LING/C SC/PSYC 438/538

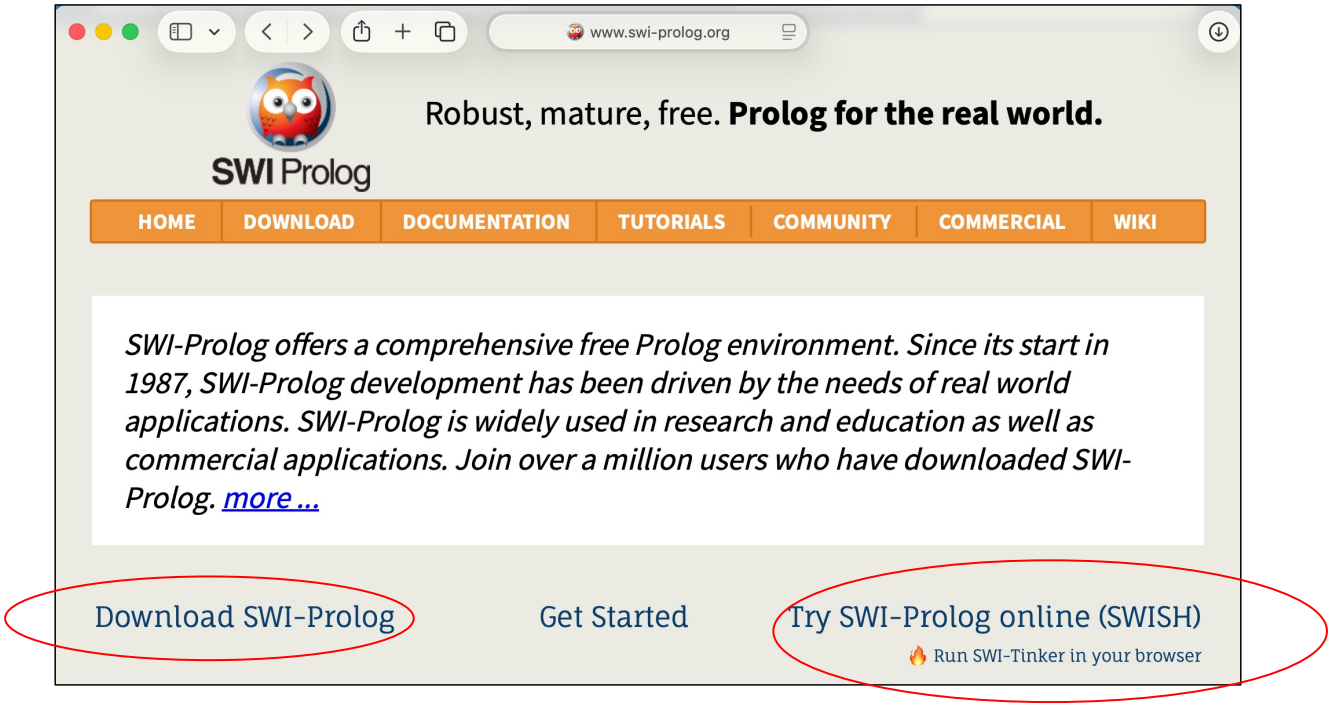
Lecture 22

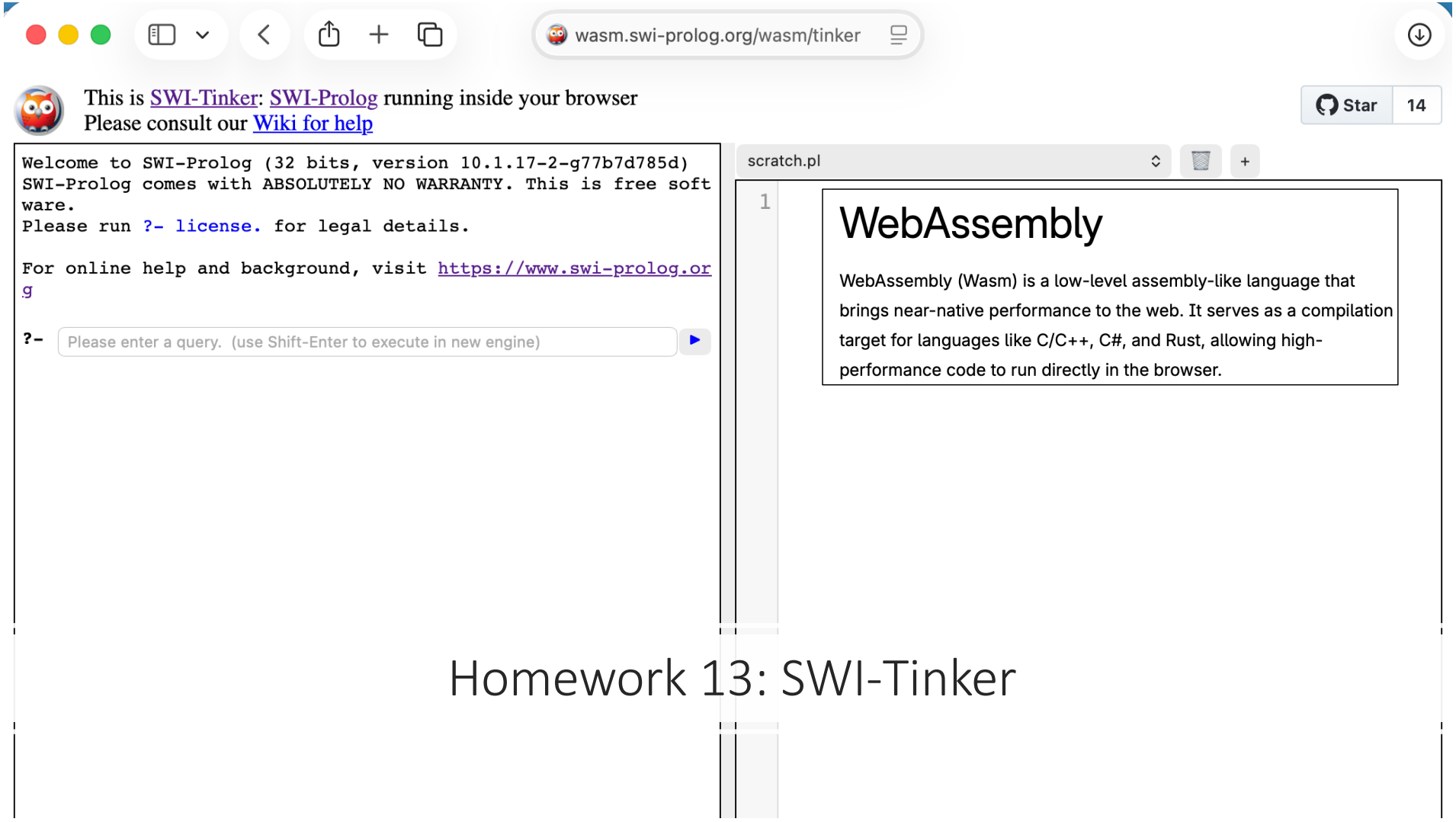
Prof. Sandiway Fong

Today's Topics

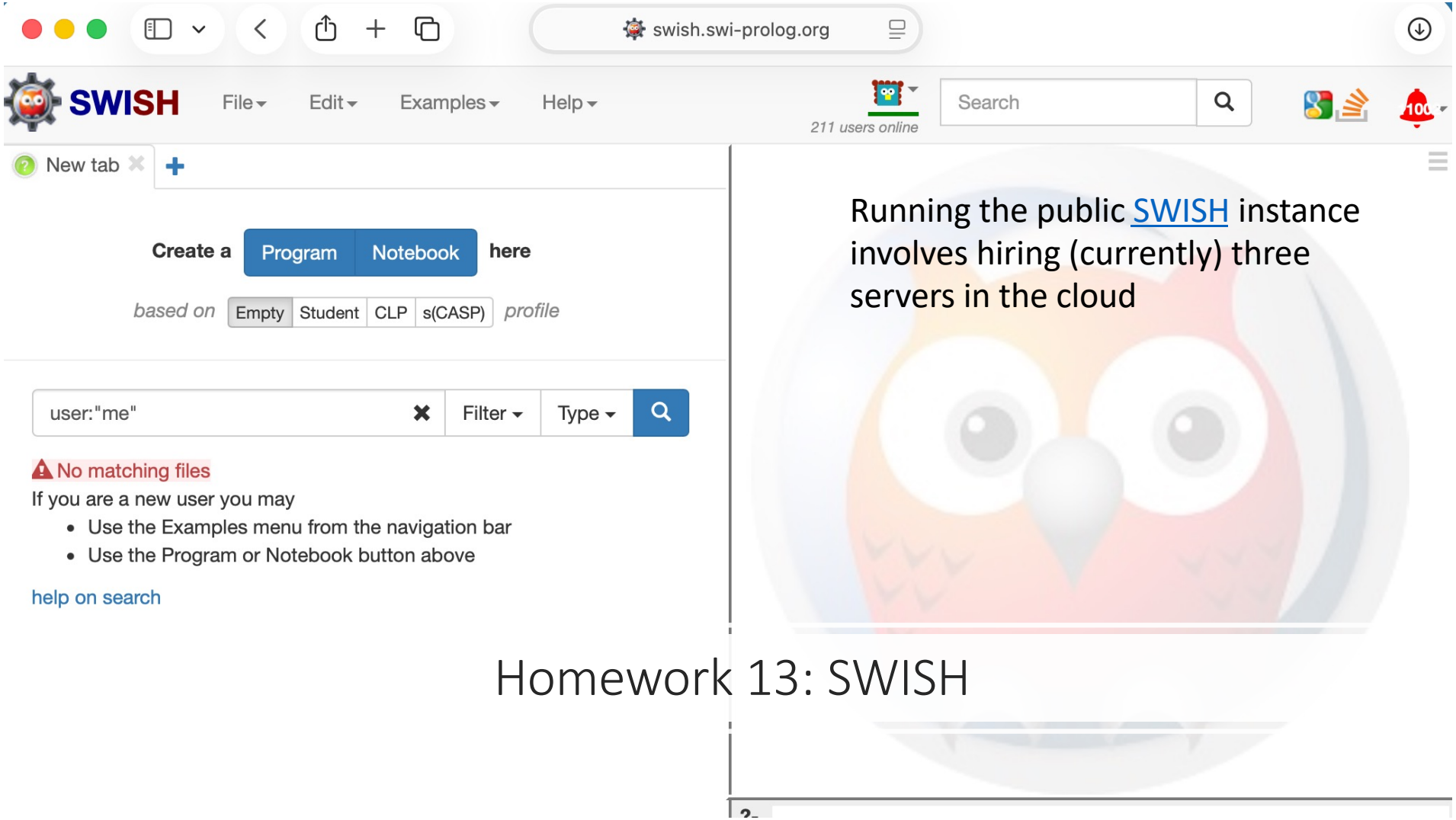
- So far for regular languages:
  - FSA: yes; regex: yes ; regular grammars: no
- Today:
  - a quick introduction to our programming language: Prolog
  - we'll be using this to explore regular grammars (*and beyond*)
