Features & Unification

Ling 571
Deep Processing Techniques for NLP
January 31, 2011
Roadmap

- Features: Motivation
  - Constraint & compactness

- Features
  - Definitions & representations

- Unification

- Application of features in the grammar
  - Agreement, subcategorization

- Parsing with features & unification
  - Augmenting the Earley parser, unification parsing

- Extensions: Types, inheritance, etc

- Conclusion
Constraints & Compactness

- Constraints in grammar
  - S → NP VP
    - They run.
    - He runs.
Constraints & Compactness

- Constraints in grammar
  - S -> NP VP
    - They run.
    - He runs.
  - But...
    - *They runs
    - *He run
    - *He disappeared the flight
Constraints & Compactness

- Constraints in grammar
  - S -> NP VP
    - They run.
    - He runs.
  - But...
    - *They runs
    - *He run
    - *He disappeared the flight
  - NP -> Det Nom
    - This flight
Constraints & Compactness

- Constraints in grammar
  - S -> NP VP
    - They run.
    - He runs.
  - But...
    - *They runs
    - *He run
    - *He disappeared the flight
  - NP -> Det Nom
    - This flight
    - These flights
Constraints & Compactness

- Constraints in grammar
  - S -> NP VP
    - They run.
    - He runs.
  - But...
    - *They runs
    - *He run
    - *He disappeared the flight
  - NP -> Det Nom
    - This flight
    - These flights
    - *This flights
Constraints & Compactness

- Constraints in grammar
  - S -> NP VP
    - They run.
    - He runs.
  - But...
    - *They runs
    - *He run
    - *He disappeared the flight
  - NP -> Det Nom
    - This flight
    - These flights
    - *This flights

- Violate agreement (number), subcategorization
Enforcing Constraints

- Enforcing constraints
Enforcing Constraints

- Enforcing constraints
- Add categories, rules
Enforcing Constraints

- Enforcing constraints
  - Add categories, rules
    - Agreement:
      - S -> NPsg3p VPsg3p,
      - S -> NPpl3p VPpl3p,
Enforcing Constraints

- Enforcing constraints
- Add categories, rules
  - Agreement:
    - $S \rightarrow NP_{sg3p} \ VP_{sg3p}$,
    - $S \rightarrow NP_{pl3p} \ VP_{pl3p}$,
  - Subcategorization:
    - $VP \rightarrow V_{trans} \ NP$,
    - $VP \rightarrow V_{intrans}$,
    - $VP \rightarrow V_{ditrans} \ NP \ NP$
Enforcing Constraints

- Enforcing constraints
  - Add categories, rules
    - Agreement:
      - S-> NPsg3p VPsg3p,
      - S-> NPpl3p VPpl3p,
    - Subcategorization:
      - VP-> Vtrans NP,
      - VP -> Vintrans,
      - VP->Vditrans NP NP

- Explosive!, loses key generalizations
Features

- person: $1^{st}$, $2^{nd}$, $3^{rd}$
  - I, we; you; he, she, they
  - am, are, is
Features

- **person:** 1\(^{st}\), 2\(^{nd}\), 3\(^{rd}\)
  - I, we; you; he, she, they
  - am, are, is
- **number:** sg, pl
  - I am; we are
Features

- **person**: 1\textsuperscript{st}, 2\textsuperscript{nd}, 3\textsuperscript{rd}
  - I, we; you; he, she, they
  - am, are, is

- **number**: sg, pl
  - I am; we are

- **case**: nom, acc
  - I, he; me, him
Features

- person: 1\textsuperscript{st}, 2\textsuperscript{nd}, 3\textsuperscript{rd}
  - I, we; you; he, she, they
  - am, are, is
- number: sg, pl
  - I am; we are
- case: nom, acc
  - I, he; me, him
- gender: masc, fem, neut
Features

- **person:** 1\textsuperscript{st}, 2\textsuperscript{nd}, 3\textsuperscript{rd}
  - I, we; you; he, she, they
  - am, are, is

- **number:** sg, pl
  - I am; we are

- **case:** nom, acc
  - I, he; me, him

- **gender:** masc, fem, neut

- **animacy:** +/-

- etc
Why features?

- Need compact, general constraints
- $S \rightarrow NP\ VP$
Why features?

- Need compact, general constraints
  - S -> NP VP
    - Only if NP and VP agree
Why features?

- Need compact, general constraints
  - S -> NP VP
    - Only if NP and VP agree

- How can we describe agreement, subcat?
Why features?

