LING/C SC/PSYC 438/538

Lecture 25

Prof. Sandiway Fong

538 Presentations! (Starts next week)

Presenter	Date	Topic	Pages
Farinaz Nasiri Ziba	Nov 24th	10.1 Language Divergences and Typology	
Han Han	Dec 1st	23.1-2 Information Retrieval and IR-based Factoid Question Answering	495–503
Hannia Rojas Bareda	Nov 24th	12.3 Some Grammar Rules for English	266-270
Arun Sanyal	Nov 24th	24.1 Properties of Human Conversation (Chatbots & Dialogue Systems)	425-431
Hyeri Lee	Nov 24th	20.1-20.3 Defining Emotion / Creating Affect Lexicons by Human Labeling	166-173

Homework 14

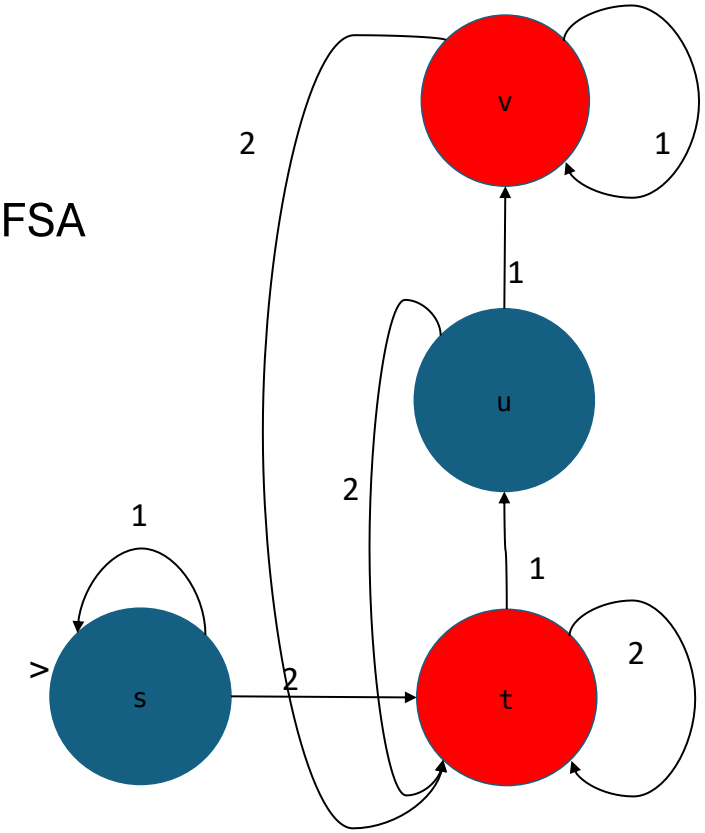
- Not yet graded … hopefully tonight or tomorrow

Homework 14 Review

Question 1:

- Give a Prolog **regular grammar** for this FSA
- How many rules do you have?
- **Hint:** `t --> [].`
  - encodes final state (*we permit this*!)
- Test it on:

```
1. s([1,1,2],[]).
2. s([1,1,2,2],[]).
3. s([2,1,2,1,1,1],[]).
4. s([1,1],[]).
5. s([2,2,1],[]).
6. s([2,1,2,1],[]).
```



Homework 14 Review

```
 1s --> [2], t.¶
 2s --> [1], s.¶
 3t --> [].¶
 4t --> [1], u.¶
 5t --> [2], t.¶
 6u --> [1], v.¶
 7u --> [2], t.¶
 8v --> [].¶
 9v --> [2], t.¶
10v --> [1], v.¶
```

```
[82 ?- [hw14].
true.

83 ?- s([1,1,2],[]).
true.

84 ?- s([1,1,2,2],[]).
true.

85 ?- s([2,1,2,1,1,1],[]).
true ;
false.

86 ?- s([1,1],[]).
false.

87 ?- s([2,2,1],[]).
false.

88 ?- s([2,1,2,1],[]).
false.
```

Homework 14 Review

- **Enumeration**: means to guarantee we print out every string in the language (*given enough time*)

```
89 ?- s(String,[]).
String = [2] ;
String = [2, 1, 1] ;
String = [2, 1, 1, 2] ;
String = [2, 1, 1, 2, 1, 1] ;
String = [2, 1, 1, 2, 1, 1, 2] ;
String = [2, 1, 1, 2, 1, 1, 2, 1, 1] ;
String = [2, 1, 1, 2, 1, 1, 2, 1, 1|…] ;
String = [2, 1, 1, 2, 1, 1, 2, 1, 1|…] [write]
String = [2, 1, 1, 2, 1, 1, 2, 1, 1|…] ;
String = [2, 1, 1, 2, 1, 1, 2, 1, 1|…] .
```

