CS 561: Artificial Intelligence

Instructor: Sofus A. Macskassy, macskass@usc.edu
TAs: Nadeesha Ranasinghe (nadeeshr@usc.edu)
     William Yeoh (wyeoh@usc.edu)
     Harris Chiu (chiciu@usc.edu)

Lectures: MW 5:00-6:20pm, OHE 122 / DEN
Office hours: By appointment
Class page: http://www.rcf.usc.edu/~macskass/CS561-Spring2010/

This class will use http://www.uscden.net/ and class webpage
- Up to date information
- Lecture notes
- Relevant dates, links, etc.

Course material:
Inference in FOL [AIMA Ch. 9]

- Reducing first-order inference to propositional inference
- Unification
- Generalized Modus Ponens
- Forward and backward chaining
- Logic programming
- Resolution
# A brief history of reasoning

<table>
<thead>
<tr>
<th>Year</th>
<th>Person</th>
<th>Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>450b.c.</td>
<td>Stoics</td>
<td>propositional logic, inference (maybe)</td>
</tr>
<tr>
<td>322b.c.</td>
<td>Aristotle</td>
<td>“syllogisms” (inference rules), quantifiers</td>
</tr>
<tr>
<td>1565</td>
<td>Cardano</td>
<td>probability theory (propositional logic+ uncertainty)</td>
</tr>
<tr>
<td>1847</td>
<td>Boole</td>
<td>propositional logic (again)</td>
</tr>
<tr>
<td>1879</td>
<td>Frege</td>
<td>first-order logic</td>
</tr>
<tr>
<td>1922</td>
<td>Wittgenstein</td>
<td>proof by truth tables</td>
</tr>
<tr>
<td>1930</td>
<td>Gödel</td>
<td>∃ complete algorithm for FOL</td>
</tr>
<tr>
<td>1930</td>
<td>Herbrand</td>
<td>complete algorithm for FOL (reduce to propositional)</td>
</tr>
<tr>
<td>1931</td>
<td>Gödel</td>
<td>¬∃ complete algorithm for arithmetic</td>
</tr>
<tr>
<td>1960</td>
<td>Davis/Putnam</td>
<td>“practical” algorithm for propositional logic</td>
</tr>
<tr>
<td>1965</td>
<td>Robinson</td>
<td>“practical” algorithm for FOL—resolution</td>
</tr>
</tbody>
</table>
Universal instantiation (UI)

- Every instantiation of a universally quantified sentence is entailed by it:

  \[ \forall v \alpha \quad \text{SUBST}(\{v/g\}, \alpha) \]

- for any variable \(v\) and ground term \(g\)

- E.g., \(\forall x \ King(x) \land Greedy(x) \Rightarrow Evil(x)\) yields

  \[
  \begin{align*}
  & King(John) \land Greedy(John) \Rightarrow Evil(John) \\
  & King(Richard) \land Greedy(Richard) \Rightarrow Evil(Richard) \\
  & King(Father(John)) \land Greedy(Father(John)) \Rightarrow Evil(Father(John)) \\
  & \ldots
  \end{align*}
  \]
Existential instantiation (EI)

- For any sentence $\alpha$, variable $v$, and constant symbol $k$ that does not appear elsewhere in the knowledge base:

$$\exists v \alpha
\underline{\text{SUBST}(\{v/k\}, \alpha)}$$

- E.g., $\exists x \text{Crown}(x) \land \text{OnHead}(x, John)$ yields $\text{Crown}(C_1) \land \text{OnHead}(C_1, John)$ provided $C_1$ is a new constant symbol, called a **Skolem constant**

- Another example: from $\exists x \frac{d(xy)}{dy}=xy$ we obtain $\frac{d(e^y)}{dy}=e^y$ provided $e$ is a new constant symbol
Existential instantiation contd.

- UI can be applied several times to add new sentences; the new KB is logically equivalent to the old.