- Need compact, general constraints
  - S -> NP VP
    - Only if NP and VP agree

- How can we describe agreement, subcat?
  - Decompose into elementary features that must be consistent
  - E.g. Agreement
Why features?

- Need compact, general constraints
  - \( S \rightarrow \text{NP VP} \)
    - Only if NP and VP agree

- How can we describe agreement, subcat?
  - Decompose into elementary features that must be consistent
  - E.g. Agreement
    - Number, person, gender, etc
Why features?

- Need compact, general constraints
  - $S \rightarrow NP \ VP$
    - Only if NP and VP agree

- How can we describe agreement, subcat?
  - Decompose into elementary features that must be consistent
  - E.g. Agreement
    - Number, person, gender, etc

- Augment CF rules with feature constraints
  - Develop mechanism to enforce consistency
  - Elegant, compact, rich representation
Feature Representations

- Fundamentally, Attribute-Value pairs
- Features: atomic symbols from a finite set
Feature Representations

- Fundamentally, Attribute-Value pairs
  - Features: atomic symbols from a finite set

- Values may be
  - Atomic symbols from a finite set

Attribute-value matrix (AVM)
Feature Representations

• Fundamentally, Attribute-Value pairs
  • Features: atomic symbols from a finite set

• Values may be
  • Atomic symbols from a finite set

Attribute-value matrix (AVM)
Feature Representations

- Fundamentally, Attribute-Value pairs
  - Features: atomic symbols from a finite set
  - Values may be
    - Atomic symbols from a finite set

Attribute-value matrix (AVM)

\[
\begin{align*}
\text{NUMBER} & \quad \text{PL} \\
\text{PERSON} & \quad 3
\end{align*}
\]
Feature Representations

- Fundamentally, Attribute-Value pairs
  - Features: atomic symbols from a finite set
  - Values may be
    - Atomic symbols from a finite set

Attribute-value matrix (AVM)
Feature Representations

- Fundamentally, Attribute-Value pairs
  - Features: atomic symbols from a finite set
  - Values may be
    - Atomic symbols from a finite set

Attribute-value matrix (AVM)
Feature Representations

- Fundamentally, Attribute-Value pairs
  - Features: atomic symbols from a finite set

- Values may be
  - Atomic symbols from a finite set
  - Values may also be feature structures themselves

Attribute-value matrix (AVM)

- CAT
- AGREEMENT
- NP
- NUMBER
- PERSON
- PL
- 3
Feature Representations

- Feature path:
  - Sequence of features through a feature structure leading to a particular value

(\(\text{CAT} \rightarrow \text{AGREEMENT} \rightarrow \text{NP} \rightarrow \text{NUMBER} \rightarrow \text{PERSON} \rightarrow 3\))
Feature Representations

- Feature path:
  - Sequence of features through a feature structure leading to a particular value

\[
\begin{align*}
\text{CAT} & \quad \text{NP} \\
\text{AGREEMENT} & \quad \text{NUMBER} & \text{PL} \\
& \quad \text{PERSON} & 3
\end{align*}
\]

\(<\text{AGREEMENT NUMBER}> \rightarrow \text{PL}\)
Feature Representations

- Feature path:
  - Sequence of features through a feature structure leading to a particular value

\[
\begin{array}{c}
\text{CAT} \\
\text{AGREEMENT} \\
\text{NP} \\
\text{NUMBER} \\
\text{PERSON} \\
\end{array}
\]

\[
\begin{array}{c}
\text{PL} \\
\text{3} \\
\end{array}
\]

\[
<\text{AGREEMENT NUMBER}> \rightarrow \text{PL}
\]

\[
<\text{AGREEMENT PERSON}> \rightarrow 3
\]
Feature Representations

- Reentrant feature structures
  - Features share some feature structure as value
    - Not merely equal values
    - Shared substructure
    - Feature paths lead to same node

```
CAT                           S
HEAD   AGREEM
T

NUMBER            PL
PERSON              3
SUBJECT
AGREEMENT
1
```
Head-Subject Agreement

CAT

HEAD AGREEM’T

SUBJECT

S

NUMBER

PERSON 3

AGREEMENT 1

PL
Feature representations

- Feature structures can also be represented as DAGs
  - Directed, acyclic graphs
    - Edges are features
    - Nodes values

![Diagram of a DAG with nodes CAT, NP, NUMBER, AGREEMENT, PERSON, and edges showing feature relationships. The diagram illustrates the structure of feature representations.]
Reentrant DAG
Unification

