The story so far...

- Propositional Logic
  - Model Checking
  - Forward Chaining
  - Backwards Chaining
  - Resolution

- First-Order Logic
  - Richer language with objects, relations
  - We practiced writing sentences in FOL
  - Reduction of FOL to PL for inference
    - Why was this problematic?
Today

- FOL Inference without reducing to PL
  - Unification – an important building block
  - Forward Chaining
  - Backwards Chaining
  - Resolution

Administrative

- PS3 extension: until Thursday Feb 24

- Midterm Feedback
  - Thanks for your comments!
  - On challenge problems...
  - Have more feedback? I want to hear it!
Midterm 1 Results

<table>
<thead>
<tr>
<th>Count</th>
<th>Sample Bar Chart</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
</tr>
</tbody>
</table>

Class median: 64.0  
Class average: 63.3  
Class std dev: 9.2  
Num scores: 76

Reasoning within FOL

- In order to reason within FOL, we need to be able to deal with Universal Instantiation without expanding for every object.

- Consider:
  - S1. IsKing(x) ^ IsGreedy(x) => IsEvil(x)
  - S2. IsKing(John)
  - S3. IsGreedy(John)

- Do IsKing(John) and IsGreedy(John) match the implication?  
  - Can we determine this without plugging in every possible object into S1?
Unification

- Unification: The process of making two FOL sentences equivalent by substituting values for variables.

  - S1. IsKing(x) ^ IsGreedy(x) => IsEvil(x)
  - S2. IsKing(John)

- The first term of S1 and S2 can be unified with:
  - \{ x/ John \}

Unification

NextTo(r1,r2) \rightarrow\leftarrow NextTo(A,B)

\{ r1 / A, r2 / B \}
**Unification Examples**

- $\text{NT}(r_1, C) \leftarrow \text{NT}(B, r_3) \leftarrow \{ r_1 / B, r_3 / C \}$
- $\text{NT}(\text{RoomA}, n) \leftarrow \text{NT}(r, \text{Hallway}(\text{DoorOf}(r))) \leftarrow \{ r / \text{RoomA}, n / \text{Hallway}(\text{DoorOf}(\text{RoomA})) \}$
- $\text{NT}(\text{RoomA}, r_1) \leftarrow \text{NT}(\text{RoomB}, r_2)$
  - **FAIL**

**Most general unifier**

- $\text{NT}(r_1, B) \leftarrow \text{NT}(r_2, r_3) \leftarrow \{ r_3 / B, r_1 / A, r_2 / A \}$
  - $\{ r_3 / B, r_1 / r_2 \}$
  - **Most general unifier**
Unification Algorithm (sketch)

- \( \theta = \text{Unify}(x, y, \emptyset) \)
  - Return substitutions that cause \( x \) and \( y \) to match, given already known substitutions.

- Recursively examine two sentences \((x, y)\)
  - Base case:
    - If both sentences are ground terms, ensure that terms are equal or fail.
    - If one sentence is a variable, unify it with the other expression.
    - If sentences have multiple parts, recursively unify each of the parts.

Unification Algorithm (sketch)

- \( \text{Unify}(\text{King}(x), \text{King}(\text{John}), \{\}) \)
  
  Functions have two parts: function symbol and argument list

- \( \text{Unify} (\text{Unify} (\text{King}, \text{King}), \text{Unify} (x, \text{John}), \{\}) \)
  \[ \{\} \quad \{x / \text{John}\} \]

- Result: \( \{x / \text{John}\} \)
Unify (in detail)

- **Function UNIFY(x, y, θ)** returns a substitution to make x and y identical
  - Inputs: x, a variable, constant, list, or compound
  - y, a variable, constant, list, or compound
  - θ, the substitution built up so far
  - if θ = failure then return failure
  - else if x = y then return θ
  - else if VARIABLE?(x) then return UNIFY-VAR(x, y, θ)
  - else if VARIABLE?(y) then return UNIFY-VAR(y, x, θ)
  - else if COMPOUND?(x) and COMPOUND?(y) then
    return UNIFY(ARGS[x], ARGS[y], UNIFY(OP[x], OP[y], θ))
  - else if LIST?(x) and LIST?(y) then
    return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y], θ))
  - else return failure

- **Function UNIFY-VAR(var, x, θ)** returns a substitution
  - Inputs: var, a variable
  - x, any expression
  - θ, the substitution built up so far
  - if [var/x] ∈ θ then return UNIFY(var, x, θ)
  - else if [x/var] ∈ θ then return UNIFY(var, x, θ)
  - else if OCCUR-CHECK?(var, x) then return failure
  - else return add {var/x} to θ