- EI can be applied once to replace the existential sentence; the new KB is not equivalent to the old, but is satisfiable iff the old KB was satisfiable.
Reduction to propositional inference

Suppose the KB contains just the following:
\[
\forall x \text{King}(x) \land \text{Greedy}(x) \Rightarrow \text{Evil}(x)
\]
\[
\text{King}(\text{John})
\]
\[
\text{Greedy}(\text{John})
\]
\[
\text{Brother(} \text{Richard, John} \text{)}
\]

Instantiating the universal sentence in all possible ways, we have
\[
\text{King}(\text{John}) \land \text{Greedy}(\text{John}) \Rightarrow \text{Evil(} \text{John} \text{)}
\]
\[
\text{King(} \text{Richard} \text{)} \land \text{Greedy(} \text{Richard} \text{)} \Rightarrow \text{Evil(} \text{Richard} \text{)}
\]
\[
\text{King(} \text{John} \text{)}
\]
\[
\text{Greedy(} \text{John} \text{)}
\]
\[
\text{Brother(} \text{Richard, John} \text{)}
\]

The new KB is propositionalized: proposition symbols are
\[
\text{King(} \text{John} \text{)}, \text{Greedy(} \text{John} \text{)}, \text{Evil(} \text{John} \text{)}, \text{King(} \text{Richard} \text{)} \text{ etc.}
\]
Reduction contd.

- Claim: a ground sentence is entailed by new KB iff entailed by original KB
- Claim: every FOL KB can be propositionalized so as to preserve entailment
- Idea: propositionalize KB and query, apply resolution, return result
- Problem: with function symbols, there are infinitely many ground terms,
  e.g., $\text{Father(Father(Father(John)))}$
Theorem: Herbrand (1930). If a sentence $\alpha$ is entailed by an FOL KB, it is entailed by a \textit{finite} subset of the propositional KB

Idea: For $n = 0$ to $\infty$ do

create a propositional KB by instantiating with depth-$n$ terms and see if $\alpha$ is entailed by KB

Problem: works if $\alpha$ is entailed, loops if $\alpha$ is not entailed

Theorem: Turing (1936), Church (1936), entailment in FOL is \textit{semidecidable}
Problems with propositionalization

- Propositionalization seems to generate lots of irrelevant sentences.
- E.g., from
  \[
  \forall x \ King(x) \land Greedy(x) \Rightarrow Evil(x) \\
  King(John) \\
  \forall y \ Greedy(y) \\
  Brother(Richard, John)
  \]
- it seems obvious that *Evil(John)*, but propositionalization produces lots of facts such as *Greedy(Richard)* that are irrelevant
- With \(p\) \(k\)-ary predicates and \(n\) constants, there are \(p \cdot n^k\) instantiations
- With function symbols, it gets much much worse!
Unification

- We can get the inference immediately if we can find a substitution \( \theta \) such that \( \text{King}(x) \) and \( \text{Greedy}(x) \) match \( \text{King}(\text{John}) \) and \( \text{Greedy}(\text{y}) \)
- \( \theta = \{x/\text{John}, y/\text{John}\} \) works
- \( \text{UNIFY}(\alpha, \beta) = \theta \) if \( \alpha \theta = \beta \theta \)

<table>
<thead>
<tr>
<th>( p )</th>
<th>( q )</th>
<th>( \theta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Knows}(\text{John}, x) )</td>
<td>( \text{Knows}(\text{John}, \text{Jane}) )</td>
<td>( {x/\text{Jane}} )</td>
</tr>
<tr>
<td>( \text{Knows}(\text{John}, x) )</td>
<td>( \text{Knows}(y,\text{OJ}) )</td>
<td>( {x/\text{OJ}, y/\text{John}} )</td>
</tr>
<tr>
<td>( \text{Knows}(\text{John}, x) )</td>
<td>( \text{Knows}(y,\text{Mother}(y)) )</td>
<td>( {y/\text{John}, x/\text{Mother}(\text{John})} )</td>
</tr>
<tr>
<td>( \text{Knows}(\text{John}, x) )</td>
<td>( \text{Knows}(x,\text{OJ}) )</td>
<td>fail</td>
</tr>
</tbody>
</table>

