The Problem of Morphological Variation
(Ranta 2011)

Example

(1) this wine is Italian

(2) these wines are Italian

fun
    Pred : Item_Sg -> Quality -> Comment ;
    Pred : Item_Pl -> Quality -> Comment ;
    This, that : Kind_Sg -> Item_Sg ;
    These : Kind_Pl -> Item_Pl ;

lin
    Pred item quality = item ++ "is" ++ quality ;
    Pred item quality = item ++ "are" ++ quality ;

In Italian, gender is to be added as well
Example (Parameters)

```javascript
param Number = Sg | Pl
```

Example (Tables)

<table>
<thead>
<tr>
<th>number</th>
<th>form</th>
</tr>
</thead>
<tbody>
<tr>
<td>singular</td>
<td>pizza</td>
</tr>
<tr>
<td>plural</td>
<td>pizze</td>
</tr>
</tbody>
</table>

```javascript
table {Sg => "pizza" ; Pl => "pizze"}
```
### Definition (Hypotheses and contexts)

A *context* is a sequence of *hypotheses*, i.e., variable-type pairs. It is written:

\[(x : T)\]

### Definition (Parameter Type Definitions)

A *parameter type definition*

\[\text{param } P = C_1 G_1 \mid \ldots \mid C_n G_n\]

defines a parameter type \( P \) with the *parameter constructors* \( C_1, \ldots, C_n \) with their respective contexts \( G_1, \ldots, G_n \).

- Dependent types are not available in parameter type definitions, so the use of variables is never necessary.
- Types in the context must themselves be parameter types.
Definition (Parameter Types)

Parameter types

- Given the judgment `param P`, `P` is a parameter type.
- A record type of parameter types is a parameter type.
- `Ints n` (integers from 0 to `n`) is a parameter type.

Parameter type may not be recursive.

Example (Inflection of an Italian adjective)

```plaintext
table {
    Masc => table {Sg => "caldo" ; Pl => "caldi"}
    Fem => table {Sg => "calda" ; Pl => "calde"}
}
```

```plaintext
table {Sg => pizza ; Pl =>"pizze"} ! Pl ↓ "pizze"
```
Variable and Inherent Features

Variable Features
In English or Italian, nouns have both singular and plural forms. The number is a variable feature of nouns.

Inherent Features
In Italian, a noun is either masculine or feminine. The gender is an inherent feature of Italian nouns.

Example (Agreement)
- Adjectival modification of nouns: the variable gender of the adjective is determined by the inherent gender of the noun.
- Determination: the variable number of the noun is determined by the inherent number of the determiner.

⇒ Asymmetry (contrary to unification grammars).
Types

**Linearization Types (Ljunglöf 2004)**

- Str, the type of strings, is a linearization type.
- If $T_1, \ldots, T_n$ are linearization types or parameter types, and at least one of them is a linearization type, then the record type $\{r_1 : T_1; \ldots; r_n : T_n\}$ is a linearization type.
- If $T$ is a linearization type and $P$ is a parameter type, then $P \Rightarrow T$ is a linearization type.
Linearization Types and Agreement

Example (English)

```plaintext
param
    Number = Sg | Pl ;

lincat
    Comment = {s: Str} ;  -- a full sentence
    Item = {s: Str ; n: Number} ;  -- a noun phrase
    Kind = {s: Number => Str} ;  -- a noun
    Quality = {s: Str} ;  -- an adjective

lin
    This kind = {s = "This" ++ kind.s ! Sg ; n = Sg } ;
    Mod qual kind = {s = table {n => qual.s ++ (kind.s ! n)}} ;

    Pred item qual =
        { s = item.s ++
            table { Sg => "is" ; Pl => "are" } ! item.n ++
            qual.s } ;
```
Abstract Types
(Ljunglöf 2004)

**Definition (Category Declarations)**

A *category declaration*  
\[ \text{cat } C = CG \]

defines the basic types of abstract syntax. A basic type is formed from a category by giving values to all variables in the *context* \( G \). If the context is empty, the basic type looks the same as the category itself. Otherwise, application syntax is used: \( C a_1 \ldots a_n \)

**Definition (Function declaration)**

A *function declaration*  
\[ \text{fun } f : T \]

defines the *syntactic constructors of abstract syntax*.

