CPSC 312 — Functional and Logic Programming

- Assignment 4 is due Thursday!
- "Logic is the beginning of wisdom, not the end."

Leonard Nimoy
"Star Trek VI: The Undiscovered Country" 1991

Plan

Done:

- Syntax and semantics of propositional definite clauses
- Model a simple domain using propositional definite clauses
- Soundness and completeness of a proof procedure
- Bottom-up proof procedure
- Bottom-up proof procedure: soundness and completeness

Today:

- Top-down proof procedure
- Procedural definition (box model)

Semantics

- An interpretation I assigns a truth value to each atom.
- rule h :- $b_1, \ldots, b_k$ is true unless
  h is false and $b_1 \ldots b_k$ are all true.
- A model of a set of clauses is an interpretation in which all the clauses (atomic facts and rules) are *true*.
- If KB is a set of clauses and g is a conjunction of atoms, g is a logical consequence of KB, written $KB \models g$, if g is *true* in every model of KB.
- That is, $KB \models g$ if there is no interpretation in which KB is *true* and g is *false*.

Proofs

- A proof is a mechanically derivable demonstration that a formula logically follows from a knowledge base.
- Given a proof procedure, $KB \vdash g$ means g can be derived from knowledge base KB.
- Recall $KB \models g$ means g is true in all models of KB.
- A proof procedure is sound if $KB \vdash g$ implies $KB \models g$.
  - If a sound proof procedure produces a result, the result is correct.
- A proof procedure is complete if $KB \models g$ implies $KB \vdash g$.
  - A complete proof procedure can produce all results.

Clicker Question

Consider the knowledge base KB:

$$\begin{array}{ll} xox \text{ :- } bar,\ fun. & aah \text{ :- } green. \\ bar \text{ :- } zed. & aah \text{ :- } blue. \\ zed. & green. \end{array}$$

What is the final consequence set in the bottom-up proof procedure run on KB?

A $\{aah, blue, green, xox, bar, fun, zed\}$

B $\{aah, blue, green, xox, bar, zed\}$

C $\{aah, green, bar, zed\}$

D $\{green, bar, zed\}$

E None of the above

Top-down Definite Clause Proof Procedure

- Idea: search backward from a query to determine if it is a logical consequence of KB.
- An answer clause is of the form:

  $$yes \text{ :- } a_1,\ a_2,\ \ldots,\ a_m$$

- The (SLD) resolution of this answer clause on atom a_1 with the clause in the knowledge base:

  $$a_1 \text{ :- } b_1,\ \ldots,\ b_p$$

  is the answer clause

  $$yes \text{ :- } b_1,\ \cdots,\ b_p,\ a_2,\ \cdots,\ a_m.$$

  A fact in the knowledge base is considered as a clause where $p = 0$.

Clicker Question

Given the answer clause

yes :- *good*, *happy*, *green*.

Which clause in a KB could this be resolved with

(i) *green* :- *good*.
(ii) *good*.
(iii) *happy* :- *green*.
(iv) *good* :- *nice*, *green*.
(v) *happy*, *green*

Click on:

A (i), (ii) and (iv) only
B (ii) and (iv) only
C (ii) only
D (iii) and (v) only
E none of the above

Clicker Question

Given the answer clause

yes :- *good*, *happy*, *green*.

What is the result of resolving this with the clause

good :- *nice*, *green*.

A *yes* :- *good*, *nice*, *green*, *happy*, *green*

B *good* :- *happy*, *green*

C *yes* :- *happy*, *green*

D *yes* :- *nice*, *green*, *happy*, *green*

E *yes* :- *nice*, *green*, *happy*

Clicker Question

Given the answer clause

yes :- *happy*, *green*, *good*.

What is the result of resolving this with the clause

happy.

A *yes* :- *good*, *nice*, *green*, *happy*, *green*

B *happy* :- *green*, *good*

C *yes* :- *happy*, *green*

D *yes* :- *happy*, *green*, *good*

E *yes* :- *green*, *good*

Derivations

- An answer is an answer clause with $m = 0$. That is, it is the answer clause *yes* :- .
- A derivation of query "?$q_1, \ldots, q_k$" from KB is a sequence of answer clauses $\gamma_0, \gamma_1, \ldots, \gamma_n$ such that
  - γ_0 is the answer clause *yes* :- $q_1, \ldots, q_k$
  - γ_i the resolution of γ_{i-1} with a clause in KB
  - γ_n is an answer.

Top-down definite clause interpreter

To solve the query $?q_1, \ldots, q_k$:

$ac :=$ "yes :- $q_1, \ldots, q_k$"
repeat
    select leftmost atom a_1 from the body of ac
    choose clause C from KB with a_1 as head
    replace a_1 in the body of ac by the body of C
until ac is an answer.

Nondeterministic Choice

- Don't-care nondeterminism If one selection doesn't lead to a solution, there is no point trying other alternatives.
  "select"
- Don't-know nondeterminism If one choice doesn't lead to a solution, other choices may.
  "choose"

Example: successful derivation

a :- b, c.
c :- e.
f :- j, e.

a :- e, f.
d :- k.
f :- c.

b :- f, k.
e.
j :- c.

