Lecture 19: Features & Categories

CS 181O
Spring 2016
Kim Bruce

Features

- So far have ignored complexities due to features, e.g., gender, number, person, case, tense, …
- Can add features to cfg to require agreement

Modifying Grammar

- Replace S → NP VP by
  - $S_{\varnothing} \rightarrow NP_{\{Sg\}}\ VP_{\varnothing}$
  - $S_{\varnothing} \rightarrow NP_{\{Sg\}}\ VP_{\{Sg\}}$
  - $S_{\varnothing} \rightarrow NP_{\{Pl\}}\ VP_{\varnothing}$
  - $S_{\varnothing} \rightarrow NP_{\{Pl\}}\ VP_{\{Pl\}}$
- If start w/cfg, then end with cfg

Features

- Should group them, but simpler to include all in same type.

```
data Feat = Masc  | Fem  | Neutr | MascOrFem   — gender
        | Sg   | Pl                            — number
        | Fst  | Snd  | Thrd                   — person
        | Nom  | AccOrDat                      — case
        | Pers  | Refl | Wh                    — pronoun type
        | Tense | Infl                         — tense
        | On   | With | By | To | From         — prep type
        deriving (Eq,Show,Ord)

type Agreement = [Feature]
```

Functions

- gender, number, person, ... check for kind of feature
- prune function eliminates redundancy
  - Want at most one feature in each category
  - Function combine lets add features together as long as at most one in each group in final.

Category

- List of features associated with a lexical item
  - data Cat = Cat Phon CatLabel Agreement [Cat] deriving Eq
  - type Phon = String — string representing word
  - type CatLabel = String — part of speech
  - Agreement is list of features
  - Last arg is subcategorization list
    - list of items can be combined with. E.g., transitive verb needs np with feature AccOrDat, ditransitive also needs prep phrase with To feature.

Extract Values from Cat

- phon :: Cat → String — Returns spelling
- catLabel :: Cat → String — returns POS
- fs :: Cat → String — returns features
- subcatList :: Cat → String

Imposing Roles

- Syntactic rules impose features on components when recognized.
  - E.g., S → NP VP, imposes Nom on NP
  - combine cat1 cat2 attempts to combine, but requires at most one entry in each type of feature
  - agree cat1 cat2 determines whether can combine 2 cats
  - assign f oldCat tries to add feature f to oldCat.
    - If compatible gives list with that new category
    - If not compatible gives empty list

Lexicon

- lexicon :: String →[Cat]
  - Associates words with the possible categorizations for them.
  - Look through definitions in text & P.hs
  - Esp, see pronouns, determiners (all vs every), verbs (esp subcategorization lists)

Using the Lexicon

- lexer :: String -> Words
- lexer = preproc . words . (map toLower) . scan
- *lexer puts white space before punctuation, converts to lower case, breaks it into words, and then gets rid of punctuation and combines/simplifies words*
  - *e.g. "at most" becomes "at_most"*

Using Lexicon

- lookUpWord db w — *looks up cat for word in db*
- collectCats db words — *returns list of words and their cats*
  - collectCats lexicon (words "he loved her")
    [[he NP[Pers,Thrd,Sg,Nom,Masc],loved VP[Tense],her NP[Pers,Thrd,Sg,AccOrDat,Fem]]]

Parsing Using Lexicon

```
prs :: String -> [ParseTree Cat Cat]
prs string = let ws = lexer string
   in  [ s | catlist <- collectCats lexicon ws,
             (s,[])  <- parseSent catlist ]
```

— *Grab lexicon entries for words in ws, parse the list to build a parse tree for a sentence. For all parsers that use up all input, return parse trees*

Building Parse Tree

- Top level function:
  - > prs "I did love her” *returns:*
  - [[.S[] [i NP[Sg,Fst,Nom,Pers],
    [.VP[] [did AUX[],[.VP[Infl] [love VP[Infl],
    her NP[Pers,Thrd,Sg,AccOrDat,Fem]]]]]]]]
  - prs "I loved her” *returns*
  - [[.S[] [i NP[Sg,Fst,Nom,Pers],
    [.VP[Tense] [loved VP[Tense],
    her NP[Pers,Thrd,Sg,AccOrDat,Fem]]]]]]

How do we build it?

Parsing with Categories

- Leaves and interior nodes will hold categories,
  - Only leaves hold actual text in phon field
- ParseTree Cat Cat
  - t2c:: ParseTree Cat Cat → Cat
    returns category at root of tree
  - agreeC t1 t2
    returns if categories at roots of t1 and t2 compatible
  - assignT f pts
    adds feature f to roots of parse trees in its root

Parse trees with Categories

- Build parsers as before, but must respect category compatibility.
  - PARSER Cat Cat
    = Parser Cat (ParseTree Cat Cat)
    = [Cat] →[(ParseTree Cat Cat, [Cat])
  - *leafP lab input creates list of parse trees from items in input whose label matches lab, e.g. leafP “NP” cs grabs noun phrases.*
  - leafP :: CatLabel → PARSER Cat Cat
  - leafP label [] = []
  - leafP label (c:cs) =
    [(Leaf c, cs) | catLabel c == label}

Parsing Sentences

```
sRule :: PARSER Cat Cat
sRule = \ xs ->        — xs is input cats
    [ (Branch (Cat "_" "S" [] []) [np',vp],zs) | — no features
      (np,ys) <- parseNP xs,            — parse NP
      (vp,zs) <- parseVP ys,            — then parse VP
      np'    <- assignT Nom np,         — make np' nominative
      agreeC np vp,                     — make sure features compatible
      subcatList (t2c vp) == [] ]       — no subcat constraints on vp
parseSent = sRule    — because only one rule
```

Parsing Noun Phrases

```
npRule = \ xs ->
  [ (Branch (Cat "_" "NP" fs []) [det,cn],zs) |
    (det,ys) <- parseDET xs,   — parse determiner
    (cn,zs)  <- parseCN  ys,   — then parse CN
    fs       <- combine (t2c det) (t2c cn), — combine features
    agreeC det cn ] — only create tree if features compatible
— recognize NP's in input cats or Det-NP pairs
parseNP :: PARSER Cat Cat
parseNP = leafP "NP" <|> npRule
```

Prepositional Phrases

```
ppRule = \ xs ->
  [ (Branch (Cat "_" "PP" fs []) [prep,np'],zs) |
    (prep,ys) <- parsePrep xs,  — parse preposition
    (np,zs)   <- parseNP ys,    — parse noun phrase
     np'      <- assignT AccOrDat np, — make np' accusative
     fs       <- combine (t2c prep) (t2c np') ] — combine features

parsePP :: PARSER Cat Cat
parsePP = ppRule
```

More Parsing

- See code in P2.hs for remaining rules.

Questions?