Artificial Intelligence

Module 4: Logic and Deduction

PART 4.4 : Deduction & Reasoning Tasks
PART 4.5 : Inference By Forward & Backward Chaining
PART 4.6 : Inferencing By Resolution Refutation
PART 4.7 : Reduction to satisfiability problem

Dr. Chandra Prakash
Assistant Professor
Department of Computer Science and Engineering

(Slides adapted from StuartJ. Russell, B Ravindran, Mausam, Dan Klein and Pieter Abbeel, Partha P Chakrabarti, Saikishor Jangiti

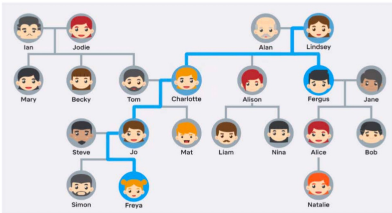
Module 4: Logic and Deduction

- PART 4.1 : Logical Agents
- PART 4.2 : Propositional logic
- PART 4.3 : Predicate Logic
  - Propositional Logic to Predicate Logic
  - Predicate Logic Fundamentals
- PART 4.4 : Deduction & Reasoning Tasks
  - Theorem Proving
- PART 4.5 : Inference By Forward & Backward Chaining
  - Representing knowledge using Prolog
- PART 4.6 : Inferencing By Resolution Refutation
- PART 4.7: Reduction to satisfiability problem : SAT Solver

Logical Reasoning with Other Fun Things

- Facts
- Rules
  - grandfather, grandmother,
  - maternalgrandfather, maternalgrandmother ,
- Query :
  - maternalgranduncle

- **Who is the maternal great uncle of Freya?**



We need that a social media platform to suggests Freya to post a picture of Fergus on the Maternal-Great-Uncle day

Logical Reasoning with Other Fun Things

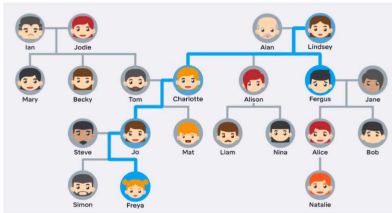
- **grandfather, grandmother, maternalgrandfather, maternalgrandmother , maternalgranduncle**

- *father(x, z), father(z, y)* ⇒ ***grandfather**(x, y)*
- *father(x, z), mother(z, y)* ⇒ ***grandmother**(x, y)*

- *mother(x, z), father(z, y)*
  ⇒ *maternalgrandfather(x, y)*
- *mother(x, z), mother(z, y)*
  ⇒ *maternalgrandmother(x, y)*

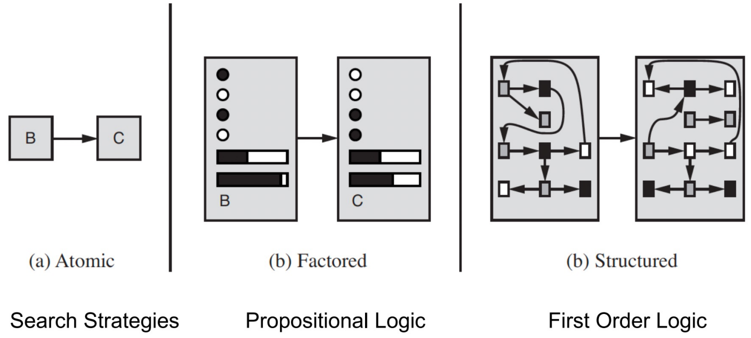
- *maternalgrandmother(x, z), mother(z, p), son(p,y)* ⇒ *maternalgreatuncle(x, y)*

- **Who is the maternal great uncle of Freya?**



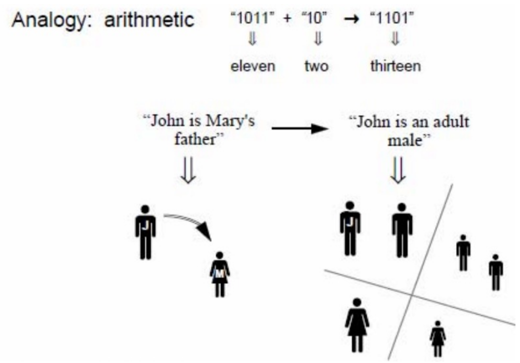
- maternalgrandmother(Freya, Charlotte), mother(Charlotte, Lindsey), son(Lindsey, Fergus) ⇒ maternalgreatuncle(Freya, Fergus)

Complexity of Representation



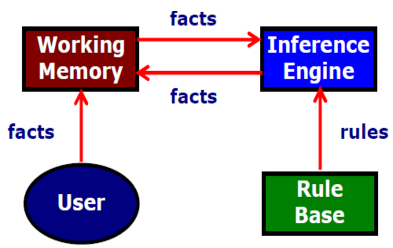
What is reasoning

- Manipulation of symbols encoding propositions to produce represenations of new propositions
- Benefits of Reasoning
  - Given
    - Patient X allergic to medication M
    - Anyone allergic to medication M is also allergic to medication M'
  - Reasoning helps us derive
    - Patient X is allergic to medication M'



Reasoning Tasks: Modeling and Deduction

- Model finding
  - KB = background knowledge
  - S = description of problem
  - Show (KB ∧ S) is satisfiable
  - A kind of constraint satisfaction
- Deduction
  - S = question
  - Prove that KB |= S
  - Two approaches:
    - Rules to derive new formulas from old (inference)
    - Show (KB ∧ ¬ S) is unsatisfiable
      - Proof by Contradiction / Resolution



Deduction : Term

- DEDUCTION
  - Commonly associated with formal logic
  - Involves reasoning from known premises to a conclusion
  - The conclusions reached are inevitable, certain, inescapable
- Validity / Tautology :
  - A sentence is valid if it is true in all models. E.g., P1,2 ∨ ¬ P1,2 is always true
- Deduction Theorem:
  - For any sentences α and β, $\alpha \models \beta$ if and only if the sentence $(\alpha \Rightarrow \beta)$ is valid
- If we prove $(\alpha \Rightarrow \beta)$ is equivalent to True, then entailment is proved
- Satisfiability: A sentence is satisfiable if it is true in some model.
  - S1 entails S2 if wherever S1 is true S2 is also true
- Unsatisfiable: If a sentence is false in all models.
  - E.g., $P_{1,2} \wedge \neg P_{1,2}$ is always false
- Boolean satisfiability problem (SAT) : mid-1990's
  - The problem of determining the satisfiability of sentences in propositional logic
  - given a formula, to check whether it is satisfiable.
  - importance in mantheoretical computer science, complexity theory, algorithmics, cryptography and AI

