Example

- KB:
  female(X) \rightarrow person(X)

- Prove KB |= person(sally)
- What does KB |= person(sally) mean?
  Means that if interpretation I models KB then it models person(sally)
- Could prove this by checking all interpretations
  - Let's do proof instead

Review

- An interpretation maps any clause to either true or false
- A model I of KB is an interpretation that maps every clause in KB to true
- KB |= g iff every model of KB makes g true

Overview

⇒ Semantics
- Queries
  - Proof Procedures
    - Bottom-up Ground Proof Procedure
    - Top-down Ground Proof Procedure
Overview

• Semantics
  ⇒ Queries

• Proof Procedures

• Bottom-up Ground Proof Procedure
• Top-down Ground Proof Procedure

More on Variables in Clauses (p. 42)

• Say parent(X,Y) ← father(X,Y)
is in KB
  - Implicit universal quantifiers around it
  - For any X and Y, if we find a true X and Y, we can infer Z
  - For any X and Y, if we find a true parent(X,Y), then must be true
  - How does Z work here (variable just in the body)?
  - For any X and Y, if we find a Z that makes body true, head must be true
  - Now it seems that Z is just existentially quantified
  - We just need to find one Z for each X and Y, and then set it true for all
  - We first need to find one Z for each X, and then set it true for all Y
  - Example: Let I = {D, φ, π}
    - So <φ(sally)> ∈ π(female)
    - Say ρ(X) = s
      - If female(X) is true for I, then person(X) must be true for I ρ for any var. assignment
      - So ρ(X) = s
      - So <s> ∈ π(person)
      - Since φ(sally) = s, person(sally) is true under I
• Say grandfather(X,Z) ← father(X,Z)∧ parent(Z,Y)
is in KB
  - This clause is true for all X, Y, Z
  - ∀X Y Z (grandfather(X,Y) ← father(X,Z)∧ parent(Z,Y)).
  - Z only appears in the body
  - How does Z work here (variable just in the body)?
  - For any X and Y, if we find a Z that makes body true, head must be true
  - Now it seems that Z is just existentially quantified
  - We just need to find one Z for each X and Y, not ensure it is true for all Z
  - ∀XY (grandfather(X,Y) ← (∃Z father(X,Z)∧parent(Z,Y)).

A Semantic Proof

1. Since φ(sally) = s, person(sally) is true under I

2. Say <s> ∈ π(female)

3. Say <s> ∈ π(person)

4. Let D be a model of KB

CSE560 Class 03: 6 c
Overview

• Semantics
• Queries
⇒ Proof Procedures

• Bottom-up Ground Proof Procedure
• Top-down Ground Proof Procedure

Queries

• You might not only want to check if something is true or false, but what value makes it true

KB:
father(william,ted)

parent(X,Y) ← father(X,Y)

• Example:
?parent(X,ted)
- Who is Ted's parent?
- Could transform this to
yes ← parent(X,ted)
- But, lets capture the variables in the body:
yes(X) ← parent(X,ted)

• An answer is either
- instance of 'yes' that is a logical consequence of KB:
yes(william)
- or no if no instance is a logical consequence of KB

Ground Queries

• A query is a way to ask if a body is a logical consequence of the knowledge base: ?b₁ ∧ ... ∧ bₘ

• Ground query (no variables) has the answer
- "yes" if the body is a logical consequence of the KB
- "no" if the body is not a logical consequence of the KB

• We do not distinguish between it being false in all models or just some

• Cannot tell if query is false in the intended interpretation

• Can do query-answering by:
- Transform query b₁ ∧ ... ∧ bₘ into yes ← b₁ ∧ ... ∧ bₘ
- Add (temporarily) yes ← b₁ ∧ ... ∧ bₘ to KB
- Check if yes is a logical consequence of KB
- This lets us view queries as just finding consequences from a KB

Questions with Variables

This lets us view queries as just finding consequences from a KB:

- Check if yes is a logical consequence of KB:
- Add (temporarily) yes ← b₁ ∧ ... ∧ bₘ to KB
- Transform query b₁ ∧ ... ∧ bₘ into yes ← b₁ ∧ ... ∧ bₘ
- Check if yes is a logical consequence of KB

