\textbf{oper}\ \mathit{closeIt}\ \mathit{obj} = \backslash\backslash \mathit{tense} \Rightarrow \mathit{close\_V}!\mathit{tense} +\!\!+\ \mathit{obj};$$

$$\mathit{close\_V} = \textbf{table}\ \mathit{Tense}\ \{\text{"close"}; \text{"closed"}; \text{"have closed"}; \ldots\};$$

Hints for Variants

- **Variants should be used with care**
- A variant in the wrong place could lead too many combinations, which is often not what you want
- A combinatorial explosion could kill the compiler with

  Out of memory

- Unnecessary combinations will slow down the parser
- Hint: use the command 'l -all'

Parameter Types Elimination

$$
\begin{aligned}
\textbf{lincat}\ NP\ &= \{s : Case \Rightarrow Str; g : Gender; n : Number; p : Person\}\\
\textbf{param}\ Case\ &= Nom|Acc|Dat;\\
Gender\ &= Masc|Fem|Neutr;\\
Number\ &= Sg|Pl;\\
Person\ &= P1|P2|P3;
\end{aligned}
$$

Table Types Elimination

A value of type $Case \Rightarrow Str$ looks like:

$$\textbf{table}\ \{Nom \Rightarrow s_1;\ Acc \Rightarrow s_2;\ Dat \Rightarrow s_3\}$$

We could replace it with tuple:

$$< s_1, s_2, s_3 >$$

Then in general type like $A \Rightarrow Str$ is equivalent to:

$$\underbrace{Str * Str * \ldots * Str}_{n \text{ times}}$$

where n is the number of values in the parameter type A.

Parameter Fields Elimination

GF

$$\textbf{lincat } NP = \{s : \ldots; g : Gender; n : Number; p : Person\}$$

GF Core

$$\begin{aligned}
\textbf{lincat } NP_1 &= Str * Str * Str; && -\ Masc; Sg, P1 \\
NP_2 &= Str * Str * Str; && -\ Masc; Sg, P2 \\
NP_3 &= Str * Str * Str; && -\ Masc; Sg, P3 \\
NP_4 &= Str * Str * Str; && -\ Masc; Pl, P1 \\
\ldots \\
NP_{18} &= Str * Str * Str; && -\ Neutr; Pl, P3
\end{aligned}$$

Note: The number of categories doesn't immediately affect the size of the compiled grammar

Counting Parametric Types

It is important to know how many possible values a given parameter type has because:

- This determines the number of fields in the core:

$$P \Rightarrow Str$$

- This determines the number of categories in the core:

$$\{ \ldots ; p : P\}$$

Counting the number of parameter values

Parameter Definition

$$\begin{aligned}\textbf{param } P \; &= P_1 \; Q_{11} \; Q_{12} \ldots Q_{1m_1} \\ &\mid P_2 \; Q_{21} \; Q_{22} \ldots Q_{2m_2} \\ &\quad \ldots \\ &\mid P_n \; Q_{n1} \; Q_{n2} \ldots Q_{nm_n}\end{aligned}$$

Values Count

$$\begin{aligned}\mathbb{C}(P) &= \mathbb{C}(Q_{11}) * \mathbb{C}(Q_{12}) \ldots \mathbb{C}(Q_{1m_1}) \\ &+ \mathbb{C}(Q_{21}) * \mathbb{C}(Q_{22}) \ldots \mathbb{C}(Q_{2m_2}) \\ &\quad \ldots \\ &+ \mathbb{C}(Q_{n1}) * \mathbb{C}(Q_{n2}) \ldots \mathbb{C}(Q_{nm_n})\end{aligned}$$

Counting Parametric Tables and Records

Parametric Records

$$\mathbb{C}(\{q_1 : Q_1; q_2 : Q_2 \ldots q_n : Q_n\}) = \mathbb{C}(Q_1) * \mathbb{C}(Q_2) \ldots \mathbb{C}(Q_n)$$

Parametric Tables

$$\mathbb{C}(P \Rightarrow Q) = \mathbb{C}(Q)^{\mathbb{C}(P)}$$

Warning: Exponentials should be avoided!!!

Hints for Parameters

- **Keep the lexicon compact:**

$$\textbf{lincat } N = \{s : NForm \Rightarrow Str;\ g : DGender\};$$

$$\begin{aligned} \textbf{param } & NForm \\ & = NF\ Number\ Species \\ & \mid NFSgDefNom \\ & \mid NFPlCount \\ & \mid NFVocative \end{aligned}$$

$$\begin{aligned} \textbf{param } & NForm \\ & = NF\ Number\ Species\ Case \\ & \mid NFPlCount \end{aligned}$$

- Linguistically accurate
- The irregularity is obvious

- Mathematically elegant
- Linguistically overgenerating

Comment

The lexical items are inflection tables. Duplication means overhead for every entry in the lexicon.

