LING 388: Language and Computers

Sandiway Fong

Lecture 7
Administrivia

• Today’s Topics
  1. Exercise 2 from last time
  2. Grammar rules using Prolog
Exercise 2

• **Exercise 4.5:**
  
database
  tran(eins,one).
  tran(zwei, two).
  tran(drei, three).
  tran(vier, four).
  tran(fuenf, five).
  tran(sechs, six).
  tran(sieben, seven).
  tran(acht, eight).
  tran(neun, nine).

• Write a predicate `listtran(G,E)` which translates a list of German number words to the corresponding list of English number words.

• For example:
  
  ?- listtran([eins,neun,zwei],X).
     X = [one,nine,two]
  
  ?- listtran(X,[one,seven,six,two]).
     X = [eins,sieben,sechs,zwei]
Exercise 2

German/English word list translation

• Part 1: *what is the base case?*
  – The empty list ...
  – Write the Prolog code

  – `listtran([],[]).`
Exercise 2

German/English word list translation

• Part 2: what is the recursive case?
  – Given two lists [G|L1] for German and [E|L2] for English, what needs to be recursively true...
  – Write the Prolog code

\[
\text{listtran}([G|L1],[E|L2]) \leftarrow \\
\text{tran}(G,E), \\
\text{listtran}(L1,L2).
\]
What is a grammar?

- **informally**
  - set of rules to parse or “diagram” a sentence

- **example**

```
  /\  
 /   
NP   VP
 / \\
the man Verb NP
  \\
  took the book
```

From a parse, we can extract a grammar
What is a grammar?

- **example**

  ![Parse Tree Diagram]

  - **production (or grammar) rule format**
    - \( \text{LHS} \rightarrow \text{RHS} \)
      - \( \text{LHS} = \) Left Hand Side
      - \( \rightarrow = \) “expands to” or “rewrites to”
      - \( \text{RHS} = \) Right Hand Side
  
  - **example**
    - Sentence \( \rightarrow \) NP VP
    - VP \( \rightarrow \) Verb NP
    - Verb \( \rightarrow \) took
    - NP \( \rightarrow \) the man
    - NP \( \rightarrow \) the book

Using a grammar, we can derive the parse tree
What is a grammar?

- **Example**
  - Sentence $\rightarrow$ NP VP
  - VP $\rightarrow$ Verb NP
  - Verb $\rightarrow$ took
  - NP $\rightarrow$ the man
  - NP $\rightarrow$ the book

- **Derivation**
  - *top-down (one of many)*
    1. Sentence
    2. NP VP
    3. NP Verb NP
    4. NP took NP
    5. the man took NP
    6. the man took the book

- **Derivation**
  - *top-down (alternative)*
    1. Sentence
    2. NP VP
    3. the man VP
    4. the man Verb NP
    5. the man took NP
    6. the man took the book
What is a grammar?

- **example**
  - Sentence $\rightarrow$ NP VP
  - VP $\rightarrow$ Verb NP
  - Verb $\rightarrow$ took
  - NP $\rightarrow$ the man
  - NP $\rightarrow$ the book

- **derivation**
  - bottom-up (*one of many*)
    1. the man took the book
    2. NP took the book
    3. NP Verb the book
    4. NP Verb NP
    5. NP VP
    6. Sentence

- **derivation**
  - bottom-up (*alternative*)
    1. the man took the book
    2. the man took NP
    3. the man Verb NP
    4. the man VP
    5. NP VP
    6. Sentence
What is a grammar?

- **example**
  - Sentence $\rightarrow$ NP VP
  - VP $\rightarrow$ Verb NP
  - Verb $\rightarrow$ took
  - NP $\rightarrow$ the man
  - NP $\rightarrow$ the book

- *this grammar can generate more than one sentence*

- **examples**
  - the man took the book
  - #the book took the man # = semantically odd
  - other sentences?

- **add new rule**
  - Verb $\rightarrow$ bought

- **examples**
  - the man took the book
  - the man bought the book
  - #the book took the man # = semantically odd
  - #the book bought the man
What is a grammar?