An abstract syntax is *context-free* if it has neither dependent types nor higher-order functions.
Questions

- Parsing with non linearity (and deletion)?

  context-free GF is strongly equivalent to PMCFG. This equivalence is shown by giving an algorithm converting cf-GF grammars into PMCFG grammars recognizing the same language; and by showing that parse results can be converted back efficiently. The conversion algorithm consists of enumerating all parameter instantiations in a linearization, and then moving the instantiated parameters to the abstract categories. Enumerating all instantiations may lead to an exponential increase of the grammar size. Therefore two alternative conversion algorithms are given, which do not enumerate all possible instantiations, but instead try to only instantiate when it is necessary. (Ljunglöf 2004)

- Parsing algorithms (Ljunglöf 2004; Angelov 2009; Ranta 2007b) and differences with (Salvati 2010).

- Differences between features at the object or at the abstract level (Ranta 2007a).

- Permutative conversions?
Abstract and Concrete Syntax

Linearization

\[ \text{lincat } C = L \quad \text{C has the linearization type of } L \]
\[ \text{lin } f \ x_1 \ldots x_\delta = t \quad \text{f has the linearization function } \lambda x_1 \ldots x_\delta. t \]
\[ \text{lindef } C \ x = t \quad \text{C has default linearization } \lambda x. t \]

\[
\begin{align*}
(C \ a_1 \ldots a_n)^\circ &= L \text{ if lincat } C = L \\
((x_1 : A_1) \to \cdots \to (x_n : A_n) \to A)^\circ &= \text{Str} \to \cdots \to A^\circ
\end{align*}
\]
Canonical Linearization

The concrete syntax of any GF grammar can be partially evaluated to a grammar in canonical form (Ranta 2004):

- All local and global definitions disappear, as well as function applications;
- all tables are instantiated (all patterns are variable-free);
- Hierarchical parameters can be flattened (assumption that parameters are declared by giving a finite set of parameter types
Canonically Linearized Term

Definition (Canonical Term)

A canonical linearization term is of the following form:

- A string constant is of type \( \text{Str} \); and a concatenation \( s_1 ++ s_2 : \text{Str} \) whenever \( s_1, s_2 : \text{Str} \);
- A constant parameter \( p : P \), whenever \( p \in P \);
- A record \( \{ r_1 = \phi_1; \ldots; r_n = \phi_n \} \) is of type \( T = \{ r_1 : T_1; \ldots; r_n : T_n \} \) whenever each \( \phi_i : T_i \);
- A record projection \( \phi. r_i : T_i \) whenever \( \phi \) is of the record type \( T = \{ r_1 : T_1; \ldots; r_n : T_n \} \);
- A table \( [ p_1 \Rightarrow \phi_1; \ldots; p_n \Rightarrow \phi_n ] \) is of type \( P \Rightarrow T \) whenever \( P = \{ p_1, \ldots, p_n \} \) and each \( \phi_i : T \);
- A table selection \( \phi ! \psi : T \) whenever \( \phi : P \Rightarrow T \) and \( \psi : P \);
- An argument variable \( x_i : B_i \). 

