Expressivity of Unification Grammars

Shuly Wintner
Department of Computer Science
University of Haifa
Haifa, Israel
shuly@cs.haifa.ac.il

LTI CMU, May 2006
Basic notions

- A *signature* consisting of finite, non-empty sets of features and atoms of atoms
- *Attribute-value matrices* (AVMs) used to depict feature structures, which are sets of ⟨feature, value⟩ pairs
- *Reentrancy tags* (or variables) are used to indicate co-indexing
- *Multi-AVMs* are sequences of AVMs with possible reentrancies among different members of the sequence.
- A *grammar* is a set of production rules, each of which is a multi-AVM, and a *lexicon* which associates a set of AVMs with each word.
### Basic notions

**Example: Lexicon**

<table>
<thead>
<tr>
<th>Word</th>
<th>Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>lamb</td>
<td>CAT : n, NUM : sg, CASE : []</td>
</tr>
<tr>
<td>love</td>
<td>CAT : v, SUBCAT : [CAT : np, CASE : acc], NUM : pl</td>
</tr>
<tr>
<td>give</td>
<td>CAT : v, SUBCAT : [CAT : np, CASE : acc], [CAT : np], NUM : pl</td>
</tr>
</tbody>
</table>
Basic notions

Example: Grammar rules

\[
\begin{align*}
\text{[CAT : } s\text{]} & \rightarrow \text{[CAT : } np\text{]} \\
& \quad \text{[CAT : } v\text{]} \\
\text{[CAT : } np\text{]} & \rightarrow \text{[CAT : } d\text{]} \\
& \quad \text{[CAT : } n\text{]} \\
\text{[CAT : } np\text{]} & \quad \text{[CAT : } d\text{]} \\
& \quad \text{[CAT : } n\text{]} \\
\text{[CAT : nom]} & \quad \text{[CAT : nom]} \\
& \quad \text{[CAT : nom]} \\
\text{[CAT : nom]} & \quad \text{[CAT : nom]} \\
& \quad \text{[CAT : nom]} \\
\text{[CAT : nom]} & \quad \text{[CAT : nom]} \\
& \quad \text{[CAT : nom]} \\
\end{align*}
\]
Expressiveness of unification grammars

- Just how expressive are unification grammars?
- What is the class of languages generated by unification grammars?
Trans-context-free languages

- A grammar, $G_{abc}$, for the language $L = \{a^n b^n c^n \mid n > 0\}$.
- Feature structures will have two features: CAT, which stands for category, and T, which “counts” the length of sequences of a-s, b-s and c-s.
- The “category” is $ap$ for strings of a-s, $bp$ for b-s and $cp$ for c-s. The categories $at$, $bt$ and $ct$ are pre-terminal categories of the words a, b and c, respectively.
- “Counting” is done in unary base: a string of length $n$ is derived by an AVM (that is, an multi-AVM of length 1) whose depth is $n$.
- For example, the string $bbb$ is derived by the following AVM:

$$\begin{bmatrix} \text{CAT} : & bp \\ T : [ & T : [ & T : \text{end}]] \end{bmatrix}$$
Trans-context-free languages

Example: A unification grammar for the language $\{a^nb^nc^n \mid n > 0\}$

The signature of the grammar consists in the features $\text{CAT}$ and $T$ and the atoms $s$, $ap$, $bp$, $cp$, $at$, $bt$, $ct$ and $end$. The terminal symbols are, of course, $a$, $b$ and $c$. The start symbol is the left-hand side of the first rule.