- Homework 13:
  - install SWI-Prolog for next time

Homework 13


www.swi-prolog.org
SWI Prolog
Robust, mature, free. Prolog for the real world.
HOME
DOWNLOAD
DOCUMENTATION
TUTORIALS
COMMUNITY
COMMERCIAL
WIKI
SWI-Prolog offers a comprehensive free Prolog environment. Since its start in 1987, SWI-Prolog development has been driven by the needs of real world applications. SWI-Prolog is widely used in research and education as well as commercial applications. Join over a million users who have downloaded SWI-Prolog. more ...
Download SWI-Prolog
Get Started
Try SWI-Prolog online (SWISH)
Run SWI-Tinker in your browser



Homework 13: SWI-Tinker

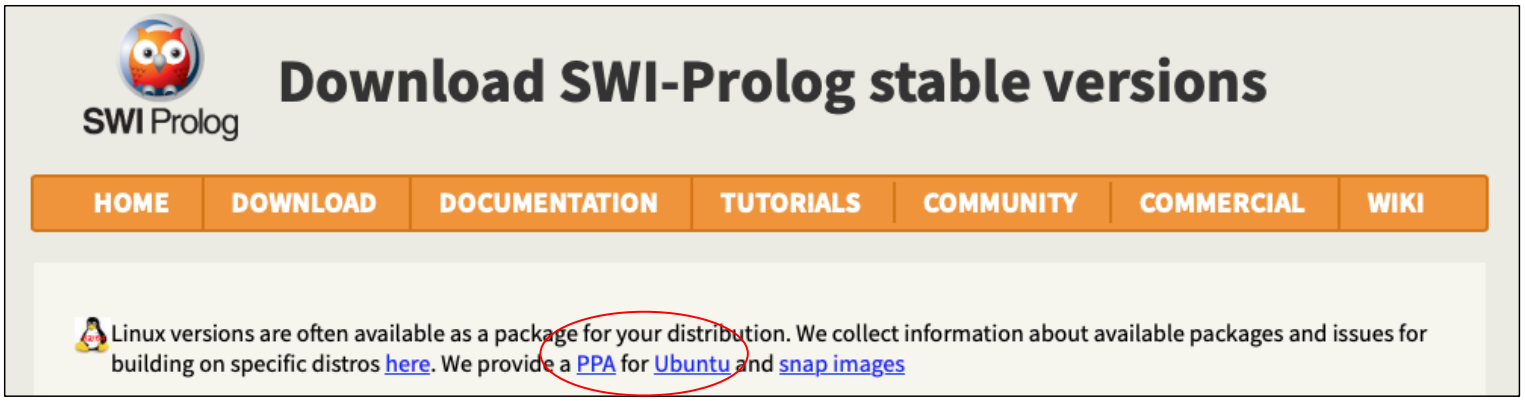


Running the public SWISH instance involves hiring (currently) three servers in the cloud

