LING/C SC/PSYC 438/538

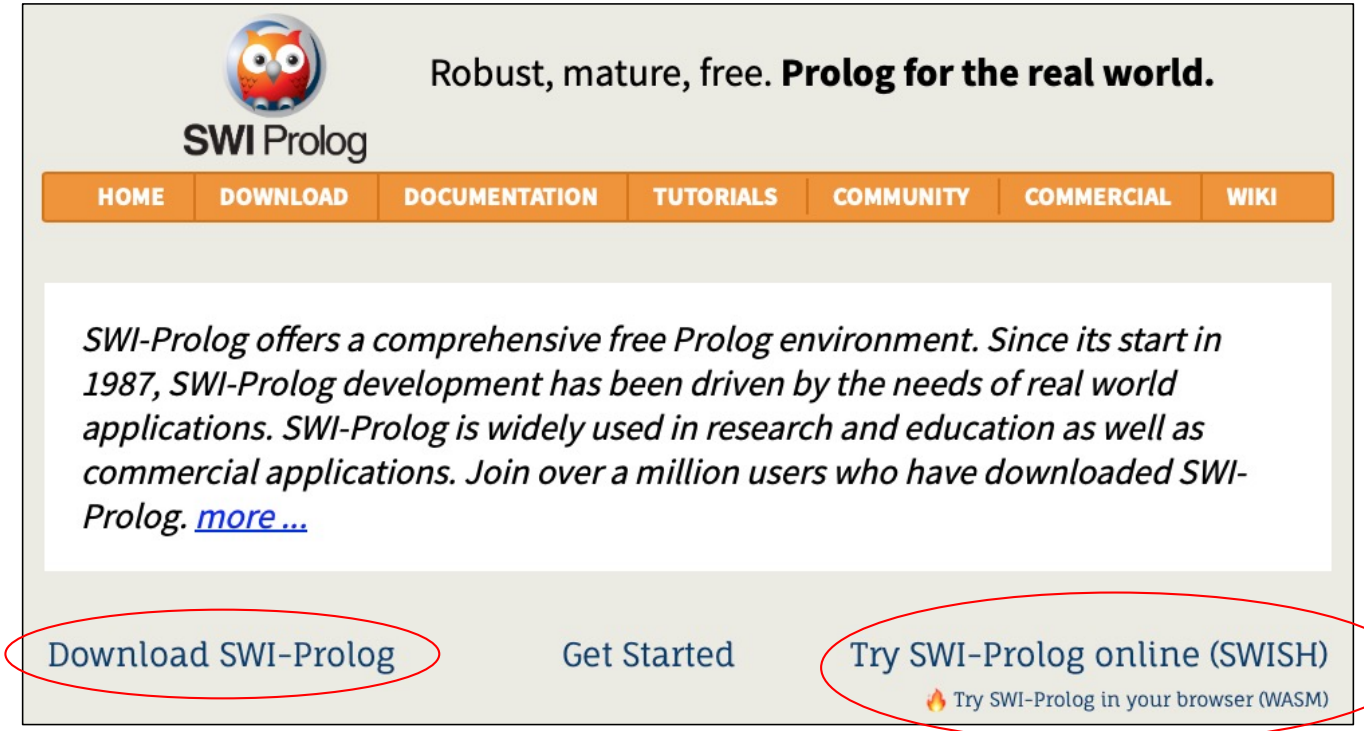
Lecture 23

Sandiway Fong

Today's Topics

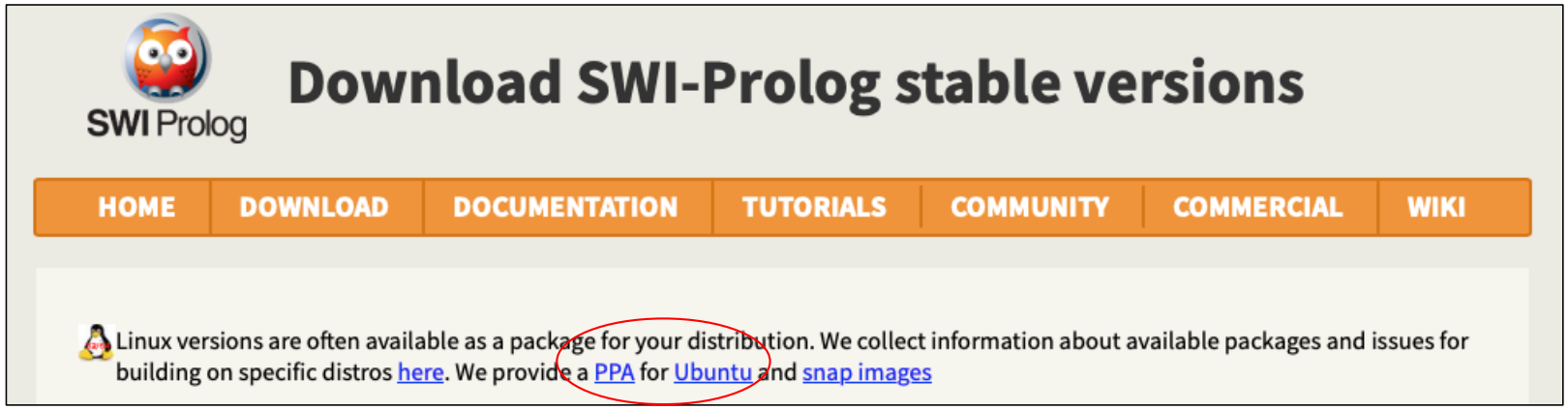
- So far for regular languages:
  - FSA: yes; regex: yes ; regular grammars: no
- Today:
  - a quick introduction to our programming language: Prolog
  - we'll be using this to explore regular grammars (*and beyond*)
- Homework 12:
  - install SWI-Prolog for next time

Homework 12


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)
Try SWI-Prolog in your browser (WASM)

Homework 12

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



Homework 12

Binaries		
	13,163,366 bytes	SWI-Prolog **9.0.4-1** for Microsoft Windows (64 bit) Self-installing executable for Microsoft's Windows 64-bit editions. Requires at least Windows 7. See the reference manual for deciding on whether to use the 32- or 64-bits version. This binary is linked against GMP 6.1.1 which is covered by the LGPL license. **SHA256**: `33758f1c2dd190df9c8828d2dcb39166ad10d31d78f1198812e6d0f33b71c73b`
	13,203,365 bytes	SWI-Prolog **9.0.4-1** for Microsoft Windows (32 bit) Self-installing executable for MS-Windows. Requires at least Windows 7. Installs **swipl-win.exe** and **swipl.exe**. This binary is linked against GMP 6.1.1 which is covered by the LGPL license. **SHA256**: `c99b7b794d14335ca6fda556f959e74c4b1b51877673a404f87c9cb68bce794c`
	51,743,650 bytes	SWI-Prolog **9.0.4-1** for MacOSX 10.14 (Mojave) and later on x86_64 and arm64 Installer with binaries created using Macports. Installs `/opt/local/bin/swipl`. Needs xquartz (X11) and the Developer Tools (Xcode) installed for running the development tools **SHA256**: `a6f32683e4c42e62ea6f8f481ac1f5f5fbfa2623b5c32eb21396a04c5ebbc197`
	28,195,489 bytes	SWI-Prolog **8.4.1-1** for MacOSX bundle on intel Installer with binaries created using Macports. Installs `/opt/local/bin/swipl`. Needs xquartz (X11) and the Developer Tools (Xcode) installed for running the development tools **SHA256**: `1b9c62caa781818a0dafd1d822ab563b8c10c7cd018ce10a3b71f900eb3a434f`