Examples: Satisfiable, UnSatisfiable &Valid

a) $P \rightarrow Q$
b) $R \rightarrow \neg R$
c) $S \land (W \land \neg S)$
d) $T \lor \neg T$
e) $X \rightarrow X$

a) Satisfiable
b) Satisfiable
c) Unsatisfiable
d) Valid /Tautology
e) Tautology

Recall: BNF grammar of sentences

$$
\begin{array}{rcl}
Sentence & \rightarrow & AtomicSentence \mid ComplexSentence \\
AtomicSentence & \rightarrow & True \mid False \mid P \mid Q \mid R \mid \ldots \\
ComplexSentence & \rightarrow & (\, Sentence\,) \mid [\, Sentence\,] \\
 & \mid & \neg\, Sentence \\
 & \mid & Sentence \land Sentence \\
 & \mid & Sentence \lor Sentence \\
 & \mid & Sentence \Rightarrow Sentence \\
 & \mid & Sentence \Leftrightarrow Sentence \\
\\
\text{OPERATOR PRECEDENCE} & : & \neg, \land, \lor, \Rightarrow, \Leftrightarrow
\end{array}
$$

Connectives and Symbols in decreasing order of operation priority

Connective	Symbols					Read as
assertion	P					"p is true"
negation	¬p	~	!		NOT	"p is false"
conjunction	p ∧ q	·	&&	&	AND	"both p and q are true"
disjunction	P v q	||	|		OR	"either p is true, or q is true, or both "
implication	p → q	⊃	⇒		if ..then	"if p is true, then q is true" " p implies q "
equivalence	↔	≡	⇔		if and only if	"p and q are either both true or both false"

P	Q	$\neg P$	$P \land Q$	$P \lor Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Notation

- $\Rightarrow$
- $\supset$
- $\rightarrow$

Implication (syntactic symbol)

- $\vdash$ **Proves**: $S1 \vdash_{ie} S2$ if \`ie' algorithm says \`S2' from S1
  - ie : inference engine
- $\models$ **Entails**: $S1 \models S2$ if wherever S1 is true S2 is also true
- Sound $\vdash \rightarrow \models$
  - nothing but the truth
- Complete $\models \rightarrow \vdash$
  - all truth

Normal Forms

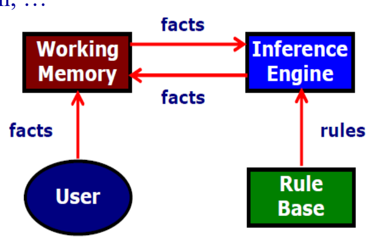
- Several ways of representing the same logical statement.
  - They are logically equivalent. e.g.
    - $P \rightarrow Q$
    - $\neg P \lor Q$ or $\neg(P \land \neg Q)$
- The convention for standardizing the representation of formulas is called a canonical or normal form.
- conjunctive normal form (CNF)
  - $(\neg P \lor Q) \land (S \lor T) \land R$
- disjunctive normal form (DNF)
  - $(\neg P \land S \land R) \lor (\neg P \land T \land R) \lor (Q \land S \land R) \lor (Q \land T \land R)$
- Given an arbitrary database of propositional formulas, it is possible to generate an equivalent database in CNF (or DNF).

Special Syntactic Forms

- General Form:
  - $((q \wedge \neg r) \rightarrow s)) \wedge \neg (s \wedge t)$
- Conjunction Normal Form (CNF)
  - $(\neg q \vee r \vee s) \wedge (\neg s \vee \neg t)$
  - Set notation: $\{ (\neg q, r, s), (\neg s, \neg t) \}$
  - empty clause () = false
- Binary clauses: 1 or 2 literals per clause
  - $(\neg q \vee r)\ (\neg s \vee \neg t)$
- Horn clauses: 0 or 1 positive literal per clause
  - $(\neg q \vee \neg r \vee s)\ (\neg s \vee \neg t)$
  - $(q \wedge r) \rightarrow s\ (s \wedge t) \rightarrow$ false

Inference Enging

- Inference engine performs 2 major tasks:
  1. Examines existing facts and rules and adds new facts when possible
  2. Decides the order in which inferences are made.
- Inference rules for propositional logic apply to FOL as well
  - Modus Ponens, And-Introduction, And-Elimination, …
- New (sound) inference rules for use with quantifiers:
  - Universal elimination
  - Existential introduction
  - Existential elimination
  - Generalized Modus Ponens (GMP)



Propositional Logic: Inference

Inference:

A mechanical process for computing new sentences

1. Backward & Forward Chaining
2. Resolution (Proof by Contradiction)
3. SAT
   1. Davis Putnam
   2. WalkSat

Inference Rules

- Inference rules that can be applied to derive a proof
  - a chain of conclusions that leads to the desired goal.
- **Inference Rules:**
  - Modus Ponens (Latin for mode that affirms) and is written $\dfrac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$
    - if (WumpusAhead $\wedge$WumpusAlive) $\Rightarrow$ Shoot and (WumpusAhead $\wedge$ WumpusAlive) are given, then Shoot can be inferred.
  - And-Elimination, says that, from a conjunction, any of the conjuncts can be inferred : $\dfrac{\alpha \wedge \beta}{\alpha}$
    - from (WumpusAhead $\wedge$ WumpusAlive), WumpusAlive can be inferred.
  - Resolution Technique :
    - assumes sentences are in conjunctive normal form (CNF) – conjunction of clauses,
    - e.g. $(\neg B1,1 \vee P1,2 \vee P2,1) \wedge (\neg P1,2 \vee B1,1) \wedge (\neg P2,1 \vee B1,1)$