$\rho_1 : \begin{array}{l}
\text{CAT : } s \\
T : 1
\end{array} \rightarrow \begin{array}{l}
\text{CAT : } ap \\
T : 1
\end{array} \begin{array}{l}
\text{CAT : } bp \\
T : 1
\end{array} \begin{array}{l}
\text{CAT : } cp \\
T : 1
\end{array}$

$\rho_2 : \begin{array}{l}
\text{CAT : } ap \\
T : 1
\end{array} \rightarrow \begin{array}{l}
\text{CAT : } at
\end{array} \begin{array}{l}
\text{CAT : } ap \\
T : 1
\end{array}$

$\rho_3 : \begin{array}{l}
\text{CAT : } ap \\
T : end
\end{array} \rightarrow \begin{array}{l}
\text{CAT : } at
\end{array}$
Example: (continued)

\[\begin{align*}
\rho_4 & : \begin{bmatrix} \text{CAT} : \quad bp \\ T : \quad [T : 1] \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} : \quad bt \\ T : \quad [1] \end{bmatrix} & \begin{bmatrix} \text{CAT} : \quad bp \\ T : \quad [1] \end{bmatrix} \\
\rho_5 & : \begin{bmatrix} \text{CAT} : \quad bp \\ T : \quad end \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} : \quad bt \end{bmatrix} \\
\rho_6 & : \begin{bmatrix} \text{CAT} : \quad cp \\ T : \quad [T : 1] \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} : \quad ct \\ T : \quad [1] \end{bmatrix} & \begin{bmatrix} \text{CAT} : \quad cp \\ T : \quad [1] \end{bmatrix} \\
\rho_7 & : \begin{bmatrix} \text{CAT} : \quad cp \\ T : \quad end \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} : \quad ct \end{bmatrix}
\end{align*}\]
### Example: (continued)

<table>
<thead>
<tr>
<th>[ \text{CAT} : \ at ]</th>
<th>$\rightarrow$</th>
<th>$a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \text{CAT} : \ bt ]</td>
<td>$\rightarrow$</td>
<td>$b$</td>
</tr>
<tr>
<td>[ \text{CAT} : \ ct ]</td>
<td>$\rightarrow$</td>
<td>$c$</td>
</tr>
</tbody>
</table>
Example: Derivation sequence of $a^2b^2c^2$

Start with a form that consists of the start symbol,

$$\sigma_0 = \left[\text{CAT} : s \right].$$

Only one rule, $\rho_1$, can be applied to the single element of the multi-AVM in $\sigma_0$, yielding:

$$\sigma_1 = \left[\begin{array}{c}
\text{CAT} : ap \\
T : 1
\end{array}\right] \quad \left[\begin{array}{c}
\text{CAT} : bp \\
T : 1
\end{array}\right] \quad \left[\begin{array}{c}
\text{CAT} : cp \\
T : 1
\end{array}\right]$$
Example: (continued)

Applying $\rho_2$ to the first element of $\sigma_1$:

$$\sigma_2 = \begin{bmatrix} \text{CAT}: \text{at} \\ T: 1 \end{bmatrix} \begin{bmatrix} \text{CAT}: \text{ap} \\ T: 1 \end{bmatrix} \begin{bmatrix} \text{CAT}: \text{bp} \\ T: 1 \end{bmatrix} \begin{bmatrix} \text{CAT}: \text{cp} \\ T: 1 \end{bmatrix}$$

Choose the third element in $\sigma_2$ and apply the rule $\rho_4$:

$$\sigma_3 = \begin{bmatrix} \text{CAT}: \text{at} \\ T: 1 \end{bmatrix} \begin{bmatrix} \text{CAT}: \text{ap} \\ T: 1 \end{bmatrix} \begin{bmatrix} \text{CAT}: \text{bt} \\ T: 1 \end{bmatrix} \begin{bmatrix} \text{CAT}: \text{bp} \\ T: 1 \end{bmatrix} \begin{bmatrix} \text{CAT}: \text{cp} \\ T: 1 \end{bmatrix}$$

Apply $\rho_6$ to the fifth element of $\sigma_3$:

$$\sigma_4 = \begin{bmatrix} \text{CAT}: \text{at} \\ T: 1 \end{bmatrix} \begin{bmatrix} \text{CAT}: \text{ap} \\ T: 1 \end{bmatrix} \begin{bmatrix} \text{CAT}: \text{bt} \\ T: 1 \end{bmatrix} \begin{bmatrix} \text{CAT}: \text{bp} \\ T: 1 \end{bmatrix} \begin{bmatrix} \text{CAT}: \text{ct} \\ T: 1 \end{bmatrix} \begin{bmatrix} \text{CAT}: \text{cp} \\ T: 1 \end{bmatrix}$$
Example: (continued)