Hints for Parameters

- **Keep the syntax elegant:**

$$\textbf{lincat}\ CN = \{s : Number \Rightarrow Species \Rightarrow Case \Rightarrow Str\};$$

Comment

The syntactic rules are closed set. Compared to the lexicon this is a small set so it is not so important to make them compact. It is much more important to have clear easy to manipulate structure.

The efficiency of the parser is not affected by the number of fields in the linearization types.

Hints for Parameters

- Minimize the number of inherent parameters

$$\textbf{lincat } N = \{s : \mathit{NForm} \Rightarrow \mathit{Str};\ g : \mathit{DGender}\};$$

$$\begin{aligned}
&\textbf{param } \mathit{DGender} \\
&\quad = \mathit{DMasc}\ \mathit{Animacy} \\
&\quad \mid \mathit{DFem} \\
&\quad \mid \mathit{DNeutr}
\end{aligned}$$

$$\begin{aligned}
&\textbf{oper } \mathit{DGender} = \{g : \mathit{Gender}; a : \mathit{Animacy}\} \\
&\textbf{param } \mathit{Gender} \\
&\quad = \mathit{Masc} \\
&\quad \mid \mathit{Fem} \\
&\quad \mid \mathit{Neutr}; \\
&\textbf{param } \mathit{Animacy} = \mathit{Animate} \mid \mathit{Inanimate};
\end{aligned}$$

- Animacy matters only for Masc
- Animacy given for all genders

Linearization Rules Transformation

GF

$$
\begin{array}{l}
\textbf{fun } AdjCN : AP \rightarrow CN \rightarrow CN; \\
\textbf{lin } AdjCN\ ap\ cn = \{ \\
\quad s = ap.s!cn.g +\!\!+\ cn.s; \\
\quad g = cn.g \\
\};
\end{array}
$$

GF Core

$$\textbf{fun } AdjCN_1 : AP \rightarrow CN_1 \rightarrow CN_1; \qquad -Masc$$

$$\textbf{lin } AdjCN_1\ ap\ cn = < ap.p1 +\!\!+\ cn.p1 >$$

$$\textbf{fun } AdjCN_2 : AP \rightarrow CN_2 \rightarrow CN_2; \qquad -Fem$$

$$\textbf{lin } AdjCN_2\ ap\ cn = < ap.p2 +\!\!+\ cn.p1 >$$

$$\textbf{fun } AdjCN_3 : AP \rightarrow CN_3 \rightarrow CN_3; \qquad -Neutr$$

$$\textbf{lin } AdjCN_3\ ap\ cn = < ap.p3 +\!\!+\ cn.p1 >$$

Counting Linearization Rules

In general linearization rule like:

$$\textbf{fun } f : A_1 \to A_2 \to \ldots \to A_n \to A;$$

produces $\mathbb{C}(f)$ rules in the core

$$\mathbb{C}(f) = \mathbb{C}(A_1) * \mathbb{C}(A_2) * \ldots * \mathbb{C}(A_n)$$

Comment

The number of rules could be reduced by reducing the number of parameters in the linearization types. The count is also reduced by the optimizations in the compiler.

No pattern matching

Allowed

$$\begin{array}{l} \textbf{oper}\ \mathit{mkN}\ \mathit{noun} = \textbf{case}\ \mathit{noun}\ \textbf{of}\ \{ \\ \quad \_ + \text{"s"} \Rightarrow < \mathit{noun}, \mathit{noun} + \text{"es"} >; \\ \quad \_ \qquad\ \Rightarrow < \mathit{noun}, \mathit{noun} + \text{"s"} > \\ \quad \}; \end{array}$$

Not Allowed

$$\begin{array}{l} \textbf{lin}\ \mathit{DetCN}\ \mathit{det}\ \mathit{cn} = \textbf{case}\ \mathit{det}.\mathit{s}\ \textbf{of}\ \{ \\ \qquad \text{""} \Rightarrow \ldots \\ \qquad \_ \Rightarrow \ldots \\ \quad \} \end{array}$$

Hint: use parameter which says whether the string is empty

No gluing

Allowed

$$
\begin{aligned}
&\textbf{lin } \mathit{DetCN}\ \mathit{det}\ \mathit{cn} = \textbf{case } \mathit{det}.\mathit{spec} \textbf{ of } \{ \\
&\quad \dots \\
&\quad\quad \mathit{Indefinite} \Rightarrow \textbf{case } \mathit{cn}.g \textbf{ of } \{\mathit{Utr} \Rightarrow \text{"en"}; \mathit{Neutr} \Rightarrow \text{"ett"}\} +\!\!+\ \mathit{cn}.s \\
&\}
\end{aligned}
$$