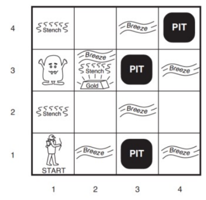
Inference Rules

- All Logical Equivalence rules can be used as inference rules

$$
\begin{array}{rcll}
(\alpha \land \beta) & \equiv & (\beta \land \alpha) & \text{commutativity of } \land \\
(\alpha \lor \beta) & \equiv & (\beta \lor \alpha) & \text{commutativity of } \lor \\
((\alpha \land \beta) \land \gamma) & \equiv & (\alpha \land (\beta \land \gamma)) & \text{associativity of } \land \\
((\alpha \lor \beta) \lor \gamma) & \equiv & (\alpha \lor (\beta \lor \gamma)) & \text{associativity of } \lor \\
\neg(\neg\alpha) & \equiv & \alpha & \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) & \equiv & (\neg\beta \Rightarrow \neg\alpha) & \text{contraposition} \\
(\alpha \Rightarrow \beta) & \equiv & (\neg\alpha \lor \beta) & \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) & \equiv & ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)) & \text{biconditional elimination} \\
\neg(\alpha \land \beta) & \equiv & (\neg\alpha \lor \neg\beta) & \text{De Morgan} \\
\neg(\alpha \lor \beta) & \equiv & (\neg\alpha \land \neg\beta) & \text{De Morgan} \\
(\alpha \land (\beta \lor \gamma)) & \equiv & ((\alpha \land \beta) \lor (\alpha \land \gamma)) & \text{distributivity of } \land \text{ over } \lor \\
(\alpha \lor (\beta \land \gamma)) & \equiv & ((\alpha \lor \beta) \land (\alpha \lor \gamma)) & \text{distributivity of } \lor \text{ over } \land
\end{array}
$$

Propositional Logic (PL) Inference – Theorem Proving

- Using Logical equivalence $(\alpha \Longrightarrow \beta) \equiv \neg\alpha \lor \beta$ and Deductive theorem .
- For any sentences α and β, $\alpha \vDash \beta$ if and only if the sentence $(\alpha \land \neg\beta)$ is unsatisfiabl
- This is Proof by Refutation or Proof by Contradiction
- Example: Given our simple KB of Wumpus World
  - R1: $\neg P_{1,1}$
  - R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$
  - R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \lor P_{2,2} \lor P_{3,1})$
  - R_4: $\neg B_{1,1}$
  - R_5: $B_{2,1}$
- Query: $\neg P_{1,2}$
- Can we prove if this sentence be entailed from KB using inference rules?



R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$
Apply Biconditional Elimination
R_6: $(B_{1,1} \Longrightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Longrightarrow B_{1,1})$
Apply And-Elimination to R6 to obtain
R_7: $((P_{1,2} \lor P_{2,1}) \Longrightarrow B_{1,1})$
Logical equivalence for contrapositives gives
R_8: $(\neg B_{1,1} \Longrightarrow \neg (P_{1,2} \lor P_{2,1}))$
Apply Modus Ponens with R8 and R4 to obtain
R_9: $\neg (P_{1,2} \lor P_{2,1})$
Finally, using De Morgan's rule
R_{10} : $\neg P_{1,2} \land \neg P_{2,1}$

PL Inference – Theorem Proving

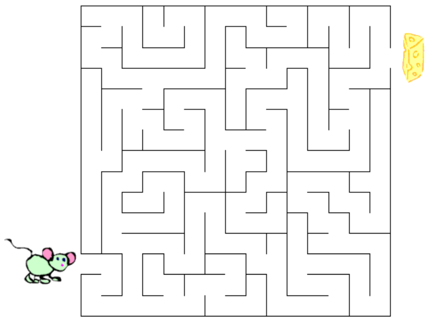
- This proof can be efficiently computed using search algorithms we discussed earlier.
- We need to define the proof problem as
  - Initial State: the initial knowledge base
  - Actions: all inference rules that could match the sentences with the top half of inference rule
  - Result: the bottom half of inference rule
  - Goal: The state containing the query sentence
  - In many practical cases, searching a proof can be more efficient because the proof can ignore irrelevant propositions
- In many practical cases, searching a proof can be more efficient because the proof can ignore irrelevant propositions

Inference 1: Forward/Backward Chaining

- Forward Chaining:
  - Based on rule of modus ponens
  - If know P_1, …, P_n & know $(P_1 \land \ldots \land P_n) \rightarrow Q$
  - Then can conclude Q
- Backward Chaining: search
  - start from the query and go backwards
- Is it sound ?
  - FC ☑ BC☑
- Is is complete ?
  - FC ☑ BC☑

Two referencing methods

- Objective is to find a path through a problem space from the initial to the final one.
- Two directions to go and find the answer:

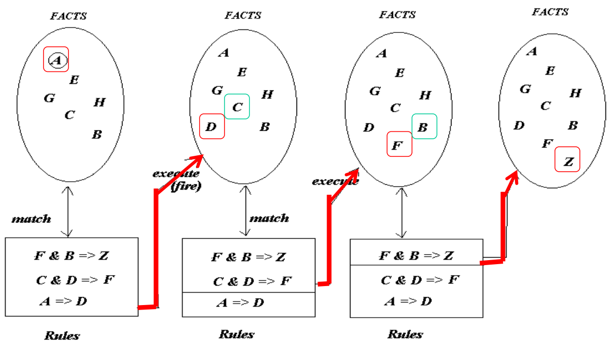


- **Forward chaining : also called data driven.**
  - It starts with the facts, and sees what rules apply.
  - to reason forward, the left sides(pre conditions) are matched against the current state and the right sides(the results) are used to generate new nodes until the goal is reached.
- **Backward chaining : also called goal driven.**
  - It starts with something to find out, and looks for rules that will help in answering it.
  - To reason backwards, the right sides are matched against the current node and the left sides are used to generate new nodes.

Forward v/s Backward Reasoning

- **Reason forward from the initial states :**
  - Begin building a tree of move sequences that might be solution by starting with the initial configuration(s) at the root of the tree.
  - Generate the next level of tree by finding all the rules whose left sides match the root node and use the right sides to create the new configurations.
  - Generate each node by taking each node generated at the previous level and applying to it all of the rules whose left sides match it.
- **Reason backward from the goal states :**
  - Begin building a tree of move sequences that might be solution by starting with the goal configuration(s) at the root of the tree.
  - Generate the next level of tree by finding all the rules whose right sides match the root node and use the left sides to create the new configurations.
  - Generate each node by taking each node generated at the previous level and applying to it all of the rules whose right sides match it. Continue. This is also called Goal-Directed Reasoning.

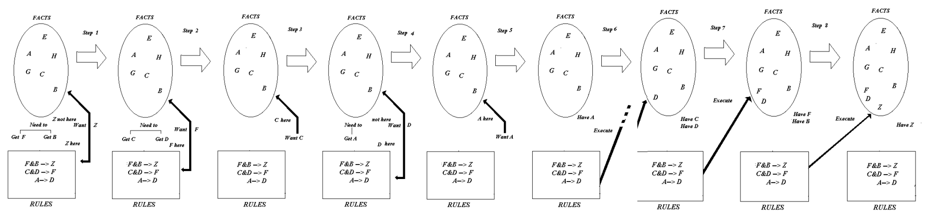
Forward Chaining:



Problem: Does situation Z exists or not ?

