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 FEATS of features and ATOMS of atoms
- *Attribute-value matrices* (AVMs) used to depict *feature structures*, which are sets of $\langle$feature, value$\rangle$ 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

$$\text{lamb} \longrightarrow \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & sg \\ \text{CASE}: & [\,] \end{bmatrix}$$

$$\text{love} \longrightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{SUBCAT}: & \left\langle \begin{bmatrix} \text{CAT}: & np \\ \text{CASE}: & acc \end{bmatrix} \right\rangle \\ \text{NUM}: & pl \end{bmatrix}$$

$$\text{give} \longrightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{SUBCAT}: & \left\langle \begin{bmatrix} \text{CAT}: & np \\ \text{CASE}: & acc \end{bmatrix}, \begin{bmatrix} \text{CAT}: & np \end{bmatrix} \right\rangle \\ \text{NUM}: & pl \end{bmatrix}$$

Basic notions

Example: Grammar rules

$$\begin{bmatrix} \text{CAT}: & s \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & \boxed{1} \\ \text{CASE}: & nom \end{bmatrix} \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{1} \\ \text{SUBCAT}: & elist \end{bmatrix}$$

$$\begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & \boxed{1} \\ \text{CASE}: & \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & \boxed{1} \end{bmatrix} \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & \boxed{1} \\ \text{CASE}: & \boxed{2} \end{bmatrix}$$

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: $\textsc{cat}$, which stands for category, and $\textsc{t}$, 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:

$$\begin{bmatrix} \textsc{cat}: & bp \\ \textsc{t}: & \begin{bmatrix} \textsc{t}: & \begin{bmatrix} \textsc{t}: & end \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Trans-context-free languages

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

The signature of the grammar consists in the features $\textsc{cat}$ and $\textsc{t}$ 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.

$$\rho_1: \quad \begin{bmatrix}\text{CAT}: & s\end{bmatrix} \to \begin{bmatrix}\text{CAT}: & ap \\ \text{T}: & \boxed{1}\end{bmatrix} \begin{bmatrix}\text{CAT}: & bp \\ \text{T}: & \boxed{1}\end{bmatrix} \begin{bmatrix}\text{CAT}: & cp \\ \text{T}: & \boxed{1}\end{bmatrix}$$

$$\rho_2: \quad \begin{bmatrix}\text{CAT}: & ap \\ \text{T}: & \begin{bmatrix}\text{T}: & \boxed{1}\end{bmatrix}\end{bmatrix} \to \begin{bmatrix}\text{CAT}: & at\end{bmatrix} \begin{bmatrix}\text{CAT}: & ap \\ \text{T}: & \boxed{1}\end{bmatrix}$$

$$\rho_3: \quad \begin{bmatrix}\text{CAT}: & ap \\ \text{T}: & end\end{bmatrix} \to \begin{bmatrix}\text{CAT}: & at\end{bmatrix}$$

Example: (continued)

$$\rho_4 : \begin{bmatrix} \text{CAT} : & bp \\ \text{T} : & \begin{bmatrix} \text{T} : & \boxed{1} \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} : & bt \end{bmatrix} \begin{bmatrix} \text{CAT} : & bp \\ \text{T} : & \boxed{1} \end{bmatrix}$$

$$\rho_5 : \begin{bmatrix} \text{CAT} : & bp \\ \text{T} : & end \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} : & bt \end{bmatrix}$$

$$\rho_6 : \begin{bmatrix} \text{CAT} : & cp \\ \text{T} : & \begin{bmatrix} \text{T} : & \boxed{1} \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} : & ct \end{bmatrix} \begin{bmatrix} \text{CAT} : & cp \\ \text{T} : & \boxed{1} \end{bmatrix}$$

$$\rho_7 : \begin{bmatrix} \text{CAT} : & cp \\ \text{T} : & end \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} : & ct \end{bmatrix}$$

Example: (continued)

$$\begin{aligned} \left[\mathrm{CAT}: \ at\right] &\rightarrow a \\ \left[\mathrm{CAT}: \ bt\right] &\rightarrow b \\ \left[\mathrm{CAT}: \ ct\right] &\rightarrow c \end{aligned}$$