Standardizing apart eliminates overlap of variables, e.g., \( \text{Knows}(z_{17}, \text{OJ}) \)
The unification algorithm

function \textsc{Unify}(x, y, \theta) \textbf{returns} a substitution to make \( x \) and \( y \) identical

\hspace{1em} \textbf{inputs:} \( x \), a variable, constant, list, or compound
\hspace{1em} \( y \), a variable, constant, list, or compound
\hspace{1em} \( \theta \), the substitution built up so far

\hspace{1em} \textbf{if} \ \theta = \text{failure} \ \textbf{then} \ \textbf{return} \ \text{failure}
\hspace{1em} \textbf{else if} \ x = y \ \textbf{then} \ \textbf{return} \ \theta
\hspace{1em} \textbf{else if} \ \textsc{variable}(x) \ \textbf{then} \ \textbf{return} \ \textsc{unify-var}(x, y, \theta)
\hspace{1em} \textbf{else if} \ \textsc{variable}(y) \ \textbf{then} \ \textbf{return} \ \textsc{unify-var}(y, x, \theta)
\hspace{1em} \textbf{else if} \ \textsc{compound}(x) \ \textbf{and} \ \textsc{compound}(y) \ \textbf{then}
\hspace{2em} \textbf{return} \ \textsc{unify}([\text{args}[x], \text{args}[y], \text{unify}([\text{op}[x], \text{op}[y], \theta)])
\hspace{1em} \textbf{else if} \ \textsc{list}(x) \ \textbf{and} \ \textsc{list}(y) \ \textbf{then}
\hspace{2em} \textbf{return} \ \textsc{unify}([\text{rest}[x], \text{rest}[y], \text{unify}([\text{first}[x], \text{first}[y], \theta)])
\hspace{1em} \textbf{else} \ \textbf{return} \ \text{failure}
The unification algorithm

function \textsc{Unify-Var}(\textit{var}, x, \theta) returns a substitution
inputs: \textit{var}, a variable
         \textit{x}, any expression
         \theta, the substitution built up so far

if \{\textit{var} / \textit{val}\} \in \theta then return \textsc{Unify}(\textit{val}, x, \theta)
else if \{\textit{x} / \textit{val}\} \in \theta then return \textsc{Unify}(\textit{var}, \textit{val}, \theta)
else if \textsc{Occur-Check?}(\textit{var}, x) then return failure
else return add \{\textit{var} / x\} to \theta
Generalized Modus Ponens (GMP)

\[ p_1', p_2', \ldots, p_n', (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q) \]

- where \( p_i'\theta = p_i\theta \) for all \( i \)
- \( p_1' \) is \( \text{King}(\text{John}) \)
- \( p_2' \) is \( \text{Greedy}(y) \)
- \( \theta \) is \( \{x/\text{John}, y/\text{John}\} \)
- \( q\theta \) is \( \text{Evil}(\text{John}) \)

GMP used with KB of definite clauses (exactly one positive literal)

All variables assumed universally quantified
Soundness of GMP

Need to show that

\[ p_1', p_2', \ldots, p_n', (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q) \models q\theta \]

provided that \( p_i'\theta = p_i\theta \) for all \( i \)

Lemma: For any definite clause \( p \), we have \( p \models p\theta \) by UI

1. \((p_1 \land p_2 \land \ldots \land p_n \Rightarrow q) \models (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q)\theta = (p_1\theta \land p_2\theta \land \ldots \land p_n\theta \Rightarrow q\theta)\)

2. \( p_1', p_2', \ldots, p_n' \models p_1' \land p_2' \land \ldots \land p_n' \models p_1'\theta \land p_2'\theta \land \ldots \land p_n'\theta \)

3. From 1 and 2, \( q \) follows by ordinary Modus Ponens
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 that Col. West is a criminal.
Example knowledge base contd.

