LING/C SC/PSYC 438/538
Computational Linguistics

Sandiway Fong
Lecture 5: 9/4
Administrivia

- Homework 1 due tonight
Correction

• For multiple matching cases:
  \$v =~ /regexp/g
  \$v a variable

• For each match, Perl does not start from the beginning
  Instead, it must “remember” where in the string it has gotten up to

• the Perl function
  – \texttt{pos \$v}
  – returns the position in characters where the next match will begin
  – first character is at position 0 (zero)

\begin{tabular}{cccccccccccc}
  h & e & e & d & h & e & a & d & b & o & o & k  \\
  0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13
\end{tabular}

• Example:
  \begin{verbatim}
  \$x = "heed head book";
  \$x =~ /([aeiou])\1/g;
  print pos \$x, "\n";
  \$x =~ /([aeiou])\1/g;
  print pos \$x, "\n";
  \$x =~ /([aeiou])\1/g;
  print pos \$x, "\n";
  \end{verbatim}
There is so much more about Perl regular expressions we have not covered

- **backtracking**
  - \$x = "aaaab"; ~/a+/aab/

- (locally) **greedy matching**
  - \$x = "\{abc\} \{de\}"; \$x =~ /\{(.*)\}/

- **lazy** (non-greedy) **matching**
  - \$x = "\{abc\} \{de\}"; \$x =~ /\{(.*)\}/

*but it’s time to move on....*
New Topic

• Regular Grammars
  – formally equivalent to regexps
A New Programming Language

• You should (have) install(ed) SWI-Prolog on your machines
  – www.swi-prolog.org
  – a free download for various platforms (Windows, MacOSX, Linux)
  – Prolog is a logic programming language that has a built-in grammar rule facility
  – can encode regular grammars and much, much more ...
Chomsky Hierarchy

Chomsky Hierarchy

• division of grammar into subclasses partitioned by “generative power/capacity”

• **Type-0 General rewrite rules**
  – Turing-complete, powerful enough to encode any computer program
  – can simulate a Turing machine
  – anything that’s “computable” can be simulated using a Turing machine

• **Type-1 Context-sensitive rules**
  – weaker, but still very power
  – $a^n b^n c^n$

• **Type-2 Context-free rules**
  – weaker still
  – a^n b^n

• **Type-3 Regular grammar rules**
  – very restricted
  – Regular Expressions a*b*
  – Finite State Automata (FSA)
Chomsky Hierarchy

DCG = Type-0

Type-0
Type-1
Type-2
Type-3

DCG
FSA
Regular Expressions
Regular Grammars
Prolog’s Grammar Rule System

• known as “Definite Clause Grammars” (DCG)
  – based on type-2 (context-free grammars)
  – but with extensions
  – powerful enough to encode the hierarchy all the way up to type-0

  – we’ll start with the bottom of the hierarchy
    • i.e. the least powerful
    • regular grammars (type-3)
Definite Clause Grammars (DCG)

• **some facts**
  – built-in feature
    • not an accident:
    • Prolog was originally implemented to support natural language processing (Colmerauer 1972)
  – notation designed to resemble grammar rules used by linguists and computer scientists
Definite Clause Grammars (DCG)

• Prolog: PROgrammation en LOGique
  – man-machine communication system
    • TOUT PSYCHIATRE EST UNE PERSONNE.
    • Every psychiatrist is a person.
    • CHAQUE PERSONNE QU'IL ANALYSE, EST MALADE.
    • Every person he analyzes is sick.
    • *JACQUES EST UN PSYCHIATRE A MARSEILLE.
    • Jacques is a psychiatrist in Marseille.

  – machine queries:
    • EST-CE QUE *JACQUES EST UNE PERSONNE?
    • Is Jacques a person?
    • OU EST *JACQUES?
    • Where is Jacques?
    • EST-CE QUE *JACQUES EST MALADE?
    • Is Jacques sick?