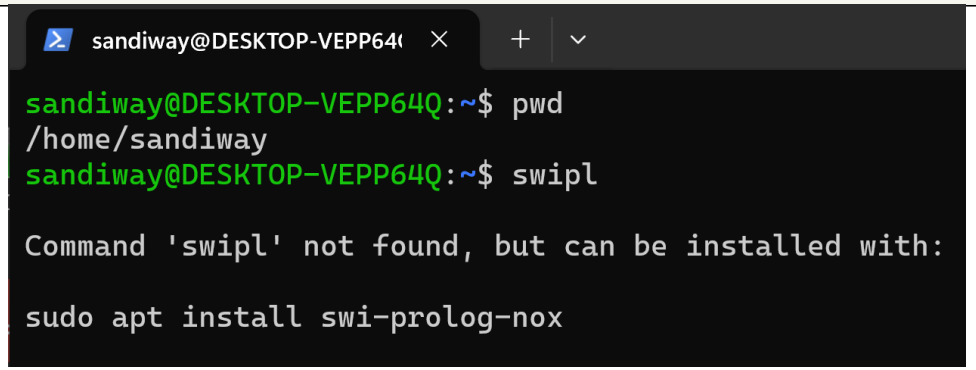
Homework 12: Ubuntu

- Ubuntu: PPA for SWI-Prolog

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

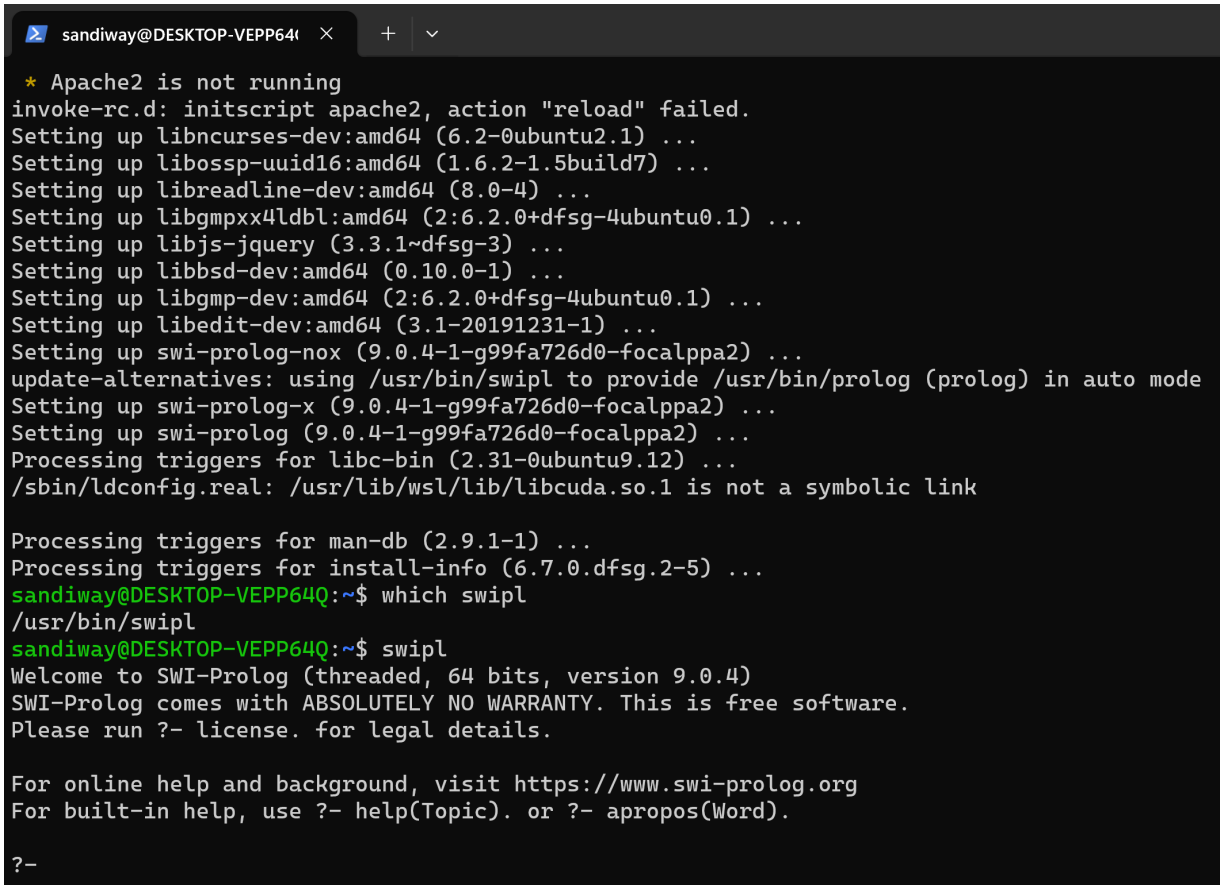
Stable versions

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