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

\[ \text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x,y,z) \land \text{Hostile}(z) \Rightarrow \text{Criminal}(x) \]

Nono ... has some missiles, i.e., \( \exists x \, \text{Owns}(\text{Nono},x) \land \text{Missile}(x) : \)

\[ \text{Owns}(\text{Nono},M1) \text{ and } \text{Missile}(M1) \]

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

\[ \forall x \, \text{Missile}(x) \land \text{Owns}(\text{Nono},x) \Rightarrow \text{Sells}(\text{West},x,\text{Nono}) \]

Missiles are weapons:

\[ \text{Missile}(x) \Rightarrow \text{Weapon}(x) \]

An enemy of America counts as “hostile”:

\[ \text{Enemy}(x,\text{America}) \Rightarrow \text{Hostile}(x) \]

West, who is American ...

\[ \text{American}(\text{west}) \]

The country Non, an enemy of America

\[ \text{Enemy}(\text{Nono},\text{America}) \]
Forward chaining algorithm

function FOL-FC-Ask(KB, α) returns a substitution or false

repeat until new is empty
    new ← {}
    for each sentence r in KB do
        (p₁ ∧ ... ∧ pₙ ⇒ q) ← STANDARDIZE-APART(r)
        for each θ such that (p₁ ∧ ... ∧ pₙ)θ = (p₁' ∧ ... ∧ pₙ')θ
            for some p₁', ..., pₙ' in KB
                q' ← SUBST(θ, q)
                if q' is not a renaming of a sentence already in KB or new then do
                    add q' to new
                    φ ← UNIFY(q', α)
                    if φ is not fail then return φ
                add new to KB
    return false
Forward chaining proof
Forward chaining proof

Diagram:

- **Weapon(M1)**
- **Sells(West,M1,Nono)**
- **Hostile(Nono)**

- **American(West)**
- **Missile(M1)**
- **Owns(Nono,M1)**

- **Enemy(Nono,America)**
Forward chaining proof

- Criminal(West)
  - Weapon(M1)
  - Sells(West,M1,Nono)
  - Hostile(Nono)
    - Enemy(Nono,America)
  - American(West)
  - Missile(M1)
  - Owns(Nono,M1)
Properties of forward chaining

- Sound and complete for first-order definite clauses (proof similar to propositional proof)
- Datalog = first-order definite clauses + no functions (e.g., crime KB)
- FC terminates for Datalog in poly iterations: at most $p \cdot n^k$ literals
- May not terminate in general if $\alpha$ is not entailed
- This is unavoidable: entailment with definite clauses is semidecidable
Efficiency of forward chaining

- Simple observation: no need to match a rule on iteration $k$ if a premise wasn't added on iteration $k-1$
  \[ \Rightarrow \text{match each rule whose premise contains a newly added literal} \]

- Matching itself can be expensive

- **Database indexing** allows $O(1)$ retrieval of known facts
  e.g., query $\text{Missile}(x)$ retrieves $\text{Missile}(M_1)$

- Matching conjunctive premises against known facts is NP-hard

- Forward chaining is widely used in **deductive databases**
**Hard matching example**

- **Colorable()** is inferred iff the CSP has a solution
- CSPs include 3SAT as a special case, hence matching is NP-hard
Backward chaining algorithm

function FOL-BC-Ask(KB, goals, θ) returns a set of substitutions

inputs: KB, a knowledge base
goals, a list of conjuncts forming a query (θ already applied)
θ, the current substitution, initially the empty substitution {

local variables: answers, a set of substitutions, initially empty

if goals is empty then return {θ}
q' ← SUBST(θ, FIRST(goals))
for each sentence r in KB
    where STANDARDIZE-APART(r) = ( p_1 \land \ldots \land p_n \Rightarrow q
    and θ' ← UNIFY(q, q') succeeds
    new_goals ← [ p_1, \ldots, p_n | REST(goals)]
    answers ← FOL-BC-Ask(KB, new_goals, COMPOSE(θ', θ)) \cup answers
return answers
Backward chaining example