Query: ?a

γ_0 : *yes* :- a
γ_1 : *yes* :- e, f
γ_2 : *yes* :- f
γ_3 : *yes* :- c

γ_4 : *yes* :- e
γ_5 : *yes* :-

Example: failing derivation

a :- b, c.
c :- e.
f :- j, e.

a :- e, f.
d :- k.
f :- c.

b :- f, k.
e.
j :- c.

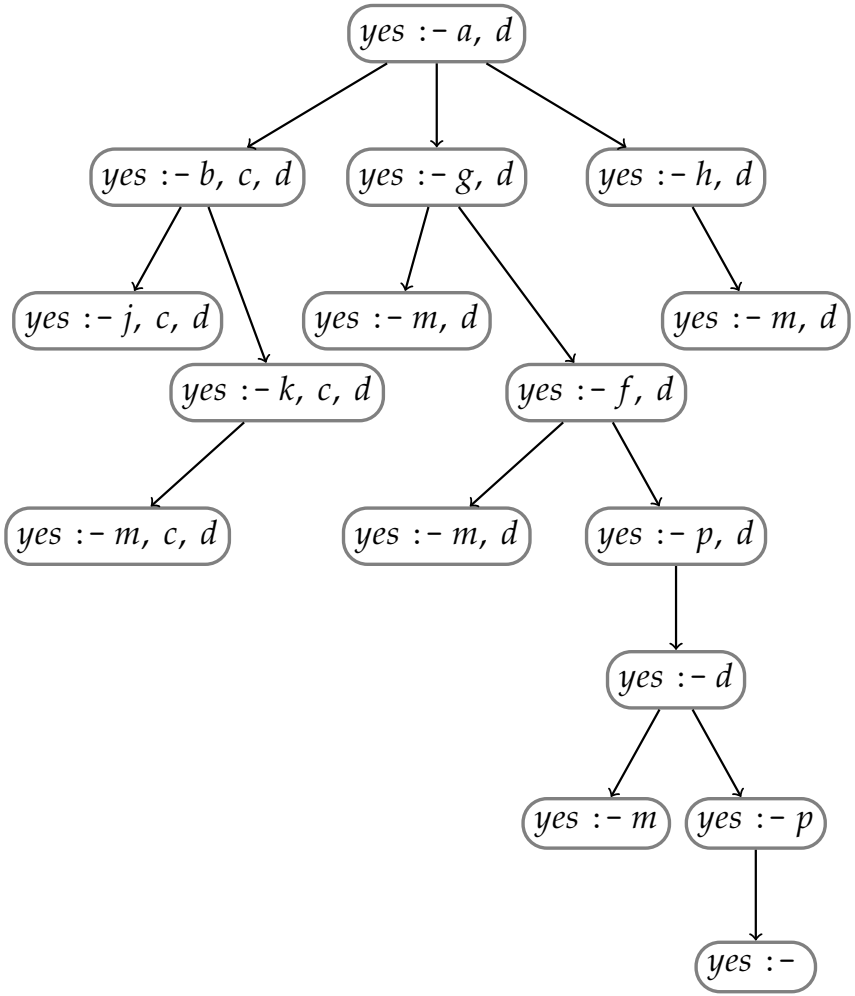
Query: ?a

γ_0 : yes :- a
γ_1 : yes :- b, c
γ_2 : yes :- f, k, c
γ_3 : yes :- c, k, c
γ_4 : yes :- e, k, c
γ_5 : yes :- k, c

Search Graph for SLD Resolution

a :- b, c.
a :- h.
b :- k.
d :- p.
f :- p.
g :- f.
h :- m.
?a, d.

a :- g.
b :- j.
d :- m.
f :- m.
g :- m.
k :- m.
p.



Soundness and completeness of top-down proof procedure

- Is top-down proof procedure with depth-first search sound?
  Yes!
- Is top-down proof procedure with depth-first search complete?
  No!

```
a :- b.
b :- a.
b :- c.
c.
?a.
```

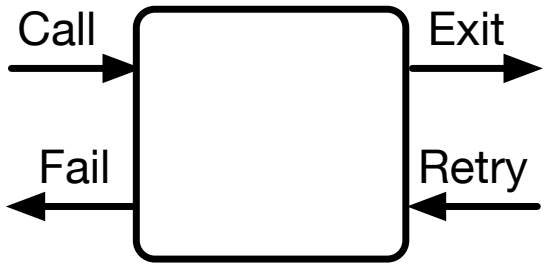
This can get stuck in an infinite loop.

Procedural View of Propositional Prolog

Informally:

- The clauses with same atom as head define a procedure.
- Each procedure either succeeds or fails.
- To call a procedure, call each body in turn until one succeeds.
- To call a body, call each atom in the body.
  - A body succeeds if all atoms in the body succeed.
- The procedure fails if no bodies succeed.

Box Model of Propositional Prolog



Try in Prolog:

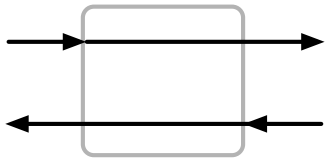
?– *trace*.