The second element of $\sigma_4$ is unifiable with the heads of both $\rho_2$ and $\rho_3$. We choose to apply $\rho_3$:


In the same way we can now apply $\rho_5$ and $\rho_7$ and obtain, eventually,


Now, let $w = aabbcc$; then $\sigma_7$ is a member of $PT_w(1, 6)$; in fact, it is the only member of the preterminal set. Therefore, $w \in L(G_{abc})$. 
Example: Derivation tree of $a^2b^2c^2$
The repetition language

Example: A unification grammar for \( \{ww \mid w \in \{a, b\}^+\} \)

The signature of the grammar consists in the features \textsc{cat}, \textsc{first} and \textsc{rest} and the atoms \textit{s}, \textit{ap}, \textit{bp}, \textit{at}, \textit{bt} and \textit{elist}. The terminal symbols are \textit{a} and \textit{b}. The start symbol is the left-hand side of the first rule.

\[
\begin{align*}
[\textsc{cat} : s] & \rightarrow [\textsc{first} : \begin{array}{c}1 \\
\textsc{rest} : \begin{array}{c}2 \end{array}\end{array} ] [\textsc{first} : \begin{array}{c}1 \\
\textsc{rest} : \begin{array}{c}2 \end{array}\end{array} ]
\end{align*}
\]
Example: (continued)

\[
\begin{align*}
\text{FIRST: } & ap \\
\text{REST: } & \begin{bmatrix} \text{FIRST:} & 1 \\
\text{REST:} & 2 \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT:} & at \\
\text{REST:} & \end{bmatrix} \\
\text{FIRST: } & bp \\
\text{REST: } & \begin{bmatrix} \text{FIRST:} & 1 \\
\text{REST:} & 2 \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT:} & bt \\
\text{REST:} & \end{bmatrix} \\
\text{FIRST: } & ap \\
\text{REST: } & \text{elist} \rightarrow \begin{bmatrix} \text{CAT:} & at \end{bmatrix} \\
\text{FIRST: } & bp \\
\text{REST: } & \text{elist} \rightarrow \begin{bmatrix} \text{CAT:} & bt \end{bmatrix} \\
\text{CAT: } & at \rightarrow a \\
\text{CAT: } & bt \rightarrow b
\end{align*}
\]
Unification grammars and Turing machines

- Unification grammars can simulate the operation of Turing machines.
- The membership problem for unification grammars is as hard as the halting problem.
A (deterministic) **Turing machine** \((Q, \Sigma, b, \delta, s, h)\) is a tuple such that:

- \(Q\) is a finite set of states
- \(\Sigma\) is an alphabet, not containing the symbols \(L, R\) and \(elist\)
- \(b \in \Sigma\) is the blank symbol
- \(s \in Q\) is the initial state
- \(h \in Q\) is the final state
- \(\delta : (Q \setminus \{h\}) \times \Sigma \to Q \times (\Sigma \cup \{L, R\})\) is a total function specifying transitions.
A configuration of a Turing machine consists of the state, the contents of the tape and the position of the head on the tape. A configuration is depicted as a quadruple \((q, w_l, \sigma, w_r)\) where \(q \in Q, w_l, w_r \in \Sigma^*\) and \(\sigma \in \Sigma\); in this case, the contents of the tape is \(\♭ \omega \cdot w_l \cdot \sigma \cdot w_r \cdot \♭ \omega\), and the head is positioned on the \(\sigma\) symbol.