Criminal(West)
Backward chaining example

```
Criminal(West)

American(x)  Weapon(y)  Sells(x,y,z)  Hostile(z)
```

{x/West}
Backward chaining example

\[
\text{Criminal}(\text{West}) \quad \{x/\text{West}\}
\]

- \text{American}(\text{West})
- \text{Weapon}(y)
- \text{Sells}(x,y,z)
- \text{Hostile}(z)
Backward chaining example

```
Criminal(West)

American(West)

Weapon(y)

Sells(x,y,z)

Hostile(z)

Missile(y)

{x/West}
```

Backward chaining example

Backward chaining example

```
American(West)

Weapon(y)

Sells(x, y, z)

Hostile(z)

Criminal(West)

{ x/West, y/M1 }

Missile(y)

{ y/M1 }
```
Backward chaining example

Criminal(West)

American(West) \{ \}

Weapon(y)

Sells(West, M1, z) \{ z/Nono \}

Missile(y) \{ y/M1 \}

Missile(M1)

Owns(Nono, M1)

Hostile(z)
Backward chaining example

```
American(West)  Weapon(y)  Sells(West,M1,z)  Hostile(Nono)
  {}             {}                  {}                  {}                  
Missile(y)  Missile(M1)  Owns(Nono,M1)  Enemy(Nono,America)
  {y/M1}        {}                    {}                    {}                  

Criminal(West)  {x/West, y/M1, z/Nono}
```
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!)

Widely used (without improvements!) for logic programming
## Logic programming

- **Sound bite:** computation as inference on logical KBs

<table>
<thead>
<tr>
<th>Logic programming</th>
<th>Ordinary programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Identify problem</td>
<td>Identify problem</td>
</tr>
<tr>
<td>2. Assemble information</td>
<td>Assemble information</td>
</tr>
<tr>
<td>3. Tea break</td>
<td>Figure out solution</td>
</tr>
<tr>
<td>4. Encode information in KB</td>
<td>Program solution</td>
</tr>
<tr>
<td>5. Encode problem as facts</td>
<td>Encode problem as data</td>
</tr>
<tr>
<td>6. Ask queries</td>
<td>Apply program to data</td>
</tr>
<tr>
<td>7. Find false facts</td>
<td>Debug procedural errors</td>
</tr>
</tbody>
</table>

Should be easier to debug `Capital(NewYork,US)` than $x := x + 2!$
Prolog systems

Basis: backward chaining with Horn clauses + bells & whistles
Widely used in Europe, Japan (basis of 5th Generation project)
Compilation techniques ⇒ approaching a billion LIPS

Program = set of clauses = head :- literal₁, … literalₙ.
   
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).

• Efficient unification by open coding
• Efficient retrieval of matching clauses by direct linking
• Depth-first, left-to-right backward chaining
• Built-in predicates for arithmetic etc., e.g., X is Y*Z+3
• Closed-world assumption ("negation as failure")
  e.g., given alive(X) :- not dead(X).
  alive(joe) succeeds if dead(joe) fails
Prolog examples

Depth-first search from a start state X:
\[ \text{dfs}(X) :\text{goal}(X). \]
\[ \text{dfs}(X) :\text{successor}(X,S),\text{dfs}(S). \]
No need to loop over S: \text{successor} succeeds for each

Appending two lists to produce a third:
\[ \text{append}([],Y,Y). \]
\[ \text{append}([X|L],Y,[X|Z]) :\text{append}(L,Y,Z). \]
query: \[ \text{append}(A,B,[1,2]) ? \]
answers: \[ A=[] \quad B=[1,2] \]
\[ A=[1,2] \quad B=[] \]
Resolution: brief summary

Full first-order version:

\[
\frac{l_1 \lor \ldots \lor l_k, m_1 \lor \ldots \lor m_n}{(l_1 \lor \ldots l_{i-1} \lor l_{i+1} \lor \ldots \lor l_k \lor m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n) \theta}
\]

where UNIFY\( (l_i, \neg m_j) = \theta \).