Backward Chaining

- With this inference method the system starts with what it wants to prove
  - that situation Z exists, and only executes rules that are relavent to establishing it.



Whether to choose forward or backward reasoning :

- Are there more possible start states or goal states?
  - We would like to go from smaller set of states to larger set of states.
- In which direction is the branching factor (the average number of nodes that can be reached directly from a single node) greater?
  - We would like to proceed in the direction with the lower branching factor.
- Will the program be asked to justify its reasoning process to the user?
  - It so, it is important to proceed in the direction that corresponds more closely with the way user will think.
- What kind of event is going to trigger a problem-solving episode?
  - If it is the arrival of a new fact , forward reasoning should be used. If it a query to which response is desired, use backward reasoning.

Quiz

- Forward v/s Backward Reasoning
  - Home to unknown place example.
  - Patients example of diagnosis
  - MYCIN
  - Prolog
  - Where are my keys?
- Bidirectional Search (The two searches must pass each other)
- Forward Rules : which encode knowledge about how to respond to certain input configurations.
- Backward Rules : which encode knowledge about how to achieve particular goals.

Forward v/s Backward Reasoning

- Forward –Chaining Rule Systems
  - data-driven
  - Automatic, unconscious processing
  - E.g., object recognition, routine decisions
  - May do lots of work that is irrelevant to the goal
- Backward-Chaining Rule Systems
  - goal-driven, appropriate for problem-solving
  - PROLOG is an example of this.
  - These are good for goal-directed problem solving.
  - Hence Prolog & MYCIN are examples of the same.

- Patients example of diagnosis.
  - In some systems ,this is only possible in reversible rules.

Question Answering

PROLOG:

- Only Horn sentences are acceptable
- The occur-check is omitted from the unification: unsound

  test ← P(x, x)

  P(x, f(x))
- Backward chaining with depth-first search: incomplete

  P(x, y) ← Q(x, y)

  P(x, x)

  Q(x, y) ← Q(y, x)

Clause

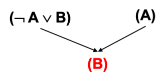
- Term
  - The set of terms of FOL is the least set satisfying these conditions:
    - every variable is a term
    - if t_l . . . . . t_n are terms, and f is a function symbol of arity n, then f(t_l . . . . . t_n) is a term
- Formula
  - The set of formulas of FOL is the least set satisfying these constraints:
    - if t_l ....t_n are terms, and P is a predicate symbol of arity n, then P(t1 . . . . . tn) is a formula;
    - if t1 and t2 are terms, then tl=t2 is a formula;
    - if α and β are formulas, and x is a variable, then ¬α, α ∨ β, α ∧ β, x α, and Exists α, are formulas.
- Atomic Formula
  - Formulas of first two types above
- Sentence
  - Any formula with no free variables

Completeness of GMP

- Literal
  - Atomic formula or its negation
- Clause
  - A finite set of literals
- A clause (i.e., a disjunction of literals)
- A Horn clause is a clause containing at most one positive literal.
- A definite clause contains exactly one positive literal.
- Examples of a Horn Clause
  - [¬Child, ¬Mail, Boy]
- Not a Horn Clause
  - [Rain, Sleet, Snow]
- Generalized Modus Ponens (GMP)
  - using forward or backward chaining is complete for KBs that contain only Horn clauses
- It is not complete for simple KBs that contain non-Horn clauses
- The following entail that S(A) is true:
  - $(\forall x)\ P(x) \rightarrow Q(x)$
  - $(\forall x)\ \neg P(x) \rightarrow R(x)$
  - $(\forall x)\ Q(x) \rightarrow S(x)$
  - $(\forall x)\ R(x) \rightarrow S(x)$
- Which one is not a Horn clause ??
  - If we want to conclude S(A), with GMP we cannot, since the second one is not a Horn clause
  - It is equivalent to $P(x) \vee R(x)$

Automating FOL inference with resolution

- Resolution Technique :
  - assumes sentences are in conjunctive normal form (CNF) – conjunction of clauses
- Resolution subsumes Modus Ponens
- $A \rightarrow B,\ A \models B$



Resolution in Propositional Logic

- Resolution is a **sound** and **complete** inference procedure for FOL
- Reminder: Resolution rule for propositional logic:
  - $P_1 \vee P_2 \vee \ldots \vee P_n$
  - $\neg P_1 \vee Q_2 \vee \ldots \vee Q_m$
  - Resolvent: $P_2 \vee \ldots \vee P_n \vee Q_2 \vee \ldots \vee Q_m$
- Examples
  - P and ¬ P ∨ Q :
    - derive Q (Modus Ponens)
  - (¬ P ∨ Q) and (¬ Q ∨ R) : derive ¬ P ∨ R
  - **P and ¬ P : derive False [contradiction!]**
  - (P ∨ Q) and (¬ P ∨ ¬ Q) : derive True

Resolution Refutation for Propositional Logic

To prove validity of

F = ((F1 Λ F2 Λ … Λ Fn) → G)

we shall attempt to prove that

~F = (F1 Λ F2 Λ … Λ Fn Λ ~G)

is unsatisfiable

Steps for Proof by Resolution Refutation:

1. Convert of Clausal Form /Conjunctive Normal Form (CNF, Product of Sums).
2. Generate new clauses using the resolution rule.
3. At the end, either False will be derived if the formula ~F is unsatisfiable implying F is valid.

If Asha is elected VP then Rajat is chosen as GSec and Bharati is chosen as Treasurer. Rajat is not chosen as G-Sec. Therefore Asha is not elected VP.

F1: (a → (b Λ c)) = (~a V b) Λ (~a V c)

F2: ~b, G: ~a, ~G: a

Clauses of Clause Form:
~F= (C1 Λ C2 Λ C3 Λ C4)
where: C1: (~a V b)
C2: (~a V c)
C3: ~b
C4: a
To prove that ~F is False

Resolution Rule: Let C1 = a V b and C2 = ~a V c

then a new clause C3 = b V c can be derived.

(Proof by showing that ((C1 Λ C2) → C3) is a valid formula).

To prove unsatisfiability use the Resolution Rule repeatedly to reach a situation where we have two contradictory clauses of the form C1 = a and C2 = ~a from which False can be derived.

If the propositional formula is satisfiable then we will not reach a contradiction and eventually no new clauses will be derivable.

For propositional logic the procedure terminates.

Resolution Rule is Sound and Complete

Applying Resolution Refutation for Propositional Logic

Resolution Rule:

Let C1 = a V b and C2 = ~a V c

then a new clause C3 = b V c can be derived.