Adapting Modus Ponens to FOL

- **PL Modus Ponens**
  \[
  \frac{\alpha, \alpha \Rightarrow \beta}{\beta}
  \]

- **FOL Modus Ponens**
  \[
  \text{King(John), King(x) } \Rightarrow \text{ Ruler(x)}
  \]
  \[
  \text{------------------------------------------}
  \]
  \[
  \text{Ruler(John)}
  \]

- Given that Subst(θ, a') = Subst(θ, a)
  \[
  \frac{\alpha', \alpha \Rightarrow \beta}{\text{Subst(θ, β)}}
  \]
Generalized Modus Ponens

- Given that $\text{Subst}(\theta, a') = \text{Subst}(\theta, a)$
  
  $\alpha', \alpha \Rightarrow \beta$
  
  $\text{Subst}(\theta, \beta)$

- Given that $\text{Subst}(\theta, a_i') = \text{Subst}(\theta, a_i)$ for all $i$
  
  $\alpha_1', \alpha_2', \ldots, \alpha_m', \alpha_1 \land \alpha_2 \land \cdots \land \alpha_m \Rightarrow \beta$
  
  $\text{Subst}(\theta, \beta)$

Your turn: Self-Shaver?

- There is a barber who shaves every man in town who does not shave himself, and nobody else.
  - Is this a paradox?

- Step 1. Translate English to FOL
- Step 2. Convert FOL to CNF
- Step 3. Perform unit resolution using unification.
  - If paradox is unresolvable, then you will arrive at a contradiction!
Practical FOL Inference

- Classic FOL language subsets
  - Datalog
  - Prolog

- Inference methods
  - Forward Chaining
  - Backwards Chaining
  - Resolution

Datalog

- First-order definite clauses
  - disjunction of literals, exactly one positive term
  - Why are these definite clauses?
    - King(x) ^ Greedy(x) => Evil(x)
    - King(John)
    - Greedy(y)

- No functions allowed

- These are like ____ clauses in PL

- Inference method?
  - Forward chaining
Forward Chaining

- Just like in PL, restrictions on sentence types allows simple inference

- Find rules that are “triggered” by known facts
  - PL: A \land B \Rightarrow X
  - FOL: King(x) \land Greedy(x) \Rightarrow Evil(x)
    - Use Unify() to match terms

- Keep matching/generating new facts until fixed point: we only derive facts we already know.

Example knowledge base

- The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

- Prove: Col. West is a criminal
Forward Chaining

```latex
function FOL-FC-Ask(KB, α) returns a substitution or false
repeat until new is empty
  new ← \{
  for each sentence r in KB do
    (p_1 \land \ldots \land p_n \Rightarrow q) ← \text{STANDARDIZE-AFORM}(r)
    for each θ such that (p_1 \land \ldots \land p_n)θ = (p'_1 \land \ldots \land p'_n)θ
      for some p'_1, \ldots, p'_n in KB
        q' ← \text{SUBST}(θ, q)
      if q' is not a renaming of a sentence already in KB or new then do
        add q' to new
        φ ← \text{UNIFY}(q', α)
      if φ is not fail then return φ
  add new to KB
return false
```

Example knowledge base contd.

... it is a crime for an American to sell weapons to hostile nations:

S1: American(x) \land Weapon(y) \land Sells(x,y,z) \land Hostile(z) \Rightarrow Criminal(x)

Nono has some missiles

\exists x \text{Owns}(Nono,x) \land \text{Missile}(x):

S2: \text{Owns}(Nono,M_1) \land \text{Missile}(M_1)

... all of its missiles were sold to it by Colonel West

S3: \text{Missile}(x) \land \text{Owns}(Nono,x) \Rightarrow \text{Sells}(West,x,Nono)

Missiles are weapons:

S4: \text{Missile}(x) \Rightarrow \text{Weapon}(x)

An enemy of America counts as “hostile”:

S5: Enemy(x,America) \Rightarrow Hostile(x)

West, who is American ...

S6: American(West)

The country Nono, an enemy of America ...

