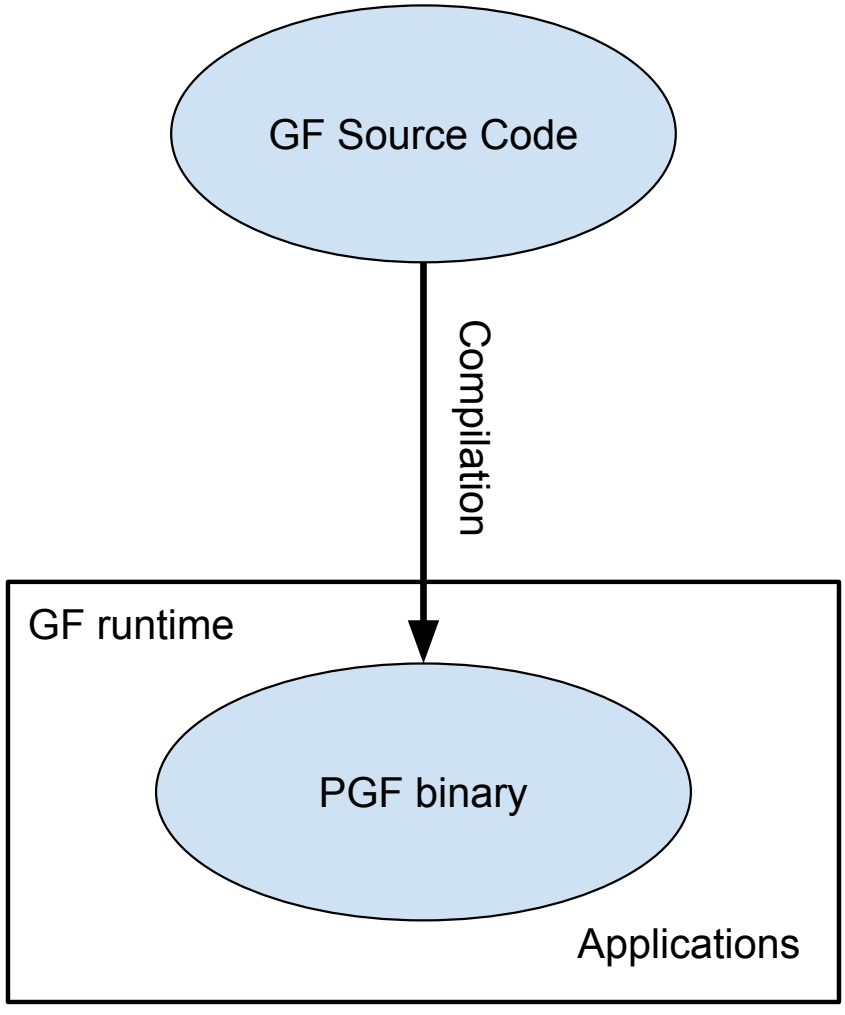
The mechanics of GF

Krasimir Angelov

University of Gothenburg

August 21, 2017

Compilation



$\text{PMCFG}^+ \equiv$ GF Core Language

The runtime understands only a subset of GF:

- The linearization types are flat tuples of strings:

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

- The tuples contain 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 $+$)

Operations elimination

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

GF

oper *mkN noun* = **case** *noun* **of** {
_ + "s" ⇒ < *noun*, *noun* + "es" >;
_ ⇒ < *noun*, *noun* + "s" >
};
lin *apple_N* = *mkN* "apple";
plus_N = *mkN* "plus";

GF Core

lin *apple_N* = < "apple", "apples" >;
plus_N = < "plus", "pluses" >;

Note: the pattern matching in mkN was eliminated

Variants elimination

The variants are just expanded:

GF

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

GF Core

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

Parameter Types Elimination

$$
\begin{aligned}
\textbf{lincat}\ & CN = \{s : Case \Rightarrow Str; g : Gender\} \\
\textbf{param}\ & Case = Nom|Acc|Dat; \\
& Gender = Masc|Fem|Neutr;
\end{aligned}
$$