(Proof by showing that ((C1 Λ C2) → C3) is a valid formula).

To prove unsatisfiability use the Resolution Rule repeatedly to reach a situation where we have two contradictory clauses of the form C1 = a and C2 = ~a from which False can be derived.

If the propositional formula is satisfiable then we will not reach a contradiction and eventually no new clauses will be derivable.

For propositional logic the procedure terminates.

Resolution Rule is Sound and Complete

If Asha is elected VP then Rajat is chosen as G-Sec and Bharati is chosen as Treasurer. Rajat is not chosen as G-Sec. Therefore Asha is not elected VP.

F1: (a → (b Λ c)) = (~a V b) Λ (~a V c)

F2: ~b

G: ~a

~G: a

Clauses of Clause Form: ~F = (C1 Λ C2 Λ C3 Λ C4)
where: C1: (~a V b)
C2: (~a V c)
C3: ~b
C4: a
To prove that ~F is False

New Clauses Derived
C5: ~a (Using C1 and C3)
C6: False (using C4 and C5)

Applying Resolution Refutation for Propositional Logic

Let C1 = a V b and C2 = ~a V c

then a new clause C3 = b V c can be derived.

(Proof by showing that ((C1 Λ C2) → C3) is a valid formula).

To prove unsatisfiability use the Resolution Rule repeatedly to reach a situation where we have two contradictory clauses of the form C1 = a and C2 = ~a from which False can be derived.

If the propositional formula is satisfiable then we will not reach a contradiction and eventually no new clauses will be derivable.

For propositional logic the procedure terminates.

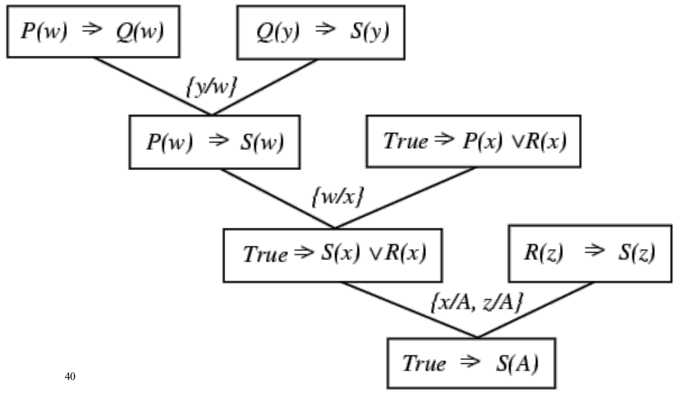
Resolution Rule is Sound and Complete

Rajesh either took the bus or came by cycle to class. If he came by cycle or walked to class he arrived late. Rajesh did not arrive late. Therefore he took the bus to class.

Resolution in first-order logic

- Given sentences
  $P_1 \vee ... \vee P_n$
  $Q_1 \vee ... \vee Q_m$
- in conjunctive normal form (CNF):
  – each P_i and Q_i is a literal, i.e., a positive or negated predicate symbol with its terms,
- if P_j and $\neg Q_k$ **unify** with substitution list θ, then derive the resolvent sentence:
  $subst(\theta, P_1 \vee ... \vee P_{j-1} \vee P_{j+1} ... P_n \vee Q_1 \vee ... Q_{k-1} \vee Q_{k+1} \vee ... \vee Q_m)$
- Example
  – from clause **$P(x, f(a)) \vee P(x, f(y)) \vee Q(y)$**
  – and clause **$\neg P(z, f(a)) \vee \neg Q(z)$**
  – derive resolvent **$P(z, f(y)) \vee Q(y) \vee \neg Q(z)$**
  – using **$\theta = \{x/z\}$**

A resolution proof tree



Resolution refutation

- If Will goes, Jane will go
- ~W V J
- If doesn't go, Jane will still go
- W V J
- Will Jane go?
- |= J?

J ∨ J = J

Don't need to use other equivalences if we use resolution in refutation style

~J	*~W*
~W ∨ J	
W ∨ J	*J*

Resolution refutation

- Given a consistent set of axioms KB and goal sentence Q, show that KB |= Q
- Proof by contradiction: Add ¬Q to KB and try to prove false.
  i.e., (KB |- Q) ↔ (KB ∨ ¬Q |- False)
- Resolution is refutation complete: it can establish that a given sentence Q is entailed by KB, but can't (in general) be used to generate all logical consequences of a set of sentences
- Also, it cannot be used to prove that Q is **not entailed** by KB.
- Resolution won't always give an answer since entailment is only semidecidable
  – And you can't just run two proofs in parallel, one trying to prove Q and the other trying to prove ¬Q, since KB might not entail either one

Resolution Refutation for Predicate Logic

Given a formula F which we wish to check for validity, we first check if there are any free variables. We then quantify all free variables universally.

Create F' = ~F and check for unsatisfiability of F'

STEPS:

Conversion to Clausal (CNF) Form:

- Handling of Variables and Quantifiers, Ground Instances

Applying the Resolution Rule:

- Concept of Unification
- Principle of Most General Unifier (mgu)
- Repeated application of Resolution Rule using mgu

F1: Vx(goes(Mary, x) → goes(Lamb, x))

F2: goes(Mary, School)

G: goes(Lamb, School)

To prove: (F1 Λ F2) → G) is valid

CONVERSION TO CLAUSAL FORM IN PREDICATE LOGIC

1. Remove implications and other Boolean symbols converting to equivalent forms using ~, V, Λ
2. Move negates (~) inwards as close as possible
3. Standardize (Rename) variables to make them unambiguous
4. Remove Existential Quantifiers by an appropriate new function /constant symbol taking into account the variables dependent on the quantifier (Skolemization)
5. Drop Universal Quantifiers
6. Distribute V over Λ and convert to CNF

Resolution Refutation for Predicate Logic

We need answers to the following questions

- How to convert FOL sentences to conjunctive normal form (a.k.a. CNF, clause form):
  - **normalization and skolemization**
- How to unify two argument lists, i.e., how to find their most general unifier (**mgu**) θ:
  - **unification**
- How to determine which two clauses in KB should be resolved next (among all resolvable pairs of clauses) :
  - **resolution (search) strategy**