Homework 13: SWISH

Homework 13

- Install SWI-Prolog
  https://www.swi-prolog.org/download/stable e.g. installer for Windows or macOS



Homework 13: Download

Binaries		
x64	36,092,393 bytes	SWI-Prolog **10.0.2-1** for Microsoft Windows (64 bit) Self-installing executable for Microsoft Windows 64-bit editions. **SHA256**: `2ec1f25be0eafb92e559004782b130774c12573fb4b7e8a917e534754a641ad6`
	34,301,763 bytes	SWI-Prolog **10.0.2-1** for MacOSX 10.15 (Catalina) and later on x86_64 and arm64 Mac OS X disk image with relocatable application bundle. The bundle also provides the commandline tools in the `Contents/MacOS` directory. Users of older MacOS versions are adviced to use Macports, Homebrew or install from source. This bundle contains universal (fat) binaries that run natively on Intel (x86_64) and Apple Silicon (M*, arm64). **SHA256**: `bf775f0b8d7880f4908dee513316013ef42a73793be392814fde2a0a8e9ddc5d`
	21,704,201 bytes	SWI-Prolog **10.0.0-1** for MacOSX bundle on intel Mac OS X disk image with relocatable application bundle. Same as the *fat* bundle, but only contains the `x86_64` binaries. Used to be faster due to using GCC rather than Apple Clang. Since version 10.0.1 the fat bundle is built using GCC 15, making this version **obsolete**. **SHA256**: `bb49b99dc9b0bf6aa0772f9593eba092fece9bb2cc7ef3339144256990304dee`

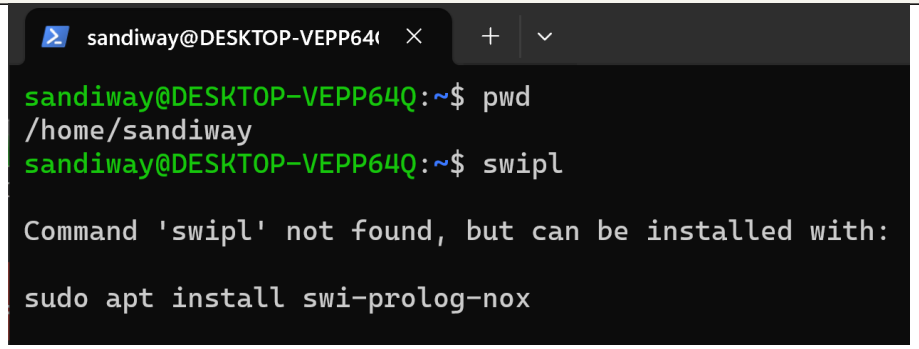
Homework 13: Ubuntu

- Ubuntu: PPA for SWI-Prolog

```
% sudo apt-get install software-properties-common
```