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, a 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

$$\textbf{lincat } CN = \{s : \ldots ; g : Gender\}$$

GF Core

$$\begin{aligned} \textbf{lincat } NP_1 &= Str * Str * Str; && -- Masc \\ NP_2 &= Str * Str * Str; && -- Fem \\ NP_3 &= Str * Str * Str; && -- Neutr \end{aligned}$$

Linearization Rules Transformation

GF

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

GF Core

$$\begin{aligned}
&\textbf{fun } AdjCN_1 : AP \to CN_1 \to CN_1; && -Masc\\
&\textbf{lin } AdjCN_1\ ap\ cn = < ap.p1 \mathbin{++} cn.p1 >\\
&\\
&\textbf{fun } AdjCN_2 : AP \to CN_2 \to CN_2; && -Fem\\
&\textbf{lin } AdjCN_2\ ap\ cn = < ap.p2 \mathbin{++} cn.p1 >\\
&\\
&\textbf{fun } AdjCN_3 : AP \to CN_3 \to CN_3; && -Neutr\\
&\textbf{lin } AdjCN_3\ ap\ cn = < ap.p3 \mathbin{++} cn.p1 >
\end{aligned}$$

No pattern matching

Allowed

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

Not Allowed

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

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

$$\textbf{lin}\ \mathit{DetCN}\ \mathit{det}\ \mathit{cn} = \textbf{case}\ \mathit{det.spec}\ \textbf{of}\ \{$$
$$\mathit{Definite} \Rightarrow \mathit{cn.s} + \textbf{case}\ \mathit{cn.g}\ \textbf{of}\ \{\mathit{Utr} \Rightarrow \text{"en"}\,;\ \mathit{Neutr} \Rightarrow \text{"et"}\};$$
$$\ldots$$
$$\}$$

Extension: BIND

- Agglutinatinative languages have pottentially infinite set of words:

  Turkish:

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

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

$$\text{"anla"} +\!\!+ \mathit{BIND} +\!\!+ \text{"mı"} +\!\!+ \mathit{BIND} +\!\!+ \text{"yor"} +\!\!+ \mathit{BIND} +\!\!+ \text{"um"}$$

- The parser uses BIND to do the right segmentation

Extension: *SOFT_BIND* and *SOFT_SPACE*

- *SOFT_BIND*

Expression	"Hi" ++ "John" ++ *SOFT_BIND* ++ ","
Accepts	"Hi John," "Hi John␣,", ...
Generates	"Hi John,"

- *SOFT_SPACE*

Expression	"10" ++ *SOFT_SPACE* ++ "000"
Accepts	"10 000" "10000", ...
Generates	"10 000"

Extension: CAPIT and ALL_CAPIT

- Capitalization in German
  $CAPIT \mathbin{+\!\!+}$ "aktions" $\mathbin{+\!\!+} BIND \mathbin{+\!\!+}$ "plan" $\Rightarrow$ "Aktionsplan"
- Acronyms
  $ALL\_CAPIT \mathbin{+\!\!+}$ "ibm" $\Rightarrow$ "IBM"

Extension: Case-insensitive Parsing

Enable case-insensitive parsing

flags *case_sensitive* = "off"

- All words in the grammar must be in lower-case
- Capitalization is achieved only with *CAPIT* and *ALL_CAPIT*
- The switching off *case_sensitive* tells the parser to lower-case the input