Homework 14 Review

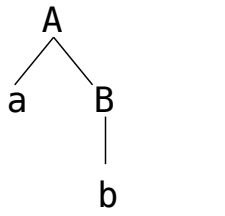
```
90 ?- set_prolog_flag(answer_write_options,[max_depth(0)]).
true.

91 ?- s(String,[]).
String = [2] ;
String = [2,1,1] ;
String = [2,1,1,2] ;
String = [2,1,1,2,1,1] ;
String = [2,1,1,2,1,1,2] ;
String = [2,1,1,2,1,1,2,1,1] ;
String = [2,1,1,2,1,1,2,1,1,2] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2] ;
String = [2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1,2,1,1
```

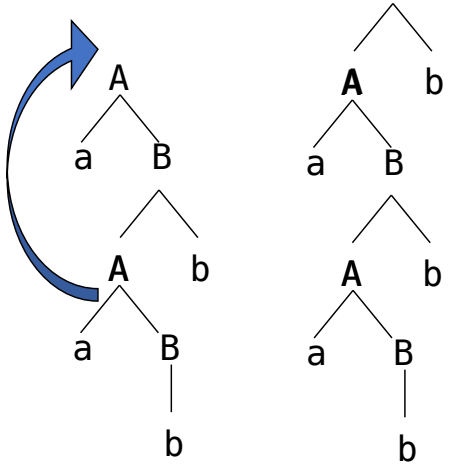
Last Time: beyond Regular Languages

- Intuition:
  - grammar implements the stacking of partial trees balanced for a's and b's:

```
1. a --> [a], b.
2. b --> [b].
3. b --> a, [b].
```



A, B: nonterminals
a, b: terminals

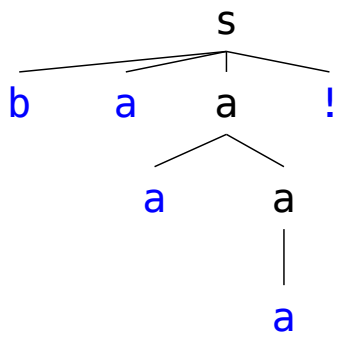


- A type-2 or context-free grammar (CFG) has no restrictions on what can go on the RHS of a grammar rule
- **Note**: *CFGs still have a single nonterminal limit for the LHS of a rule.*
- Example:

```
1. s --> [a], [b].
2. s --> [a], s, [b].
```

Parse Trees using Terms

- **Tree:**

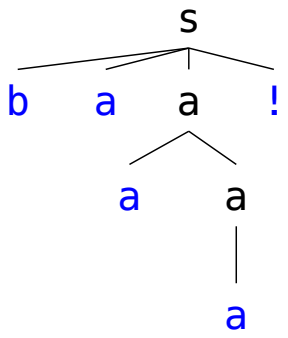


`s(b,a,a(a,a(a)),!)`

- Prolog term data structure:
  - `functor(arg`$_1$`,..,arg`$_n$`)`
  - each `arg`$_i$ could (recursively) be another term or simple atom (symbol).
  - encodes hierarchy
  - permits a sequence of arguments

Extra Argument: Parse Tree

- **Recovering a parse tree**
  - when want Prolog to return more than just **true/false** answers
  - in case of **true**, we can *incrementally* compute a Prolog term representation of the parse
  - by adding an extra argument to each nonterminal
  - applies to all rules (*not just regular grammars*)



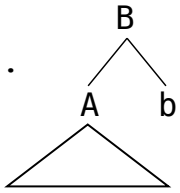
Extra Argument: Parse Tree

- **DCG**
  - `a --> [a], b.` (start symbol a)
  - `b --> [b].` (base case)
  - `b --> a, [b].` (left recursive case)

- **base case**
  - `b --> [b].`
  - `b(`*`subtree`*`) --> [b].`
  - `b(b(b)) --> [b].`

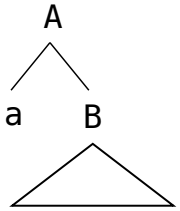


- **recursive case**
  - `b --> a, [b].`
  - `b(`*`subtree`*`) --> a(`*`subtree'`*`), [b].`
  - `b(b(A,b)) --> a(A), [b].`



- **Idea**:
  - for each nonterminal, add an argument to store its subtree

- **start symbol case**
  - `a --> [a], b.`
  - `a(`*`tree`*`) --> [a], b(`*`subtree'`*`).`
  - `a(a(a,B)) --> [a], b(B).`