S7: Enemy(Nono,America)
Forward chaining proof

American(\textit{West}) \quad S6

Missile(M1) \quad S2

Own(\textit{None}, M1) \quad S2

Enemy(\textit{None}, \textit{American}) \quad S7

Forward chaining proof

Weapon(M1) \quad S4

Self(\textit{West}, M1, \textit{None}) \quad S3

Heat(\textit{None}) \quad S5

American(\textit{West}) \quad S6

Missile(M1) \quad S2

Own(\textit{None}, M1) \quad S2

Enemy(\textit{None}, \textit{American}) \quad S7
Forward chaining proof

\[ \begin{align*}
S1 & \quad \text{Criminal(West)} \\
S2 & \quad \text{Sell(West, ML, None)} \\
S3 & \quad \text{Sellers(ML, None)} \\
S4 & \quad \text{Weapon(M1)} \\
S5 & \quad \text{Heavily(None)} \\
S6 & \quad \text{American(West)} \\
S7 & \quad \text{Enemy(None, America)}
\end{align*} \]

Forward Chaining

- **Performance:**
  - Worst case: only learn one thing per iteration
  - \( p = \) # of predicates ("King")
  - \( n = \) # of ground symbols ("John")
  - \( k = \) maximum arity (# of arguments of predicate)

  \[ \text{think: How many facts could we learn from Siblings(x, y)} \]
  - \( pn^k \)

- Infinite domains (i.e., if KB includes Peano axioms)?
  - Herbrand’s theorem to the rescue again.
Forward Chaining: Practical Issues

- Described approach spends lots of time trying to match premises of implications.

- Incremental forward chaining: no need to match a rule on iteration $k$ if a premise wasn't added on iteration $k-1$
  - match each rule whose premise contains a newly added positive literal

- Database indexing allows $O(1)$ retrieval of known facts
  - e.g., query $Missile(x)$ retrieves $Missile(M_1)$
  - What data structure do we use for the index?

- Suppose we learn $Sells(West, M_1, Nono)$. How should we index it?

Backwards Chaining

- Most widely used form of automated reasoning

- Similar to PL version
  - Depth first search
    - What kind of problems do you anticipate?
Backward chaining algorithm

```latex
function FOL-BC-Ask(KB, goals, \theta) returns a set of substitutions
inputs: KB, a knowledge base
goals, a list of conjuncts forming a query
\theta, the current substitution, initially the empty substitution \{ \}
local variables: ans, a set of substitutions, initially empty
if goals is empty then return \{\theta\}
q' \leftarrow \text{subst}(\theta, \text{first}(goals))
for each \( r \) in KB where \text{standardize-}\text{-}\text{apart}(r) = (p_1 \land \ldots \land p_n \Rightarrow q)
and \( \theta' \leftarrow \text{unify}(q, q') \) succeeds
\text{ans} \leftarrow \text{FOL-BC-Ask}(\text{KB}, [p_1, \ldots, p_n, \text{rest}(goals)], \text{compose}(\theta, \theta')) \cup \text{ans}
return \text{ans}
```

\[
\text{subst}((\theta_2 \circ \theta_1), p) = \text{subst}(\theta_2, \text{subst}(\theta_1, p))
\]

Backward chaining example

\[\text{Granular Wenzl}\]
Backward chaining example

1. \( \text{Criminal \ Wear} \)
2. \( S1 \) (to\text{\cite{Wear}})
   - \( \text{American\text{\cite{upgrade}}} \)
   - \( \text{Weapon\text{\cite{upgrade}}} \)
   - \( \text{Sell\text{\cite{upgrade},2}} \)
   - \( \text{Hostile\text{\cite{upgrade}}} \)

Backward chaining example

1. \( \text{Criminal \ Wear} \)
2. \( S1 \) (to\text{\cite{Wear}})
   - \( S6 \)
     - \( \text{American\text{\cite{upgrade}}} \)
     - \( \text{Weapon\text{\cite{upgrade}}} \)
     - \( \text{Sell\text{\cite{upgrade},2}} \)
     - \( \text{Hostile\text{\cite{upgrade}}} \)
Backward chaining example

1. \( S_1 \) \( \{ x \text{Wear} \} \)
2. \( S_4 \)
3. \( S_6 \) \( \{ \} \)
4. \( S_4 \)
5. \( S_2 \)
Backward chaining example

\[ \text{S1} \{x/West, y/M1, z/None}\]

\[ \text{S6} \{\text{American/West}\} \]
\[ \text{S4} \{\text{Weapon}()\} \]
\[ \text{S2} \{\text{Messile}()\} \]
\[ \text{S3} \{\text{Sell it, M1, z/None}\} \]
\[ \text{S5} \{\text{Messile/None}\} \]

