

11-711 Algorithms for NLP

Parsing with Unification Grammars

Readings:

(1) James Allen: “NL Understanding”,

Chapter 4.

(2) Tomita and Knight: “Pseudo-Unification
and Full-Unification”, CMT Tech Report.

Augmenting CFGs with Features

- Certain linguistic constraints are not naturally described via CFGs
- Example: *Number Agreement* between constituents - “a boys”
- Possible to describe using refined CF rules:

NP-Sing --> ART-Sing N-Sing

NP-Plu --> ART-Plu N-Plu

- Much more natural to describe via a *single* feature-augmented CF rule:

NP --> ART N

((x1 number = x2 number))

- Describing a large set of such feature constraints using only CF rules is not practical

Feature Structures

- *Constituents* can be viewed as *structures* (collections) of *features* that have assigned *values*
- Features can be *shared between constituents*
- Linguistic constraints express rules about how the feature-structure of a constituent is formed from its sub-constituents
- Some basic features for English:
 - Number, Gender and Person agreement
 - Verb form features and sub-categorizations
- Complex Feature Structures: Feature values can themselves be feature structures

Unification of Feature Structures

- Unification Grammars (such as LFG) establish a complete linguistic theory for a language via a set of relationships between feature structures of constituents
- Key concept - *extension* relationship between two FSs:
 F_2 *extends* F_1 if every feature-value pair in F_1 is also in F_2
- Two FSs F_1 and F_2 *unify* if there exists a FS F that is an extension of both of them
- *The Most General Unifier* is the minimal FS F that extends both
- The *Unification* operation allows easy expression of grammatical relationships among constituent feature structures

Unification-based Grammars

- Grammar rules can be completely specified using unification
- Example:

```
x0 --> x1 x2
((x0 cat = S)
 (x1 cat = NP)
 (x2 cat = VP)
 (x1 agr = x2 agr)
 (x0 subj = X1))
```

- If a feature (such as `cat`) is always specified, it can be associated with the non-terminal of a CFG rule
- Example:

```
S --> NP VP
((x1 agr = x2 agr)
 (x0 subj = X1))
```

Unification-based Grammars

Example:

- The grammar rule:

```
NP --> ART N
      ((x1 agr) = (x2 agr))
      ((x0 spec) = (x1 spec))
      (x0 = x2))
```

- The Feature Structures:

```
ART: ((agr *3s, *3p)  N: ((agr *3s)  NP: ((agr *3s)
      (root *the)      (root *boy))      (spec *def)
      (spec *def))    (root *boy))
```

Unification of Feature Structures

- Feature Structures are commonly represented as DAGs
- Unification between FSs can be described as an operation that constructs a new DAG representing the unified structure
- the Algorithm:

To unify a DAG rooted at node N_i with a DAG rooted at node N_j :

1. If N_i equals N_j , then return N_i and succeed.
2. If both N_i and N_j are sink nodes, then if their labels have a non-null intersection, return a new node with the intersection as its label. Otherwise, the DAGs do not unify.
3. If N_i and N_j are not sinks, then create a new node N . For each arc labeled F leaving N_i to node NF_i ,
 - 3a. If there is an arc labeled F leaving N_j to node NF_j , then recursively unify NF_i and NF_j . Build an arc labeled F from N to the result of the recursive call.
 - 3b. If there is no arc labeled F from N_j , build an arc labeled F from N to NF_i .
 - 3c. For each arc labeled F from N_j to node NF_j where there is no F arc leaving N_i , create a new arc labeled F from N to NF_j .

Given a rule $X_0 \rightarrow X_1 \dots X_n$ and set of feature equations of form $F_i = V$, where SC_1, \dots, SC_n are the subconstituents corresponding to X_1, \dots, X_n , this algorithm builds a DAG that satisfies all the feature equations.

1. Create a node CC_0 to be the root of the new feature structure.
2. Make a copy of each DAG rooted by SC_i (call the new root of each CC_i), and add an arc labeled i from CC_0 to each CC_i .
3. For each feature equation of form $F_i = V$, where V is a value, follow the F link from node CC_i to a node N_i , and unify N_i with V .
4. For each feature equation (of form $F_i = G_j$),
 - 4a. If there is an F link from CC_i , and a G link from CC_j , then
 - i. follow the F link to node N_i and the G link to N_j ;
 - ii. unify N_i and N_j , using the graph unification algorithm, to create new node X ;
 - iii. change all arcs pointing to either N_i or N_j to point to X ;
 - 4b. If there is no F link from CC_i , but there is a G link from CC_j to node N_j , create an F link from CC_i to N_j ;
 - 4c. If there is no G link from CC_j , but there is an F link from CC_i to N_i , create a G link from CC_j to N_i .