Extra Argument: Parse Tree

- **Prolog grammar:**
  - anbn.prolog
  - `a --> [a], b.`
  - `b --> [b].`
  - `b --> a, [b].`
- **Equivalent Prolog grammar computing a parse**
  - anbn2.prolog
  - `a(a(a,B)) --> [a], b(B).`
  - `b(b(b)) --> [b].`
  - `b(b(A,b)) --> a(A), [b].`

Every nonterminal *x* has an extra argument *Term*: *x*(*Term*)

Extra Argument: Parse Tree

- anbn2.prolog
  1. `a(a(a,B)) --> [a], b(B).`
  2. `b(b(b)) --> [b].`
  3. `b(b(A,b)) --> a(A), [b].`
- To run the code, we also have to add the extra argument (for our parse) to the query.

```
?- [anbn2].
true.

?- a(Tree, [a,b], []).
Tree = a(a, b(b)) ;
false.

?- a(Tree, [a,a,b], []).
false.

?- a(Tree, [a,a,b,b], []).
Tree = a(a, b(a(a, b(b)), b)) ;
false.
```

Use of Extra Arguments

- **Store a Parse tree**: no change in expressive power
- But, extra arguments are powerful
  - they allow us to impose (grammatical) constraints and change the expressive power of the system
    - if used as read-able memory (cf. *Turing Machine discussion*)
- **Example**:
  - $a^n b^n c^n$ n>0 is not a **context-free language** (Chomsky Hierarchy type-2)
  - *i.e. you cannot write rules of the form* n –––> *RHS for this language*
  - in fact, it's **context-sensitive** (Chomsky Hierarchy type-1)

Extra arguments

- Let's start by writing a context-free grammar (**CFG**) + extra argument for the parse for the language { $a^p b^q c^r$ | p, q, r > 0}
- `File: abc.prolog`

```
1. s(s(A,B,C)) --> a(A), b(B), c(C).
2. a(a(a)) --> [a].
3. a(a(a,A)) --> [a], a(A).
4. b(b(b)) --> [b].
5. b(b(b,B)) --> [b], b(B).
6. c(c(c)) --> [c].
7. c(c(c,C)) --> [c], c(C).
```

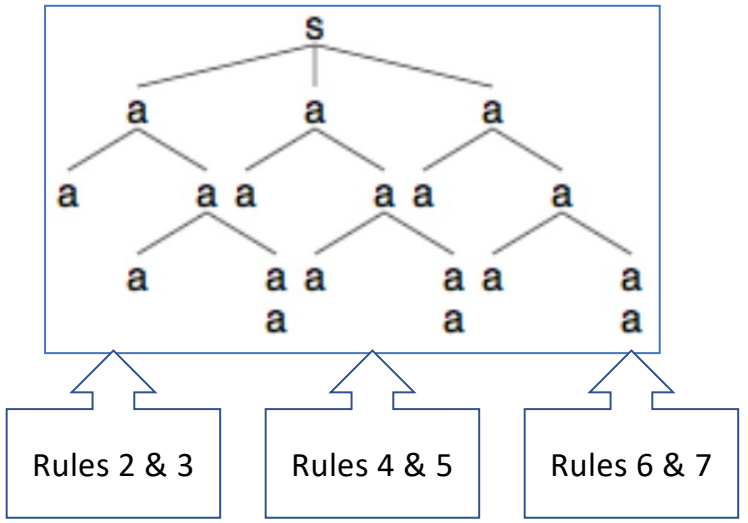
Extra arguments

- { $a^p b^q c^r$ | p, q, r > 0}: `abc.prolog`
- `?- [abc].`
- `?- s(Parse,[],[]).`
- **`false.`**
- `?- s(Parse,[a,b,c],[]).`
- `Parse = s(a(a), b(b), c(c)) ;`
- **`false.`**
- `?- s(Parse,[a,b,b,c,c,c],[]).`
- `Parse = s(a(a), b(b, b(b)), c(c, c(c, c(c)))) ;`
- **`false.`**
- `?- s(Parse,[a,b,b,c,a],[]).`
- **`false.`**

Extra arguments

- A context-free grammar (CFG) + extra argument (EA) for the context-sensitive language { $a^n b^n c^n$ | n>0}, `abc2.prolog`