A given configuration yields a next configuration, determined by the transition function \(\delta\), the current state and the character on the tape that the head points to.
Let

$$\text{first}(\sigma_1 \cdots \sigma_n) = \begin{cases} 
\sigma_1 & n > 0 \\
\varnothing & n = 0 
\end{cases}$$

$$\text{but-first}(\sigma_1 \cdots \sigma_n) = \begin{cases} 
\sigma_2 \cdots \sigma_n & n > 1 \\
\varepsilon & n \leq 1 
\end{cases}$$

$$\text{last}(\sigma_1 \cdots \sigma_n) = \begin{cases} 
\sigma_n & n > 0 \\
\varnothing & n = 0 
\end{cases}$$

$$\text{but-last}(\sigma_1 \cdots \sigma_n) = \begin{cases} 
\sigma_1 \cdots \sigma_{n-1} & n > 1 \\
\varepsilon & n \leq 1 
\end{cases}$$
Then the next configuration of a configuration \((q, w_l, \sigma, w_r)\) is defined iff \(q \neq h\), in which case it is:

\[(p, w_l, \sigma', w_r)\] if \(\delta(q, \sigma) = (p, \sigma')\) where \(\sigma' \in \Sigma\)

\[(p, w_l \sigma, \text{first}(w_r), \text{but-first}(w_r))\] if \(\delta(q, \sigma) = (p, R)\)

\[(p, \text{but-last}(w_l), \text{last}(w_l), \sigma w_r)\] if \(\delta(q, \sigma) = (p, L)\)
A next configuration is only defined for configurations in which the state is not the final state, $h$.

Since $\delta$ is a total function, there always exists a unique next configuration for every given configuration.

We say that a configuration $c_1$ yields the configuration $c_2$, denoted $c_1 \vdash c_2$, iff $c_2$ is the next configuration of $c_1$. 
Program:

- define a unification grammar $G_M$ for every Turing machine $M$ such that the grammar generates the word $\text{halt}$ if and only if the machine accepts the empty input string:

$$L(G_M) = \begin{cases} \{\text{halt}\} & \text{if } M \text{ terminates for the empty input} \\ \emptyset & \text{if } M \text{ does not terminate on the empty input} \end{cases}$$

- if there were a decision procedure to determine whether $w \in L(G)$ for an arbitrary unification grammar $G$, then in particular such a procedure could determine membership in the language of $G_M$, simulating the Turing machine $M$.

- the procedure for deciding whether $w \in L(G)$, when applied to the problem $\text{halt} \in L(G_M)$, determines whether $M$ terminates for the empty input, which is known to be undecidable.
Feature structures will have three features: \textsc{curr}, representing the character under the head; \textsc{right}, representing the tape contents to the right of the head (as a list); and \textsc{left}, representing the tape contents to the left of the head, in a reversed order.

All the rules in the grammar are unit rules; and the only terminal symbol is \texttt{halt}. Therefore, the language generated by the grammar is necessarily either the singleton \{\texttt{halt}\} or the empty set.
Let $M = (Q, \Sigma, b, \delta, s, h)$ be a Turing machine. Define a unification grammar $G_M$ as follows:

- $\text{Feats} = \{\text{cat, left, right, curr, first, rest}\}$
- $\text{Atoms} = \Sigma \cup \{\text{start,elist}\}$.
- The start symbol is $[\text{CAT : start}]$.
- the only terminal symbol is $\text{halt}$. 
Two rules are defined for every Turing machine:

\[
\text{[CAT : start]} \quad \rightarrow \quad \begin{bmatrix}
\text{CAT} : & s \\
\text{CURR} : & b \\
\text{RIGHT} : & \text{elist} \\
\text{LEFT} : & \text{elist}
\end{bmatrix}
\]

\[h \quad \rightarrow \quad \text{halt}\]
For every \( q, \sigma \) such that \( \delta(q, \sigma) = (p, \sigma') \) and \( \sigma' \in \Sigma \), the following rule is defined:

\[
\begin{bmatrix}
\text{CAT} : & q \\
\text{CURR} : & \sigma \\
\text{RIGHT} : & 1 \\
\text{LEFT} : & 2 \\
\end{bmatrix} \rightarrow \begin{bmatrix}
\text{CAT} : & p \\
\text{CURR} : & \sigma' \\
\text{RIGHT} : & 1 \\
\text{LEFT} : & 2 \\
\end{bmatrix}
\]
For every $q, \sigma$ such that $\delta(q, \sigma) = (p, R)$ we define two rules:

