The mechanics of GF

Krasimir Angelov

University of Gothenburg

August 22, 2013

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 - $\{a^n b^n c^n | n \geq 0\}$:

GF Core Language ≡ PMCFG

The parser uses a language which is a subset 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 $+$)

$\{a^n b^n c^n | n \geq 0\}$ in PMCFG

cat N, S

fun $z : N$

$s : N \rightarrow N$

$c : N \rightarrow S$

lincat $N = Str * Str * Str$

$S = Str$

lin $z \quad = < "", "", "" >$

$s\ x \quad = < "a" +\!\!+\ x.p1, "b" +\!\!+\ x.p2, "c" +\!\!+\ x.p3 >$

$c\ x \quad = \ x.p1 +\!\!+\ x.p2 +\!\!+\ x.p3$

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

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

GF Core

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

Note: the pattern matching in mkN was eliminated

Variants elimination

The variants are just expanded:

GF

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

GF Core

$$\textbf{lin}\ girl\_N_1 = mkN\ \text{"tjej"};$$
$$girl\_N_2 = mkN\ \text{"flicka"};$$

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 the 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

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

Linearization Rules Transformation

GF

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

GF Core

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

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

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

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

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

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

No pattern matching

Allowed

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

Not Allowed

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

Hint: use parameter which says whether the string is empty

No gluing

Allowed

$$\textbf{lin}\ \mathit{DetCN}\ \mathit{det}\ \mathit{cn} = \textbf{case}\ \mathit{det.spec}\ \textbf{of}\ \{$$

$$\ldots$$

$$\mathit{Indefinite} \Rightarrow \textbf{case}\ \mathit{cn.g}\ \textbf{of}\ \{\mathit{Utr} \Rightarrow \text{"en"};\ \mathit{Neutr} \Rightarrow \text{"ett"}\} +\!\!+\ \mathit{cn.s}$$

$$\}$$

Not Allowed

$$\textbf{lin}\ \mathit{DetCN}\ \mathit{det}\ \mathit{cn} = \textbf{case}\ \mathit{det.spec}\ \textbf{of}\ \{$$

$$\mathit{Definite} \Rightarrow \mathit{cn.s} + \textbf{case}\ \mathit{cn.g}\ \textbf{of}\ \{\mathit{Utr} \Rightarrow \text{"en"};\ \mathit{Neutr} \Rightarrow \text{"et"}\};$$

$$\ldots$$

$$\}$$

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

Agglutinatination

- 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

Summary

- GF $\Rightarrow$ (GF Core $\equiv$ PMCFG)
- Linearization is overload resolution
- Parsing is search