Inference in PL and FOL

Chapters 7, 8 and 9
+ Prolog Redux

Long lecture ahead
Outline: PL Inference

- Enumerative methods
- Resolution in CNF
  - Sound and Complete
- Forward and Backward Chaining using Modus Ponens in Horn Form
  - Sound and Complete
Proof methods

Proof methods divide into (roughly) two kinds:

- **Application of inference rules**
  - Legitimate (sound) generation of new sentences from old
  - **Proof** = a sequence of inference rule applications
    - Can use inference rules as operators in a standard search algorithm
  - Typically require transformation of sentences into a normal form

- **Model checking**
  - truth table enumeration (always exponential in $n$)
  - improved backtracking, e.g., Davis-Putnam-Logemann-Loveland (DPLL)
  - heuristic search in model space (sound but incomplete)
    - e.g., min-conflicts like hill-climbing algorithms
Efficient propositional inference

Two families of efficient algorithms for propositional inference:

Complete backtracking search algorithms
- DPLL algorithm (Davis, Putnam, Logemann, Loveland)
- Incomplete local search algorithms
  - WalkSAT algorithm
The DPLL algorithm

Determine if an input propositional logic sentence (in CNF) is satisfiable.

Improvements over truth table enumeration:

1. Early termination
   A clause is true if any literal is true.
   A sentence is false if any clause is false.

2. Pure symbol heuristic
   Pure symbol: always appears with the same "sign" in all clauses.
   e.g., In the three clauses \((A \lor \neg B), (\neg B \lor \neg C), (C \lor A)\), \(A\) and \(B\) are pure, \(C\) is impure.
   Make a pure symbol literal true.

3. Unit clause heuristic
   Unit clause: only one literal in the clause
   The only literal in a unit clause must be true.

What are correspondences between DPLL and in general CSPs?
The DPLL algorithm

```
function DPLL-SATISFIABLE?(s) returns true or false
    inputs: s, a sentence in propositional logic
    clauses ← the set of clauses in the CNF representation of s
    symbols ← a list of the proposition symbols in s
    return DPLL(clauses, symbols, [])

function DPLL(clauses, symbols, model) returns true or false
    if every clause in clauses is true in model then return true
    if some clause in clauses is false in model then return false
    P, value ← FIND-PURE-SYMBOL(symbols, clauses, model)
    if P is non-null then return DPLL(clauses, symbols−P, [P = value|model])
    P, value ← FIND-UNIT-CLAUSE(clauses, model)
    if P is non-null then return DPLL(clauses, symbols−P, [P = value|model])
    P ← FIRST(symbols); rest ← REST(symbols)
    return DPLL(clauses, rest, [P = true|model]) or
    DPLL(clauses, rest, [P = false|model])
```
The WalkSAT algorithm

- Incomplete, local search algorithm
- Evaluation function: The min-conflict heuristic of minimizing the number of unsatisfied clauses
- Balance between greediness and randomness
The **WalkSAT** algorithm

```plaintext
function WalkSAT(clauses, p, max-flips) returns a satisfying model or failure
  inputs: clauses, a set of clauses in propositional logic
  p, the probability of choosing to do a “random walk” move
  max-flips, number of flips allowed before giving up

  model ← a random assignment of true/false to the symbols in clauses
  for i = 1 to max-flips do
    if model satisfies clauses then return model
    clause ← a randomly selected clause from clauses that is false in model
    with probability p flip the value in model of a randomly selected symbol
    from clause
    else flip whichever symbol in clause maximizes the number of satisfied clauses
  return failure
```

Let’s ask ourselves: Why is it **incomplete**?
Hard satisfiability problems

Consider random 3-CNF sentences. e.g.,

\[(\neg D \lor \neg B \lor C) \land (B \lor \neg A \lor \neg C) \land (\neg C \lor \neg B \lor \neg E) \land (E \lor \neg D \lor B) \land (B \lor E \lor \neg C)\]

- \(m = \text{number of clauses}\)
- \(n = \text{number of symbols}\)

- Hard problems seem to cluster near \(m/n = 4.3\) (critical point)
Hard satisfiability problems
Hard satisfiability problems

- Median runtime for 100 satisfiable random 3-CNF sentences, \( n = 50 \)
Proof methods

Proof methods divide into (roughly) two kinds:

- Application of inference rules
  - Legitimate (sound) generation of new sentences from old
  - Proof = a sequence of inference rule applications
    Can use inference rules as operators in a standard search algorithm
  - Typically require transformation of sentences into a normal form