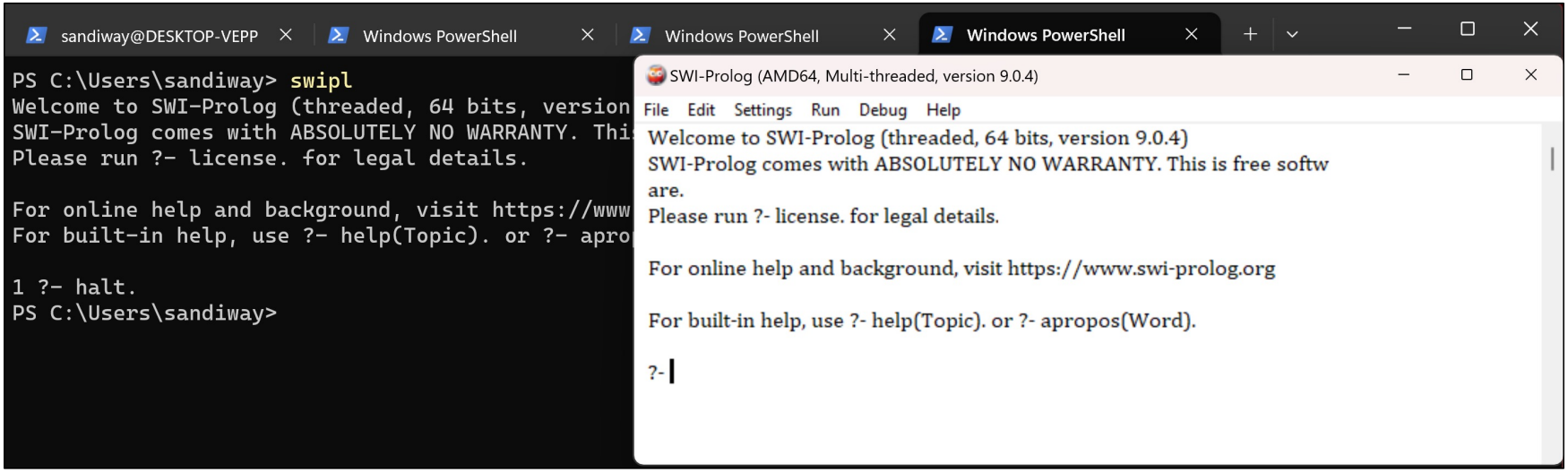


Homework 12: Ubuntu

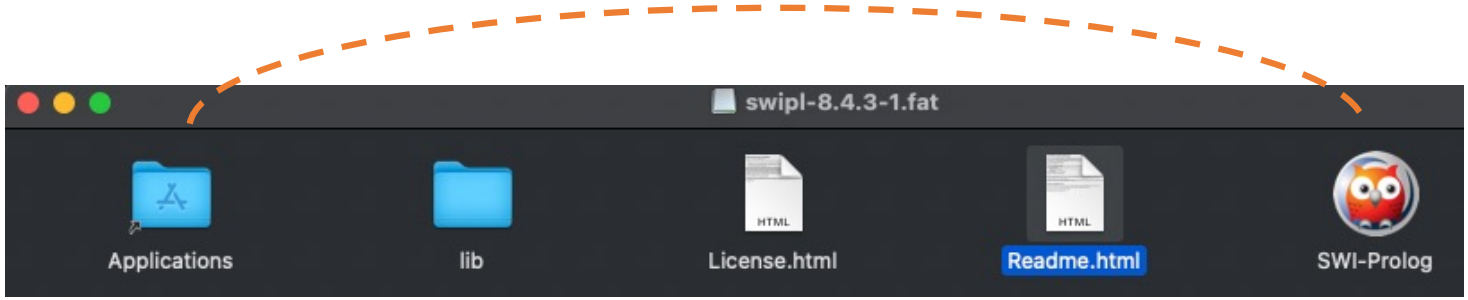
```
sudo apt-get install swi-prolog
```



Homework 12


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 12: 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