Figure 4.24 An algorithm to build a new constituent using feature equations

Unification of Feature Structures

Example:

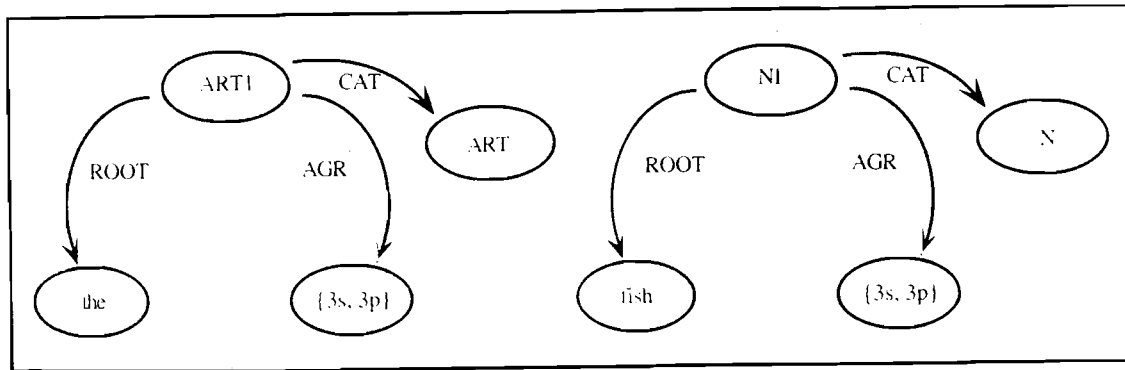


Figure 4.25 Lexical entries for *the* and *fish*

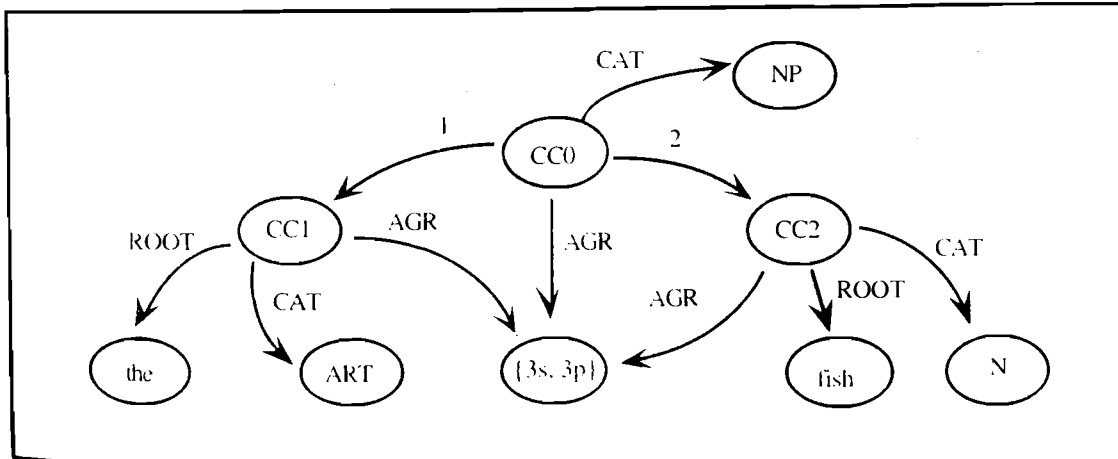


Figure 4.26 The graph for the NP *the fish*

CFG Parsing with Feature Unification

- *Back-bone* CFG is augmented with a functional description that describes unification constraints between grammar constituents
- The FS corresponding to the “root” of the grammar is constructed compositionally during parsing
- The Algorithm for building the FS of a LHS constituent of a grammar rule: For a rule $X_0 \rightarrow X_1 X_2 \dots$
 1. Create a new FS labeled X_0
 2. For each equation of the form $(x_n \dots) = (x_m \dots)$
 - (i) Follow the DAG links of x_n and x_m to obtain the two values
 - (ii) Unify the two values
 - (iii) Modify the two original FSs to point to the new unified FS

Unification Augmented GLR Parsing

- CFG is augmented with LFG-style unification equations
- The unification equations for each rule are compiled into a function that corresponds to the rule
- During parse time - the parser maintains a FS associated with each created parse node
- Whenever the parser performs a reduction by rule i - the function associated with rule i is activated and given the FSs of the RHS constituents as arguments
- The function computes the FS associated with the LHS constituent of the rule and attaches it to the parse node created for the LHS constituent.
- If the unification function fails - the reduction “fails” - LHS constituent is *not created*
- Augmented GLR uses *pseudo-unification* - assigns values instead of sharing actual structures