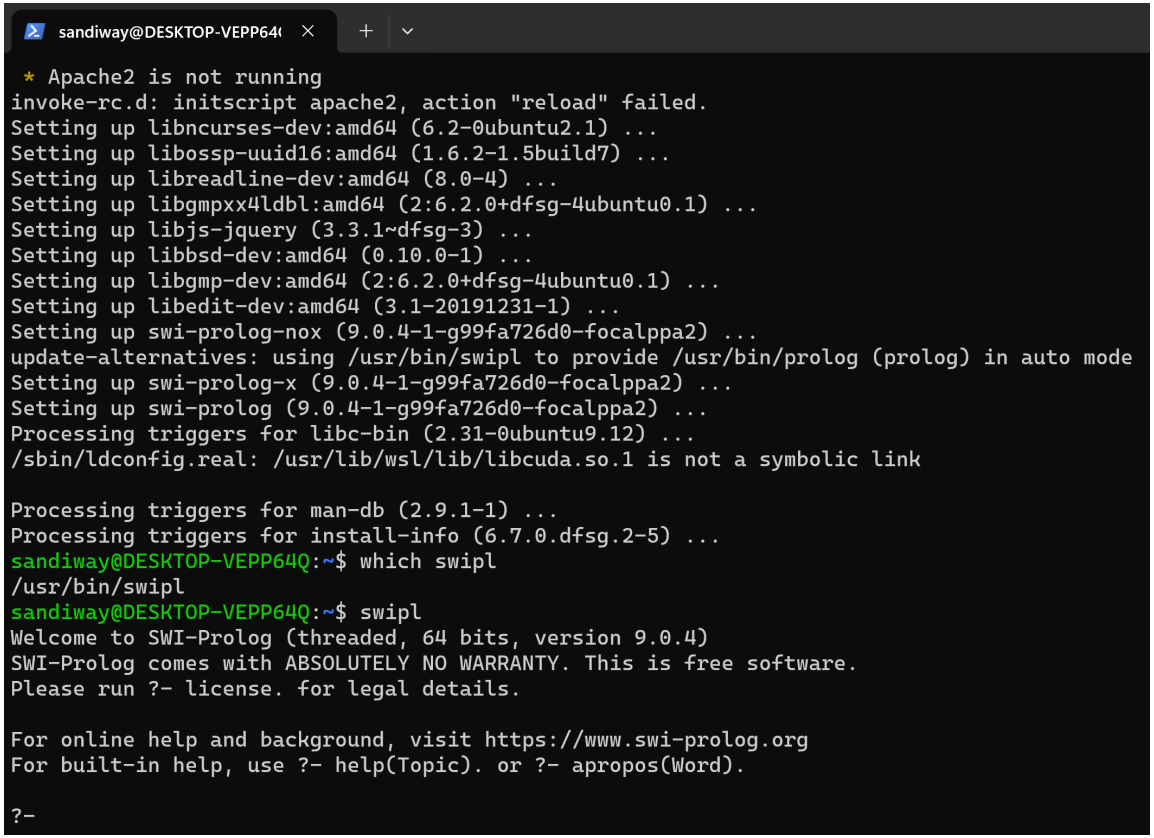
Stable versions

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

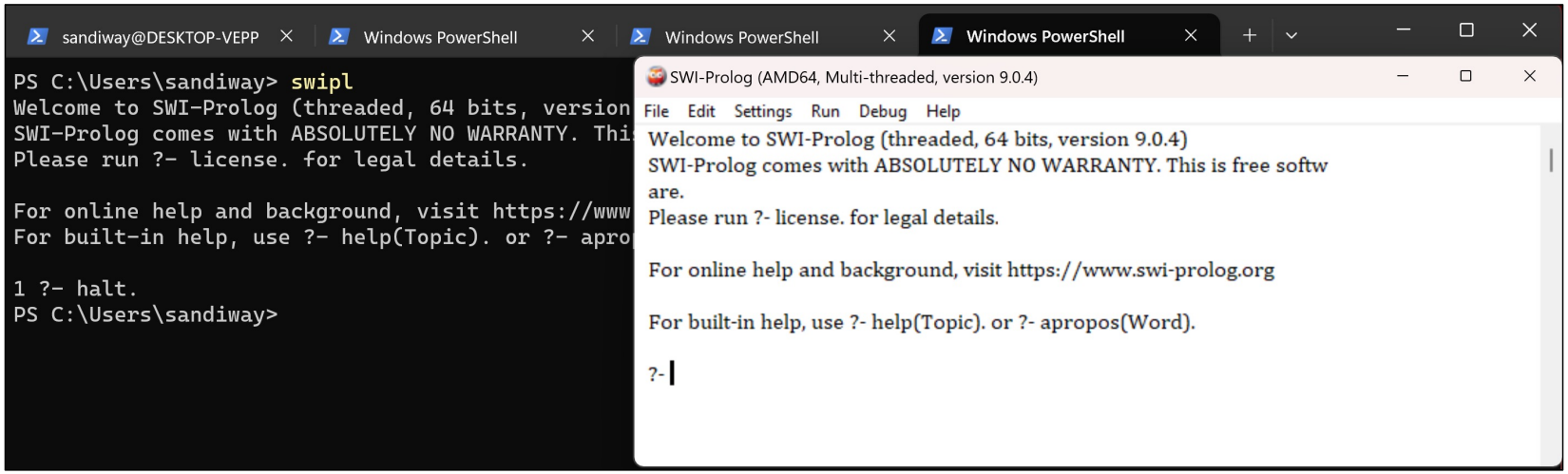


Homework 13: Ubuntu

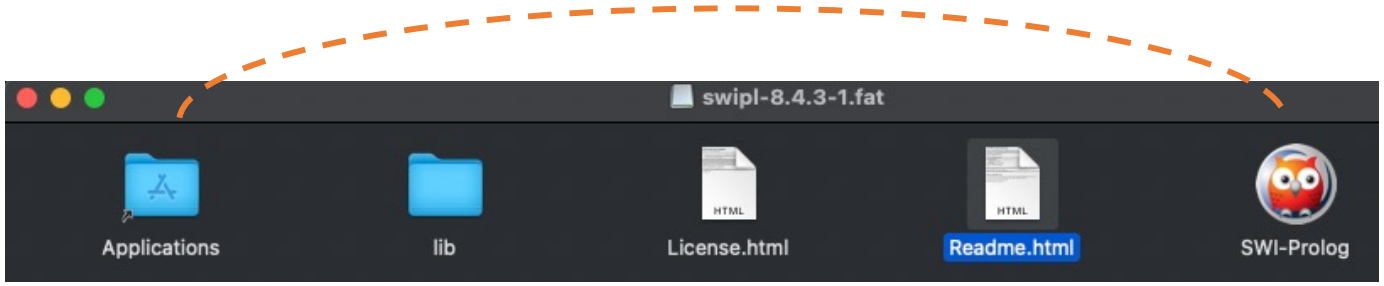
```
sudo apt-get install swi-prolog
```



Homework 13


sandiway@DESKTOP-VEPP
Windows PowerShell
Windows PowerShell
Windows PowerShell
PS C:\Users\sandiway> swipl
Welcome to SWI-Prolog (threaded, 64 bits, version
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. Thi
Please run ?- license. for legal details.
For online help and background, visit https://www
For built-in help, use ?- help(Topic). or ?- apro
1 ?- halt.
PS C:\Users\sandiway>
