- Assignment 4 is due next Thursday! From last class, you know enough to do questions 1 and 2.
- "Consequently he who wishes to attain to human perfection, must therefore first study Logic, next the various branches of Mathematics in their proper order, then Physics, and lastly Metaphysics."

  Maimonides 1135–1204

Plan

- Logic Programming
  - Propositional logic programs
    - Semantics
    - Bottom-up and top-down proof procedures
  - Datalog
  - Logic programs with function symbols
  - Applications (e.g., natural language processing)
  - Semantic web

Propositional Logic Program Syntax

- An atom is of the form p, a word that (can contain letters, digits and underscore _) and starts with a lower-case letter.
- A body is either
  - an atom or
  - (b_1, b_2) where b_1 and b_2 are bodies. (Parentheses are optional). A comma in a body means "and".
- A definite clause is either
  - an atomic clause: an atom or
  - a rule: h :- b where h is an atom and b is a body.

    :- means "if"

  An atomic clause is treated as a rule with an empty body.
  All definite clauses ends with a period "."
- A logic program or knowledge base is a set of definite clauses
- A query is a body that is asked at the Prolog prompt (ended with a period).
- comments start with % to end of line

Datalog Syntax

(Needed for assignment 4)
Syntax is same as for propositional logic programs, with expanded definition of atom:

- An atom is of the form $p(t_1, \ldots, t_n)$ or p
- p is a predicate symbol, starts with lower-case letter (e.g., mother, parent, instructor, ta)
- each t_i is a term which is either:
  - a constant: a number or a word starting with a lower-case letter (e.g., justin, cs312, 2024)
  - a logical variable: a word starting with an upper-case letter
- In a query a variable X means "for what value of X is the query a logical consequence". Another answer can be obtained using semicolon ";"

See family.pl for example queries.

Human's view of semantics

Step 1 Begin with a task domain.

Step 2 Choose atoms in the computer to denote propositions. These atoms have meaning to the KB designer.

Step 3 Tell the system knowledge about the domain.

Step 4 Ask the system questions.

— The system gives answers.
— Person can interpret the answer with the meaning associated with the atoms.

A progression of logical languages

- Propositional logic programs: atoms only have constants as arguments (no variables).
- Datalog: allow for logical variables in clauses.
- Pure Prolog: Datalog + function symbols

Semantics

- An interpretation I assigns a truth value to each atom.
- True of compound propositions in interpretation is derived from truth table:

p	q	$p,\ q$	p :- q
true	*true*	*true*	*true*
true	*false*	*false*	*true*
false	*true*	*false*	*false*
false	*false*	*false*	*true*

- A body $(b_1,\ b_2)$ is true in I if b_1 is true in I and b_2 is true in I.
- A rule h :- b is false in I if b is true in I and h is false in I. The rule is true otherwise.
- A knowledge base KB is true in I if and only if every clause in KB is true in I.

Models and Logical Consequence

- 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*.

Clicker Question

Consider the knowledge base KB:

$$
\begin{array}{ll}
happy \;\text{:-}\; good. & foo \;\text{:-}\; bar,\; fun. \\
happy \;\text{:-}\; green. & bar \;\text{:-}\; zed. \\
green. & zed.
\end{array}
$$

Which of the following are true
($KB \not\models g$ means "g is a not a logical consequence of KB")

A $KB \models happy$ and $KB \models foo$

B $KB \models happy$ and $KB \not\models foo$

C $KB \not\models happy$ and $KB \models foo$

D $KB \not\models happy$ and $KB \not\models foo$

E I'm not sure, please explain it again.

Clicker Question

Consider the knowledge base KB:

$$\begin{array}{ll} happy \;\text{:-}\; good. & foo \;\text{:-}\; bar,\; fun. \\ happy \;\text{:-}\; green. & bar \;\text{:-}\; zed. \\ green. & zed. \end{array}$$

What is the set of all atoms that are logical consequences of KB?

A $\{happy, good, green, foo, bar, fun, zed\}$

B $\{happy, good, green, foo, bar, zed\}$

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

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

E None of the above

User's view of Semantics

1. Choose a task domain: intended interpretation.
2. Associate an atom with each proposition you want to represent.
3. Tell the system clauses that are true in the intended interpretation: axiomatizing the domain.
4. Ask questions about the intended interpretation.
5. If $KB \models g$, then g must be true in the intended interpretation.
6. Users can interpret the answer using their intended interpretation of the symbols.