- Two key roles:
Unification

- Two key roles:
  - Merge compatible feature structures
Unification

- Two key roles:
  - Merge compatible feature structures
  - Reject incompatible feature structures
Unification

- Two key roles:
  - Merge compatible feature structures
  - Reject incompatible feature structures

- Two structures can unify if
Unification

- Two key roles:
  - Merge compatible feature structures
  - Reject incompatible feature structures

- Two structures can unify if
  - Feature structures are identical
    - Result in same structure
Unification

- Two key roles:
  - Merge compatible feature structures
  - Reject incompatible feature structures

- Two structures can unify if
  - Feature structures are identical
    - Result in same structure
  - Feature structures match where both have values, differ in missing or underspecified
    - Resulting structure incorporates constraints of both
Subsumption

- Relation between feature structures
  - Less specific f.s. subsumes more specific f.s.
  - F.s. F subsumes f.s. G iff
    - For every feature x in F, F(x) subsumes G(x)
    - For all paths p and q in F s.t. F(p)=F(q), G(p)=G(q)
Subsumption

• Relation between feature structures
  • Less specific f.s. subsumes more specific f.s.
  • F.s. $F$ subsumes f.s. $G$ iff
    • For every feature $x$ in $F$, $F(x)$ subsumes $G(x)$
    • For all paths $p$ and $q$ in $F$ s.t. $F(p)=F(q)$, $G(p)=G(q)$

• Examples:
  • A: [Number SG], B: [Person 3]
  • C: [Number SG]
    • [Person 3]
Subsumption

- Relation between feature structures
  - Less specific f.s. subsumes more specific f.s.
  - F.s. F subsumes f.s. G iff
    - For every feature x in F, F(x) subsumes G(x)
    - For all paths p and q in F s.t. F(p)=F(q), G(p)=G(q)

- Examples:
  - A: [Number SG], B: [Person 3]
  - C:[Number SG]
    - [Person 3]
  - A subsumes C
Subsumption

• Relation between feature structures
  • Less specific f.s. subsumes more specific f.s.
  • F.s. F subsumes f.s. G iff
    • For every feature x in F, F(x) subsumes G(x)
    • For all paths p and q in F s.t. F(p)=F(q), G(p)=G(q)

• Examples:
  • A: [Number SG], B: [Person 3]
  • C:[Number SG]
    • [Person 3]
  • A subsumes C; B subsumes C
Subsumption

• Relation between feature structures
  • Less specific f.s. subsumes more specific f.s.
  • F.s. F subsumes f.s. G iff
    • For every feature x in F, F(x) subsumes G(x)
    • For all paths p and q in F s.t. F(p)=F(q), G(p)=G(q)

• Examples:
  • A: [Number SG], B: [Person 3]
  • C:[Number SG]
    • [Person 3]
  • A subsumes C; B subsumes C; B,A don’t subsume
    • Partial order on f.s.
Subsumption

- Relation between feature structures
  - Less specific f.s. subsumes more specific f.s.
  - F.s. F subsumes f.s. G iff
    - For every feature x in F, F(x) subsumes G(x)
    - For all paths p and q in F s.t. F(p)=F(q), G(p)=G(q)

- Examples:
  - A: [Number SG], B: [Person 3]
  - C:[Number SG]
    - [Person 3]
  - A subsumes C; B subsumes C; B,A don’t subsume
  - Partial order on f.s.
Unification

• Two structures can unify if
  • Feature structures are identical
    • Result in same structure

  • Feature structures match where both have values, differ in missing or underspecified
    • Resulting structure incorporates constraints of both
Unification Examples

- Identical
- [Number SG] U [Number SG]
Unification Examples

- Identical
  - \([\text{Number SG}] \cup [\text{Number SG}] = [\text{Number SG}]\)

- Underspecified
  - \([\text{Number SG}] \cup [\text{Number [ ]]}\)
Unification Examples

- Identical
  - \([\text{Number SG}] \cup [\text{Number SG}] = [\text{Number SG}]\)

- Underspecified
  - \([\text{Number SG}] \cup [\text{Number [ ]}] = [\text{Number SG}]\)

- Different specification
  - \([\text{Number SG}] \cup [\text{Person 3}]\)
Unification Examples

- **Identical**
  - [Number SG] U [Number SG] = [Number SG]

- **Underspecified**
  - [Number SG] U [Number []] = [Number SG]

- **Different specification**
  - [Number SG] U [Person 3] = [Number SG]
  - [Person 3]
  - [Number SG] U [Number PL]
Unification Examples

- Identical
  - \([\text{Number SG}] \cup [\text{Number SG}] = [\text{Number SG}]\)