SWI-Prolog (AMD64, Multi-threaded, version 9.0.4)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free softw
are.
Please run ?- license. for legal details.
For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).
?-

Homework 13: macOS



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`

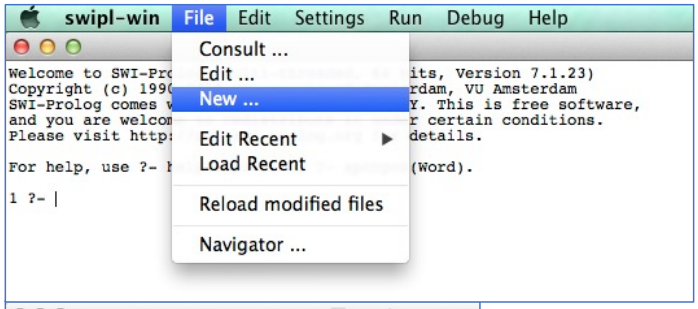
Homework 13: macOS

- Mac problems:



- option-click on application







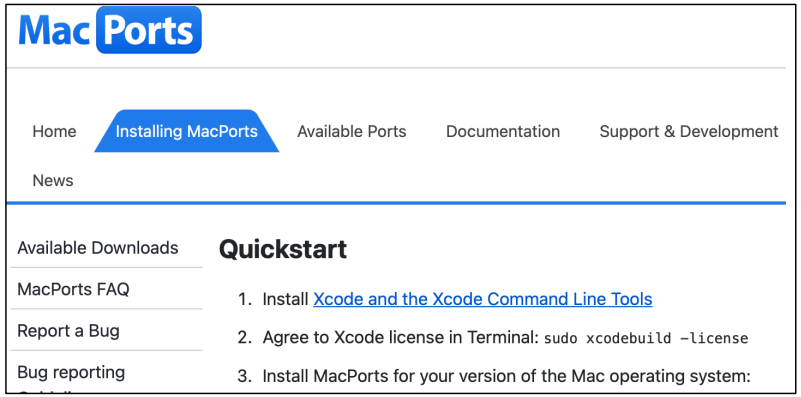
https://www.xquartz.org

- also Xcode needed

Homework 13: macOS

- Command line alternative to the application:

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




```
port install swi-prolog
```

