Lecture 21:
Parsing with Features

CS 181O
Spring 2016
Kim Bruce

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 (P2.hs is subset)
  - Esp, see pronouns, determiners (all vs every), verbs (esp subcategorization lists)
  - Examples:
    - lexicon "i" = [Cat "i" "NP" [Pers,Fst,Sg,Nom] []]
    - lexicon "kick" = [Cat "kick" "VP" [Infl] [Cat "_" "NP" [AccOrDat] [],
      Cat "_" "PP" [With] []],
      Cat "kick" "VP" [Infl] [Cat "_" "NP" [AccOrDat] []]]

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 first elt in input if label matches lab, e.g. leafP "NP" cs grabs first noun phrase.*
  - 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 cat list
    [ (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) == [] ]     — make sure no subcat
                                      — constraints left 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.

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*

Intensional Logic

Intension vs Extension

- Propositional and predicate logic: extensional logics
  - expressions with the same reference (or extension) may be freely substituted for each other:

    $$\phi \leftrightarrow \phi' \models \psi \leftrightarrow \psi[\phi'/\phi]$$

  - *Variant*: Leibniz's law of the indiscernability of identicals:

    $$s = t \models \psi \leftrightarrow \psi[t/s]$$

Intension vs Extension

- Not always work!!
  - “The morning star is the evening star” versus “The morning star is the morning star”.
  - “John’s mother is looking for David Oxtoby” versus “John’s mother is looking for the Pomona College president.”
- “Intensions” of the phrases are distinct
  - Frege: sense vs reference
  - “sense” is how you get to the reference
  - proposition it expresses vs. truth of proposition

Intension

- Frege:
  - Expressions do not have their normal references in intensional constructions, but refer instead to their senses.
  - They have an “indirect reference” which is to their senses.
- Truth can depend on their context (including time)
  - Barack Obama is president of the USA.

Intension

- Contrast:
  - The intension of a phrase is its conceptual content
  - The extension comprises all that exemplifies the conceptual content – i.e., all the elements satisfying the intension.
- The intension of a phrase is a mapping from the context to the extension in that context.

Why Care?

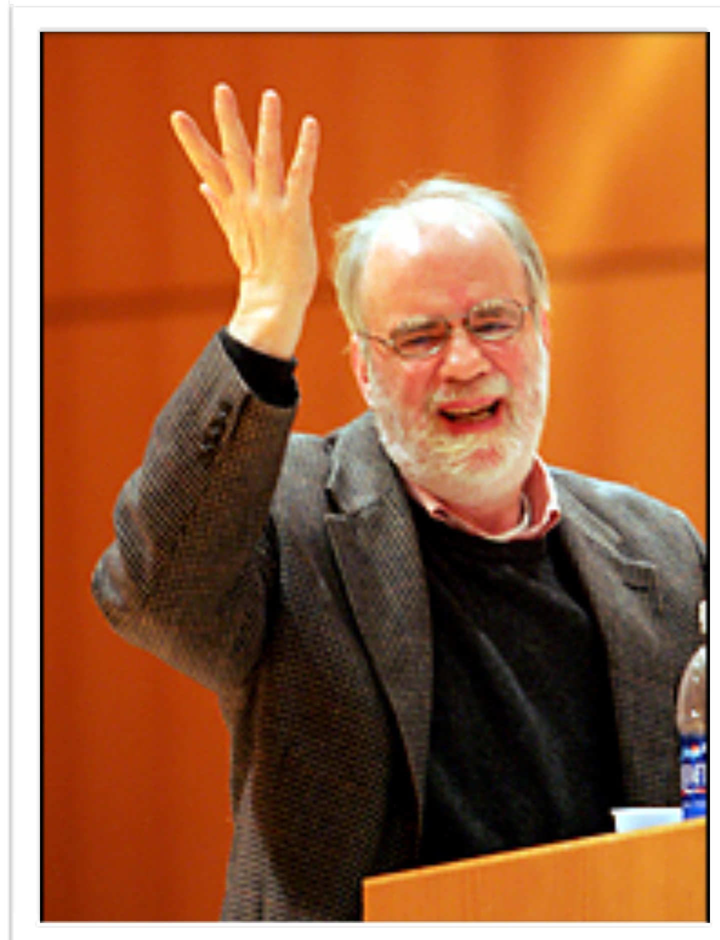
- Intensional models help interpret
  - adjectives like “fake”, “former”,
  - attitude verbs like “want”, “hope”
  - “must”, “may”, “necessarily”, “possibly”

Intensional Propositional Logic

- If p is a proposition letter, then p is a formula
- If ϕ and ψ are formulas then so are $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \rightarrow \psi$, $\neg\phi$, and $O\phi$.
  - Meaning of $O\phi$ will depend on the set of contexts that we are interested in
    - Necessarily ϕ, always in the future ϕ, ...

