LING/C SC/PSYC 438/538

Lecture 22

Sandiway Fong

Today's Topics

- 538 Presentations
- Homework 11 Review
- So far for regular languages:
  - FSA: yes; regex: yes ; regular grammars: no
- Today:
  - a quick introduction to our programming language: Prolog
  - we'll be using this to explore regular grammars (*and beyond*)
- Homework 12:
  - install SWI-Prolog for next time

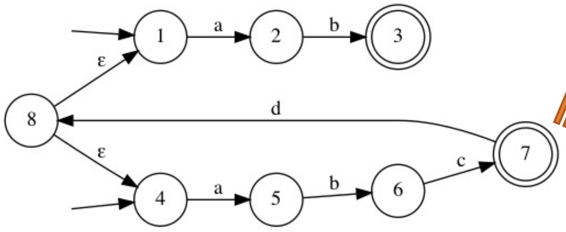
538 Presentations

- Do NOT pick an entire chapter!
  - Maybe pick one or two sub-sections (*depending on size of topic*).
- Remember you have 7 mins!
- Chapters available:
- 3 N-gram Language Models
- 6 Vector Semantics and Embeddings
- 8 Sequence Labeling for Parts of Speech and Named Entities
- 12 Constituency Grammars
- 13 Constituency Parsing
- 14 Dependency Parsing (*do not choose material we have covered in class*)
- 15 Logical Representations of Sentence Meaning
- 17 Information Extraction
- 18 Word Senses and WordNet
- 19 Semantic Role Labeling
- 20 Lexicons for Sentiment, Affect, and Connotation
- 21 Coreference Resolution
- 22 Discourse Coherence
- 23 Question Answering
- 24 Chatbots & Dialogue Systems

538 Presentations

- Instructions:
  - Choose your top 3 topics from draft `jm3.pdf`.
  - Supply section title and numbers.
  - First come, first served
  - Choose Dec 5th or Dec 7th presentation date
  - Send to me
  - Subject: 538 Presentation *YOUR NAME*
  - When approved, make slides PPTX/PDF suitable for 7 mins
  - Then send me your slides

Homework 11 Review

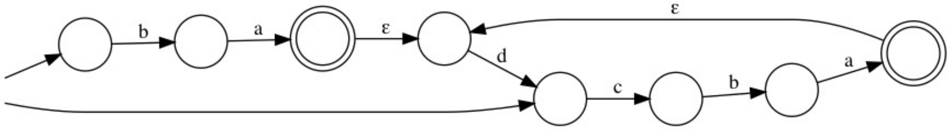


8 states; 8 edges; 2 start; 2 final

Q1-Q5

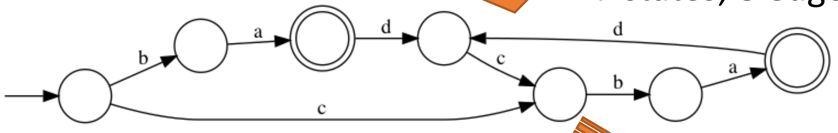
Q2

8 states; 8 edges; 2 start; 2 final

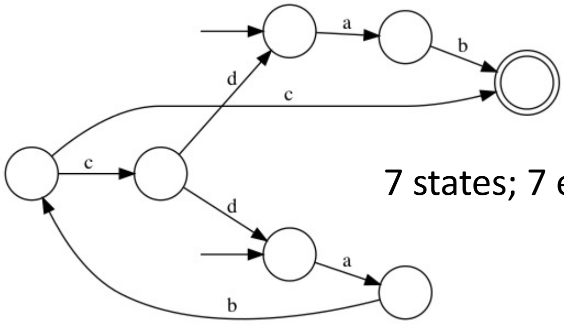


Q3

7 states; 8 edges; 1 start; 2 final

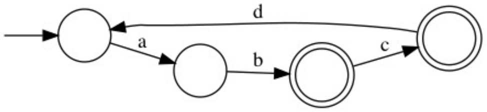


Q4



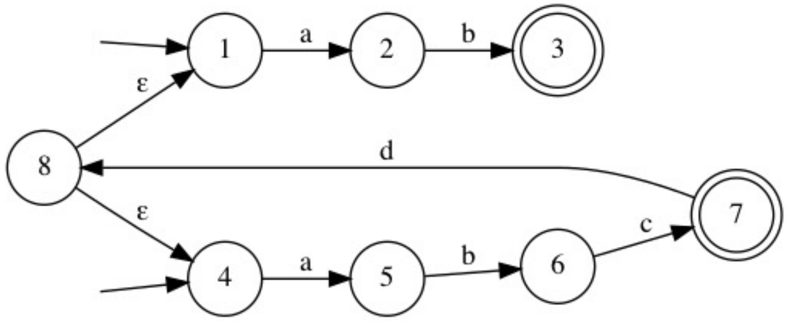
7 states; 7 edges; 2 start; 1 final

Q5



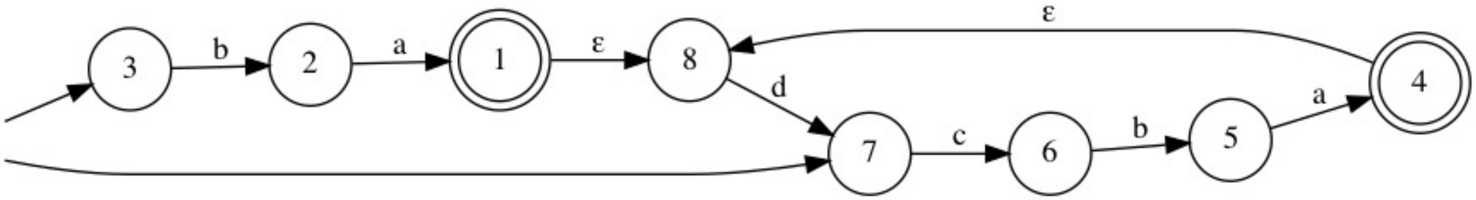
4 states; 4 edges; 1 start; 2 final

Homework 11 Review



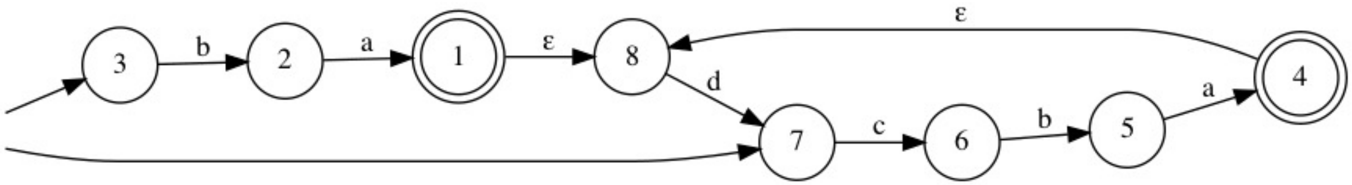
Q2: Reverse the machine:

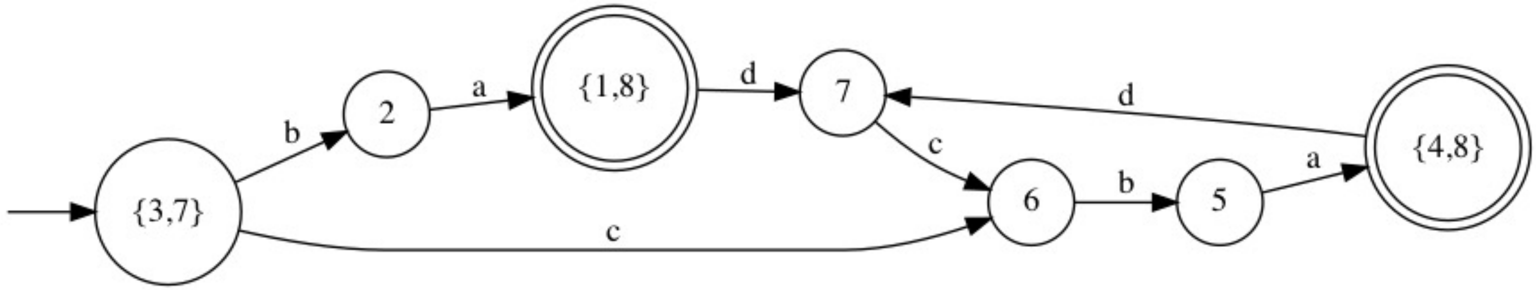
- End states: 3, 7 => Start states: 3, 7
- Start states: 1, 4 => End states: 1, 4



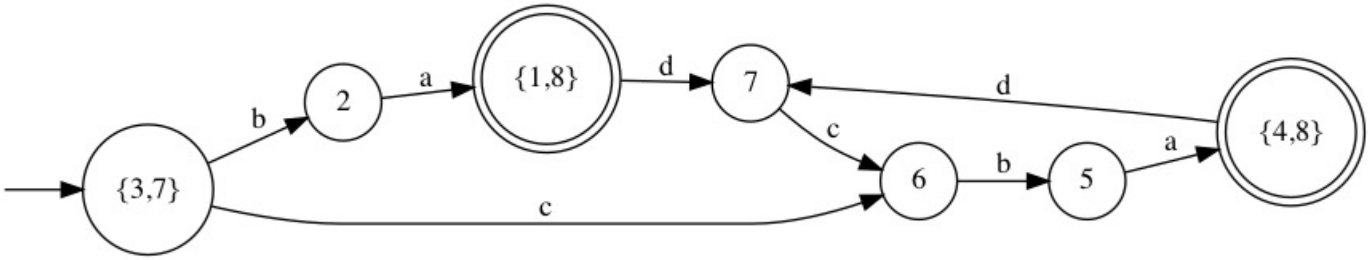
Homework 11 Review

- Q3: Determinize the machine



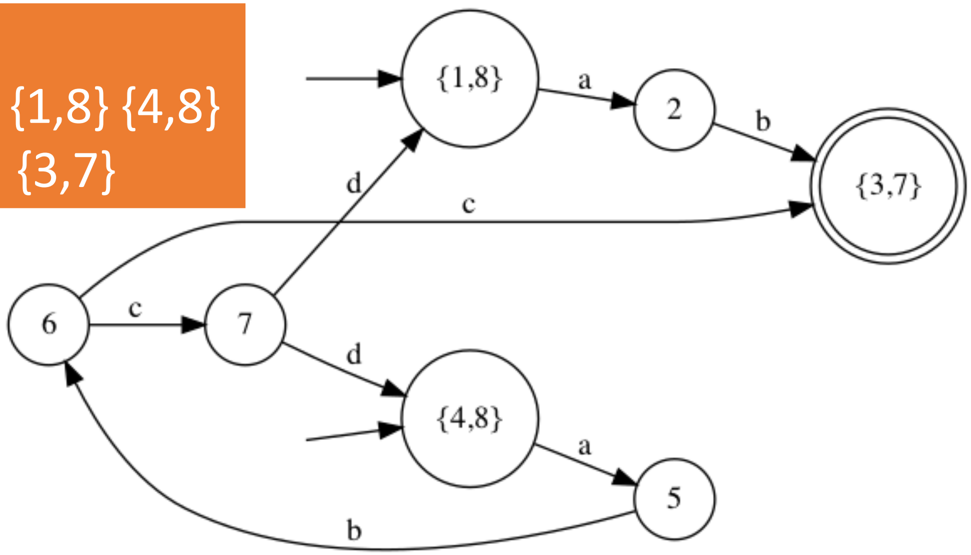


Homework 11 Review

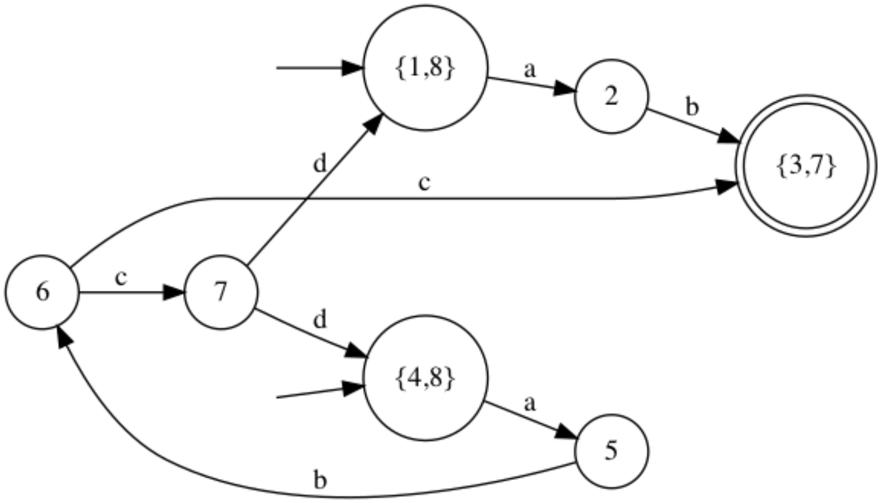


Q4: Reverse the machine:

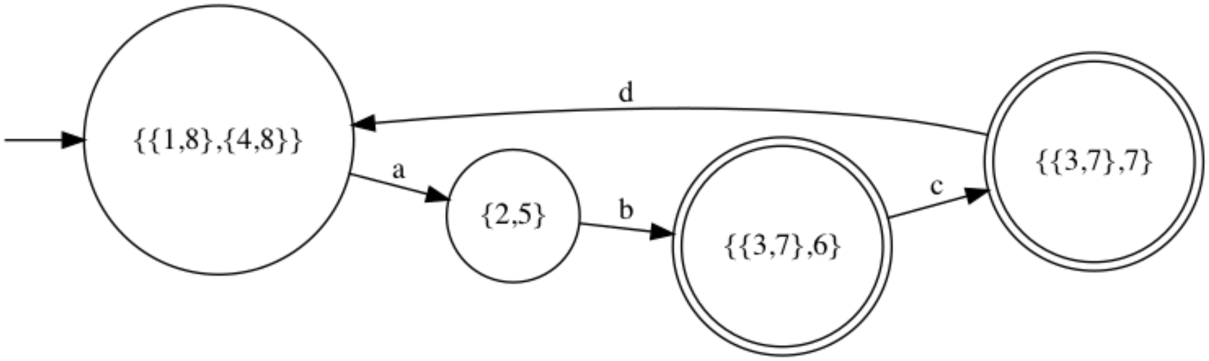
- End states: {1,8} {4,8} => Start states: {1,8} {4,8}
- Start states: {3,7} => End states: {3,7}