For example,

\[
\frac{\neg Rich(x) \lor Unhappy(x)}{Rich(Ken)} \quad \frac{}{Unhappy(Ken)}
\]

with \( \theta = \{x/Ken\} \)

Apply resolution steps to \( CNF(KB \land \neg \alpha) \); complete for FOL
Conversion to CNF

Everyone who loves all animals is loved by someone:
\[ \forall x \ [\forall y \ Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y \ Loves(y, x)] \]

1. Eliminate biconditionals and implications

\[ \forall x \ [\neg \forall y \ \neg Animal(y) \vee Loves(x, y)] \vee [\exists y \ Loves(y, x)] \]

2. Move \( \neg \) inwards:

\[ \neg \forall x, p \equiv \exists x \ \neg p, \ \neg \exists x, p \equiv \forall x \ \neg p: \]

\[ \forall x \ [\exists y \ \neg (\neg Animal(y) \vee Loves(x, y))] \vee [\exists y \ Loves(y, x)] \]
\[ \forall x \ [\exists y \ \neg \neg Animal(y) \wedge \neg Loves(x, y)] \vee [\exists y \ Loves(y, x)] \]
\[ \forall x \ [\exists y \ Animal(y) \wedge \neg Loves(x, y)] \vee [\exists y \ Loves(y, x)] \]
3. Standardize variables: each quantifier should use a different one

\[ \forall x \ [\exists y \ Animal(y) \land \neg Loves(x, y)] \lor [\exists z \ Loves(z, x)] \]

4. Skolemize: a more general form of existential instantiation. Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

\[ \forall x \ [Animal(F(x)) \land \neg Loves(x, F(x))] \lor Loves(G(x), x) \]

5. Drop universal quantifiers:

\[ [Animal(F(x)) \land \neg Loves(x, F(x))] \lor Loves(G(x), x) \]

6. Distribute \( \land \) over \( \lor \):

\[ [Animal(F(x)) \lor Loves(G(x), x)] \land [\neg Loves(x, F(x)) \lor Loves(G(x), x)] \]
Resolution proof: definite clauses

\[ \neg \text{American}(x) \lor \neg \text{Weapon}(y) \lor \neg \text{Sells}(x,y,z) \lor \neg \text{Hostile}(z) \lor \text{Criminal}(x) \]

\[ \neg \text{Criminal}(\text{West}) \]

\[ \text{American}(\text{West}) \]

\[ \neg \text{American}(\text{West}) \lor \neg \text{Weapon}(y) \lor \neg \text{Sells}(\text{West},y,z) \lor \neg \text{Hostile}(z) \]

\[ \neg \text{Missile}(x) \lor \text{Weapon}(x) \]

\[ \neg \text{Missile}(x) \lor \neg \text{Owns}(\text{Nono},x) \lor \text{Sells}(\text{West},x,\text{Nono}) \]

\[ \neg \text{Sells}(\text{West},\text{M1},z) \lor \neg \text{Hostile}(z) \]

\[ \text{Missile}(\text{M1}) \]

\[ \neg \text{Missile}(\text{M1}) \lor \neg \text{Owns}(\text{Nono},\text{M1}) \lor \neg \text{Hostile}(\text{Nono}) \]

\[ \text{Owns}(\text{Nono},\text{M1}) \]

\[ \neg \text{Enemy}(x,\text{America}) \lor \text{Hostile}(x) \]

\[ \neg \text{Hostile}(\text{Nono}) \]

\[ \text{Enemy}(\text{Nono},\text{America}) \]

\[ \text{Enemy}(\text{Nono},\text{America}) \]