Saul Kripke

- Research in modal logic as a high school student in Omaha.
- Published as a freshman at Harvard.
- No advanced degrees.
- NYT: “the world’s greatest living philosopher, perhaps the greatest since Wittgenstein.”



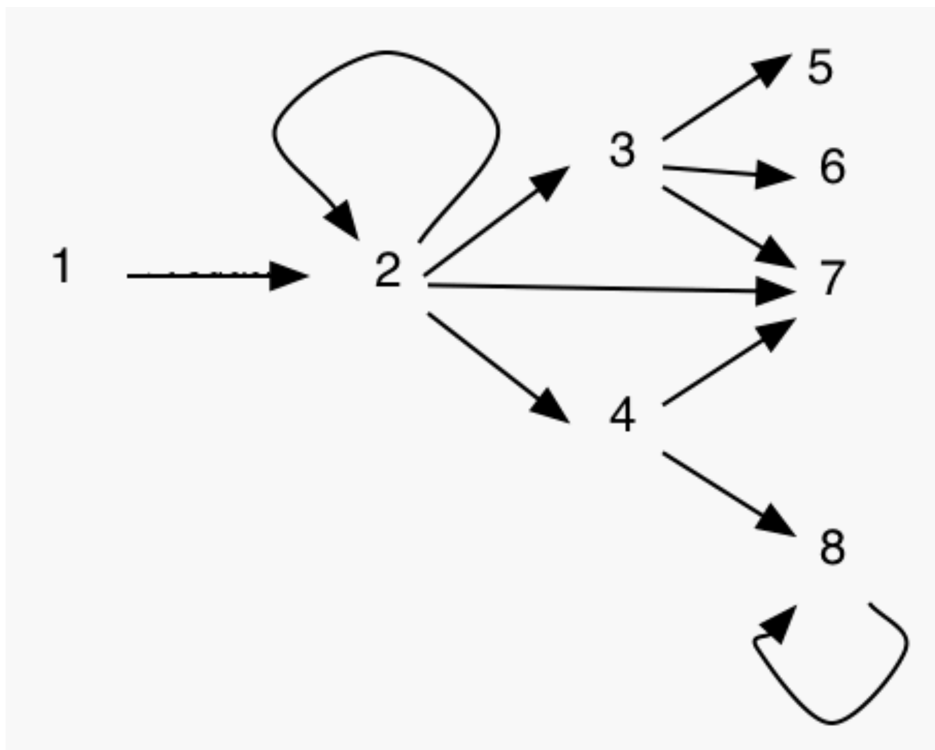
http://www.nytimes.com/2006/01/28/books/28krip.html?_r=2&

Models for Intensional Logic

- A (Kripke) model M consists of
  - a non-empty set W of contexts,
  - a binary relation R on W, the accessibility relation
  - A valuation function V which assigns a truth value $V_w(p)$ to every proposition letter p in each context w.
- Contexts referred to as possible worlds
- Combination of W,R called a "frame"

Contexts & Accessibility

- Accessibility relation:
  - R = {(v1, v2), (v2, v2), (v2, v3), (v2, v4), (v2, v7), (v3, v5), (v3, v6), (v3, v7), (v4, v7), (v4, v8), (v8, v8)}
  - or



Truth in Intensional Propositional Logic

- Let M be model with W as set of possible worlds, R as accessibility relation, and V as valuation, then $V_{M,w}(\phi)$, the truth value of ϕ in w given M is defined as follows:
- $V_{M,w}(p) = V_w(p)$ for all proposition letters p.
- $V_{M,w}(\neg\phi)$ = true iff $V_w(\phi)$ = false.
- $V_{M,w}(\phi \rightarrow \psi)$ = true iff
  $V_{M,w}(\phi)$ = false or $V_{M,w}(\psi)$ = true.
- ...
- $V_{M,w}(O\phi)$ = true iff $\forall w' \in W$ s.t. $\langle w,w' \rangle \in R$,
  $V_{M,w'}(\phi)$ = true.

Modal Logic: Necessity

- Replace $O\phi$ by $\Box\phi$, dual $\Diamond\phi \equiv \neg\Box\neg\phi$
  - $\Box\phi$ means "necessarily ϕ"
  - $\Diamond\phi$ means "possibly ϕ"
  - If ϕ stands for "you understand me", then translate: "It is possible that you understand me, but it isn't necessary" as $\Diamond\phi \wedge \neg\Box\phi$

Questions?