Homework 11 Review

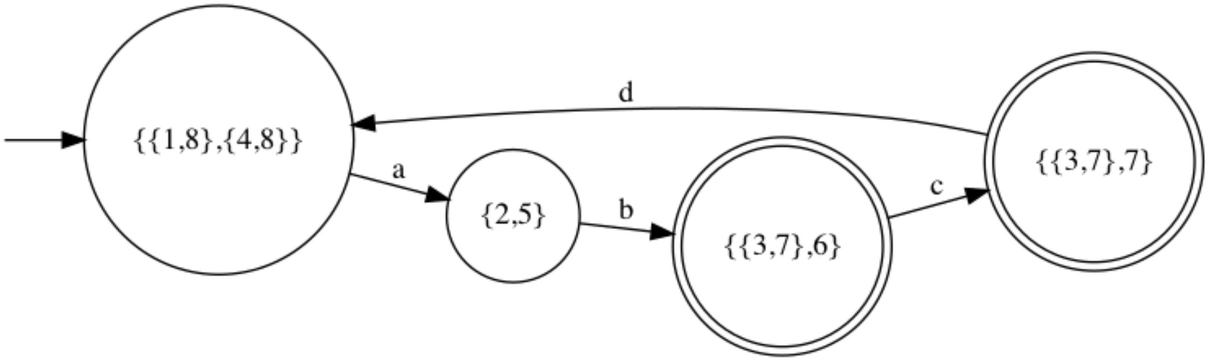


- Q5: Determinize the machine



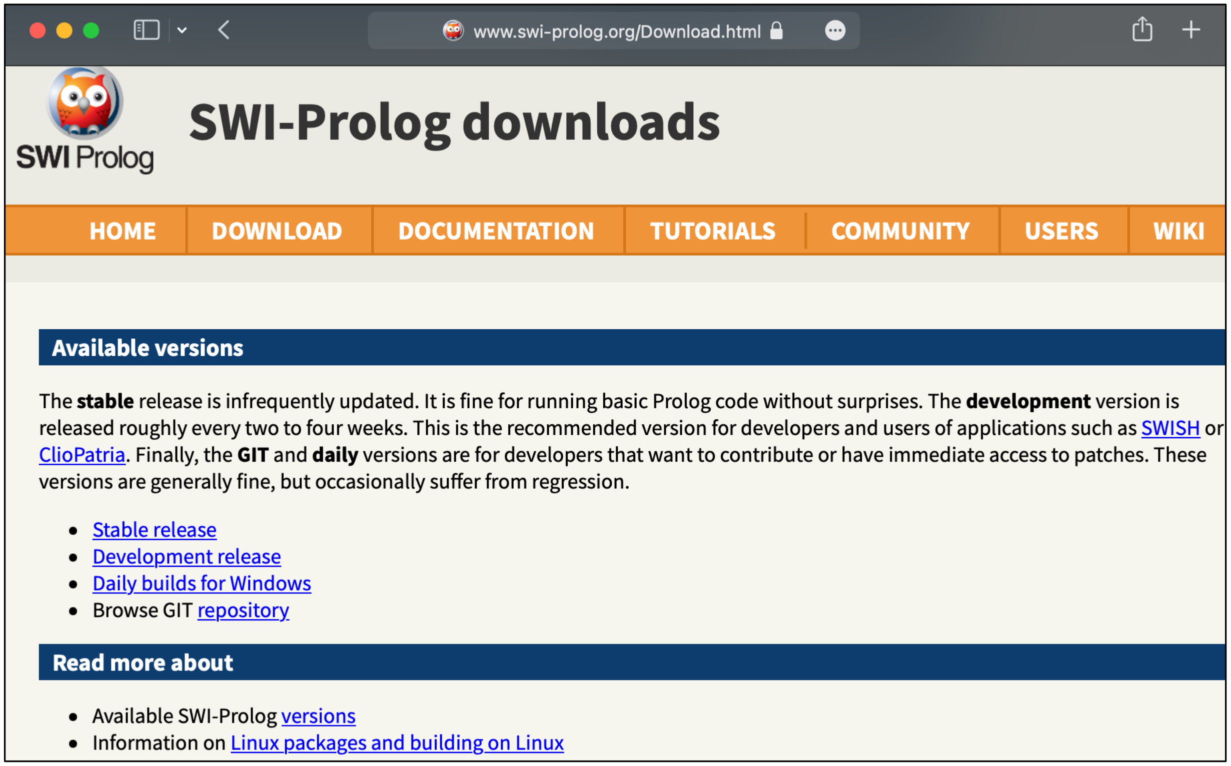
Homework 11 Review

- Do you think you could build a machine for L (= L^{RR}) with fewer states? NOPE
  - Brzozowski, J.A. Canonical regular expressions and minimal state graphs ford efinite events. In *Proc. Sympos. Math. Theory of Automata (New York, 1962)*, pages 529–561. Polytechnic Press of Polytechnic Inst. of Brooklyn, Brooklyn, N.Y., 1963.
  - Basically, the set of states construction can be used to optimize the number of stat



Homework 12

Install SWI-Prolog
www.swi-prolog.org
e.g.
Windows 10
installer



SWI-Prolog downloads

Available versions

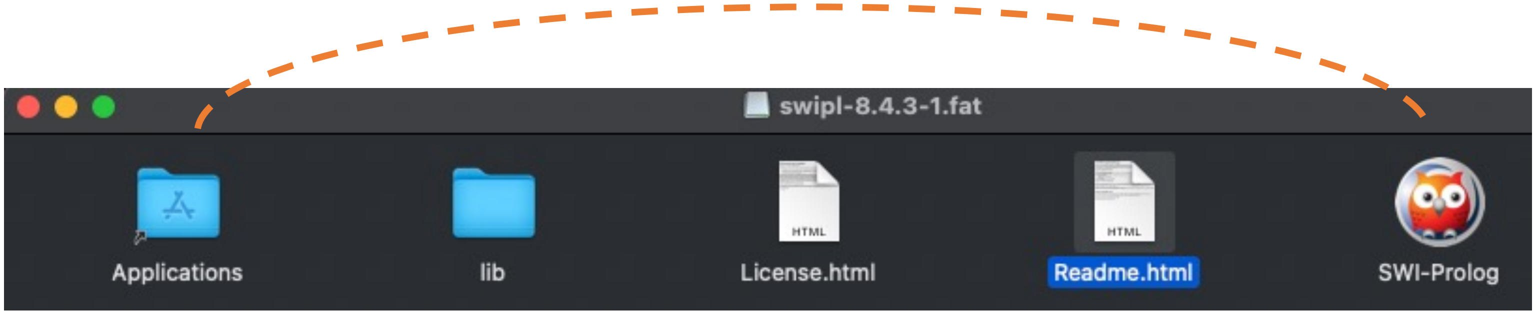
The **stable** release is infrequently updated. It is fine for running basic Prolog code without surprises. The **development** version is released roughly every two to four weeks. This is the recommended version for developers and users of applications such as SWISH or ClioPatria. Finally, the **GIT** and **daily** versions are for developers that want to contribute or have immediate access to patches. These versions are generally fine, but occasionally suffer from regression.

- Stable release
- Development release
- Daily builds for Windows
- Browse GIT repository

Read more about

- Available SWI-Prolog versions
- Information on Linux packages and building on Linux

Homework 12



Using the commandline tools

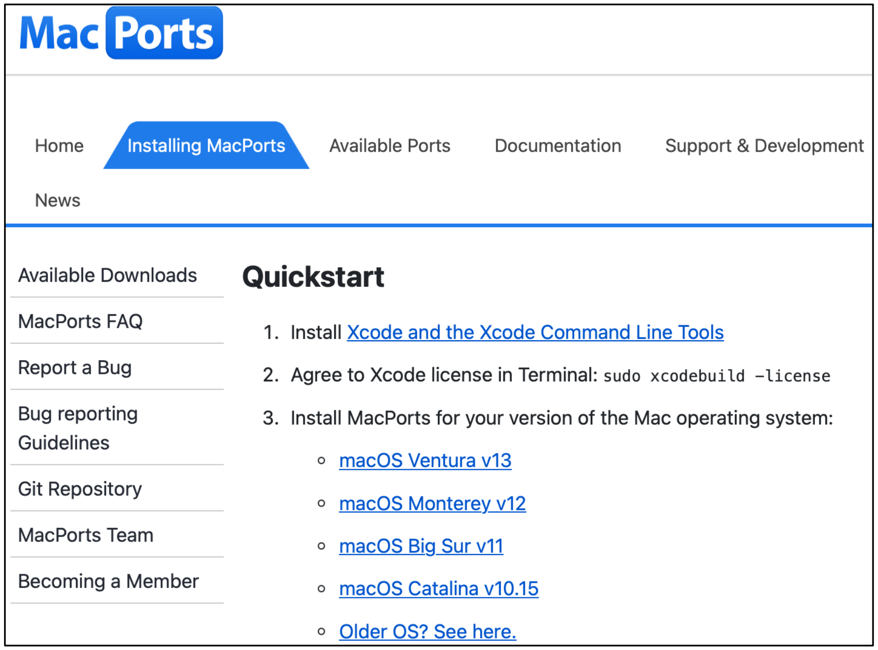
The traditional command line tools are included in the app. To access them from the Terminal application, add the directory

- `/Applications/SWI-Prolog.app/Contents/MacOS`

to $PATH

Homework 12

An alternative to the OSX application:



Quickstart

1. Install Xcode and the Xcode Command Line Tools
2. Agree to Xcode license in Terminal: `sudo xcodebuild -license`
3. Install MacPorts for your version of the Mac operating system:
   - macOS Ventura v13
   - macOS Monterey v12
   - macOS Big Sur v11
   - macOS Catalina v10.15
   - Older OS? See here.