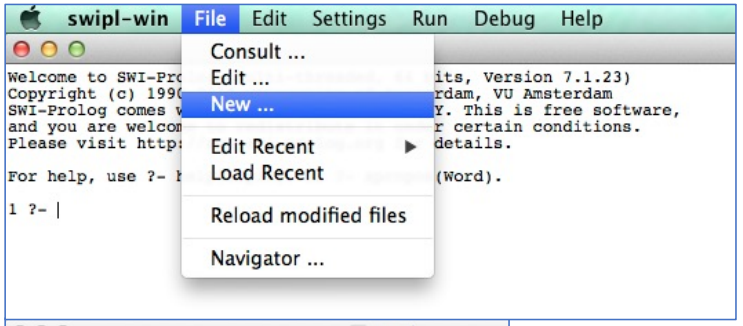
- `/Applications/SWI-Prolog.app/Contents/MacOS`

to $PATH

Homework 12: macOS

- Mac problems:

- option-click on application



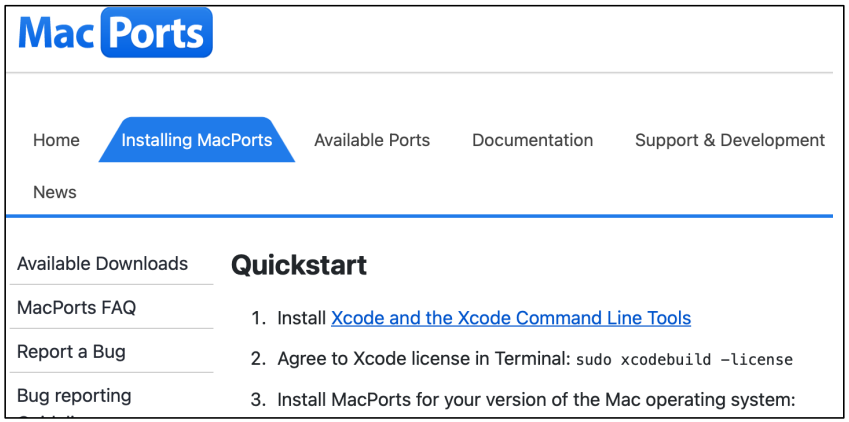
https://www.xquartz.org

- also Xcode needed

Homework 12: macOS

- Command line alternative to the application:

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




```
port install swi-prolog
```