Backward chaining example

\[ \text{S1} \{x/West, y/M1, z/None\} \]

\[ \text{S6} \{\text{American/West}\} \]
\[ \text{S4} \{\text{Weapon}()\} \]
\[ \text{S2} \{\text{Messile}()\} \]
\[ \text{S3} \{\text{Sell it, M1, z/None}\} \]
\[ \text{S5} \{\text{Messile/None}\} \]

M. Wellman & E. Olson
Properties of backward chaining

- Depth-first recursive proof search: space is linear in size of proof
- Incomplete due to infinite loops
  - fix by checking current goal against every goal on stack
- Inefficient due to repeated subgoals (both success and failure)
  - fix using caching of previous results (extra space)
  - “memoization”
- Widely used for logic programming

Prolog

- Datalog + functions
- Specific, widely-used syntax
  - Variables are UPPERCASE
  - Literals are lowercase.

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
```

- Depth-first, left-to-right backwards chaining
Prolog: Failure Modes

path(X,Z) :- link(X,Z).
path(X,Z) :- path(X,Y), link(Y,Z)

Consider a path through three nodes:

\[
\text{a --- b --- c}
\]

Link(a,b)
Link(b,c)

Query: path(a,c)?
Prolog: Failure Mode

path(X,Z) :- path(X,Y), link(Y,Z)  (Order Switched)
path(X,Z) :- link(X,Z).

Resolution

- As with PL, we can use Resolution in FOL
- No restrictions on clauses (don’t need to be definite clauses)
- Sound
- Complete? (We’ll come back to this)
Standardize Variables

- Everything is great or something is wrong.
  \[ \forall x. \text{Great}(x) \lor \exists x. \text{Wrong}(x) \]

We saw that, with unification, all occurrences of a variable need the same binding. And in substitution, we must make the same substitution for all occurrences of a variable.

\[ \forall x. \text{Great}(x) \lor \exists y. \text{Wrong}(y) \]

---

Standardize Variables Example

1. A father of someone isn't a woman.
   \[ \forall x,y. \neg \text{Father}(x,y) \lor \neg \text{Woman}(x) \]
2. A mother of someone is a woman.
   \[ \forall x,y. \neg \text{Mother}(x,y) \lor \text{Woman}(x) \]
3. Mary is the mother of Chris.
   \[ \text{Mother}(\text{Mary}, \text{Chris}) \]

Resolve 1&2, \{x / x\}
4. \[ \forall x,y. \neg \text{Father}(x,y) \lor \neg \text{Mother}(x,y) \]
Resolve 4&3, \{x / \text{Mary}, y / \text{Chris}\}
5. \[ \neg \text{Father}(\text{Mary}, \text{Chris}) \]

OK, but why didn’t we conclude that Mary wasn’t anyone’s father?
Standardize Variables Example

1. A father of someone isn’t a woman.
\[ \forall x_1, y_1. \neg \text{Father}(x_1, y_1) \lor \neg \text{Woman}(x_1) \]
2. A mother of someone is a woman.
\[ \forall x_2, y_2. \neg \text{Mother}(x_2, y_2) \lor \text{Woman}(x_2) \]
3. Mary is the mother of Chris.
\[ \text{Mother}(\text{Mary}, \text{Chris}) \]

Resolve 1&2, \{x_1 / x_2\}
4. \[ \forall x_2, y_1, y_2. \neg \text{Father}(x_2, y_1) \lor \neg \text{Mother}(x_2, y_2) \]
Resolve 4&3, \{x_2 / \text{Mary}, y_2 / \text{Chris}\}
5. \[ \forall y_1. \neg \text{Father}(\text{Mary}, y_1) \]

Better. But to ensure standardized variables, we’d even want:
4. \[ \forall x_4, y_4, z_4. \neg \text{Father}(x_4, y_4) \lor \neg \text{Mother}(x_4, z_4) \]
5. \[ \forall y_5. \neg \text{Father}(\text{Mary}, y_5) \]

Conversion to Conjunctive Normal Form

- Translate bidirectionals to implications
- Translate implications to disjunctions.
- Move negations inward
  - Use De Morgan’s laws
  - until only atoms are negated
- Standardize variables.
- Eliminate existentials via \textit{skolemization}.
- Drop universal quantifiers.
- Distribute and associate into CNF.
**Skolemization (revisited)**

\[ \exists c. \text{IsMother}(\text{Jen}, c) \xrightarrow{\text{skolem}} \text{IsMother}(\text{Jen}, \text{JensKid}) \]