\( \text{Str} \) denotes the type of strings.
Example (Ljunglöf 2004, p.47)

$$vp^\circ(x, y) = \{ s = [z \Rightarrow x.s ! z ++ y.s] \}$$

and

$$vp^\circ(x, y) = \{ s = [Sg \Rightarrow x.s ! Sg ++ y.s; P1 \Rightarrow x.s ! P1 ++ y.s] \}$$

Computation Rules

$$s_1 ++ s_2 = s_1s_2$$

$$\{\ldots; r = t; \ldots \}.r = t$$

$$[\ldots; p \Rightarrow t; \ldots]! p = t$$
Generalized Context-Free Grammars (Pollard 1984)

Abstract Grammar A tuple \((\mathcal{C}, S, \mathcal{F}, \mathcal{R})\). For each function symbol \(f \in \mathcal{F}\) there is an associated context-free syntax rule:

\[
A \rightarrow f[B_1, \ldots, B_\delta]
\]

Concrete Interpretation To each function symbol \(f i\) associated a partial linearization function \(f^\circ\)

\[
f^\circ \in b_1^\circ \times \cdots \times B_\delta^\circ \rightarrow A^\circ
\]

Variable-Free Notation For a rule \(A \rightarrow f[A_1, \ldots, A_\delta]\) and a linearization \(f^\circ(x_1, \ldots, x_\delta) = \phi\) can be rewritten as:

\[
A \rightarrow f[A_1, \ldots, A_\delta] := \hat{\phi}
\]

where each occurrence of the variable \(x_i\) in \(\phi\) is replaced by the term \(A_i\) in \(\hat{\phi}\).

Example

\[
A \rightarrow f[B^1, A, B^2] := aB^1 AbB^3
\]

\[
f^\circ(x, y, z) = axyb\]
Definition (Part of a term)

If there is a bijective function $\pi : T \rightarrow P_1 \times \cdots \times P_n$, $\Pi$ is said to form a partition of $T$.
Given a term $t : T$, a projected term $p_k : P_k$ is a part of $t$ if there is some partition $\pi$ of $T$ such that $p_k = \pi_k(\pi(t))$.

Subclasses of GCFG

Given a GCFG rule $A \rightarrow f[B_1, \ldots, B_\delta]$ with its linearization $f(x_1, \ldots, x_\delta) = \phi$, the rule is said:

- Parallel if some part of $x_i$ is mentioned twice in $\phi$
- Linear if no part of $x_i$ is mentioned twice
- Erasing if some part of $x_i$ is not mentioned at all in $\phi$
- Non erasing if all parts of $x_i$ are mentioned in $\phi$
- Suppressing if $x_i$ is not mentioned at all in $\phi$
Parallel Multiple Context-Free Grammars (Kasami, Seki, and Fujii 1989; Seki et al. 1991)

Definition (PMCFG)

A GCFG such that:

- Linearization types are restricted to tuples of strings
- The only allowed operations in linearization functions are tuple projections and string concatenation

\[
f^\circ \left( \langle x_1,1, \ldots, x_1,d_1 \rangle, \ldots, \langle x_\delta,1, \ldots, x_\delta,d_\delta \rangle \right) = \langle \alpha_1, \ldots, \alpha_n \rangle
\]

where each \( \alpha_i \) is a sequence of variables \( x_{j,k} \) or constant strings.

Or, in record notation,

\[
f^\circ (x_1, \ldots, x_\delta) = \{ 1 = \hat{\alpha}_1, \ldots; \mathbf{d} = \hat{\alpha}_d \}
\]

where each \( x_{j,k} \) in \( \alpha_i \) is replaced by the projection \( x_{j,k} \) in \( \hat{\alpha}_i \).
Theorem (Ljunglöf 2004)

Every PMCFG is equivalent to a context-free GF grammar.

Theorem (Ljunglöf 2004)

And vice-versa.

Proof.

Easy if, in GF grammars:

- all tables and table selections are instantiated (canonical linearization)
- records containing parameters are not allowed

If there are records that contains a parameter (e.g., $d_m^o = \{ s = ' many'; n = P1 \}$), requires more work...
What about ACG?

Theorem

*2nd-order ACG are equivalent to linear PMCFG.*

What to look at next?

“Building PMCFG Parsers as Datalog Program Transformations” (Ball et al. 2014).

Questions: What extensions to ACGs to make them equivalent to these programs?
Features at the object level (Ranta 2007a)

See the demo.
Bibliography I


References

Bibliography II