Converting to CNF

Converting sentences to CNF

1. (a) Eliminate all ↔ connectives

   $(P \leftrightarrow Q) \Rightarrow ((P \rightarrow Q) \wedge (Q \rightarrow P))$

   (b) Eliminate all → connectives

   $(P \rightarrow Q) \Rightarrow (\neg P \vee Q)$

2. Reduce the scope of each negation symbol to a single predicate

   $\neg\neg P \Rightarrow P$

   $\neg(P \vee Q) \Rightarrow \neg P \wedge \neg Q$

   $\neg(P \wedge Q) \Rightarrow \neg P \vee \neg Q$

   $\neg(\forall x)P \Rightarrow (\exists x)\neg P$

   $\neg(\exists x)P \Rightarrow (\forall x)\neg P$

3. Standardize variables: rename all variables so that each quantifier has its own unique variable name

   $(\forall x: P(x)) \vee (\exists x: Q(x)) \equiv (\forall x: P(x)) \vee (\exists y: Q(y))$

Converting sentences to clausal form
Skolem constants and functions

4. Move all quantifiers to the left without changing their relative order.

$(\forall x: P(x)) \vee (\exists y: Q(y)) \equiv \forall x: \exists y: (P(x) \vee (Q(y))$

Eliminate existential quantification $\exists$ by introducing Skolem constants/functions (Skolemization).

$(\exists x)P(x) \Rightarrow P(c)$

c is a Skolem constant (a brand-new constant symbol that is not used in any other sentence)

$(\forall x)(\exists y)P(x,y) \Rightarrow (\forall x)P(x, f(x))$

since $\exists$ is within the scope of a universally quantified variable, use a **Skolem function f** to construct a new value that **depends on** the universally quantified variable

f must be a brand-new function name not occurring in any other sentence in the KB.

E.g., $(\forall x)(\exists y)loves(x,y) \Rightarrow (\forall x)loves(x,f(x))$

In this case, f(x) specifies the person that x loves

Converting sentences to clausal form

5. Remove universal quantifiers by (1) moving them all to the left end; (2) making the scope of each the entire sentence; and (3) dropping the "prefix" part

   Ex: $(\forall x)P(x) \Rightarrow P(x)$

6. Put into conjunctive normal form (conjunction of disjunctions) using distributive and associative laws

   $(P \wedge Q) \vee R \Rightarrow (P \vee R) \wedge (Q \vee R)$

   $(P \vee Q) \vee R \Rightarrow (P \vee Q \vee R)$

7. Split conjuncts into separate clauses
8. Standardize variables so each clause contains only variable names that do not occur in any other clause

Exercise : Conversion to Clausal Form

1. Remove implications and other Boolean symbols converting to equivalent forms using ~, V, Λ
2. Move negates (~) inwards as close as possible
3. Standardize (Rename) variables to make them unambiguous
4. Remove Existential Quantifiers by an appropriate new function /constant symbol taking into account the variables dependent on the quantifier (Skolemization)
5. Drop Universal Quantifiers
6. Distribute V over Λ and convert to CNF

$\forall x(\forall y(student(y) \rightarrow likes(x, y)) \rightarrow (\exists z(likes(z,x)))$

Exercise :Converting sentences to CNF

1. $(\forall x)(P(x) \rightarrow ((\forall y)(P(y) \rightarrow P(f(x,y))) \wedge \neg(\forall y)(Q(x,y) \rightarrow P(y))))$
2. Anyone who likes all animals is loved by someone:

Solution: Example 1 CNF

$(\forall x)(P(x) \rightarrow ((\forall y)(P(y) \rightarrow P(f(x,y))) \wedge \neg(\forall y)(Q(x,y) \rightarrow P(y))))$

1. Eliminate →

$(\forall x)(\neg P(x) \vee ((\forall y)(\neg P(y) \vee P(f(x,y))) \wedge \neg(\forall y)(\neg Q(x,y) \vee P(y))))$

2. Reduce scope of negation

$(\forall x)(\neg P(x) \vee ((\forall y)(\neg P(y) \vee P(f(x,y))) \wedge (\exists y)(Q(x,y) \wedge \neg P(y))))$

3. Standardize variables

$(\forall x)(\neg P(x) \vee ((\forall y)(\neg P(y) \vee P(f(x,y))) \wedge (\exists z)(Q(x,z) \wedge \neg P(z))))$

4. Eliminate existential quantification

$(\forall x)(\neg P(x) \vee ((\forall y)(\neg P(y) \vee P(f(x,y))) \wedge (Q(x,g(x)) \wedge \neg P(g(x)))))$

5. Drop universal quantification symbols

$(\neg P(x) \vee ((\neg P(y) \vee P(f(x,y))) \wedge (Q(x,g(x)) \wedge \neg P(g(x)))))$

6. Convert to conjunction of disjunctions

$(\neg P(x) \vee \neg P(y) \vee P(f(x,y))) \wedge (\neg P(x) \vee Q(x,g(x))) \wedge (\neg P(x) \vee \neg P(g(x)))$

7. Create separate clauses

$\neg P(x) \vee \neg P(y) \vee P(f(x,y))$

$\neg P(x) \vee Q(x,g(x))$

$\neg P(x) \vee \neg P(g(x))$

8. Standardize variables

$\neg P(x) \vee \neg P(y) \vee P(f(x,y))$

$\neg P(z) \vee Q(z,g(z))$

$\neg P(w) \vee \neg P(g(w))$

Solution Example 2 : CNF

Original sentence:

Anyone who likes all animals is loved by someone:

$\forall x\, [\forall y\, Animal(y) \Rightarrow Likes(x,y)] \Rightarrow [\exists y\, Loves(y,x)]$

1. Eliminate biconditionals and implications

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

2. Move ¬ inwards:

Recall: $\neg\forall x\ p \equiv \exists x\ \neg p$, $\neg \exists x\ p \equiv \forall x\ \neg p$

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

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

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

Either there is some animal that x doesn't like if that is not the case then someone loves x

Solution Example 2 : CNF cont.

3. **Standardize variables: each quantifier should use a different one**

$\forall x\, [\exists y\, Animal(y) \wedge \neg Likes(x,y)] \vee [\exists z\, Loves(z,x)]$

4. **Skolemize:**

$\forall x\, [Animal(A) \wedge \neg Likes(x,A)] \vee Loves(B,x)$

Everybody fails to love a particular animal A or is loved by a particular person B

Animal(cat) , Likes(marry, cat), Loves(john, marry)

Likes(cathy, cat), Loves(Tom, cathy)

a more general form of existential instantiation.

Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

$\forall x\, [Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$

(reason: animal y could be a different animal for each x.)

Solution Example 2 : CNF cont.