https://www.macports.org/install.php

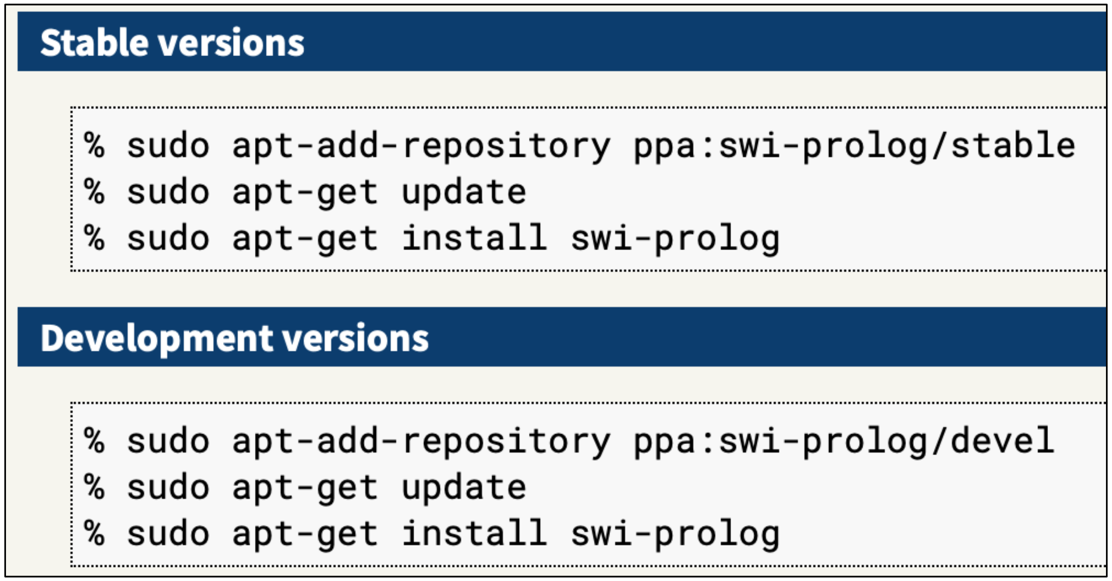
```
port install swi-prolog
```

Yet another one is Homebrew

https://brew.sh

```
brew install swi-prolog
```

Homework 12



Stable versions

```
% sudo apt-add-repository ppa:swi-prolog/stable
% sudo apt-get update
% sudo apt-get install swi-prolog
```

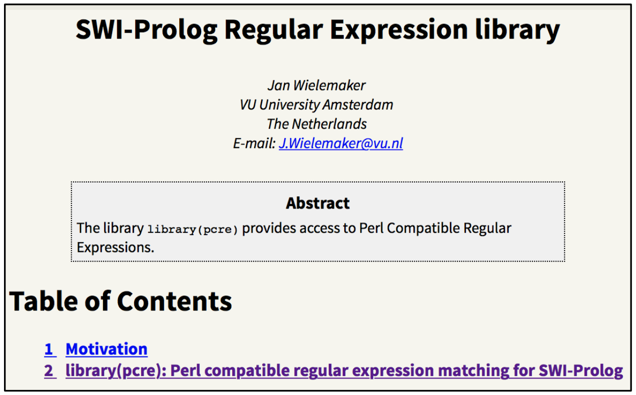
Development versions

```
% sudo apt-add-repository ppa:swi-prolog/devel
% sudo apt-get update
% sudo apt-get install swi-prolog
```

Linux: Debian-based
e.g. Ubuntu

SWI-Prolog

- **Good for**:
  1. formal logic
  2. directly handling **non-determinism** (through backtracking)
  3. phrase structure grammars (PSG)
  4. partially instantiated data structures (lists, terms)
- **Not good for**:
  - regex (*there is a library though*)
  - math (linear algebra: arrays etc.)
  - looping



SWI-Prolog Regular Expression library

Jan Wielemaker
VU University Amsterdam
The Netherlands
E-mail: J.Wielemaker@vu.nl

Abstract

The library `library(pcre)` provides access to Perl Compatible Regular Expressions.

Table of Contents

SWI Prolog Cheatsheet

- **At the prompt** ?-
  1. `halt.` `^D`
  2. `listing.` `listing(`*name*`).`
  3. `[`*filename*`].` loads *filename.pl*
  4. `trace.`
  5. `notrace.`
  6. `debug.`
  7. `nodebug.`
  8. `spy(`*name*`).`
  9. `pwd.`
  10. `working_directory(_,Y).`
      switch directories to Y
- Anytime
  - ^C (then **a**(bort) or **h**(elp) for other options)

Notation:

`\+`	negation
`,`	conjunction
`;`	disjunction
`:-`	if

Facts:

predicate(*Args*).

```
sing(man).
```

Rules:

p(*Args*) :- q(*Args*)

```
sing(X) :- human(X).
sing(X) :- bird(X).
```

Data structures:

list: `[a,..b]`

empty list: `[]`

head/tail: `[`*head*`|`*List*`]`

```
[the, man, sings]
```

Atom:

name, number

```
man, 12
```

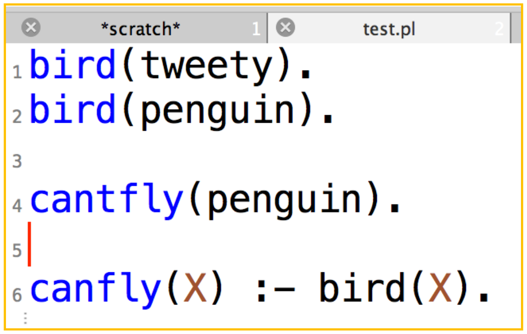
Term:

```
s(np(dt(the),nn(man)),vp(vbz(sings)))
```

functor(arguments)

arguments: comma-separated terms/atoms

Example: as a logic programming language



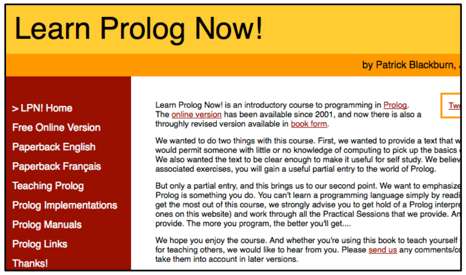
```
bird(tweety).
bird(penguin).

cantfly(penguin).

canfly(X) :- bird(X).
```

Learn Prolog Now!