Ground Query (no variables) has the answer

A query is a way to ask if a body is a logical consequence of the KB.
Two Types of Proof Procedures

- Bottom-up
  - Forward-Chaining
  - KB ⊢ g means g can be derived from KB with the proof procedure
- Top-down
  - Backward-Chaining
  - Query

Proof Procedures

- Proof: a mechanically derivable demonstration that a formula logically follows from a KB
- Proof procedure: an algorithm that constructs proofs
- KB ⊢ g means g can be derived from KB with the proof procedure
- KB | = g means g is true in all models of KB
- Proof procedure can be nondeterministic

Properties of Proof Procedure

- Correctness: if KB | = g then g ∈ KB
- Soundness: if KB ⊢ g then KB | = g

Terminology:

- Semantic proof: | = , logically follows, logically entails, models
- Syntactic proof: ⊢ , derives

Semantics Is Not Enough

- We have KB
- But, don't yet have a mechanical way of checking if YF | = 6
- Can extract this so we can use facts with variables as well
- Can infer facts from KB, if YF | = 6
- KB | = 6 if 6 is true in all models of KB
- We know what conclusions are valid to make the KB

- Computer can only access the YF
- Can't, just check the user's intended information
- Checking all interpretations is very expensive

- Semantic proof = logically follows, logically entails, models
- Syntactic proof = derives

Soundness:

- if KB | = g then KB ⊢ g

Completeness:

- if KB ⊢ g then KB | = g

Proof Procedure: an algorithm that constructs proofs

- Proof a mechanically derivable demonstration that a formula

- KB ⊢ g means g can be derived from KB with the proof procedure

- KB | = g means g is true in all models of KB

- Correctness: if KB | = g then g ∈ KB

- Soundness: if KB ⊢ g then KB | = g

- Properties of Proof Procedure

- Proof Procedures

- Semantic Proof:
  - KB | = g
  - KB ⊢ g

- Syntactic Proof:
  - KB ⊢ g
  - KB | = g
Non-deterministic Specification

- When order should we pick clauses from KB to try?
- Haven't specified the exact order that things should be done in

Bottom-up Ground Proof Procedure

- For now, only consider ground facts and ground rules
- Bottom-up or forward chaining procedure: starts from KB and works towards query
- Forward chaining rule:
  \[ h ← b_1 ∧ ... ∧ b_m \]
  is a clause in the KB
  and each \( b_i \) has been derived
  \( h \) can be derived
- Forward chaining rule also works if \( h \) is a fact in KB
  \( m = 0 \)
  \( h \) can be derived
  \( h \) can be a clause in the KB

Consequence set (C)

Call the set of derivables the consequence set (C)

- None of derivables are connected
- Forward chaining rule also works if \( h \) is a fact in KB
  \( m = 0 \)
  \( h \) can be derived
  \( h \) can be a clause in the KB

Bottom-up Ground Proof Procedure

- Top-down Ground Proof Procedure
  = Bottom-up Ground Proof Procedure

- Procedures
- Queries
- Semantics

Overview
**Is it Complete?**

- Is an interpretation, but is also a model of KB.
- Hence, also a subset of KB.

**Is it Sound?**

- Everything in KB is a model of KB.
- Interpretations are a subset of KB.

**Example**

- a ← b ∧ c.
- b ← d ∧ e.
- b ← g ∧ e.
- c ← e.
- d.
- e.

What is the consequence set?
Overview

• Semantics
• Queries
• Proof Procedures
  ⇒ Bottom-up Ground Proof Procedure
  ⇒ Top-down Ground Proof Procedure

Final Step in Completeness Proof

• Let $g$ be atomic and $KB \models g$
  • Need to make sure that $KB \vdash g$
  • Since $KB \models g$, $g$ must be in every model of $KB$
  • So, it is in the interpretation defined by the Consequence set
  • Since $g$ is atomic and it is true in the interpretation,
  • it must be in consequence set
  • So $KB \vdash g$