\[ \forall c \exists p. \text{IsParent}(p, c) \xrightarrow{\text{skolem}} \forall c \text{IsParent}(\text{Pof}(c), c) \]

\[ \forall g, c. \text{IsGP}(g, c) \Rightarrow \exists p. \text{IsP}(g, p) \land \text{IsP}(p, c) \xrightarrow{\text{skolem}} \]

\[ \forall g, c. \text{IsGP}(g, c) \Rightarrow \text{IsP}(g, \text{Sk3}(g, c)) \land \text{IsP}(\text{Sk3}(g, c), c) \]

Why, in the last example, was the same Skolem function used? Why does it take both \( g \) and \( c \) as arguments?

---

**Resolution Example**

- Everyone who loves all animals is loved by someone
  \[ \forall x. [\forall y. \text{Animal}(y) \Rightarrow \text{Loves}(x, y)] \Rightarrow \exists y. \text{Loves}(y, x) \]

- Anyone who kills an animal is loved by no one.
  \[ \forall x. [\exists y. \text{Animal}(y) \land \text{Kills}(x, y)] \Rightarrow [\forall z. \neg \text{Loves}(z, x)] \]

- Jack loves all animals.
  \[ \forall x. \text{Animal}(x) \Rightarrow \text{Loves}(\text{Jack}, x) \]

- Either Jack or Curiosity killed the cat, who is named Tuna.
  \[ \text{Kills}(\text{Jack}, \text{Tuna}) \lor \text{Kills}(\text{Curiosity}, \text{Tuna}) \]
  \[ \forall x. \text{Cat}(x) \Rightarrow \text{Animal}(x) \quad \text{(background knowledge)} \]

- Did Curiosity kill the cat?
  \[ \neg \text{Kills}(\text{Curiosity}, \text{Tuna}) \]
Conversion to CNF

- Everyone who loves all animals is loved by someone
  \[ \forall x \left[ \forall y \text{Animal}(y) \Rightarrow \text{Loves}(x,y) \right] \Rightarrow \exists y. \text{Loves}(y,x) \]

Resolution Example

A1. \text{Animal}(F(x)) \lor \text{Loves}(G(x),x)
A2. \neg\text{Loves}(x,F(x)) \lor \text{Loves}(G(x),x)
B. \neg\text{Animal}(y) \lor \neg\text{Kills}(x,y) \lor \neg\text{Loves}(z,x)
C. \neg\text{Animal}(x) \lor \text{Loves}(\text{Jack},x)
D. \text{Kills}(\text{Jack}, \text{Tuna}) \lor \text{Kills}(\text{Curiosity}, \text{Tuna})
E. \text{Cat}(\text{Tuna})
F. \neg\text{Cat}(x) \lor \text{Animal}(x)
G. \neg\text{Kills}(\text{Curiosity}, \text{Tuna})
Resolution

Non-constructive Proofs

- Suppose we asked “Who killed the cat?”
  - There exists a ‘w’ such that Kills(w, Tuna)
  - Query: -Kills(w, Tuna)

- Kills(w,Tuna) , Kills(Jack, Tuna) v Kills(Curiosity, Tuna) => Kills(Jack, Tuna) {w / Curiosity}
- Kills(Jack, Tuna), -Kills(w, Tuna) => () {w / Jack}

- Resolution tells us our query is true: There does exist a w that killed Tuna. (But what was w?)

- Note that the proof assigned multiple values to w: we can detect this and reject those proofs!
Gottfried Wilhelm Leibnitz (1646-1716)... again!

... if we could find characters or signs appropriate for expressing all our thoughts as definitely and as exactly as arithmetic expresses numbers..., we could in all subjects in so far as they are amenable to reasoning accomplish what is done in Arithmetic... For all inquiries... would be performed by the transposition of characters and by a kind of calculus, which would immediately facilitate the discovery of beautiful results...

—Dissertio de Arte Combinatoria, 1666

Incompleteness Theorem

- Godel:
  - In any consistent KB involving an inductive schema, there are true sentences that cannot be proved.
  - Arithmetic defined in terms of inductive schema
    - S(0), S(S(0)), S(S(S(0))), ...
    - Godel's theorem applies

- Bad news for Leibnitz!
  - Can't resolve every argument via inference.

- Practical limitation?
  - Hasn’t stopped theorem provers from proving many open problems!
Next Time

- Done with Logic...
- Planning!