P. Blackburn, J. Bos & K. Striegnitz
free online version
http://www.learnprolognow.org



Learn Prolog Now!

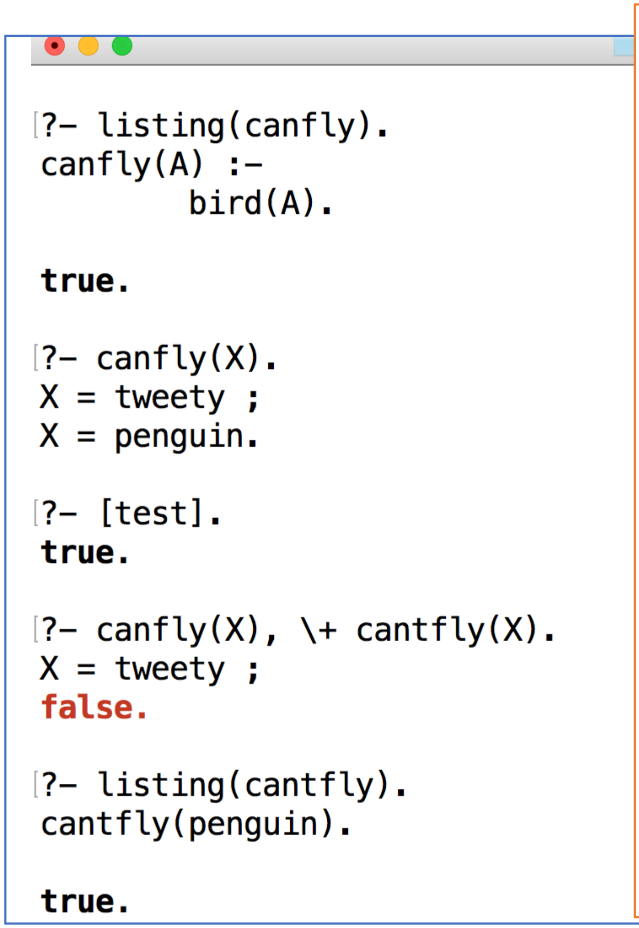
by Patrick Blackburn, J

Learn Prolog Now! is an introductory course to programming in Prolog.
The online version has been available since 2001, and now there is also a
throughly revised version available in book form.

We wanted to do two things with this course. First, we wanted to provide a text that w
would permit someone with little or no knowledge of computing to pick up the basics o
We also wanted the text to be clear enough to make it useful for self study. We believ
associated exercises, you will gain a useful partial entry to the world of Prolog.

But only a partial entry, and this brings us to our second point. We want to emphasize
Prolog is something you do. You can't learn a programming language simply by readi
get the most out of this course, we strongly advise you to get hold of a Prolog interpre
ones on this website) and work through all the Practical Sessions that we provide. An
provide. The more you program, the better you'll get....

We hope you enjoy the course. And whether you're using this book to teach yourself
for teaching others, we would like to hear from you. Please send us any comments/c
take them into account in later versions.



```
?- listing(canfly).
canfly(A) :-
        bird(A).

true.

?- canfly(X).
X = tweety ;
X = penguin.

?- [test].
true.

?- canfly(X), \+ cantfly(X).
X = tweety ;
false.

?- listing(cantfly).
cantfly(penguin).

true.
```

Notation:

`\+`	negation
`,`	conjunction
`;`	disjunction
`:-`	if

Facts:
`predicate(`*`Args`*`).`

Rules:
`p(`*`Args`*`) :- q(`*`Args`*`) ,.., r(`*`Args`*`).`

Data structures:

list: [a,..b]

empty list: []

head/tail: [h|List]

Atom:

name, number

Term:

`functor(arguments)`
`arguments:`
comma-separated terms/atoms

Prolog Recursion

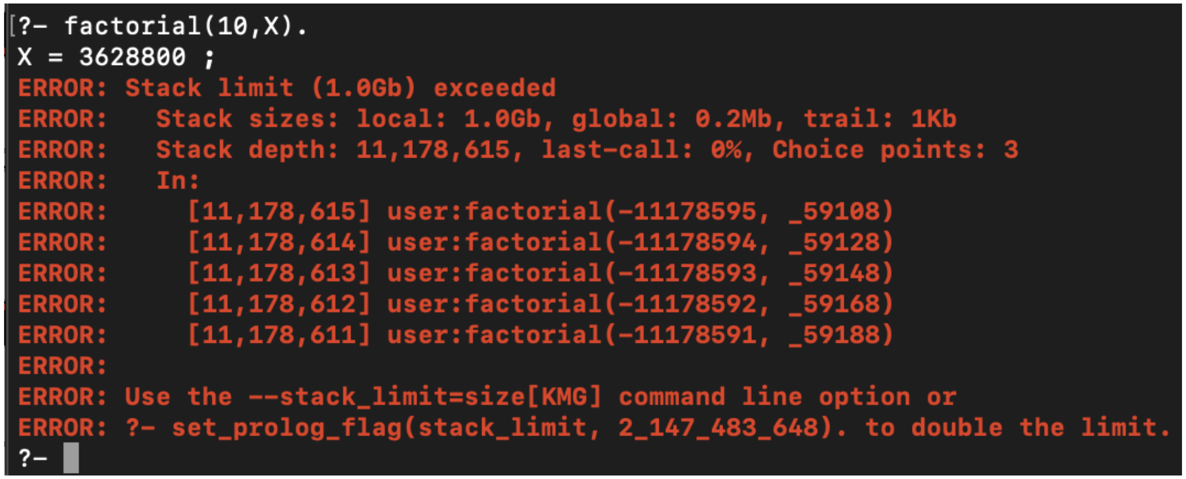
- **Example** (factorial):
  - 0! = 1
  - n! = n * (n-1)! for n>0
- In Prolog:
  - `factorial(0,1).`
  - `factorial(N,NF) :- M is N-1, factorial(M,MF), NF is N * MF.`
- Prolog arithmetic built-in `is/2`:
  - `X is` *`<math expr>`*
  - compute *expr* and assign value to variable `X`
- Run

```
?- factorial(5,X).
```