\[
\begin{align*}
&\text{CAT : } q \\
&\text{CURR : } \sigma \\
&\text{RIGHT : } \text{elist} \\
&\text{LEFT : } 1
\end{align*}
\rightarrow
\begin{align*}
&\text{CAT : } p \\
&\text{CURR : } b \\
&\text{RIGHT : } \text{elist} \\
&\text{LEFT : } \text{FIRST : } \sigma \\
&\text{REST : } 1
\end{align*}
\]

\[
\begin{align*}
&\text{CAT : } q \\
&\text{CURR : } \sigma \\
&\text{RIGHT : } \text{FIRST : } 1 \\
&\text{REST : } 2 \\
&\text{LEFT : } 3
\end{align*}
\rightarrow
\begin{align*}
&\text{CAT : } p \\
&\text{CURR : } 1 \\
&\text{RIGHT : } 2 \\
&\text{LEFT : } \text{FIRST : } \sigma \\
&\text{REST : } 3
\end{align*}
\]
For every \( q, \sigma \) such that \( \delta(q, \sigma) = (p, L) \) we define two rules:

\[
\begin{align*}
\text{CAT} &: q \\
\text{CURR} &: \sigma \\
\text{RIGHT} &: [1] \\
\text{LEFT} &: \text{elist}
\end{align*}
\]

\[\rightarrow\]

\[
\begin{align*}
\text{CAT} &: p \\
\text{CURR} &: \emptyset \\
\text{RIGHT} &: [\text{FIRST}: \sigma, \text{REST}: [1]] \\
\text{LEFT} &: \text{elist}
\end{align*}
\]

\[
\begin{align*}
\text{CAT} &: q \\
\text{CURR} &: \sigma \\
\text{RIGHT} &: [1] \\
\text{LEFT} &: [\text{FIRST}: [2], \text{REST}: [3]]
\end{align*}
\]

\[\rightarrow\]

\[
\begin{align*}
\text{CAT} &: p \\
\text{CURR} &: [2] \\
\text{RIGHT} &: [\text{FIRST}: \sigma, \text{REST}: [1]] \\
\text{LEFT} &: [3]
\end{align*}
\]
### Lemma
Let $c_1, c_2$ be configurations of a Turing machine $M$, and $A_1, A_2$ be AVMs encoding these configurations, viewed as multi-AVMs of length 1. Then $c_1 \vdash c_2$ iff $A_1 \Rightarrow A_2$ in $G_m$.

### Theorem
A Turing machine $M$ halts for the empty input iff $\text{halt} \in L(G_M)$.

### Corollary
The universal recognition problem for unification grammars is undecidable.
In order to ensure decidability of the recognition problem, several constraints on grammars, commonly known as the off-line parsability constraints (OLP), were suggested, such that the recognition problem is decidable for OLP unification grammars.

The motivation behind all OLP definitions is to rule out grammars which license trees in which unbounded amount of material is generated without expanding the frontier word.

This can happen due to two kinds of rules: ε-rules, whose bodies are empty, and unit rules, whose bodies consist of a single element.
Off-line parsability

With context-free grammars the removal of rules which can cause an unbounded growth is always possible. In particular, one can always remove cyclic sequences of unit rules.

However, with unification grammars it is not trivial to determine when a sequence of unit rules is, indeed, cyclic; and when a rule is redundant.
Off-line parsability

- Several definitions of off-line parsability are known.
- Some simple proposals:
  - Disallow $\epsilon$-rules and unit-rules
  - Require a finitely ambiguous context-free skeleton
- The state of the art: allow only unit-rules which are not cyclicly-unifiable (i.e., cannot feed themselves).
Highly constrained unification grammars

Two recent results:

- Non-reentrant grammars generate exactly the class of context-free languages;
- One-reentrant grammars generate exactly the class of mildly context-sensitive languages.