- Model checking
  - truth table enumeration (always exponential in $n$)
  - improved backtracking, e.g., Davis-Putnam-Logemann-Loveland (DPLL)
  - heuristic search in model space (sound but incomplete)
    e.g., min-conflicts like hill-climbing algorithms
Resolution

Conjunctive Normal Form (CNF)
conjunction of disjunctions of literals
clauses
E.g., \((A \lor \neg B) \land (B \lor \neg C \lor \neg D)\)

- **Resolution** inference rule (for CNF):

\[
\frac{\ell_i \lor \ldots \lor \ell_k}{\ell_i \lor \ldots \lor \ell_{i-1} \lor \ell_{i+1} \lor \ldots \lor \ell_k \lor m_1 \lor \ldots \lor m_n}
\]

where \(\ell_i\) and \(m_j\) are complementary literals.
E.g., \(P_{1,3} \lor P_{2,2}, \neg P_{2,2}\)

- Resolution is sound and complete
for propositional logic
Resolution example

- \( KB = (B_{1,1} \iff (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1} \)
- \( \alpha = \neg P_{1,2} \) (negate the premise for proof by refutation)
The power of false

- Given: \((P) \land (\neg P)\)
- Prove: \(Z\)

| \(\neg P\) | Given |
| P          | Given |
| \(\neg Z\) | Given |

- Can we prove \(\neg Z\) using the givens above?
Applying inference rules

Equivalent to a search problem

- KB state = node
- Inference rule application = edge
Inference

- Define: $KB \models_i \alpha = \text{sentence \( \alpha \) can be derived from } KB \text{ by procedure } i$

- **Soundness**: $i$ is sound if whenever $KB \models_i \alpha$, it is also true that $KB \models \alpha$

- **Completeness**: $i$ is complete if whenever $KB \models \alpha$, it is also true that $KB \models_i \alpha$

- Preview: we will define a logic (first-order logic) which is expressive enough to say almost anything of interest, and for which there exists a sound and complete inference procedure.

- That is, the procedure will answer any question whose answer is in the set of inference operators complete and sound?

- Do the operators make conclusions that aren’t always true?
Completeness: i is complete if whenever $KB \vdash \alpha$, it is also true that $KB \models_i \alpha$

- An incomplete inference algorithm cannot reach all possible conclusions
  - Equivalent to completeness in search (chapter 3)
Resolution

Conjunctive Normal Form (CNF)

- conjunction of disjunctions of literals
  - clauses

  E.g., \((A \lor \neg B) \land (B \lor \neg C \lor \neg D)\)

- Resolution inference rule (for CNF):

  \[
  \frac{\ell_i \lor \ldots \lor \ell_k \quad m_1 \lor \ldots \lor m_n}{\ell_i \lor \ldots \lor \ell_{i-1} \lor \ell_{i+1} \lor \ldots \lor \ell_k \lor m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n}
  \]

  where \(\ell_i\) and \(m_j\) are complementary literals.

  E.g., \(P_{1,3} \lor P_{2,2}, \neg P_{2,2}\)

  \[
  \frac{P_{1,3}}{P_{1,3}}
  \]

- Resolution is **sound** and **complete**
- for propositional logic
Resolution

Soundness of resolution inference rule:

\[\neg (l_i \lor \ldots \lor l_{i-1} \lor l_{i+1} \lor \ldots \lor l_k) \Rightarrow l_i\]
\[\neg m_j \Rightarrow (m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n)\]
\[\neg (l_i \lor \ldots \lor l_{i-1} \lor l_{i+1} \lor \ldots \lor l_k) \Rightarrow (m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n)\]

where \(l_i\) and \(m_j\) are complementary literals.

- What if \(l_i\) and \(\neg m_j\) are false?
- What if \(l_i\) and \(\neg m_j\) are true?
Completeness of Resolution

- That is, that resolution can decide the truth value of S

- \( S = \) set of clauses

- \( RC(S) = \text{Resolution closure of } S = \) Set of all clauses that can be derived from S by the resolution inference rule.

- \( RC(S) \) has finite cardinality (finite number of symbols \( P_1, P_2, \ldots P_k \)), thus resolution refutation must terminate.
Completeness of Resolution (cont)

- Ground resolution theorem = if S unsatisfiable, RC(S) contains empty clause.
- Prove by proving contrapositive:
  - i.e., if RC(S) doesn’t contain empty clause, S is satisfiable
  - Do this by constructing a model:
    - For each $P_i$, if there is a clause in RC(S) containing $\neg P_i$ and all other literals in the clause are false, assign $P_i = \text{false}$
    - Otherwise $P_i = \text{true}$
  - This assignment of $P_i$ is a model for S.
Other Reasoning Patterns

- Resolution works by refutation
- What about proving propositions directly?