Not Allowed

$$
\begin{aligned}
&\textbf{lin } \mathit{DetCN}\ \mathit{det}\ \mathit{cn} = \textbf{case } \mathit{det}.\mathit{spec} \textbf{ of } \{ \\
&\quad\quad \mathit{Definite} \Rightarrow \mathit{cn}.s + \textbf{case } \mathit{cn}.g \textbf{ of } \{\mathit{Utr} \Rightarrow \text{"en"}; \mathit{Neutr} \Rightarrow \text{"et"}\}; \\
&\quad \dots \\
&\}
\end{aligned}
$$

Hint: for agglutinative languages (Turkish, Finnish, Estonian, Hungarian, ...) use custom lexer

Agglutination

- Some languages have pottentially infinite set of words:

  Turkish:

  anlamiyorum = anla(root) -mi(negation) -yor(continuous) -um(first person)
  I don't understand

- The grammar could be based on roots and suffixes instead of on words:

  "anla" ++ "&+" ++ "mi" ++ "&+" ++ "yor" ++ "&+" ++ "um"

- The lexer/unlexer are responsible to produce the real words

Optimizations

Three main optimizations reduce the exponential size of the grammar:

- Common Subexpressions Optimization
- Common Functions Optimization
- Coercion Rules

Note: the optimizations cannot be expressed in GF Core. PMCFG is needed.

Common Subexpressions Optimization

GF Core

$$\begin{aligned} \textbf{lin}\ u\ x\ y &= < x.p1, x.p2 +\!\!+\ y.p1 > \\ v\ x\ y &= < "a", x.p2 +\!\!+\ y.p1 > \end{aligned}$$

PMCFG

$$\begin{aligned} F_1 &:= (S_1, S_2) \quad [u] \\ F_2 &:= (S_3, S_2) \quad [v] \end{aligned}$$

$$\begin{aligned} S_1 &:= \langle 0; 0 \rangle \\ S_2 &:= \langle 0; 1 \rangle \langle 1; 0 \rangle \\ S_3 &:= "a" \end{aligned}$$

Common Subexpressions Optimization in the Lexicon

GF Core

lin *good_A* = < "dobăr" , "dobra" , "dobro" , "dobre" >
beautiful_A = < "hubav" , "hubava" , "hubavo" , "hubavo" >

PMCFG

$$F_1 := (S1, S2, S3, S4) \quad [good\_A]$$
$$F_2 := (S5, S6, S7, S7) \quad [beautiful\_A]$$

$$
\begin{array}{ll}
S_1 := \text{"dobăr"} & S_5 := \text{"hubav"} \\
S_2 := \text{"dobra"} & S_6 := \text{"hubava"} \\
S_3 := \text{"dobro"} & S_7 := \text{"hubavo"} \\
S_4 := \text{"dobre"} &
\end{array}
$$

Common Functions Optimization

The function symbols in PMCFG could be reused in different productions

PMCFG

$$C_1 \leftarrow F_1[C_2, C_3]$$
$$C_1 \leftarrow F_1[C_4, C_5]$$

$$F_1 := (S1, S2, S3, S4) \quad [u]$$

Coercion Rules

PMCFG

$$C_1 \leftarrow F_1[C_2, C_{31}, C_{41}, C5]$$
$$C_1 \leftarrow F_1[C_2, C_{32}, C_{41}, C5]$$
$$C_1 \leftarrow F_1[C_2, C_{33}, C_{41}, C5]$$
$$C_1 \leftarrow F_1[C_2, C_{31}, C_{42}, C5]$$
$$C_1 \leftarrow F_1[C_2, C_{32}, C_{42}, C5]$$
$$C_1 \leftarrow F_1[C_2, C_{33}, C_{42}, C5]$$
$$C_1 \leftarrow F_1[C_2, C_{31}, C_{43}, C5]$$
$$C_1 \leftarrow F_1[C_2, C_{32}, C_{43}, C5]$$
$$C_1 \leftarrow F_1[C_2, C_{33}, C_{43}, C5]$$

Coercion Rules

PMCFG

$$
\begin{aligned}
C_1 &\leftarrow F_1[C_2, C_3, C_4, C_5] \\
C_3 &\leftarrow \_[C_{31}] \\
C_3 &\leftarrow \_[C_{32}] \\
C_3 &\leftarrow \_[C_{33}] \\
C_4 &\leftarrow \_[C_{41}] \\
C_4 &\leftarrow \_[C_{42}] \\
C_4 &\leftarrow \_[C_{43}]
\end{aligned}
$$

Debugging the Compiler

You can dump the PMCFG representation of the grammar with the following command:

```
c:\gf> gf -make -output-format=pmcfg_pretty LangMy.gf
Reading Lang.pgf...
Refusing to overwrite Lang.pgf
Writing LangEng.pmcfg...
```

This will produce one file with extension .pmcfg with four interesting sections:

- productions
- functions
- sequences
- startcats

Thank You

Thank You and Have Fun !!!