PROLOG and Definite Clause Grammars

Intro to PROLOG:

- A logic programming language useful for many AI applications
- Main structure is a *predicate* - a relation between entities
- Examples: `dog(sparky)` , `word(the, 1, 2)`
- Program is a database of *facts* and *deduction rules*:
- a *fact* is an axiom predicate, assumed to be true
Example: `dog(sparky)`
- a *deduction rule* is a rule by which we infer some predicate
Example:
`sibling(bob,dan) :- parent(sue,bob),parent(sue,dan)`
- Variables can be used as schemes for “ground” entities:
`sibling(X,Y) :- parent(Z,X),parent(Z,Y)`
- Variables are instantiated via unification

PROLOG and Definite Clause Grammars

Intro to PROLOG:

- Computation is a proof: using the set of facts and deduction rules to verify whether a goal predicate is true
- A goal is true if there is an instantiation of the variables by which it can be deduced from the database
- Search for a proof using DFS is built into the PROLOG interpreter

- Example: The “program”:

```
sibling(X,Y):- parent(Z,X),parent(Z,Y)
```

```
parent(tom,bob)
```

```
parent(tom,jane)
```

The goal: ?- sibling(bob,jane) will return “true”

The goal: ?- sibling(tom,jane) will return “false”

The goal: ?- sibling(X,Y) will return sibling(bob,jane)

Parsing with PROLOG

Simple Top-down CFG Parser is trivial to implement:

- Create a database of the grammar rules:

Example:

```
S(I1,I2):- NP(I1,I3),VP(I3,I2)
```

```
NP(I1,I2):- det(I1,I3),n(I3,I2)
```

```
det(I1,I2):- word(X,I1,I2),isdet(X)
```

```
isdet(a)
```

```
isdet(the)
```

The input: `word(a,1,2) word(man,2,3) ...`

The goal: `?- S(1,N)`

- Parsing is done via the PROLOG DFS search for a proof for the goal!

Parsing with PROLOG

Adding Feature Structures to a PROLOG Parser:

- Add the features as arguments to the predicates:

Example:

```
S(I1,I2,Agr):- NP(I1,I3,Agr),VP(I3,I2,Agr)
NP(I1,I2,Agr):- det(I1,I3,Agr),n(I3,I2,Agr)
det(I1,I2,Agr):- word(X,I1,I2),isdet(X,Agr)
isdet(a,3s)
isdet(the,3s)
isdet(the,3p)
```

The input: word(a,1,2) word(man,2,3) ...

The goal: ?- S(1,N,Agr)

- The feature structures can be represented in the form of a list:

Example: [[agr A] [vform V]...]

- Grammar rules can then look like:

```
S(I1,I2,FS):- NP(I1,I3,FS),VP(I3,I2,FS)
```

Parsing with PROLOG

Adding Parse Tree Construction:

- Add an extra argument variable to the predicates for the parse tree:

Example:

```
S(I1, I2, FS, s(Np, Vp)) :- NP(I1, I3, FS, Np), VP(I3, I2, FS, Vp)
```

```
NP(I1, I2, FS, np(D, N)) :- det(I1, I3, FS, D), n(I3, I2, FS, N)
```

```
det(I1, I2, FS, d(X)) :- word(X, I1, I2), isdet(X, FS)
```

```
isdet(a, 3s)
```

```
isdet(the, 3s)
```

```
isdet(the, 3p)
```

The input: `word(a, 1, 2) word(man, 2, 3) ...`

The goal: `?- S(1, K, FS, PT)`

- Produced parse tree will look something like:

```
s(np(d(a), n(man)), vp(v(cried)))
```

Definite Clause Grammars

- An convenient abbreviated format for writing grammar rules for logic-based parsing systems
- Rules can then automatically be compiled into a full database of PROLOG clauses
- Omit technical arguments that can automatically be inserted by the system:
 - word positions and constituent spans
 - explicit predicates for the lexicon entries
- Resulting grammar format looks something like:

$S(s(Np, Vp), FS) \rightarrow NP(Np, FS), VP(Vp, FS)$

$NP(np(Art, Noun), FS) \rightarrow ART(Art, FS), N(Noun, FS)$

$VP(vp(Verb), FS) \rightarrow V(Verb, FS)$