Trans-context-free languages

Example: Derivation sequence of $a^2b^2c^2$

Start with a form that consists of the start symbol,

$$\sigma_0 = \begin{bmatrix} \text{CAT} : s \end{bmatrix}.$$

Only one rule, ρ_1, can be applied to the single element of the multi-AVM in σ_0, yielding:

$$\sigma_1 = \begin{bmatrix} \text{CAT}: & ap \\ \text{T}: & \boxed{1} \end{bmatrix} \begin{bmatrix} \text{CAT}: & bp \\ \text{T}: & \boxed{1} \end{bmatrix} \begin{bmatrix} \text{CAT}: & cp \\ \text{T}: & \boxed{1} \end{bmatrix}$$

Example: (continued)

Applying ρ_2 to the first element of σ_1:

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

Choose the third element in σ_2 and apply the rule ρ_4:

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

Apply ρ_6 to the fifth element of $\sigma 3$:

$$\sigma_4 = \begin{bmatrix}\text{CAT}: & \textit{at}\end{bmatrix} \begin{bmatrix}\text{CAT}: & \textit{ap}\\ \text{T}: & \boxed{1}\end{bmatrix} \begin{bmatrix}\text{CAT}: & \textit{bt}\end{bmatrix} \begin{bmatrix}\text{CAT}: & \textit{bp}\\ \text{T}: & \boxed{1}\end{bmatrix} \begin{bmatrix}\text{CAT}: & \textit{ct}\end{bmatrix} \begin{bmatrix}\text{CAT}: \\ \text{T}: \end{bmatrix}$$

Example: (continued)

The second element of σ_4 is unifiable with the heads of both ρ_2 and ρ_3. We choose to apply ρ_3:

$$\sigma_5 = \begin{bmatrix}\text{CAT}: & at\end{bmatrix} \begin{bmatrix}\text{CAT}: & at\end{bmatrix} \begin{bmatrix}\text{CAT}: & bt\end{bmatrix} \begin{bmatrix}\text{CAT}: & bp \\ \text{T}: & end\end{bmatrix} \begin{bmatrix}\text{CAT}: & ct\end{bmatrix} \begin{bmatrix}\text{CAT}: \\ \text{T}:\end{bmatrix}$$

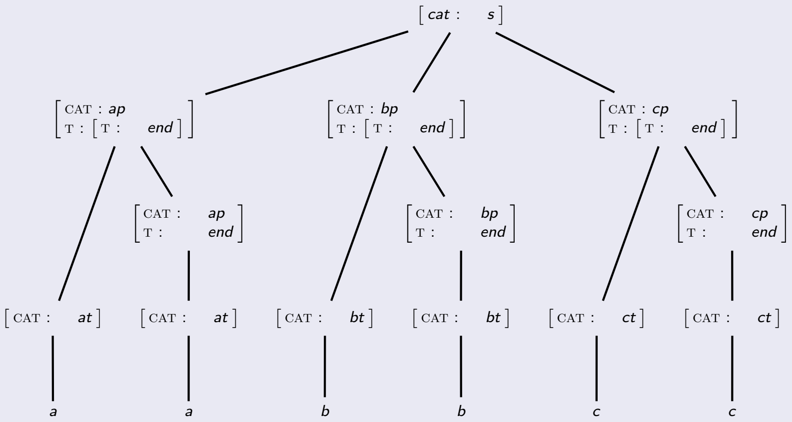
In the same way we can now apply ρ_5 and ρ_7 and obtain, eventually,

$$\sigma_7 = \begin{bmatrix}\text{CAT}: & at\end{bmatrix} \begin{bmatrix}\text{CAT}: & at\end{bmatrix} \begin{bmatrix}\text{CAT}: & bt\end{bmatrix} \begin{bmatrix}\text{CAT}: & bt\end{bmatrix} \begin{bmatrix}\text{CAT}: & ct\end{bmatrix} \begin{bmatrix}\text{CAT}: & c\end{bmatrix}$$

Now, let $w = aabbcc$; then σ_7 is a member of $PT_w(1,6)$; in fact, it is the only member of the preterminal set. Therefore, $w \in L(G_{abc})$.