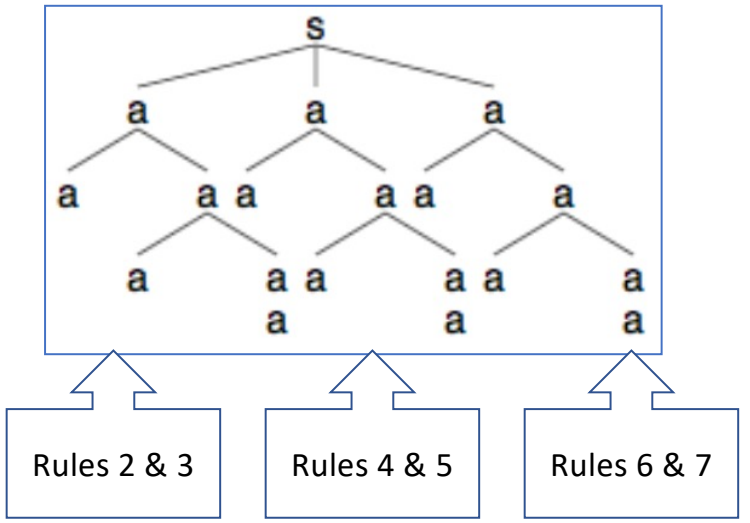
```
1. s(s(A,B,C)) --> a(A), b(B), c(C).
2. a(a(a)) --> [a].
3. a(a(a,X)) --> [a], a(X).
4. b(a(a)) --> [b]. % cf. b(b)
5. b(a(a,X)) --> [b], b(X).
6. c(a(a)) --> [c]. % cf. c(c)
7. c(a(a,X)) --> [c], c(X).
```



Extra arguments

- A context-free grammar (CFG) + extra argument (EA) for the context-sensitive language { $a^n b^n c^n$ | n>0}, `abc2.prolog`

```
1. s(s(A,A,A)) --> a(A), b(A), c(A).
2. a(a(a)) --> [a].
3. a(a(a,X)) --> [a], a(X).
4. b(a(a)) --> [b]. % cf. b(b)
5. b(a(a,X)) --> [b], b(X).
6. c(a(a)) --> [c]. % cf. c(c)
7. c(a(a,X)) --> [c], c(X).
```

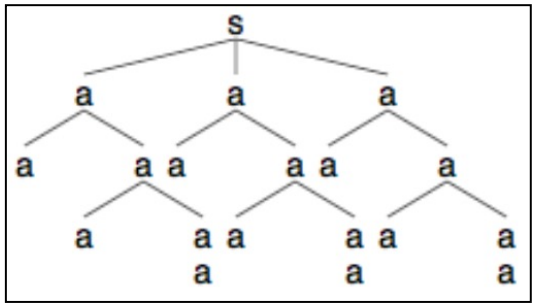


Extra arguments

- A CFG+EA for $a^n b^n c^n$ n>0:

Set membership question

```
?- [abc2].
?- s(Parse,[a,a,a,b,b,b,c,c,c],[]).
Parse = s(a(a, a(a, a(a))), a(a, a(a, a(a))), a(a, a(a, a(a)))) ;
false.
?- s(Parse,[a,a,a,b,b,b,c,c],[]).
false.
?- s(Parse,[a,a,a,b,b,c,c],[]).
false.
?- s(Parse,[a,a,b,b,c,c],[]).
Parse = s(a(a, a(a)), a(a, a(a)), a(a, a(a))) ;
false.
?- s(Parse,[],[]).
false.
```



Extra arguments

- A CFG+EA for $a^n b^n c^n$ n>0:

```
?- s(_,[a,a,b,b,c,c,c],[]).
false.

?- s(_,[a,a,b,b,c,c],[]).
true .

?- s(_,[a,a,b,b,c],[]).
false.

?- s(_,[a,a,b,b,c,c,c],[]).
false.

?- s(_,[a,a,a,b,b,b,c,c,c],[]).
true .
```

Set membership question

- underscore (_): a variable
  - *don't care what its value is,*
  - *don't tell me.*

Left Recursion and Set Enumeration

- Example (`lrrg.prolog`):

```
1. s --> a, [!].
2. a --> ba, [a].
3. a --> a, [a].
4. ba --> b, [a].
5. b --> [b].
```

- Grammar is:
  - a regular grammar
  - left recursive (*nonterminal on left*)

- **Question**
  - What is the language of this grammar?
- Answer: *Sheeptalk*!
  - $ba^n!$ (# a's = n > 1)
- Sentential forms:
  - s
  - a!
  - baa!
  - baa!
  - baa!