5. **Drop universal quantifiers:**

$[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$

(all remaining variables assumed to be universally quantified)

6. **Distribute ∨ over ∧ :**

$[Animal(F(x)) \vee Loves(G(x),x)] \wedge [\neg Loves(x,F(x)) \vee Loves(G(x),x)]$

Original sentence is now in CNF form – can apply same ideas to all sentences in KB to convert into CNF

Also need to include negated query. Then use resolution to attempt to derive the empty clause which show that the query is entailed by the KB

Unification

Unification

- Unification is a **"pattern-matching"** procedure
  - Takes two atomic sentences, called literals, as input
  - Returns "Failure" if they do not match and a substitution list, θ, if they do
- That is, *unify(p,q) = θ* means *subst(θ, p) = subst(θ, q)* for two atomic sentences, *p* and *q*
- **θ** is called the **most general unifier** (mgu)
- All variables in the given two literals are implicitly universally quantified
- To make literals match, replace (universally quantified) variables by terms
- *Unify* is a linear-time algorithm that returns the most general unifier (mgu), i.e., the shortest-length substitution list that makes the two literals match.
- In general, there is not a **unique** minimum-length substitution list, but unify returns one of minimum length
- A variable can never be replaced by a term containing that variable

  Example: x/f(x) is illegal.
- This "occurs check" should be done in the above pseudo-code before making the recursive calls

Unification examples

- Example:
  - parents(x, father(x), mother(Bill))
  - parents(Bill, father(Bill), y)
  - {x/Bill, y/mother(Bill)}
- Example:
  - parents(x, father(x), mother(Bill))
  - parents(Bill, father(y), z)
  - {x/Bill, y/Bill, z/mother(Bill)}
- Example:
  - parents(x, father(x), mother(Jane))
  - parents(Bill, father(y), mother(y))
  - Failure

Substitution, Unification, Resolution

Consider clauses:

- C1: ~studies(x,y) V passes(x,y)
- C2: studies(Madan,z)
- C3: ~passes(Chetan, Physics)
- C4: ~passes(w, Mechanics)

What new clauses can we derive by the resolution principle?

Ground Clause and a more general clause

Concept of substitution / unification

and the Most General Unifier (mgu)

Resolution Rule for Predicate

Calculus: Repeated Application of Resolution using mgu

Vx(Vy(student(y) → likes(x, y)) →(∃z(likes(z,x)))

Exercise :

F1: Vx(contractor(x) → ~dependable(x))
F2: ∃x(engineer(x) Λ contractor(x))
G: ∃x(engineer(x) Λ ~dependable(x))

F1: Vx(dancer(x) → graceful (x))
F2: student(Ayesha),
F3: dancer(Ayesha)
G: ∃x(student (x) Λ graceful(x))

Practice example

- Jack owns a dog.
- Every dog owner is an animal lover.
- No animal lover kills an animal.
- Either Jack or Curiosity killed the cat, who is named Tuna.
- Did Curiosity kill the cat?

Example: *Did Curiosity kill the cat*

- *Jack owns a dog. Every dog owner is an animal lover. No animal lover kills an animal. Either Jack or Curiosity killed the cat, who is named Tuna. Did Curiosity kill the cat?*
- These can be represented as follows:

A. $(\exists x)$ Dog(x) $\wedge$ Owns(Jack,x)
B. $(\forall x)$ $((\exists y)$ Dog(y) $\wedge$ Owns(x, y)) $\rightarrow$ AnimalLover(x)
C. $(\forall x)$ AnimalLover(x) $\rightarrow$ $((\forall y)$ Animal(y) $\rightarrow$ $\neg$Kills(x,y))
D. Kills(Jack,Tuna) $\vee$ Kills(Curiosity,Tuna)
E. Cat(Tuna)
F. $(\forall x)$ Cat(x) $\rightarrow$ Animal(x)
G. Kills(Curiosity, Tuna) ← *GOAL*

Convert to clause form

A1. (Dog(D)) ← *D is a skolem constant*
A2. (Owns(Jack,D))
B. ($\neg$Dog(y), $\neg$Owns(x, y), AnimalLover(x))
C. ($\neg$AnimalLover(a), $\neg$Animal(b), $\neg$Kills(a,b))
D. (Kills(Jack,Tuna), Kills(Curiosity,Tuna))
E. Cat(Tuna)
F. ($\neg$Cat(z), Animal(z))

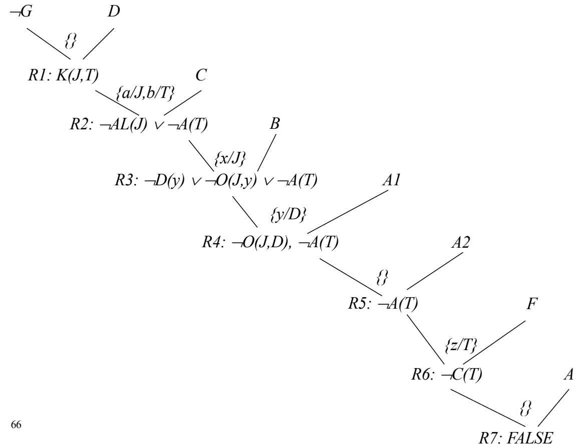
- **Add the negation of query:**

$\neg$G: ($\neg$Kills(Curiosity, Tuna))

The resolution refutation proof

R1: ¬G, D, {}	(Kills(Jack, Tuna))
R2: R1, C, {a/Jack, b/Tuna}	(~AnimalLover(Jack), ~Animal(Tuna))
R3: R2, B, {x/Jack}	(~Dog(y), ~Owns(Jack, y), ~Animal(Tuna))
R4: R3, A1, {y/D}	(~Owns(Jack, D), ~Animal(Tuna))
R5: R4, A2, {}	(~Animal(Tuna))
R6: R5, F, {z/Tuna}	(~Cat(Tuna))
R7: R6, E, {}	FALSE

- **The proof tree**



- Basic FOL inference algorithm (satisfiability check).

1. Use Skolemization to eliminate quantifiers
   1. Only universal quantifiers remain.
2. Convert to clausal form.
3. Use resolution + unification.

- This algorithm is **complete** (Gödel 1929).

Logic Programming

- Logic programming is a programming language paradigm in which logical assertions are viewed as programs, e.g : PROLOG
- A PROLOG program is described as a series of logical assertions, each of which is a Horn Clause.
  - A Horn Clause is a clause that has at most one positive literal.
  - Eg p, ¬ p V q etc are also Horn Clauses.
- The fact that PROLOG programs are composed only of Horn Clauses and not of arbitrary logical expressions has two important consequences.
- Because of uniform representation a simple & effective interpreter can be written.
- The logic of Horn Clause systems is decidable.