- Underspecified
  - \([\text{Number SG}] \cup [\text{Number [ ]}] = [\text{Number SG}]\)

- Different specification
  - \([\text{Number SG}] \cup [\text{Person 3}] = [\text{Number SG}]\)
  - \([\text{Person 3}]\)

- Mismatched
  - \([\text{Number SG}] \cup [\text{Number PL}] \rightarrow \text{Fails!}\)
More Unification Examples

\[
\begin{align*}
(A G R E E M E N T \ [1] \\
S U B J E C T \ (A G R E E M E N T \ [1])) & \ U \\
S U B J E C T \ & \ A G R E E M E N T \ (P E R S O N N U M B E R \ 3 \ S G) \\
A G R E E M E N T \ [1] \ & \ (P E R S O N N U M B E R \ 3 \ S G)
\end{align*}
\]
Features in CFGs: Agreement

- Goal:
  - Support agreement of NP/VP, Det Nominal

- Approach:
  - Augment CFG rules with features
  - Employ head features
    - Each phrase: VP, NP has head
      - Head: child that provides features to phrase
      - Associates grammatical role with word
      - VP – V; NP – Nom, etc
Agreement with Heads and Features

VP -> Verb NP
<VP HEAD> = <Verb HEAD>

NP -> Det Nominal
<NP HEAD> = <Nominal HEAD>
<Det HEAD AGREEMENT> = <Nominal HEAD AGREEMENT>

Nominal -> Noun
<Nominal HEAD> = <Noun HEAD>

Noun -> flights
<Noun HEAD AGREEMENT NUMBER> = PL

Verb -> serves
<Verb HEAD AGREEMENT NUMBER> = SG
<Verb HEAD AGREEMENT PERSON> = 3
Feature Applications

- Subcategorization:
  - Verb-Argument constraints
    - Number, type, characteristics of args (e.g. animate)
    - Also adjectives, nouns

- Long distance dependencies
  - E.g. filler-gap relations in wh-questions, rel
Implementing Unification

- Data Structure:
  - Extension of the DAG representation
  - Each f.s. has a content field and a pointer field
    - If pointer field is null, content field has the f.s.
    - If pointer field is non-null, it points to actual f.s.
Implementing Unification: II

- Algorithm:
  - Operates on pairs of feature structures
    - Order independent, destructive
  - If fs1 is null, point to fs2
  - If fs2 is null, point to fs1
  - If both are identical, point fs1 to fs2, return fs2
    - Subsequent updates will update both
  - If non-identical atomic values, fail!
Implementing Unification:

III

• If non-identical, complex structures
  • Recursively traverse all features of fs2
  • If feature in fs2 is missing in fs1
    • Add to fs1 with value null
• If all unify, point fs2 to fs1 and return fs1
Unification

function UNIFY(f1-orig, f2-orig) returns f-structure or failure

f1 ← Dereferenced contents of f1-orig
f2 ← Dereferenced contents of f2-orig

if f1 and f2 are identical then
    f1 pointer ← f2
    return f2
else if f1 is null then
    f1 pointer ← f2
    return f2
else if f2 is null then
    f2 pointer ← f1
    return f1
else if both f1 and f2 are complex feature structures then
    f2 pointer ← f1
    for each f2-feature in f2 do
        f1-feature ← Find or create a corresponding feature in f1
        if UNIFY(f1-feature.value, f2-feature.value) returns failure then
            return failure
    return f1
else return failure
Example

\[
\begin{align*}
&\text{AGREEMENT [1]} & &\text{NUMBER SG} \\
&\text{SUBJECT} & &\text{AGREEMENT [1]} \\
&\text{SUBJECT} & &\text{AGREEMENT} & &\text{PERSON 3} \\
&\text{[ AGREEMENT [1]] U [AGREEMENT [PERSON 3]]} \\
&\text{[NUMBER SG] U [PERSON 3]} \\
&\text{[NUMBER SG] U [PERSON 3]} \\
&\text{[PERSON NULL]} 
\end{align*}
\]
Unification Example

Grammar entry for sentence

(From S.F., 2010)
Unification Example

Grammar entry for NP

(From S.F., 2010)
Unification Example

Lexical entries

(From S.F., 2010)
Unification Example

Unifying a noun phrase with a determiner

(From S.F., 2010)
Unification Example

Unifying NP with Determiner

(From S.F., 2010)
Unification Example

(From S.F., 2010)
Unification and the Earley Parser

- Employ constraints to restrict addition to chart
- Actually pretty straightforward
  - Augment rules with feature structure
  - Augment state (chart entries) with DAG
    - Prediction adds DAG from rule
    - Completion applies unification (on copies)
    - Adds entry only if current DAG is NOT subsumed
Parsing with Features