(hit **;** for more answers)

Prolog Recursion

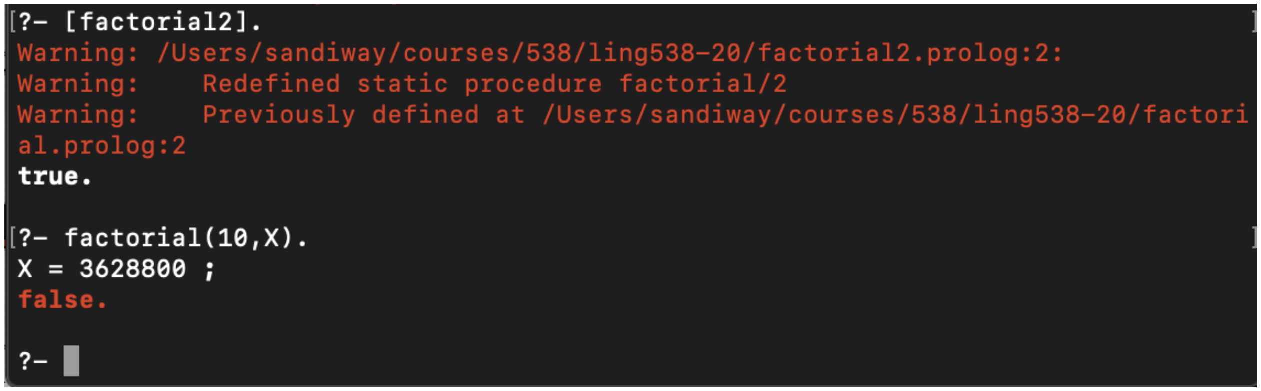
- In Prolog:
  - `factorial(0,1).`
  - `factorial(N,NF) :- M is N-1, factorial(M,MF), NF is N * MF.`
- Problem: *infinite loop when* you press **;** for more answers



```
[?- factorial(10,X).
X = 3628800 ;
ERROR: Stack limit (1.0Gb) exceeded
ERROR:   Stack sizes: local: 1.0Gb, global: 0.2Mb, trail: 1Kb
ERROR:   Stack depth: 11,178,615, last-call: 0%, Choice points: 3
ERROR:   In:
ERROR:     [11,178,615] user:factorial(-11178595, _59108)
ERROR:     [11,178,614] user:factorial(-11178594, _59128)
ERROR:     [11,178,613] user:factorial(-11178593, _59148)
ERROR:     [11,178,612] user:factorial(-11178592, _59168)
ERROR:     [11,178,611] user:factorial(-11178591, _59188)
ERROR:
ERROR: Use the --stack_limit=size[KMG] command line option or
ERROR: ?- set_prolog_flag(stack_limit, 2_147_483_648). to double the limit.
?- 
```

Prolog Recursion

- In Prolog:
  - `factorial(0,1).`
  - `factorial(N,NF) :- M is N-1, factorial(M,MF), NF is N * MF.`
- Fix: 2nd case only applies to numbers > 0
  - `factorial(N,NF) :- N>0, M is N-1, factorial(M,MF), NF is N * MF.`



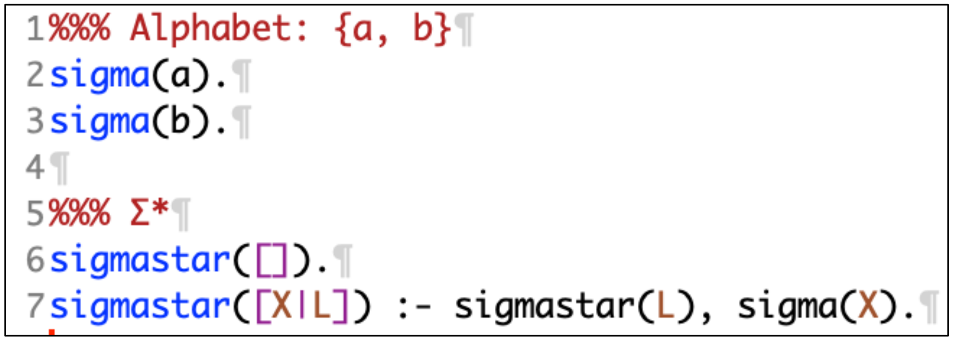
```
[?- [factorial2].
Warning: /Users/sandiway/courses/538/ling538-20/factorial2.prolog:2:
Warning:    Redefined static procedure factorial/2
Warning:    Previously defined at /Users/sandiway/courses/538/ling538-20/factori
al.prolog:2
true.

[?- factorial(10,X).
X = 3628800 ;
false.

?- 
```

Prolog Recursion

- Formal language example:
    - Suppose alphabet Σ={a, b}, enumerate Σ*



```prolog
%%% Alphabet: {a, b}
sigma(a).
sigma(b).

%%% Σ*
sigmastar([]).
sigmastar([X|L]) :- sigmastar(L), sigma(X).
```

Run (hit ; for more answers)

```
?- sigmastar(L).
```

backtracking

```
L = [] ;
L = [a] ;
L = [b] ;
L = [a, a] ;
L = [b, a] ;
L = [a, b] ;
L = [b, b] ;
L = [a, a, a] ;
L = [b, a, a] ;
L = [a, b, a] ;
L = [b, b, a] ;
L = [a, a, b] ;
L = [b, a, b] ;
L = [a, b, b] ;
L = [b, b, b] ;
…
```