Extension: nonExist

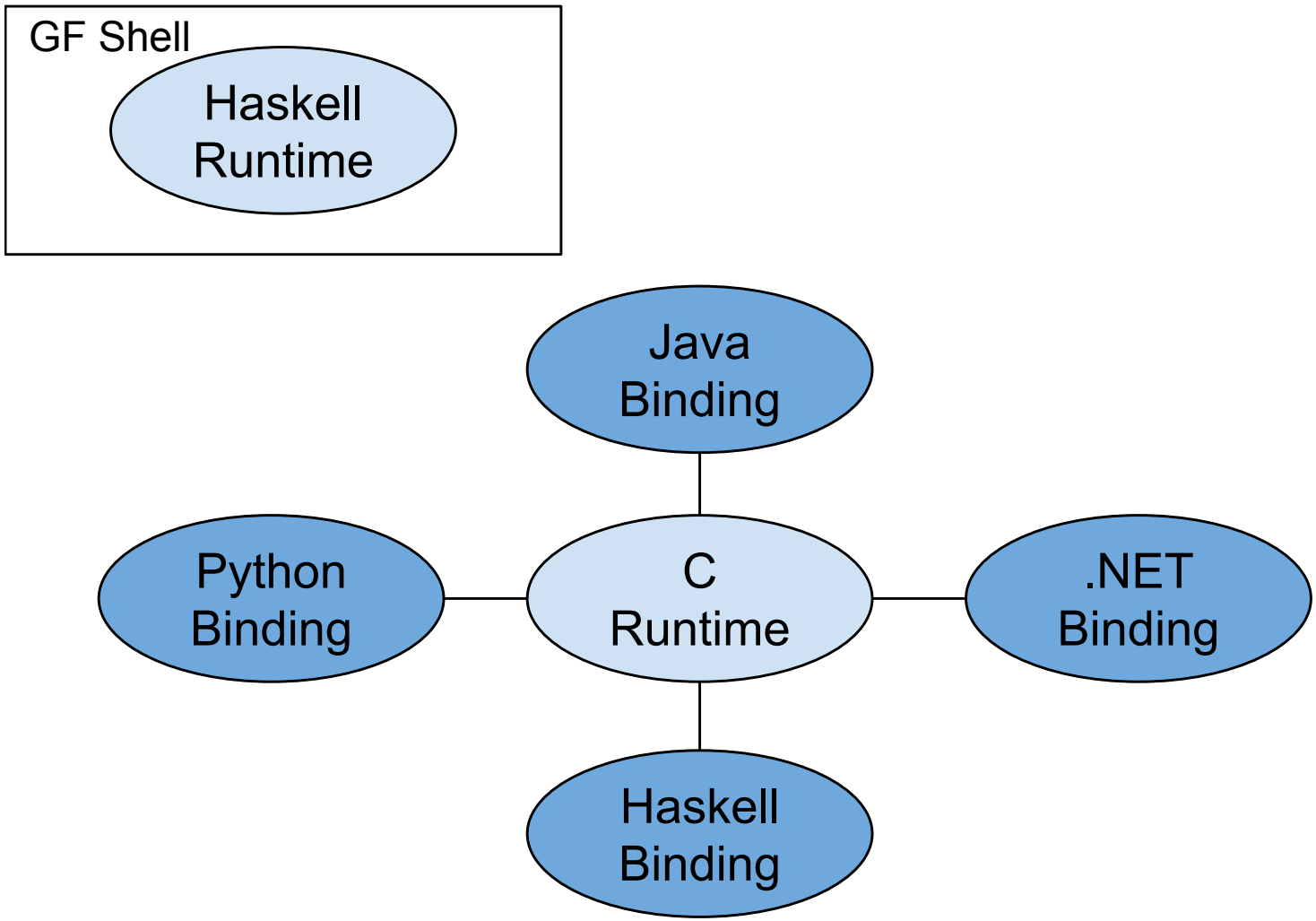
Lexical gaps

$$
\begin{aligned}
&\textbf{lin } \mathit{foo\_N} = \textbf{table } \{ \\
&\quad \mathit{Sg} \Rightarrow \text{"foo"} ; \\
&\quad \mathit{Pl} \Rightarrow \mathit{nonExist} ; \\
&\}
\end{aligned}
$$