Computer's view of semantics

- The computer doesn't have access to the intended interpretation.
- All it knows is the knowledge base.
- The computer can determine if a formula is a logical consequence of KB.
- If $KB \models g$ then g must be true in the intended interpretation.
- If $KB \not\models g$ then there is a model of KB in which g is false. This could be the intended interpretation.

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.

Aside: Gödel's incompleteness theorem

Gödel's incompleteness theorem [1930]:
No proof system for a sufficiently rich logic can be both sound and complete.
sufficiently rich = can represent arithmetic
Proof sketch:
Consider the statement "this statement cannot be proven".

- If it is true then system is incomplete.
- If it is false then system is unsound.
- The alternative is that statement cannot be represented.
- the state of a computer can be seen as a (big) integer, and all operations as arithmetic operations
- We can write a proof system that can represent that statement in a computer.

Bottom-up Proof Procedure for propositional definite clauses

One rule of derivation, a generalized form of *modus ponens*:

> *If "h :- $b_1, \ldots, b_m$" is a clause in the knowledge base, and each b_i has been derived, then h can be derived.*

This is forward chaining on this clause.
(An atomic fact is treated as a clause with empty body ($m = 0$).)

Bottom-up proof procedure

$KB \vdash g$ if $g \in C$ at the end of this procedure:

$C := \{\}$;
repeat
    select fact h or a rule "h :- $b_1, \ldots, b_m$" in KB such that
        $b_i \in C$ for all i, and
        $h \notin C$;
    $C := C \cup \{h\}$
until no more clauses can be selected.

Example

a :- b, c.

a :- e, f.

b :- f, k.

c :- e.

d :- k.

e.

f :- j, e.

f :- c.

j :- c.

Clicker Question

Consider the knowledge base KB:

$happy \;{:}{-}\; good.$	$foo \;{:}{-}\; bar,\; fun.$
$happy \;{:}{-}\; green.$	$bar \;{:}{-}\; zed.$
$green.$	$zed.$

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

- A $\{happy, good, green, foo, bar, fun, zed\}$
- B $\{happy, good, green, foo, bar, zed\}$
- C $\{happy, green, bar, zed\}$
- D $\{green, bar, zed\}$
- E None of the above

Soundness of bottom-up proof procedure

If $KB \vdash g$ then $KB \models g$.

- Suppose there is a g such that $KB \vdash g$ and $KB \not\models g$.
- Then there must be a first atom added to C that isn't true in every model of KB. Call it h.
  Suppose h isn't *true* in model I of KB.
- h was added to C, so there must be a clause in KB

  $$h \;\texttt{:-}\; b_1,\; \ldots,\; b_m$$

  where each b_i is in C, and so true in I.
  h is false in I (by assumption)
  So this clause is false in I.
  Therefore I isn't a model of KB.
- Contradiction. Therefore there cannot be such a g.

Fixed Point

- The C generated at the end of the bottom-up algorithm is called a fixed point.
- Let I be the interpretation in which every element of the fixed point is true and every other atom is false.
- Claim: I is a model of KB.
  Proof: suppose h :- $b_1, \ \ldots, \ b_m$ in KB is false in I.
  Then h is false and each b_i is true in I.
  Thus h can be added to C.
  Contradiction to C being the fixed point.
- I is called a Minimal Model.

Completeness

If $KB \models g$ then $KB \vdash g$.

- Suppose $KB \models g$. Then g is true in all models of KB.
- Thus g is true in the minimal model.
- Thus g is in the fixed point.
- Thus g is generated by the bottom up algorithm.
- Thus $KB \vdash g$.

Clicker Question

Suppose there at some atom *aaa* such that
$KB \vdash aaa$ and
$KB \not\models aaa$.
What can be inferred?

A The proof procedure is not sound

B The proof prodecure is not complete

C The proof procedure is sound and complete

D The proof procedure is either sound or complete

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:

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

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

$$a_1 \;\texttt{:-}\; b_1,\; \ldots,\; b_p$$

  is the answer clause

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

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

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 is obtained by resolving 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. $\quad$ a :- e, f. $\quad$ b :- f, k.
c :- e. $\quad$ d :- k. $\quad$ e.
f :- j, e. $\quad$ f :- c. $\quad$ 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.
a :- e, f.
b :- f, k.
c :- e.
d :- k.
e.
f :- j, e.
f :- c.
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