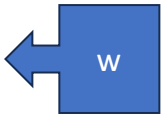
Underscores used here to indicate a nonterminal

Left Recursion and Set Enumeration

```
?- [lrrg].
true.

?- s(String,[]).
String = [b, a, a, !] ;
String = [b, a, a, a, !] ;
String = [b, a, a, a, a, !] ;
String = [b, a, a, a, a, a, !] ;
String = [b, a, a, a, a, a, a, !] ;
String = [b, a, a, a, a, a, a, a, !] ;
String = [b, a, a, a, a, a, a, a, a|...] ;
String = [b, a, a, a, a, a, a, a, a|...] [write]
String = [b, a, a, a, a, a, a, a, a, a, !] .
```

lrrg.prolog works for enumeration!



Left Recursion and Set Enumeration

- It also partially works for set membership:

```
?- s([b,a,a,!],[]).
true ;
ERROR: Stack limit (1.0Gb) exceeded
ERROR:   Stack sizes: local: 0.9Gb, global: 84.2Mb, trail: 0Kb
ERROR:   Stack depth: 11,029,028, last-call: 0%, Choice points: 4
ERROR:   Probable infinite recursion (cycle):
ERROR:     [11,029,026] user:a([length:4], _22083396)
ERROR:     [11,029,025] user:a([length:4], _22083422)
   Exception: (11,029,026) a([b, a, a, !], _22083324) ? abort
% Execution Aborted
?-
```

Left Recursion and Set Enumeration

- What happens if we present a string not in Sheeptalk!?

lrrg.prolog doesn't work as a decider: response should be yes/no.

```
?- s([b,a,a,b,a,a,!],[]).
ERROR: Stack limit (1.0Gb) exceeded
ERROR:   Stack sizes: local: 0.9Gb, global: 84.1Mb, trail: 0Kb
ERROR:   Stack depth: 11,028,563, last-call: 0%, Choice points: 4
ERROR:   Probable infinite recursion (cycle):
ERROR:     [11,028,561] user:a([length:7], _22059520)
ERROR:     [11,028,560] user:a([length:7], _22059546)
   Exception: (11,028,561) a([b, a, a, b, a, a, !], _22059448) ? abort
% Execution Aborted
```

Left Recursion and Set Enumeration

- left recursive regular grammar `lrrg.prolog`:

```
1. s --> a, [!].
2. a --> ba, [a].
3. a --> a, [a].
4. ba --> b, [a].
5. b --> [b].
```

- Summary of Behavior:
  1. enumerates the language
  2. halts when presented with a string that is in the language
  3. doesn't halt when faced with a string not in the language
  - *unable to decide the language membership question*

Left Recursion and Set Enumeration

- left recursive regular grammar:

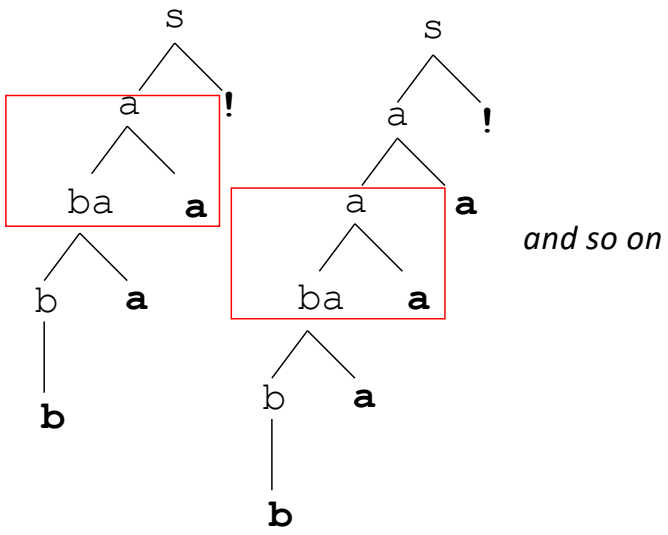
```
1. s --> a, [!].
2. a --> ba, [a].
3. a --> a, [a].
4. ba --> b, [a].
5. b --> [b].
```

Choice point

- Behavior
  - halts when presented with a string that is in the language
  - doesn't halt when faced with a string not in the language

- derivation tree for `?- s(L,[]).` [Powerpoint animation]

L = [b,a,a,!]



Left Recursion and Set Enumeration

- However, this **slightly re-ordered** left recursive regular grammar:

```
1. s --> a, [!].
2. a --> a, [a].
3. a --> ba, [a].
4. ba --> b, [a].
5. b --> [b].
```

- (*rules 2 and 3 swapped*) won't halt when enumerating

- **Why**?

[Powerpoint animation]