Given(s) \[\begin{array}{c}
A \implies B, A \\
B \\
B \land A
\end{array}\]

Conclusion

Rules that allow us to introduce new propositions while preserving truth values: logically equivalent

Two Examples:
- Modus Ponens
- And Elimination
Forward and backward chaining

- **Horn Form (restricted)**
  - KB = conjunction of Horn clauses
  - Horn clause =
    - proposition symbol; or
    - (conjunction of symbols) ⇒ symbol
  - E.g., $C \land (B \Rightarrow A) \land (C \land D \Rightarrow B)$

- **Modus Ponens (for Horn Form):** complete for Horn KBs
  \[
  \text{If } a_1, \ldots, a_n, a_1 \land \ldots \land a_n \Rightarrow \beta
  \text{ then } \beta
  \]

- Can be used with forward chaining or backward chaining.
- These algorithms are very natural and run in linear time
Forward chaining

- Idea: fire any rule whose premises are satisfied in the \( KB \),
  - add its conclusion to the \( KB \), until query is found

\[
\begin{align*}
P \Rightarrow Q \\
L \land M & \Rightarrow P \\
B \land L & \Rightarrow M \\
A \land P & \Rightarrow L \\
A \land B & \Rightarrow L \\
A & \\
B
\end{align*}
\]
Forward chaining algorithm

function PL-FC-ENTAILS?(KB, q) returns true or false

local variables: count, a table, indexed by clause, initially the number of premises
inferred, a table, indexed by symbol, each entry initially false
agenda, a list of symbols, initially the symbols known to be true

while agenda is not empty do
    p ← POP(agenda)
    unless inferred[p] do
        inferred[p] ← true
        for each Horn clause c in whose premise p appears do
            decrement count[c]
            if count[c] = 0 then do
                if HEAD[c] = q then return true
            end if
        end for
    end unless
    PUSH(HEAD[c], agenda)
end while

return false

- Forward chaining is sound and complete for Horn KB
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example
Forward chaining example
Proof of completeness

FC derives every atomic sentence that is entailed by $KB$ (only for clauses in Horn form)

1. FC reaches a fixed point (the deductive closure) where no new atomic sentences are derived
2. Consider the final state as a model $m$, assigning true/false to symbols
3. Every clause in the original $KB$ is true in $m$
   \[ a_1 \land \ldots \land a_k \Rightarrow b \]
4. Hence $m$ is a model of $KB$
5. If $KB \models q$, $q$ is true in every model of $KB$, including $m$
Backward chaining example
Backward chaining example
Backward chaining example
Inference in first-order logic

Chapter 9
Outline

- Reducing first-order inference to propositional inference
- Unification
- Generalized Modus Ponens
- Forward chaining
- Backward chaining
- Resolution
Universal instantiation (UI)

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

\[
\forall v \alpha \\
\rightarrow \\
\text{Subst}\{v/g\}, \alpha
\]

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

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

\[
\text{King}(\text{John}) \land \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})
\]
\[
\text{King}(\text{Richard}) \land \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})
\]
\[
\text{King}(\text{Father}(\text{John})) \land \text{Greedy}(\text{Father}(\text{John})) \Rightarrow \text{Evil}(\text{Father}(\text{John}))
\]
Existential instantiation (EI)

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

$$\exists v \alpha \quad \text{Subst}\{\{v/k\}, \alpha\}$$

E.g., $\exists x \ Crown(x) \land OnHead(x, John)$ yields:

$$Crown(C_1) \land OnHead(C_1, John)$$

provided $C_1$ is a new constant symbol, called a Skolem constant
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}, \text{John})

- 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{ King}(\text{Richard}) \land \text{ Greedy}(\text{Richard}) \Rightarrow \text{ Evil}(\text{Richard})
  \text{ King}(\text{John})
  \text{ Greedy}(\text{John})
  \text{ Brother}(\text{Richard}, \text{John})

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

- Every FOL KB can be propositionalized so as to preserve entailment

- (A ground sentence is entailed by new KB iff entailed by original KB)

- Idea: propositionalize KB and query, apply resolution, return result

- Problem: with function symbols, there are infinitely many ground terms,
  - e.g., Father(Father(Father(John))))
Reduction con’td.