Example (simpeg.pl)

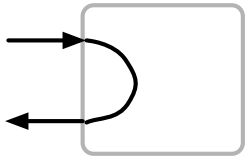
```
a :- b, c.
a :- e, f.
b :- f, k.
c :- e.
d :- k.
e.
f :- j, e.
f :- c.
j :- c.
```

Wiring Boxes

- Fact (some atom is just a fact).

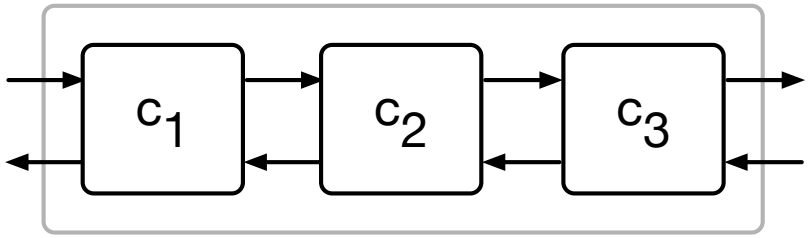


- Atom not defined



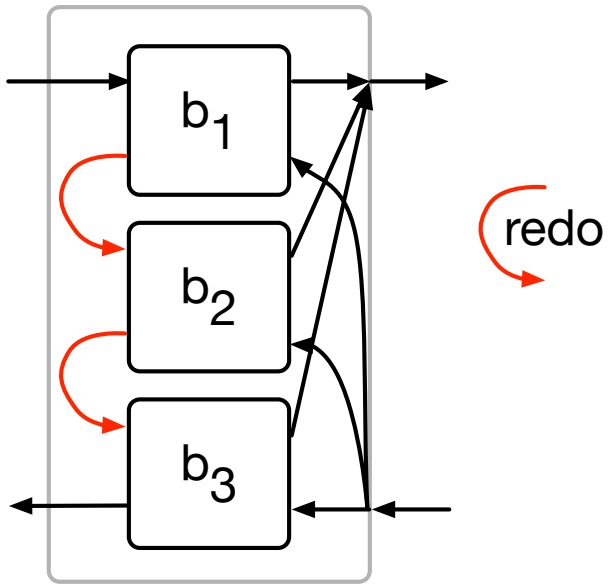
Wiring Boxes

- conjunction c_1, c_2, c_3



- Rules:

a :- b_1
a :- b_2
a :- b_3



It redoes whichever body exited. (It remembers this on "local stack").

Prolog Debugging

- If there are no clauses for an atom, Prolog assumes there is an error. Add a declaration

  ```
  :- dynamic atom.
  ```

  for an atom that has no clauses on purpose.
- To start a trace Prolog do:

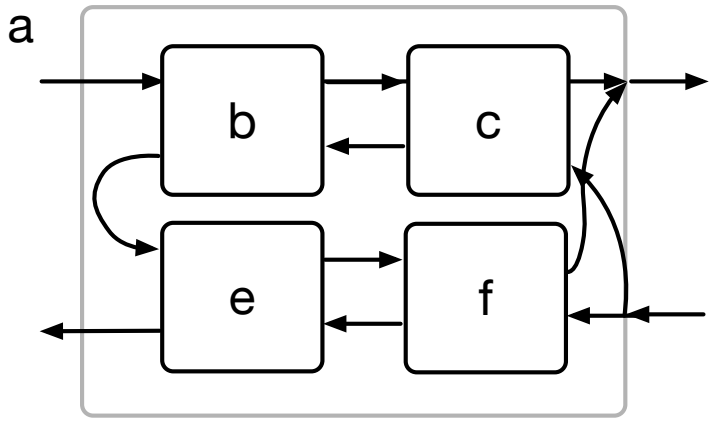
  ```
  ?- trace.
  ```
- Prolog displays call and exit following the box model.
- It does not diplay retry/fail unless there is another clause to try, where it displays a redo.
- Prolog gives one answer for each proof.
  Type ; to find another proof.
- Sometimes the debug trace misses parts becuase the compiler recognizes that these parts are unnecessary.

Example (simpeg.pl)

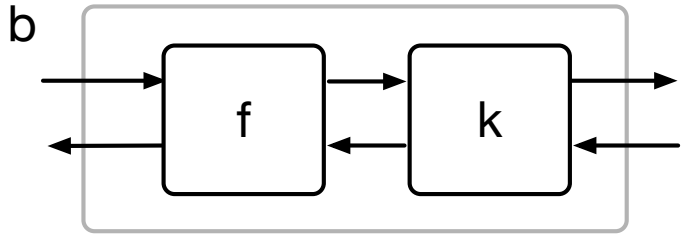
```
a :- b, c.
a :- e, f.
b :- f, k.
c :- e.
d :- k.
e.
f :- j, e.
f :- c.
j :- c.
```

Box examples

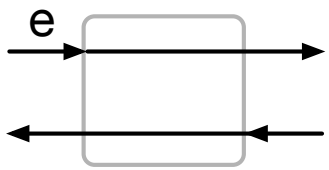
a :- b, c.
a :- e, f.



b :- f, k.



e.



no clauses for k