Prolog Recursion and Non-determinism

width: 6 x 13 = 78

```
?- sigmastar(X), length(X,N), (N>5 -> ! ; format('~|~t~p~13+',[X]), fail).
           []          [a]          [b]        [a,a]        [b,a]        [a,b]
        [b,b]      [a,a,a]      [b,a,a]      [a,b,a]      [b,b,a]      [a,a,b]
      [b,a,b]      [a,b,b]      [b,b,b]    [a,a,a,a]    [b,a,a,a]    [a,b,a,a]
    [b,b,a,a]    [a,a,b,a]    [b,a,b,a]    [a,b,b,a]    [b,b,b,a]    [a,a,a,b]
    [b,a,a,b]    [a,b,a,b]    [b,b,a,b]    [a,a,b,b]    [b,a,b,b]    [a,b,b,b]
    [b,b,b,b]  [a,a,a,a,a]  [b,a,a,a,a]  [a,b,a,a,a]  [b,b,a,a,a]  [a,a,b,a,a]
  [b,a,b,a,a]  [a,b,b,a,a]  [b,b,b,a,a]  [a,a,a,b,a]  [b,a,a,b,a]  [a,b,a,b,a]
  [b,b,a,b,a]  [a,a,b,b,a]  [b,a,b,b,a]  [a,b,b,b,a]  [b,b,b,b,a]  [a,a,a,a,b]
  [b,a,a,a,b]  [a,b,a,a,b]  [b,b,a,a,b]  [a,a,b,a,b]  [b,a,b,a,b]  [a,b,b,a,b]
  [b,b,b,a,b]  [a,a,a,b,b]  [b,a,a,b,b]  [a,b,a,b,b]  [b,b,a,b,b]  [a,a,b,b,b]
  [b,a,b,b,b]  [a,b,b,b,b]  [b,b,b,b,b]
X = [a, a, a, a, a, a],
N = 6.
```

If-Then-Else: (*`Condition -> Then ; Else`*)

`!` (cut: cut off previous choice points),

`fail` (cause backtracking)

Formatted output: https://www.swi-prolog.org/pldoc/man?predicate=format/2

Prolog Recursion and Non-determinism

```
?- findall(X, (sigmastar(X), length(X,N), (N>5 -> !, fail ; true)), List).
List = [[], [a], [b], [a, a], [b, a], [a, b], [b, b], [a|...],
[...|...]|...].
```

- abbreviated output (…), change it with:

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

Explanation:

- `findall(Variable, Goal, List)` – accumulate solutions for `Variable` in `Goal` into a `List`.
- `true` : nop (No operation) – does nothing.

Prolog Recursion and Non-determinism

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

- A list of solutions:

```
?- findall(X, (sigmastar(X), length(X,N), (N>5 -> !, fail ; true)), List).
List = [[], [a], [b], [a,a], [b,a], [a,b], [b,b], [a,a,a], [b,a,a], [a,b,a], [b,b,a], [a,a,b],
[b,a,b], [a,b,b], [b,b,b], [a,a,a,a], [b,a,a,a], [a,b,a,a], [b,b,a,a], [a,a,b,a], [b,a,b,a],
[a,b,b,a], [b,b,b,a], [a,a,a,b], [b,a,a,b], [a,b,a,b], [b,b,a,b], [a,a,b,b], [b,a,b,b],
[a,b,b,b], [b,b,b,b], [a,a,a,a,a], [b,a,a,a,a], [a,b,a,a,a], [b,b,a,a,a], [a,a,b,a,a],
[b,a,b,a,a], [a,b,b,a,a], [b,b,b,a,a], [a,a,a,b,a], [b,a,a,b,a], [a,b,a,b,a], [b,b,a,b,a],
[a,a,b,b,a], [b,a,b,b,a], [a,b,b,b,a], [b,b,b,b,a], [a,a,a,a,b], [b,a,a,a,b], [a,b,a,a,b],
[b,b,a,a,b], [a,a,b,a,b], [b,a,b,a,b], [a,b,b,a,b], [b,b,b,a,b], [a,a,a,b,b], [b,a,a,b,b],
[a,b,a,b,b], [b,b,a,b,b], [a,a,b,b,b], [b,a,b,b,b], [a,b,b,b,b], [b,b,b,b,b]].
```

Prolog Recursion and Non-determinism

```
?- findall(X, (sigmastar(X), length(X,N), (N>5 -> !, fail ; true)), List), length(List, M).

List = [[], [a], [b], [a,a], [b,a], [a,b], [b,b], [a,a,a], [b,a,a], [a,b,a], [b,b,a], [a,a,b],
[b,a,b], [a,b,b], [b,b,b], [a,a,a,a], [b,a,a,a], [a,b,a,a], [b,b,a,a], [a,a,b,a], [b,a,b,a],
[a,b,b,a], [b,b,b,a], [a,a,a,b], [b,a,a,b], [a,b,a,b], [b,b,a,b], [a,a,b,b], [b,a,b,b],
[a,b,b,b], [b,b,b,b], [a,a,a,a,a], [b,a,a,a,a], [a,b,a,a,a], [b,b,a,a,a], [a,a,b,a,a],
[b,a,b,a,a], [a,b,b,a,a], [b,b,b,a,a], [a,a,a,b,a], [b,a,a,b,a], [a,b,a,b,a], [b,b,a,b,a],
[a,a,b,b,a], [b,a,b,b,a], [a,b,b,b,a], [b,b,b,b,a], [a,a,a,a,b], [b,a,a,a,b], [a,b,a,a,b],
[b,b,a,a,b], [a,a,b,a,b], [b,a,b,a,b], [a,b,b,a,b], [b,b,b,a,b], [a,a,a,b,b], [b,a,a,b,b],
[a,b,a,b,b], [b,b,a,b,b], [a,a,b,b,b], [b,a,b,b,b], [a,b,b,b,b], [b,b,b,b,b]],

M = 63.
```

Prolog Recursion and Non-determinism

```
?- set_prolog_flag(answer_write_options,[max_depth(10)]).
true.

?- findall(X, (sigmastar(X), length(X,N), (N>5 -> !; true)),
List), length(List, M).
List =
[[],[a],[b],[a,a],[b,a],[a,b],[b,b],[a|...],[...|...]|...],
M = 63.
```

Prolog Recursion and Non-determinism

Is 63 the right answer?

- $L = \{s \mid s \in \Sigma^*, |s| \leq 5, \Sigma = \{a, b\}\}$
- length 0: [] (1)
- length 1: choice of either a or b (2)
- length 2: (4)
- length 3: (8)
- length 4: (16)
- length 5: (32)
- 32 + (16 + 8 + 4 + 2) + 1 = 63

$$\boxed{2^{n+1} - 1}$$