Trans-context-free languages

Example: Derivation tree of $a^2 b^2 c^2$



The repetition language

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

The signature of the grammar consists in the features CAT, FIRST and REST and the atoms *s*, *ap*, *bp*, *at*, *bt* and *elist*. The terminal symbols are *a* and *b*. The start symbol is the left-hand side of the first rule.

$$\begin{bmatrix}\text{CAT}: & s\end{bmatrix} \longrightarrow \begin{bmatrix}\text{FIRST}: & \boxed{1}\\ \text{REST}: & \boxed{2}\end{bmatrix} \begin{bmatrix}\text{FIRST}: & \boxed{1}\\ \text{REST}: & \boxed{2}\end{bmatrix}$$

Example: (continued)

$$\begin{bmatrix} \text{FIRST}: & ap \\ \text{REST}: & \begin{bmatrix} \text{FIRST}: & \boxed{1} \\ \text{REST}: & \boxed{2} \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & at \end{bmatrix} \begin{bmatrix} \text{FIRST}: & \boxed{1} \\ \text{REST}: & \boxed{2} \end{bmatrix}$$

$$\begin{bmatrix} \text{FIRST}: & bp \\ \text{REST}: & \begin{bmatrix} \text{FIRST}: & \boxed{1} \\ \text{REST}: & \boxed{2} \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & bt \end{bmatrix} \begin{bmatrix} \text{FIRST}: & \boxed{1} \\ \text{REST}: & \boxed{2} \end{bmatrix}$$

$$\begin{bmatrix} \text{FIRST}: & ap \\ \text{REST}: & elist \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & at \end{bmatrix}$$

$$\begin{bmatrix} \text{FIRST}: & bp \\ \text{REST}: & elist \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & bt \end{bmatrix}$$

$$\begin{bmatrix} \text{CAT}: & at \end{bmatrix} \rightarrow a$$

$$\begin{bmatrix} \text{CAT}: & bt \end{bmatrix} \rightarrow b$$

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.

Unification grammars and Turing machines

A (deterministic) **Turing machine** $(Q, \Sigma, \flat, \delta, s, h)$ is a tuple such that:

- Q is a finite set of states
- Σ is an alphabet, not containing the symbols L, R and *elist*
- $\flat \in \Sigma$ 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.

Unification grammars and Turing machines

- 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, σ, 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 $\flat^\omega \cdot w_l \cdot \sigma \cdot w_r \cdot \flat^\omega$, and the head is positioned on the σ symbol.
- A given configuration yields a *next configuration*, determined by the transition function δ, the current state and the character on the tape that the head points to.

Unification grammars and Turing machines

Let

$$
\begin{array}{rcl}
\mathit{first}(\sigma_1 \cdots \sigma_n) & = & \begin{cases} \sigma_1 & n > 0 \\ \flat & n = 0 \end{cases} \\
\mathit{but\text{-}first}(\sigma_1 \cdots \sigma_n) & = & \begin{cases} \sigma_2 \cdots \sigma_n & n > 1 \\ \epsilon & n \leq 1 \end{cases} \\
\mathit{last}(\sigma_1 \cdots \sigma_n) & = & \begin{cases} \sigma_n & n > 0 \\ \flat & n = 0 \end{cases} \\
\mathit{but\text{-}last}(\sigma_1 \cdots \sigma_n) & = & \begin{cases} \sigma_1 \cdots \sigma_{n-1} & n > 1 \\ \epsilon & n \leq 1 \end{cases}
\end{array}
$$

Unification grammars and Turing machines

Then the next configuration of a configuration (q, w_l, σ, w_r) is defined iff $q \neq h$, in which case it is:

$$
\begin{array}{ll}
(p, w_l, \sigma', w_r) & \text{if } \delta(q,\sigma) = (p, \sigma') \text{ where } \sigma' \in \Sigma \\
(p, w_l\sigma, \mathit{first}(w_r), \mathit{but\text{-}first}(w_r)) & \text{if } \delta(q,\sigma) = (p, R) \\
(p, \mathit{but\text{-}last}(w_l), \mathit{last}(w_l), \sigma w_r) & \text{if } \delta(q,\sigma) = (p, L)
\end{array}
$$

Unification grammars and Turing machines

- A next configuration is only defined for configurations in which the state is not the final state, h.
- Since δ 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.

Unification grammars and Turing machines

Program:

- define a unification grammar G_M for every Turing machine M such that the grammar generates the word 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 halt$\in L(G_M)$, determines whether M terminates for the empty input, which is known to be undecidable.

Unification grammars and Turing machines

- Feature structures will have three features: CURR, representing the character under the head; RIGHT, representing the tape contents to the right of the head (as a list); and 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 halt. Therefore, the language generated by the grammar is necessarily either the singleton $\{$halt$\}$ or the empty set.