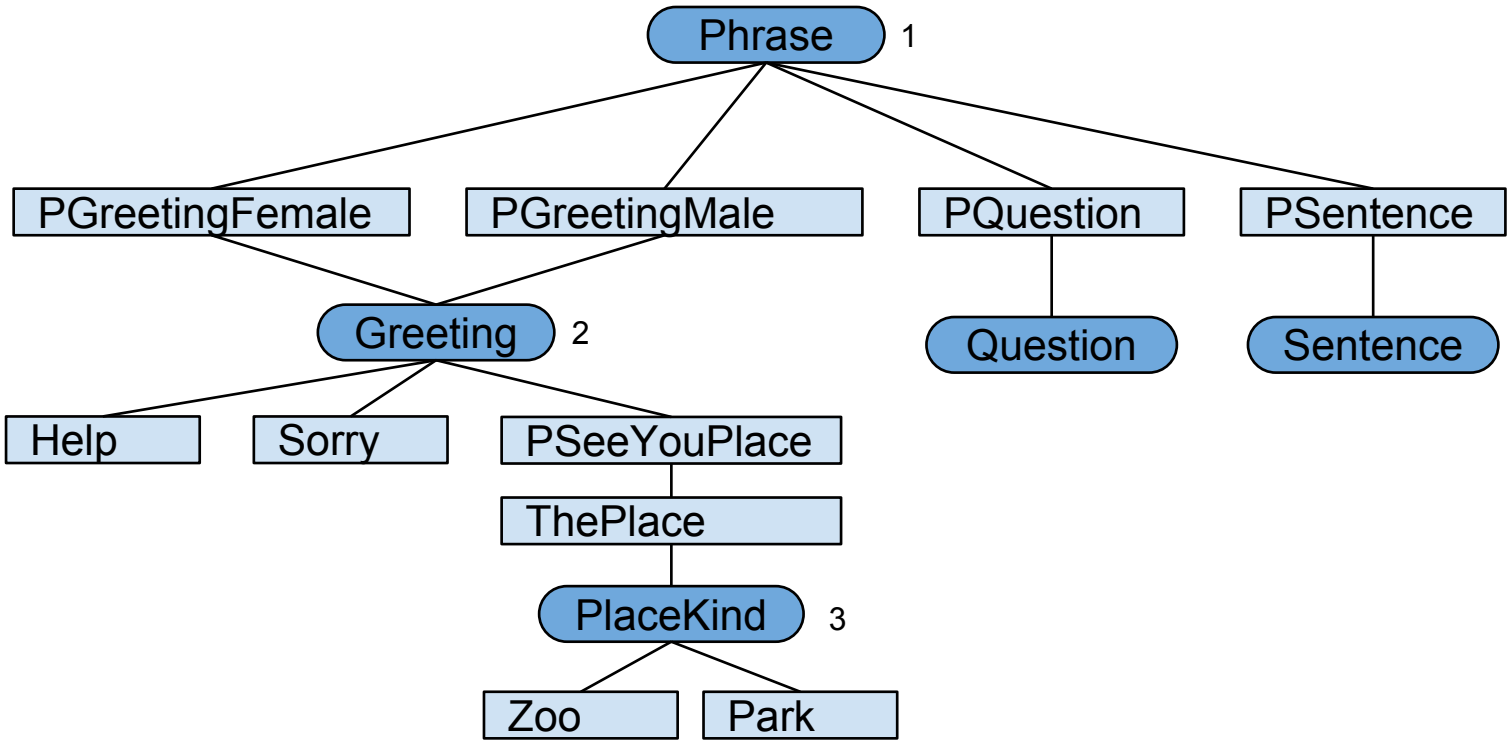
nonExist

- generates exception in linearization
- blocks parsing with the corresponding function

Types of Runtime



Choice Graph



Summary

- GF $\Rightarrow$ (GF Core $\equiv$ PMCFG)
- Embedded Grammars:
  Haskell, Python, Java, C, .NET
- User Interfaces