ALE Lexical Rules

Kilian Evang

2008-01-24

Questions?

Presenting:

[Penn et al., 2003]

Outline

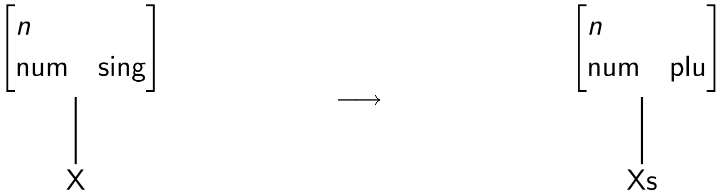
Reminder: Lexical entries

```
john --->
  synsem:(syn:np,
          sem:j),
  qstore:e_list.
```

A redundant lexicon

```
root ---> n, num:sing.
roots ---> n, num:plu.
cloud ---> n, num:sing.
clouds ---> n, num:plu.
```

What we need:



What we need:

- A device to create new lexical entries from existing lexical entries - automatically.
- Example: We only specify the singular of a word and get its plural for free.
- In a derivation, the **form** of a word can change as well as its **category** (which is a typed feature structure).
- Lexical rules do that.

Regular Plural of Nouns

```
plural_n lex_rule
   (n,
    num:sing)
    **> (n,
           num:plu)
   morphs
     X becomes (X,s).
```

Regular Plural of Nouns

- **Input pattern:** Is simply `X`, therefore the rule applies to entries independently of their word.
- **Ouptut pattern:** A new entry is generated where the word is suffixed with `s`.
- **Input description:** The rule applies only to entries of categories of type `n` with the `num` feature having a `plu` value.
- **Output description:** The generated entry is of category $\begin{bmatrix} \mathit{n} & \\ \text{num} & \text{plu} \end{bmatrix}$

Plural of Nouns

```
plural_n lex_rule
  (n,
   num:sing)
   **> (n,
        num:plu)
  morphs
    [g,o,o,s,e] becomes [g,e,e,s,e],
    german becomes germans,
    (X,man) becomes (X,men),
    (X,F) becomes (X,F,es) when fricative(F),
    (X,C,y) becomes (X,C,[i,e,s]) when consonant(C),
    X becomes (X,s).
```

Plural of Nouns

```
plural_n lex_rule
  (n,
   num:sing)
   **> (n,
         num:plu)
 morphs
   [g,o,o,s,e] becomes [g,e,e,s,e],
   german becomes germans,
   (X,man) becomes (X,men),
   (X,F) becomes (X,F,es) when fricative(F),
   (X,C,y) becomes (X,C,[i,e,s]) when consonant(C),
   X becomes (X,s).
```

```
fricative([s]).
fricative([c,h]).
fricative([s,h]).
fricative([x]).
consonant([t]).
```

Verbal Inflection

section 3.2.2 - peter likes them

Patterns

- match lists of characters = words
- are sequences of
  - lists of characters (match equal lists)
  - variables (match any list)
- Lists of characters can be abbreviated to atoms
- Sequences of length 1 can be written without parentheses
- Variables get instantiated during matching and used like backreferences in building the output strings

Greedy Matching?

```
(X,Y) becomes X when bseq(Y).
bseq([b]).
bseq([b,b]).
```

$$\text{abb} \longrightarrow$$

Greedy Matching?

```
(X,Y) becomes X when bseq(Y).
bseq([b]).
bseq([b,b]).
```

abb $\longrightarrow$ a ?
ab ?

Special cases

- No morphological effect:
  `X becomes X`

Special cases

- No morphological effect:
  `X becomes X`
- Lexical operations stated wholly in Prolog:
  `X becomes Y when morph_plural(X,Y)`

Application of Lexical Rules

1. Every lexical entry is checked to see if its **category** satisfies the **input description**.
2. If it does and there is a **definite clause**, the latter is invoked.
3. For each solution, the **most general** feature structure satisfying the **output description** - with the respective variables instantiated - is generated.
4. The **morphological productions** are tried one by one.
5. The first one where
   - the left-hand side matches the input word *and*
   - the goal of the `when`-clause (if present) succeeds

   is applied.

Multiple Solutions

fantasy example

The full syntax (1)

```
<lex_rule> ::= <lex_rule_name> lex_rule <lex_rewrite>
                               morphs <morphs>.

<lex_rewrite> ::= <desc> **> <desc>
                | <desc> **> <desc> if <goal>

<morphs> ::= <morph> | <morph>, <morphs>
```

The full syntax (2)

```
<morph> ::= (<string_pattern>) becomes (<string_pattern>)
          | (<string_pattern>) becomes (<string_pattern>)
                               when <prolog_goal>

<string_pattern> ::= <atomic_string_pattern>
                   | <atomic_string_pattern>, <string_pattern>

<atomic_string_pattern> ::= <atom>
                          | <var>
                          | <list(<var_char>)>

<var_char> ::= <char>
             | <var>
```

A Simplified Example from HPSG

```
extraction lex_rule
  local:(cat:(head:H,
              subcat:Xs),
         cont:C),
  nonlocal:(to_bind:Bs,
            inherited:Is)
  **> local:(cat:(head:H,
                  subcat:Xs2),
             cont:C),
      nonlocal:(to_bind:Bs,
                inherited:[G|Is])
  if
    select(G,Xs,Xs2)
  morphs
    X becomes X.
```

A Simplified Example from HPSG

```
extraction lex_rule
  local:(cat:(head:H,
              subcat:Xs),
         cont:C),
  nonlocal:(to_bind:Bs,
            inherited:Is)
  **> local:(cat:(head:H,
                  subcat:Xs2),
             cont:C),
      nonlocal:(to_bind:Bs,
                inherited:[G|Is])
  if
    select(G,Xs,Xs2)
  morphs
    X becomes X.
```

```
select(X,(hd:X,tl:Xs),Xs) if true.
select(X,[Y|Xs],[Y|Ys]) if
  select(X,Xs,Ys).
```

Passivization

section 3.2.3 - she is being liked

Things to Note

- The descriptions of the input/output categories must also mention information that stays the same, as the output category is the **most general** category satisfying the output description.

Things to Note

- The descriptions of the input/output categories must also mention information that stays the same, as the output category is the **most general** category satisfying the output description.
- Cuts should not be used in conditions on lexical rules.

Things to Note

- The descriptions of the input/output categories must also mention information that stays the same, as the output category is the **most general** category satisfying the output description.
- Cuts should not be used in conditions on lexical rules.
- ALE does not check for redundancies or for entries that subsume each other.

Things to Note

- The descriptions of the input/output categories must also mention information that stays the same, as the output category is the **most general** category satisfying the output description.
- Cuts should not be used in conditions on lexical rules.
- ALE does not check for redundancies or for entries that subsume each other.
- ALE does not apply lexical rules to empty categories.

Things to Note

- Lexical rules can apply to the output of a lexical rule.
- The depth of sequential application can be restrained globally (default is 2):
  `lex_rule_depth(2).`

Gerald Penn et al. (2003):
ALE. The Attribute Logic Engine. User's Guide with TRALE Extensions.

http://www.ale.cs.toronto.edu/docs/

Walt Detmar Meurers (2000):
Lexical Generalizations in the Syntax of German Non-Finite Constructions.
Phil. Dissertation, Eberhard-Karls-Universität Tübingen.
Published as: Arbeitspapiere des SFB 340, Nr. 145.

http://www.sfs.nphil.uni-tuebingen.de/sfb/reports/berichte/145/145-2up.ps

Thank you!