Software Tools for Morphological and Syntactic Analysis of Natural Language Texts

Feature structures are widely used in Natural Language Processing. They are commonly used:

To hold initial properties of lexical entries in the dictionary

To put constraints on parser rules. Certain operations defined on feature structures are used for this purpose.

To pass data across different levels of analysis

We use following notation to represent feature structures in our formalism. List of “Attribute – Value” pairs is enclosed in square braces. Attributes and values are separated by colon “:”. In example:

S = [A: V1

B: [C: V2]]

It is possible to use short-hand notation for constructing feature structures. We can rewrite above example this way:

T1 = [A: V1]

T2 = [C: V2]

S = [(S, T1) B: T2]

Feature Structures

- <feature_structure> ::= “[”[<initialization_part>] [<list_of_pairs>] “]”
- <initialization_part> ::= “(” {<initializer>} “)”
- <initializer> ::= <variable_reference> | <constant_reference>
- <list_of_pairs> ::= { <pair> }
- <pair> ::= <name> “:” <value>
- <name> ::= <identifier><value> ::= “+” | “-” | <number> | <identifier> | <string> | <feature_structure>
- . . .

- A := B (Assignment) Content of the RHS (Right Hand Side) operand (B) is assigned to the LHS (Left Hand Side) operand (A). Thus their content becomes equal after the assignment. The assignment operation always returns “true” value.
- A = B (Check on equality) This operation does not modify content of the operands. Result of the operation is “true” when both operands (A and B) have the same fields (attributes) with identical values. If there is a field in one feature structure which is not represented in the second feature structure or the same fields does not have an equal values then the result is “false”.
- A <== B (Unification) Unification returns “true” when the values of the similar field in each feature structure does not conflict with each other. That means, either the values are equal, or one of the value is undefined. Otherwise the result of the unification operator is “false”. Fields, that are not defined in LHS feature structure and are defined in RHS feature structure are copied and added to the LHS operand. If there is an undefined value in LHS feature structure, and the same field in the RHS feature structure is defined, that value is assigned to the corresponding LHS feature structure field.
- A == B (Check on unification) Returns the same truth value as unification operator, but the content of operands is not modified.
- Check on equality or unification operations (“=” and “==”) may take multiple arguments. In example:
- X == (A, B, C)

Constraints

- S -> A1 { C1 } A2 { C2 } … AN {CN}

- <constraint> ::= <constraint_term> “|”<constraint_term>
- <constraint_term> ::= <constraint_fact> “&” <constraint_fact>
- <constraint_fact> ::= [“~”] (<logical_constant> | “+” | “-” | <constraint_operation> | “(” <constraint_fact> “)”)
- <logical_constant> ::= “0” | “1”
- <constraint_operation> ::= < constraint_operator> | <constraint_function>
- <constraint_operator> ::= <constraint_argument> (“:=” | “==” | “<==”, “=”) (<constraint_argument> | <list_of_constraint_arguments>)
- <constraint_function> ::= <identifier> <constraint_function_arguments>
- . . .

Morphological analyzer

- @M1 =
- {
- “morpheme_1” [… features …]
- “morpheme_2” [… features …]
- . . .
- “morpheme_N” [… features …]
- }
- It is possible to declare empty morpheme, which means that the morpheme class may be omitted in morphological rules. Below is formal syntax for morpheme class definition:
- <morphem_definition> ::= “@” <identifier> “=” “{”
- <list_of_morphemes> “}”
- <list_of_morphemes> ::= <morpheme> { “,” <morpheme> }
- <morpheme> ::= <string> <feature_structure>
- Morphological rules are defined following way:
- word -> M1 { C1 } M2 { C2 } . . . MN { CN }
- Where Mi are morpheme classes, and Ci (i=1,…,N) are constraints (optional).

Syntactic analyzer

- S -> A1 { C1 } A2 { C2 } . . . AN { CN } ;
- S -> A1 A2 . . . AN : R { C } ;
- Where S is an LHS non-terminal symbol, Ai (i=1,…,N) are RHS terminal or non-terminal symbols, C and Ci (i=1,…,N) are constraints, and R is a set of symbol position regulators. Position regulators declare order of RHS symbols in the rule, thus making non-fixed word ordering. There are two types of position regulators:
- Ai < Aj means that symbol Ai must be placed somewhere before the symbol Aj
- Ai – Aj means that symbol Ai must be placed exactly before the symbol Aj

- **Example of a syntactic analysis**
- Below is a sample sentence given to the syntactic analyzer:
- “cnobili mSenebeli saxls uSenebs megobars” (Georgian, Latin encoding)
- “Famious builder builds a home for his friend”
- Result produced by the syntactic analyzer:
- &> Parsing: cnobili(ZS) mSenebeli(AS) saxls(AS) uSenebs(Z) megobars(AS)
- 1 solution(s) was(were) found.
- Parse Tree 1:
- |
- ZJG3P:1
- |-------------------------------|------------|-------------|
- SPNS:2 SPNS:3 ZJG:4 SPNS:5
- | | | |
- SJGM:6 SJGM:7 Z:8 (uSenebs) SJGM:9
- | | |
- SJG:10 SJG:11 SJG:12
- |---------------| | |
- AT:13 SJG:14 AS:15 (saxls) AS:16 (megobars)
- | |

Symbols translation

- ZS Adjective
- AS Noun
- Z Verb
- ZJG3P Verb group
- 3SPNS Noun or pronoun
- ZJG Verb group
- SJGM Driven noun group
- SJG Noun group
- AT Attribute
- brunva Case
- piri Person
- ricxvi Number
- dro Tense
- ir_obj Indirect object
- pir_obj Direct object