- **example**
  - Sentence $\rightarrow$ NP VP
  - VP $\rightarrow$ Verb NP
  - Verb $\rightarrow$ took
  - NP $\rightarrow$ the man
  - NP $\rightarrow$ the book

- **formally:** *a grammar contains the following 4 things*
  - $\langle N, T, P, S \rangle$
    - a set of non-terminal symbols ($N$)
    - a set of terminal symbols ($T$)
    - production rules ($P$) of the form
    - a designated start symbol ($S$)

- **example**
  - Non-terminals: \{Sentence, VP, NP, Verb\}
  - Terminals: \{the, man, book, took\}
  - Start symbol: Sentence
  - Production rules: \*set of LHS $\rightarrow$ RHS rules\*

![Diagram](image.png)
Prolog Grammar Rules

• Good news!
  – Prolog supports grammar rules
  – it knows how to interpret them (directly)
  – it can use the grammar rules to construct a derivation automatically

Prolog’s computation rule implies a top-down, left-to-right derivation

knowing the order will be important when we deal with left and right recursive rules
Prolog Grammar Rules

- Prolog’s version of grammar rules =
- Definite Clause Grammar (DCG)
- **Prolog’s format**
  - terminals and non-terminal symbols **both** begin with lowercase letters
    - e.g. sentence, vp, np, book, took
    - *Note: variables begin with an uppercase letter (or underscore)*
  - -->
    - is the rewrite symbol
  - terminals are enclosed in square brackets to distinguish them from non-terminals (list notation)
    - e.g. [the], [book], [took]
    - list of terminals are possible, e.g. [good, luck]
  - comma (,) is the concatenation symbol
  - semicolon (;) is the disjunction symbol
  - a period (.) is required at the end of all DCG rules
Prolog Grammar Rules

- **example**
  - Sentence $\rightarrow$ NP VP
  - VP $\rightarrow$ Verb NP
  - Verb $\rightarrow$ took
  - NP $\rightarrow$ the man
  - NP $\rightarrow$ the book

- **Prolog DCG version**
  - `sentence --> np, vp.`
  - `vp --> verb, np.`
  - `verb --> [took].`
  - `np --> [the], [man].`
  - `np --> [the], [book].`

- **Important Note**
  - don’t enter these rules into the database using `assert/1`.
  - Must use a file and consult it.

*reason*: Prolog needs to translate DCG rules into underlying (regular) Prolog rules.
Prolog Grammar Rules

- **example**
  - `sentence --> np, vp.`
  - `vp --> verb, np.`
  - `verb --> [took].`
  - `np --> [the], [man].`
  - `np --> [the], [book].`

- **query format**
  - `?- sentence(S, []).`
  - `S = sentence (as a list)`
  - `[ ] = empty list`

  - i.e. call the start symbol as a predicate and
  - supply two arguments, a list and an empty list
Prolog Grammar Rules

- **example**
  
  - sentence --> np, vp.
  - vp --> verb, np.
  - verb --> [took].
  - np --> [the], [man].
  - np --> [the], [book].

- **example queries**
  
  - ?- sentence([the, man, took, the, book], []).
  - Yes
  - “the man took the book” *is a member of the language generated by the grammar*

  - ?- sentence([man, took, the, book], []).
  - No
  - “man took the book” *is not in the grammar*
  - “man took the book” *is not generated* by the grammar
Prolog Grammar Rules

• **example**
  - sentence --> np, vp.
  - vp --> verb, np.
  - verb --> [took].
  - np --> [the], [man].
  - np --> [the], [book].

other queries
• ?- sentence([the, man, took, X, book], []).  
• X = the

• ?- sentence(S, []).  
• S = [the, man, took, the, man] ;
• S = [the, man, took, the, book] ;
• S = [the, book, took, the, man] ;
• S = [the, book, took, the, book] ;
• No
Prolog Grammar Rules

- **example**
  - sentence → np, vp.
  - vp → verb, np.
  - verb → [took].
  - np → [the], [man].
  - np → [the], [book].

- **notes**
  - np → [the, man].  OK
  - np → [the, book].  OK

- **more grammar**
  - \textit{det} = \textit{determiner}
  - np → det, [man].
  - np → det, [book].
  - det → [the].
  - det → [a].
Prolog Grammar Rules

- **example**
  - sentence --> np, vp.
  - vp --> verb, np.
  - verb --> [took].
  - np --> det, [man].
  - np --> det, [book].
  - det --> [the].
  - det --> [a].