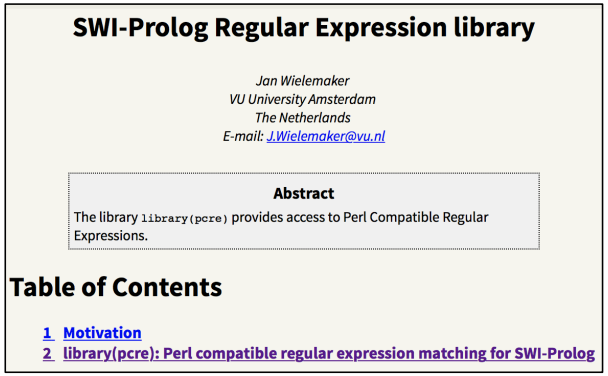
- Another one is Homebrew

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

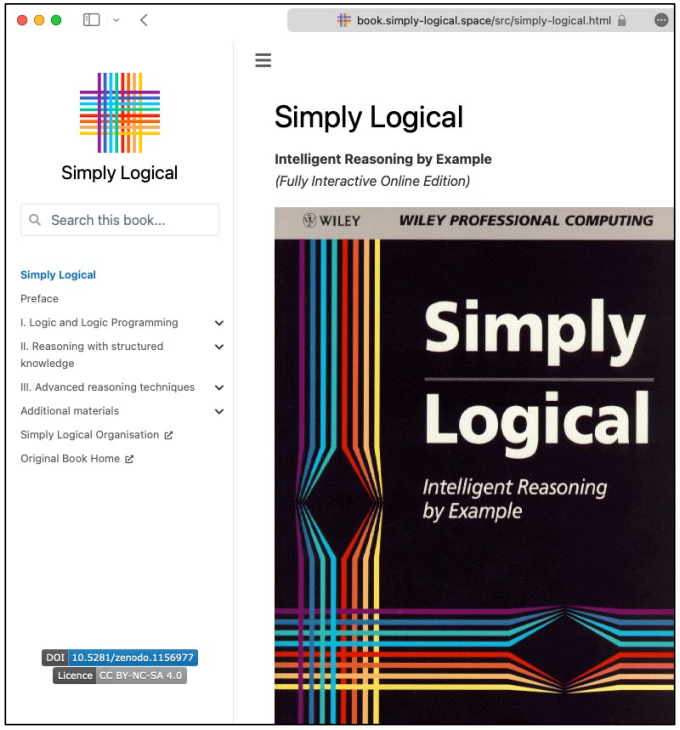
Jan Wielemaker
VU University Amsterdam
The Netherlands
E-mail: J.Wielemaker@vu.nl

Abstract

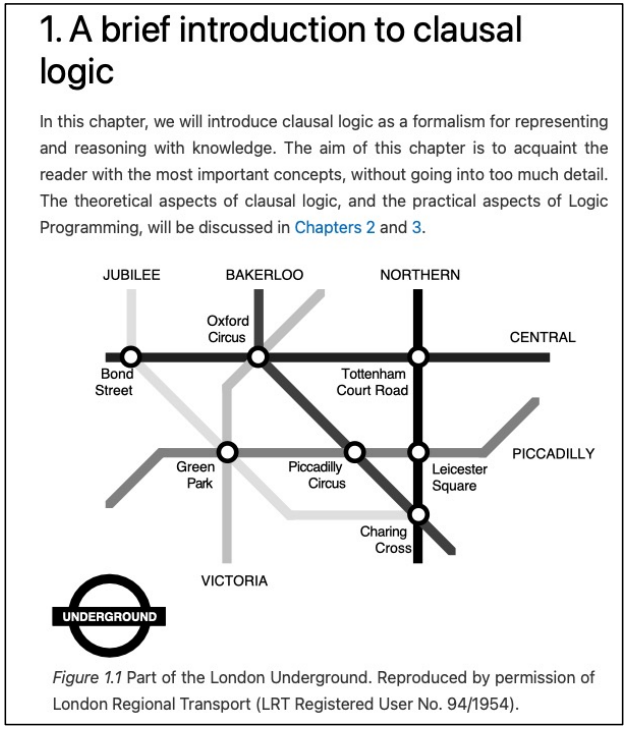
The library `library(pcre)` provides access to Perl Compatible Regular Expressions.