- Another one is Homebrew

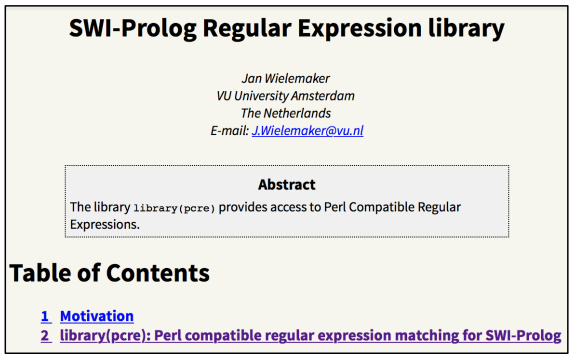


https://brew.sh

```
brew install swi-prolog
```

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

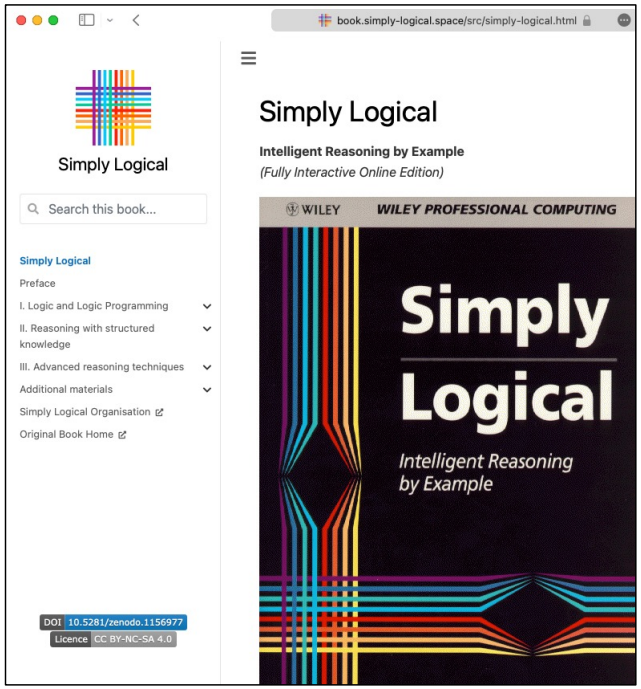
Abstract

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

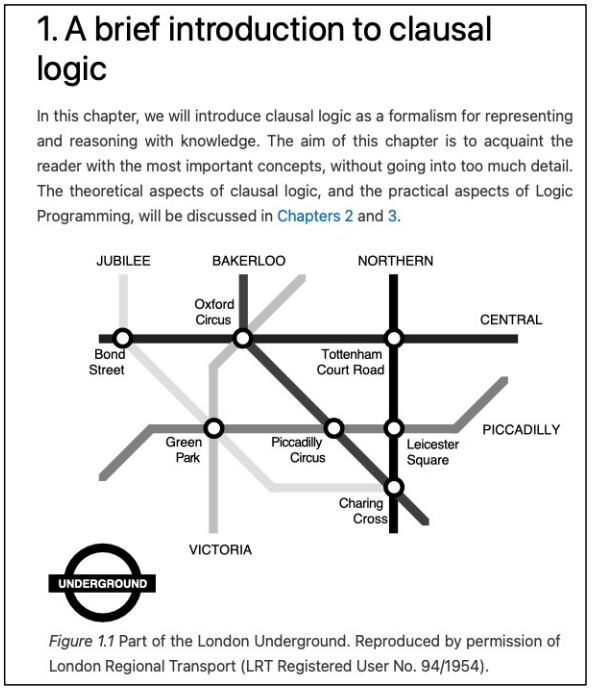
Table of Contents

1 Motivation
2 library(pcre): Perl compatible regular expression matching for SWI-Prolog

SWI-Prolog



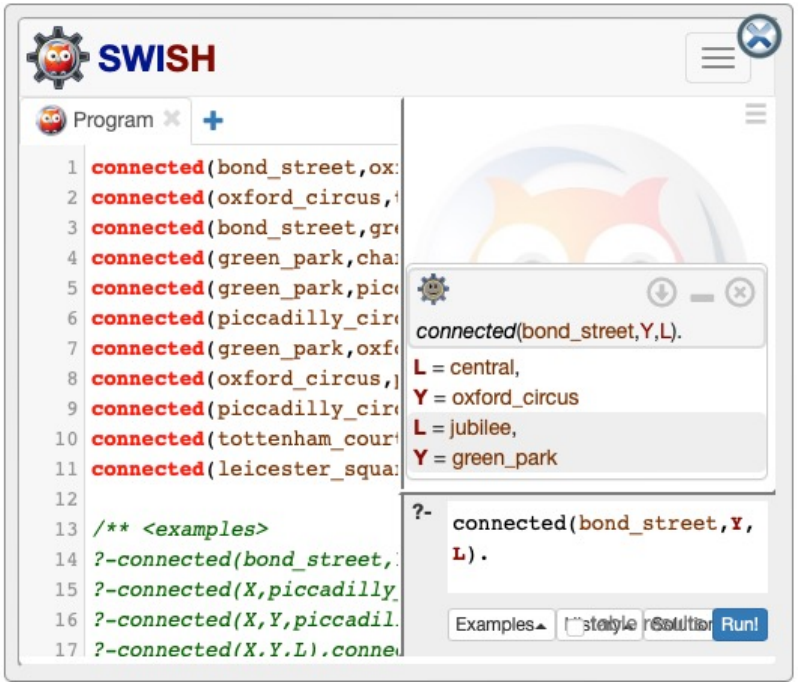
https://book.simply-logical.space/src/simply-logical.html