Proof that Consequence Set is a Model

• Proof by Contradiction: Let $g \in KB$ but where $I(g)$ is false
  • Since $g \in KB$, $g$ must have the form $h \leftarrow b_1 \wedge \ldots \wedge b_m$
  • So $h \leftarrow b_1 \wedge \ldots \wedge b_m$ is false in $I$
  • Remember, definition of $\leftarrow$ comes from Datalog, not $I$
  • So all of the $b_i$ must be true in $I$
  • Since $I$ is a model, all of the $b_i$ must be true in $I$
  • Hence $g$ is true in $I$

Proof that Consequence Set is a Model

• Let $g \in C$ but where $I(g)$ is false
  • So $g \leftarrow b_1 \wedge \ldots \wedge b_m$ is false in $I$
  • Since $g \leftarrow b_1 \wedge \ldots \wedge b_m$ is false in $I$
  • Hence $g$ is false in $I$

⇒ Bottom-up Ground Proof Procedure
⇒ Top-down Ground Proof Procedure

Semantics

Queries

Proof Procedures

Overview
Example

\[ a \rightarrow f \]
\[ e \rightarrow c \]
\[ p \]
\[ c \rightarrow e \]
\[ q \rightarrow q \]
\[ \neg p \rightarrow q \]
\[ \neg q \rightarrow a \]

\( KB \)

Now for some definitions

- **Answer clause** is
  \[ \text{yes} \rightarrow a_1 \land \ldots \land a_m \]

- **Answer** is answer clause with \( m = 0 \)

- **Derivation** of a query \( \text{yes} \rightarrow q_1 \land \ldots \land q_k \) from \( KB \) is a sequence of
  
  \[ \gamma_0, \gamma_1, \ldots, \gamma_n \]
  
  - \( \gamma_0 \) is the answer clause corresponding to the original query
  - \( \gamma_i \) is obtained by resolving \( \gamma_{i-1} \) with a clause in \( KB \)
  - \( \gamma_n \) is the answer clause with \( m = 0 \)

- **Nondeterminism**
  
  - In choosing which clause from \( KB \) to resolve with
  
  - Can find all derivations by systematically considering all different choices (see Chapter 4)

**Top-Down Ground Proof Procedure**

- **Alternative to bottom-up (forward-chaining)**
- **Top-down (backward-chaining)**
  
  - Start with goal, work toward facts in \( KB \)

- **Definite Clause Resolution for Ground Case**

  \[
  \begin{align*}
  \text{yes} & \rightarrow a_1 \land \ldots \land a_m \\
  a_i & \rightarrow b_1 \land \ldots \land b_p \\
  \text{yes} & \rightarrow a_1 \land \ldots \land a_{i-1} \land b_1 \land \ldots \land b_p \land a_{i+1} \land \ldots \land a_m 
  \end{align*}
  \]
Bottom-up versus Top-down

- Both top-down and bottom-up are sufficient for datalog.
- We will explore some of these later in the course.
- E.g. Unification

There are many other ways of doing proofs:
- e.g. Unit resolution
- We will explore some of these later in the course.
- But top-down and bottom-up are sufficient for datalog.

---

**Bottom-Up**

1. \( a \) \( \in \) C
2. \( c \)
3. \( d \)
4. \( e \)
5. \( f \)
6. \( \{ a \} \) \( \in \) C
7. \( \{ e, c \} \) \( \in \) C
8. \( \{ e, c \} \) \( \in \) C
9. \( \{ e, c, d \} \) \( \in \) C
10. \( \{ e, c, d \} \) \( \in \) C
11. \( \{ e, c, d, b \} \) \( \in \) C
12. \( \{ e, c, d, b, a \} \) \( \in \) C

---

**Top-Down**

1. \( a \)
2. \( b \)
3. \( c \)
4. \( d \)
5. \( e \)
6. \( f \)
7. \( \{ a \} \)
8. \( \{ e, c \} \)
9. \( \{ e, c, d \} \)
10. \( \{ e, c, d, b \} \)
11. \( \{ e, c, d, b, a \} \)

---

This is a demonstration of how bottom-up and top-down proofs can be constructed for the same KB.