Table of Contents

SWI-Prolog



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



1. A brief introduction to clausal logic

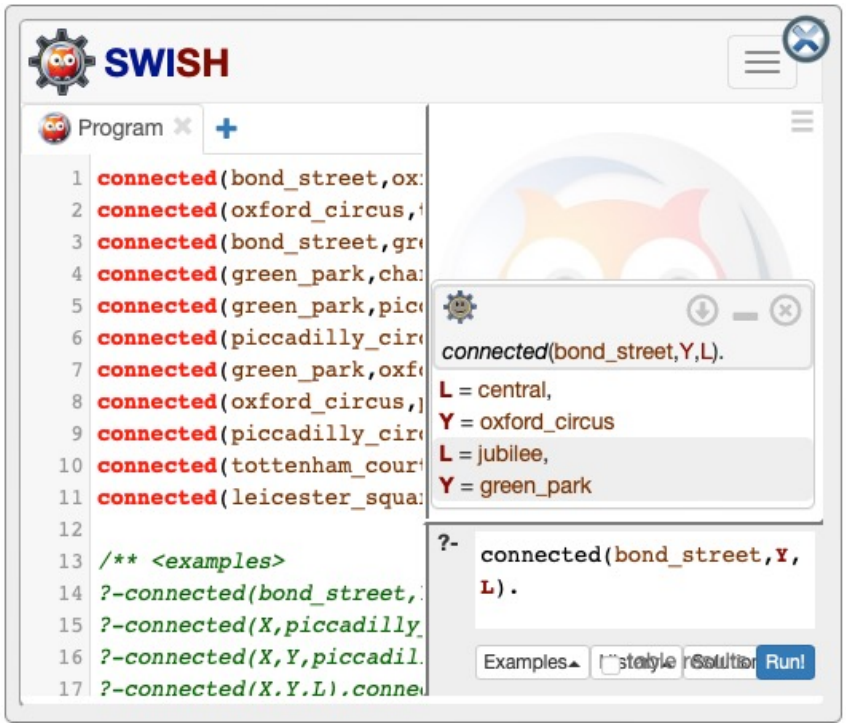
In this chapter, we will introduce clausal logic as a formalism for representing and reasoning with knowledge. The aim of this chapter is to acquaint the reader with the most important concepts, without going into too much detail. The theoretical aspects of clausal logic, and the practical aspects of Logic Programming, will be discussed in Chapters 2 and 3.

Figure 1.1 Part of the London Underground. Reproduced by permission of London Regional Transport (LRT Registered User No. 94/1954).

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:

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

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

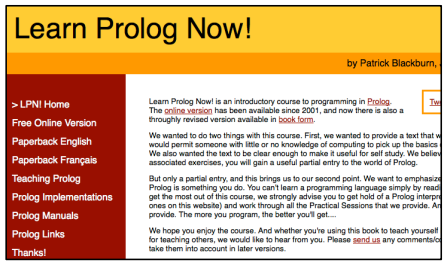
Example: as a logic programming language

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.

?-
```

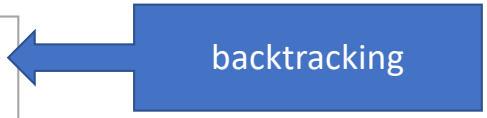
Prolog Recursion

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

```
1%%% Alphabet: {a, b}¶
2sigma(a).¶
3sigma(b).¶
4¶
5%%% Σ*¶
6sigmastar([]).¶
7sigmastar([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).
                []            [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,a,b,a]     [b,a,b,a]     [a,a,a,a]     [b,a,a,a]     [a,b,a,a]
    [b,b,a,a]     [a,b,a,b]     [b,b,a,b]     [a,b,b,a]     [b,b,b,a]     [a,a,a,b]
    [b,a,a,b]   [a,a,a,a,a]   [b,a,a,a,a]   [a,b,a,a,a]   [b,a,a,a,a]   [a,b,b,b]
    [b,b,b,b] [a,b,b,a,a]   [b,b,b,a,a]   [a,a,a,b,a]   [b,b,b,b,a]   [a,a,b,a,a]
  [b,a,b,a,a] [a,a,b,b,a]   [b,a,b,b,a]   [a,b,b,b,a]   [b,a,b,b,a]   [a,b,a,b,a]
  [b,b,a,b,a] [a,b,a,a,b]   [b,b,a,a,b]   [a,a,b,a,b]   [b,b,a,a,b]   [a,a,a,a,b]
  [b,a,a,a,b] [a,a,a,b,b]   [b,a,a,b,b]   [a,b,a,b,b]   [b,a,a,b,b]   [a,b,b,a,b]
  [b,b,b,a,b] [a,b,b,b,b]   [b,b,b,b,b]   [a,a,b,b,b]   [b,b,a,b,b]   [a,a,b,b,b]
  [b,a,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

Built-in predicates:

- `Goal`
  - can succeed or fail
- `findall(Variable, Goal, List)`
  - accumulate values for `Variable` in Prolog `Goal` when it succeeds into a `List`
  - always succeeds
- `true`
  - succeeds (No operation) – does nothing.
- `fail`
  - fails – purpose is to initiate backtracking to find more answers (if they exist).

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 | s ∈ Σ*, |s| ≤ 5, Σ = {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

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