SWI-Prolog

SWISH

```
connected(bond_street,oxford_circus,central).
connected(oxford_circus,tottenham_court_road,central).
connected(bond_street,green_park,jubilee).
connected(green_park,charing_cross,jubilee).
connected(green_park,piccadilly_circus,piccadilly).
connected(piccadilly_circus,leicester_square,piccadilly).
connected(green_park,oxford_circus,victoria).
connected(oxford_circus,piccadilly_circus,bakerloo).
connected(piccadilly_circus,charing_cross,bakerloo).
connected(tottenham_court_road,leicester_square,northern).
connected(leicester_square,charing_cross,northern).
```



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:
`functor(arguments)`

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

`arguments:` comma-separated terms/atoms

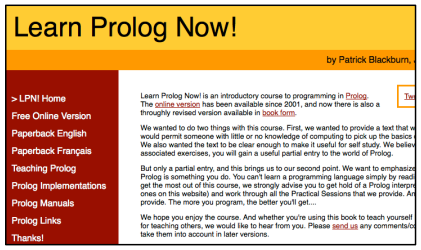
Example: as a logic programming language

scratch | test.pl

```
1 bird(tweety).
2 bird(penguin).
3
4 cantfly(penguin).
5
6 canfly(X) :- bird(X).
```

Learn Prolog Now!

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




```
?- 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

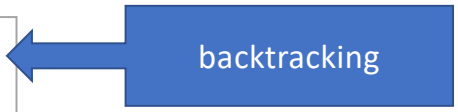
- Formal language example:
  - Suppose alphabet Σ={a, b}, enumerate Σ*

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

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

Run (hit ; for more answers)

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

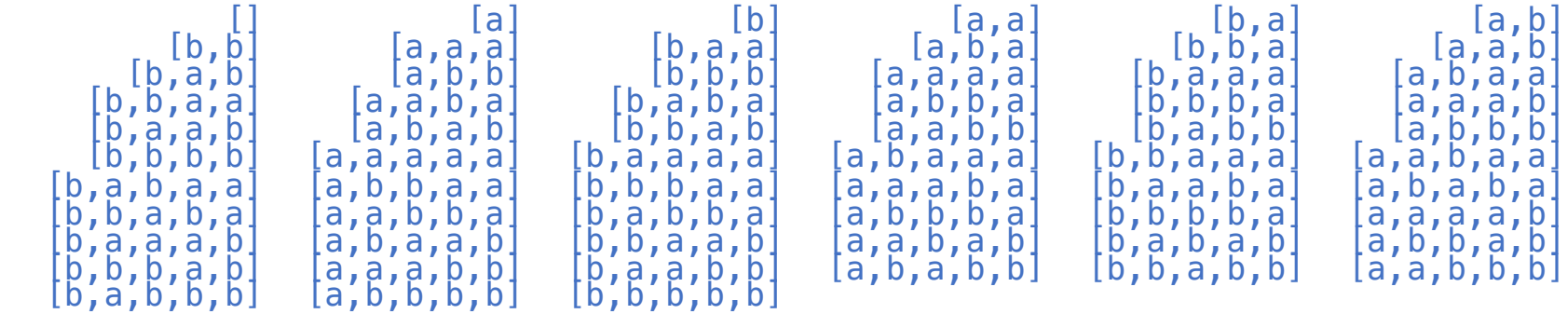



```
L = [] ;
L = [a] ;
L = [b] ;
L = [a, a] ;
L = [b, a] ;
L = [a, b] ;
L = [b, b] ;
L = [a, a, a] ;
L = [b, a, a] ;
L = [a, b, a] ;
L = [b, b, a] ;
L = [a, a, b] ;
L = [b, a, b] ;
L = [a, b, b] ;
L = [b, b, b] ;
…
```

Prolog Recursion and Non-determinism

width: 6 x 13 = 78

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




```
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