- One strategy:
  - Parse as usual
  - Test completed parses for unification constraints
Parsing with Features

- One strategy:
  - Parse as usual
  - Test completed parses for unification constraints

- Pros:
  - Simple, requires little modification
Parsing with Features

- One strategy:
  - Parse as usual
  - Test completed parses for unification constraints

- Pros:
  - Simple, requires little modification

- Cons:
  - Wasted effort
  - Builds many partial parses that can’t unify
Parsing with Features

- One strategy:
  - Parse as usual
  - Test completed parses for unification constraints

- Pros:
  - Simple, requires little modification

- Cons:
  - Wasted effort
  - Builds many partial parses that can’t unify

- Integrate unification in parse construction
Augment existing Earley parser for unification
  • Fairly straightforward

Modify representations:
  • Augment CFG rules with constraints
    • Use constraints to create feature structure as DAG
  • Add DAG to state representation
    • E.g., S -> • NP VP, [0,0],[],Dag
Integrating Unification

• Main change: Completer
  • Advances in rules where next constituent matches a just-completed constituent

• Now, unifies Dag from completed constituent with the part of the feature structure in rules advanced
  • If fails, no new entry in chart

• Second change:
  • Only add state if NOT subsumed by states in chart
function EARLEY-PARSE(words, grammar) returns chart

ADDTOCHART((γ → • S, [0, 0], dagγ), chart[0])
for i ← from 0 to LENGTH(words) do
  for each state in chart[i] do
    if INCOMPLETE?(state) and
      NEXT-CAT(state) is not a part of speech then
        PREDICTOR(state)
    elseif INCOMPLETE?(state) and
      NEXT-CAT(state) is a part of speech then
        SCANNER(state)
    else
      COMPLETER(state)
  end
end
return(chart)
procedure PREDICTOR((A → α • B β, [i, j], dagA))
  for each (B → γ) in GRAMMAR-RULES-FOR(B, grammar) do
    AddToChart((B → • γ, [j, j], dagB), chart[j])
  end

procedure SCANNER((A → α • B β, [i, j], dagA))
  if B ∈ PARTS-OF-SPEECH(word[j]) then
    AddToChart((B → word[j]•, [j, j + 1], dagB), chart[j + 1])
  end

procedure COMPLETER((B → γ •, [j, k], dagB))
  for each (A → α • B β, [i, j], dagA) in chart[j] do
    if new-dag ← UNIFY-STATES(dagB, dagA, B) ≠ Fails!
      AddToChart((A → α B • β, [i, k], new-dag), chart[k])
    end
  end

procedure UNIFY-STATES(dag1, dag2, cat)
  dag1-cp ← COPYDAG(dag1)
  dag2-cp ← COPYDAG(dag2)
  UNIFY(FOLLOW-PATH(cat, dag1-cp), FOLLOW-PATH(cat, dag2-cp))

procedure AddToChart(state, chart-entry)
  if state is not subsumed by a state in chart-entry then
    Push-ON-END(state, chart-entry)
  end
Unification Parsing

- Abstracts over categories
  - $S \rightarrow NP \ VP =$>
    - $X_0 \rightarrow X_1 \ X_2; <X_0 \text{ cat}> = S; <X_1 \text{ cat}> = NP;$
    - $<X_2 \text{ cat}> = VP$
  - Conjunction:
    - $X_0 \rightarrow X_1 \ and \ X_2; <X_1 \text{ cat}> = <X_2 \text{ cat}>;$
    - $<X_0 \text{ cat}> = <X_1 \text{ cat}>$

- Issue: Completer depends on categories

- Solution: Completer looks for DAGs which unify with the just-completed state’s DAG
Extensions

- Types and inheritance
  - Issue: generalization across feature structures
    - E.g. many variants of agreement
      - More or less specific: 3rd vs sg vs 3rdsg
  - Approach: Type hierarchy
    - Simple atomic types match literally
    - Multiple inheritance hierarchy
      - Unification of subtypes is most general type that is more specific than two input types
    - Complex types encode legal features, etc
Conclusion

- Features allow encoding of constraints
  - Enables compact representation of rules
  - Supports natural generalizations

- Unification ensures compatibility of features
  - Integrates easily with existing parsing mech.

- Many unification-based grammatical theories