- **query**
  - ?- sentence(S, []).
  - generates 16 different answers for S
  - 2 choices for det a, the
  - 2 choices for head noun man, book
  - total of 4 different choices for NP (a | (the)) ((man) | (book))
  - 2 choices for NP as subject, as object
  - total = $4^2 = 16$
Prolog Grammar Rules

**example**
- `sentence --> np, vp.`
- `vp --> verb, np.`
- `verb --> [took].`
- `np --> det, [man].`
- `np --> det, [book].`
- `det --> [the].`
- `det --> [a].`

**query**
- `?- sentence([the,man,took|L],[]).`
- `L = [the, man] ;`
- `L = [a, man] ;`
- `L = [the, book] ;`
- `L = [a, book] ;`
- `No`
Prolog Grammar Rules

- **example**
  - sentence --> np, vp.
  - vp --> verb, np.
  - verb --> [took].
  - np --> det, [man].
  - np --> det, [book].
  - det --> [the].
  - det --> [a].

- **query**
  - X = the ;
  - X = a ;
  - No

2 choices
Prolog Grammar Rules

• The computation rule for DCG rules
  – *each time we look for a matching rule, we pattern-match against the database from the 1st rule on down*

• example
  – sentence --> np, vp.
  – vp --> verb, np.
  – verb --> [took].
  – np --> det, [man].
  – np --> det, [book].
  – det --> [the].
  – det --> [a].

• listing
  
  vp(A, B) :-
      verb(A, C),
      np(C, B).
  np(A, B) :-
      det(A, C),
      'C'(C, man, B).
  np(A, B) :-
      det(A, C),
  verb([took|A], A).
  det([the|A], A).
  det([a|A], A).
  sentence(A, B) :-
      np(A, C),
      vp(C, B).
Prolog Grammar Rules

• derivation
  • ?- sentence([the, man, took, the, book], []).  
    Call: (7) sentence([the, man, took, the, book], []) ? creep
  • Call: (8) np([the, man, took, the, book], _G353) ? creep
  • Call: (9) det([the, man, took, the, book], _G353) ? creep
  • Exit: (9) det([the, man, took, the, book], [man, took, the, book]) ? creep
  • Call: (9) 'C'(man, took, the, book), man, _G353 ? creep
  • Exit: (9) 'C'(man, took, the, book), man, [took, the, book]) ? creep
  • Exit: (8) np([the, man, took, the, book], [took, the, book]) ? creep
  • Call: (8) vp([took, the, book], []) ? creep
  • Call: (9) verb([took, the, book], _G353) ? creep
  • Exit: (9) verb([took, the, book], [took, the, book]) ? creep
  • Call: (9) np([the, book], []) ? creep
  • Call: (10) det([the, book], _G353) ? creep
  • Exit: (10) det([the, book], [book]) ? creep
  • Call: (10) 'C'(book, man, []) ? creep
  • Fail: (10) 'C'(book, man, []) ? creep
  • Redo: (10) det([the, book], _G353) ? creep
  • Fail: (10) det([the, book], _G353) ? creep
  • Redo: (9) np([the, book], []) ? creep
  • Call: (10) det([the, book], _G353) ? creep
  • Exit: (10) det([the, book], [book]) ? creep
  • Call: (10) 'C'(book, book, []) ? creep
  • Exit: (10) 'C'(book, book, []) ? creep
  • Exit: (9) np([the, book], []) ? creep
  • Exit: (8) vp([took, the, book], []) ? creep
  • Exit: (7) sentence([the, man, took, the, book], []) ? creep

• Yes

• listing

  vp(A, B) :-
    verb(A, C),
    np(C, B).

  np(A, B) :-
    det(A, C),
    'C'(C, man, B).

  np(A, B) :-
    det(A, C),

  verb([took|A], A).

  det([the|A], A).

  det([a|A], A).

  sentence(A, B) :-
    np(A, C),
    vp(C, B).
Exercise

• Add grammar rules to handle:
  – I saw the boy with a telescope

• Test your grammar on
  (NB. have to add a lexical rule for *me*)
  – the boy with a telescope saw me