Theorem: Herbrand (1930). If a sentence $\alpha$ is entailed by an FOL KB, it is entailed by a finite subset of the propositionalized KB.

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

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

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

Theorem: Turing (1936), Church (1936) Entailment for FOL is semi-decidable (algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every non-entailed sentence.)
Problems with propositionalization

- Propositionalization seems to generate lots of irrelevant sentences.

- E.g., from:
  \[ \forall x \text{King}(x) \land \text{Greedy}(x) \Rightarrow \text{Evil}(x) \]
  \[ \text{King}(\text{John}) \]
  \[ \forall y \text{Greedy}(y) \]
  \[ \text{Brother}(\text{Richard}, \text{John}) \]

- it seems obvious that \text{Evil}(\text{John}), but propositionalization produces lots of facts such as \text{Greedy}(\text{Richard}) that are irrelevant

- With \( p \) \( k \)-ary predicates and \( n \) constants, there are \( p \cdot n^k \) instantiations.
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 $\text{Greedy}(y)$.

$\theta = \{x/\text{John}, y/\text{John}\}$ works.

- **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>Knows(John,x)</td>
<td>Knows(John,Jane)</td>
<td></td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(y,OJ)</td>
<td></td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(y,Mother(y))</td>
<td></td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(x,OJ)</td>
<td></td>
</tr>
</tbody>
</table>

- **Standardizing apart** eliminates overlap of variables, e.g., $\text{Knows}(z_{17}, \text{OJ})$
Unification

- To unify $\text{Knows}(John,x)$ and $\text{Knows}(y,z)$, $\theta = \{y/John, x/z\}$ or $\theta = \{y/John, x/John, z/John\}$

- The first unifier is more general than the second.

- There is a single most general unifier (MGU) that is unique up to renaming of variables. $\text{MGU} = \{y/John, x/z\}$
The unification algorithm

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 UnifyARGS[x, ARGS[y], Unify(Op[x], Op[y], θ)]
else if List?(x) and List?(y) then
    return UnifyREST[x, REST[y], UnifyFIRST[x], FIRST[y], θ])
else return failure
The unification algorithm

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/val\} ∈ θ then return UNIFY(val, x, θ)
else if \{x/val\} ∈ θ then return UNIFY(var, val, θ)
else if OCCUR-CHECK?(var, x) then return failure
else return add \{var/x\} to θ
Generalized Modus Ponens (GMP)

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q)}{q\theta}$$

where $$p_i\theta = p_i$$ for all $$i$$

- $$p_1'$$ is $$\text{King}(\text{John})$$
- $$p_1$$ is $$\text{King}(x)$$
- $$p_2'$$ is $$\text{Greedy}(y)$$
- $$p_2$$ is $$\text{Greedy}(x)$$
- $$\theta$$ is $$\{x/\text{John}, y/\text{John}\}$$
- $$q$$ is $$\text{Evil}(x)$$
- $$q\theta$$ 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
  \[ p_1', ..., p_n', (p_1 \land ... \land p_n \Rightarrow q) \models q \theta \]
  provided that \( p_i'\theta = p_i\theta \) for all \( i \)

- Lemma: For any sentence \( p \), we have \( p \models p\theta \) by UI

1. \( (p_1 \land ... \land p_n \Rightarrow q) \models (p_1 \land ... \land p_n \Rightarrow q)\theta = (p_1\theta \land ... \land p_n\theta \Rightarrow q\theta) \)
2. \( p_1', ..., p_n' \models p_1' \land ... \land p_n' \models p_1'\theta \land ... \land p_n'\theta \)
3. From 1 and 2, \( q\theta \) follows by ordinary Modus Ponens
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 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(Nono,x)} \land \text{Missile(x)}:
\[
\text{Owns(Nono,M_1)} \land \text{Missile(M_1)}
\]

... all of its missiles were sold to it by Colonel West
\[
\text{Missile}(x) \land \text{Owns(Nono,x)} \Rightarrow \text{Sells(West,x,Nono)}
\]

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 …
\[
\text{American(West)}
\]

The country Nono, an enemy of America …
\[
\text{Enemy(Nono,\text{America})}
\]
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 = (p'_₁ ∧ ... ∧ p'_ₙ)θ
                for some p'_₁, ..., p'_ₙ 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

\text{American(West)} \quad \text{Missile(M1)} \quad \text{Owns(Nono,M1)} \quad \text{Enemy(Nono,America)}
Forward chaining proof

Diagram:

- \text{Weapon}(M1)
- \text{Sells}(\text{West}, M1, \text{Nono})
- \text{Hostile}(\text{Nono})
- \text{American}(\text{West})
- \text{Missile}(M1)
- \text{Owns}(\text{Nono}, M1)
- \text{Enemy}(\text{Nono}, \text{America})
Forward chaining proof
Properties of forward chaining

- Sound and complete for first-order definite clauses
- **Datalog** = first-order definite clauses + no functions
- FC terminates for Datalog in finite number of iterations
- May not terminate in general if $\alpha$ is not entailed
- This is unavoidable: entailment with definite clauses is semidecidable
Efficiency of forward chaining

Incremental forward chaining: no need to match a rule on iteration \( k \) if a premise wasn't added on iteration \( k-1 \)

\[ \Rightarrow \] match each rule whose premise contains a newly added positive literal

Matching itself can be expensive:

**Database indexing** allows \( O(1) \) retrieval of known facts

- e.g., query \( \text{Missile}(x) \) retrieves \( \text{Missile}(M_1) \)

Forward chaining is widely used in **deductive databases**
Backward chaining algorithm

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 \(\text{Standardize-Apart}(r) = (p_1 \land \ldots \land p_n \Rightarrow q)\)
and \(\theta' \leftarrow \text{Unify}(q, q')\) succeeds
\(ans \leftarrow \text{FOL-BC-Ask}(KB, [p_1, \ldots, p_n|\text{Rest}(goals)], \text{Compose}(\theta, \theta')) \cup ans\)
return \(ans\)

\(\text{Subst}(\text{Compose}(\theta_1, \theta_2), p) = \text{Subst}(\theta_2, \text{Subst}(\theta_1, p))\)
Backward chaining example

\textit{Criminal(West)}
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example
Backward chaining example

![Diagram]

- **Criminal(West)**
  - **American(West)**
    - {}
  - **Weapon(y)**
    - {}
  - **Sells(West, ML, z)**
    - \{ z/Nono \}
    - **Hostile(z)**
  - **Missile(y)**
    - \{ y/ML \}
  - **Missile(ML)**
  - **Owns(Nono, ML)**
Backward chaining example
Prolog Inference

Q: which model do you think Prolog uses for inference?
Properties of backward chaining

- Depth-first recursive proof search: space is linear w.r.t. size of proof

- Incomplete due to infinite loops
  \[ \Rightarrow \text{fix by checking current goal against every goal on stack} \]

- Inefficient due to repeated subgoals (both success and failure)
  \[ \Rightarrow \text{fix using caching of previous results (extra space)} \]
Prolog Execution

Prolog needs to choose which goal to pursue first, although logically it doesn’t matter. Why?

- Treats goals in order, leftmost first.

A :- B,C,D.
B :- E,F.
?- A.

- B is tried first, then C, then D.
- E and F are pushed onto the stack, before C and D. Why?
Prolog Execution

Prolog also needs to choose which clause to pursue first.

- Treats clauses in order, top-most first.

  G.
  A :- B, C, D.
  B :- E, F.
  B :- G.

- To satisfy goal B, prolog tries E, F before G.
Procedural Prolog Programming

- Order of Prolog clauses and goals crucial, can affect running times immensely
  - Order of goals tell which get executed first
  - Order of clauses tell which control branches are tried first.
A Singaporean example

likes(hari,X) :- makan(X), consumes(hari,X).
likes(min,X) :- likes(hari,X).
makan(meeSiam).
makan(rojak).
minum(rootBeerFloat).
consumes(hari,meeSiam).

\[
\begin{align*}
\text{likes}(\text{min}, \text{X}) &
\quad \text{likes}(\text{min}, \text{X}) \\
\quad \text{likes}(\text{hari}, \text{X1}) &
\quad \text{likes}(\text{hari}, \text{X1}) \\
\quad \text{makan}(\text{X2}), \text{consumes}(\text{hari}, \text{X2}) &
\quad \text{makan}(\text{X2}), \text{consumes}(\text{hari}, \text{X2}) \\
\end{align*}
\]

\[
\begin{align*}
\text{X2} = \text{meeSiam} &
\quad \text{consumes}(\text{hari}, \text{meeSiam}). \\
\end{align*}
\]
Whew! That was a loooooooong lecture. What did we learn?

- Enumeration: DPLL rules are similar to CSP heuristics.
- Resolution is proof by refutation, used in PL.
- Other forms of reasoning: Modus Ponens which requires Horn form.
- FOL uses unification to find solutions, requires Skolem constants and functions.
- Forward (undirected) and Backward (directed) chaining patterns to apply an inference mechanism.