Unification grammars and Turing machines: signature

Let $M = (Q, \Sigma, \flat, \delta, s, h)$ be a Turing machine. Define a unification grammar G_M as follows:

- $\textsc{Feats} = \{\textsc{cat}, \textsc{left}, \textsc{right}, \textsc{curr}, \textsc{first}, \textsc{rest}\}$
- $\textsc{Atoms} = \Sigma \cup \{\mathit{start}, \mathit{elist}\}$.
- The start symbol is $\left[\textsc{cat}: \ \mathit{start}\right]$.
- the only terminal symbol is halt.

Unification grammars and Turing machines: rules

Two rules are defined for every Turing machine:

$$\begin{array}{lcl} \begin{bmatrix}\text{CAT}: & \textit{start}\end{bmatrix} & \rightarrow & \begin{bmatrix}\text{CAT}: & \textit{s} \\ \text{CURR}: & \flat \\ \text{RIGHT}: & \textit{elist} \\ \text{LEFT}: & \textit{elist}\end{bmatrix} \\ h & \rightarrow & \text{halt} \end{array}$$

Unification grammars and Turing machines: rules

For every q, σ such that $\delta(q, \sigma) = (p, \sigma')$ and $\sigma' \in \Sigma$, the following rule is defined:

$$\begin{bmatrix} \text{CAT}: & q \\ \text{CURR}: & \sigma \\ \text{RIGHT}: & \boxed{1} \\ \text{LEFT}: & \boxed{2} \end{bmatrix} \longrightarrow \begin{bmatrix} \text{CAT}: & p \\ \text{CURR}: & \sigma' \\ \text{RIGHT}: & \boxed{1} \\ \text{LEFT}: & \boxed{2} \end{bmatrix}$$

Unification grammars and Turing machines: rules

For every q, σ such that $\delta(q, \sigma) = (p, R)$ we define two rules:

$$\begin{bmatrix} \text{CAT}: & q \\ \text{CURR}: & \sigma \\ \text{RIGHT}: & \textit{elist} \\ \text{LEFT}: & \boxed{1} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & p \\ \text{CURR}: & \flat \\ \text{RIGHT}: & \textit{elist} \\ \text{LEFT}: & \begin{bmatrix} \text{FIRST}: & \sigma \\ \text{REST}: & \boxed{1} \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} \text{CAT}: & q \\ \text{CURR}: & \sigma \\ \text{RIGHT}: & \begin{bmatrix} \text{FIRST}: & \boxed{1} \\ \text{REST}: & \boxed{2} \end{bmatrix} \\ \text{LEFT}: & \boxed{3} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & p \\ \text{CURR}: & \boxed{1} \\ \text{RIGHT}: & \boxed{2} \\ \text{LEFT}: & \begin{bmatrix} \text{FIRST}: & \sigma \\ \text{REST}: & \boxed{3} \end{bmatrix} \end{bmatrix}$$

Unification grammars and Turing machines: rules

For every q, σ such that $\delta(q, \sigma) = (p, L)$ we define two rules:

$$\begin{bmatrix} \text{CAT}: & q \\ \text{CURR}: & \sigma \\ \text{RIGHT}: & \boxed{1} \\ \text{LEFT}: & \textit{elist} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & p \\ \text{CURR}: & \flat \\ \text{RIGHT}: & \begin{bmatrix} \text{FIRST}: & \sigma \\ \text{REST}: & \boxed{1} \end{bmatrix} \\ \text{LEFT}: & \textit{elist} \end{bmatrix}$$

$$\begin{bmatrix} \text{CAT}: & q \\ \text{CURR}: & \sigma \\ \text{RIGHT}: & \boxed{1} \\ \text{LEFT}: & \begin{bmatrix} \text{FIRST}: & \boxed{2} \\ \text{REST}: & \boxed{3} \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & p \\ \text{CURR}: & \boxed{2} \\ \text{RIGHT}: & \begin{bmatrix} \text{FIRST}: & \sigma \\ \text{REST}: & \boxed{1} \end{bmatrix} \\ \text{LEFT}: & \boxed{3} \end{bmatrix}$$

Unification grammars and Turing machines: results

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 halt $\in L(G_M)$.

Corollary

The universal recognition problem for unification grammars is undecidable.

Off-line parsability

- 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 ϵ-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.