Definite Clause Grammars (DCG)

- **background**
  - a “typical” formal grammar contains 4 things
  - \(<N,T,P,S>\)
    - **a set of non-terminal symbols (N)**
      - appear on the left-hand-side (LHS) of production rules
      - these symbols will be expanded by the rules
    - **a set of terminal symbols (T)**
      - appear on the right-hand-side (RHS) of production rules
      - consequently, these symbols cannot be expanded
    - **production rules (P) of the form**
      - LHS $\rightarrow$ RHS
      - In regular and CF grammars, LHS must be a single non-terminal symbol
      - RHS: a sequence of terminal and non-terminal symbols: possibly with restrictions, e.g. for regular grammars
    - **a designated start symbol (S)**
      - a non-terminal to start the derivation
Definite Clause Grammars (DCG)

Background

• a “typical” formal grammar contains 4 things
• \(<N,T,P,S>\)
  – a set of non-terminal symbols (N)
  – a set of terminal symbols (T)
  – production rules (P) of the form LHS → RHS
  – a designated start symbol (S)

Example

S → aB
B → aB
B → bC
B → b
C → bC
C → b

Notes:

• Start symbol: S
• Non-terminals: \{S,B,C\} (uppercase letters)
• Terminals: \{a,b\} (lowercase letters)

regular grammars are restricted to two kinds of rules only
Definite Clause Grammars (DCG)

- **Example**
  - **Formal grammar**
    - \( S \rightarrow aB \)
    - \( B \rightarrow aB \)
    - \( B \rightarrow bC \)
    - \( B \rightarrow b \)
    - \( C \rightarrow bC \)
    - \( C \rightarrow b \)
  - **Prolog format**
    - \( s \rightarrow [a],b. \)
    - \( b \rightarrow [a],b. \)
    - \( b \rightarrow [b],c. \)
    - \( b \rightarrow [b]. \)
    - \( c \rightarrow [b],c. \)
    - \( c \rightarrow [b]. \)

- **Notes:**
  - Start symbol: \( S \)
  - Non-terminals: \{S,B,C\} *(uppercase letters)*
  - Terminals: \{a,b\} *(lowercase letters)*

**Prolog format:**
- both terminals and non-terminal symbols begin with lowercase letters
  - Variables begin with an uppercase letter (or underscore)
- \( \rightarrow \) is the rewrite symbol
- terminals are enclosed in square brackets *(list notation)*
- nonterminals don’t have square brackets surrounding them
- the comma (, : and) represents the concatenation symbol
- a period (.) is required at the end of every DCG rule
Definite Clause Grammars (DCG)

• What language does our grammar generate?

  **Answer:** the set of strings containing one or more a’s followed by one or more b’s

• by writing the grammar in Prolog,
• we have a ready-made recognizer program
  – *no need to write a separate program* (in this case)

• Example queries
  – `?- s([a,a,b,b,b],[]).`  Yes
  – `?- s([a,b,a],[]).`  No

• **Note:**
  – Query uses the start symbol `s` with two arguments:
  – (1) sequence (as a list) to be recognized and
  – (2) the empty list `[]`

[a,b,a] is an example of Prolog’s list notation
Definite Clause Grammars (DCG)

• **Top-down derivation**
  – begin at the designated start symbol
  – expand rules (LHS -> RHS)
  – search space of possibilities
  – until input string is matched

  – *Prolog implements a top-down left-to-right depth-first search strategy*
SWI-Prolog

• how to start it?
  – from the Windows Program menu
  – interpreter window pops up and
  – is ready to accept database queries (?-)

• how to see what’s loaded into Prolog?
  – ?- listing.

• Load program
  – use the menu in SWI-Prolog
  – or ?- [filename].

• how to see what the current working directory is?
  – (the working directory is where your files are stored)
  – important: every machine in the lab is different
  – ?- working_directory(X,Y).
    – X: current working directory, Y: new working directory

• how to change to a new working directory?
  – ?- working_directory(X,NEW).
Definite Clause Grammars (DCG)

- Language: *Sheeptalk*
  - baa!
  - baaa!
  - ba...a!

- not in language
  - b!
  - ba!
  - baba!
  - !

Perl regular expression:
```
/baa+!/
```
DefiniteClause Grammars (DCG)

- **language**: *Sheeptalk*
  - baa!
  - baaa!
  - ba...a!

- **grammar**
  - $s \rightarrow [b], [a], a, [\text{`!'}]$.  
    (base case)
  - $a \rightarrow [a]$.  
    (recursive case)
  - $a \rightarrow a, [a]$.

*Encode Sheeptalk using a regular grammar?*

*This grammar is not a regular grammar*