Logic Programming

- Even PROLOG works on backward reasoning.
- The program is read top to bottom, left to right and search is performed depth-first with backtracking.
- Syntactic difference between the logic and the PROLOG representations :
  - PROLOG interpreter has a fixed control strategy, so assertions in the PROLOG program define a particular search path to answer any question.
  - Where as Logical assertions define set of answers that they justify, there can be more than one answers, it can be forward or backward tracking .
- Control Strategy for PROLOG states that we begin with a problem statement, which is viewed as a goal to be proved.
- Look for the assertions that can prove the goal.
- To decide whether a fact or a rule can be applied to the current problem, invoke a standard unification procedure.
- Reason backward from that goal until a path is found that terminates with assertions in the program.
- Consider paths using a depth-first search strategy and use backtracking.
- Propagate to the answer by satisfying the conditions.

Prolog

- A logic programming language based on Horn clauses
  - Resolution refutation
  - Control strategy: goal-directed and depth-first
    - always start from the goal clause
    - always use the new resolvent as one of the parent clauses for resolution
    - backtracking when the current thread fails
    - complete for Horn clause KB
  - Support answer extraction (can request single or all answers)
  - Orders the clauses and literals within a clause to resolve non-determinism
    - Q(a) may match both Q(x) <= P(x) and Q(y) <= R(y)
    - A (sub)goal clause may contain more than one literals, i.e., <= P1(a), P2(a)
  - Use "closed world" assumption (negation as failure)
    - If it fails to derive P(a), then assume ~P(a)

Representation in logic

- $\forall x : pet(x) \wedge small(x) \rightarrow apartment(x)$
- $\forall x : cat(x) \vee dog(x) \rightarrow pet(x)$
- $\forall x : poodle(x) \rightarrow dog(x) \wedge small(x)$
- Poodle(abs)

Representation in PROLOG

- Apartment (x) :- pet(x), small(x)
- Pet (x) :- dog (x)
- Dog (x) :- poodle (x)
- Small(x) :- poodle (x)
- Poodle(abs)

Syntax and Rule

- .pl files contain lists of clauses
- Clauses can be either facts or rules

Predicate, arity 1 (male/1)

```
male(bob).
```

Terminates a clause

```
male(harry).
```

Argument to predicate

```
child(bob,harry).
son(X,Y):-
```

Indicates a rule

```
     male(X),child(X,Y).
```

"and"

- Rules combine facts to increase knowledge of the system

```
son(X,Y):-
  male(X),child(X,Y).
```

- X is a son of Y if X is male and X is a child of Y

Questions

- In Prolog the queries are statements called directive
- Syntactically, directives are clauses with an empty left-hand side.
  - Example : ? - grandparent(X, W).
  - This query is interpreted as : Who is a grandparent of X?
- The result of executing a query is either success or failure
  - Success, means the goals specified in the query holds according to the facts and rules of the program.
  - Failure, means the goals specified in the query does not hold according to the facts and rules of the program
- Ask the Prolog virtual machine questions
- Composed at the ?- prompt
- Returns values of bound variables and yes or no

```
?- son(bob, harry).
yes
?- king(bob, france).
no
```

- Can bind answers to questions to variables
- Who is bob the son of? (X=harry)

```
?- son(bob, X).
```

- Who is male? (X=bob, harry)

```
?- male(X).
```

- Is bob the son of someone? (yes)

```
?- son(bob, _).
```

- No variables bound in this case!

Backtracking

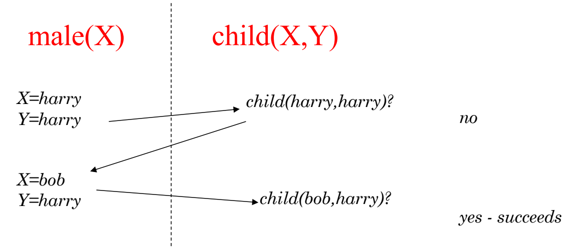
- How are questions resolved?

?- son(X,harry).

- Recall the rule:

```
son(X,Y):- male(X),child(X,Y).
```

- Y is bound to the atom "harry" by the question.



Applications

- Intelligent systems
- Complicated knowledge databases
- Natural language processing
- Logic data analysis

Prolog Exercise : Knowledge Base in FOL

- 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.
- **Exercise: Formulate this knowledge in FOL.**

Query: Criminal(West)?

Knowledge Base in FOL

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.

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

American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

Nono … has some missiles, i.e., ∃x Owns(Nono,x) ∧ Missile(x):

Owns(Nono,M_1) and Missile(M_1)

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

Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

Missiles are weapons:

Missile(x) ⇒ Weapon(x)

An enemy of America counts as "hostile“:

Enemy(x,America) ⇒ Hostile(x)

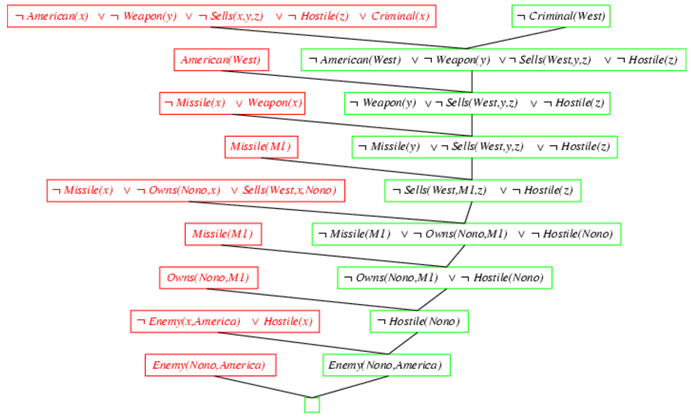
West, who is American …

American(West)

The country Nono, an enemy of America …

Enemy(Nono,America)

Resolution proof



Logic programming: Prolog

- Program = set of clauses = `head :- literal`$_1$`, … literal`$_n$.

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
Missile(m1).
Owns(nono,m1).
Sells(west,X,nono):- Missile(X) Owns(nono,X).
weapon(X):- missile(X).
hostile(X) :- enemy(X,america).
american(west)
```

`Query : criminal(west)?`

`Query: criminial(X)?`

Limitations of Resolution (Evolution of Natural Deduction)

- The previous method of resolution brings uniformity, everything looks the same. Hence at times, it becomes very difficult to pick the statement that may be useful in solving the problem.
- As we convert everything into clause form, we loose important heuristic information.
- Eg. We believe that all judges who are not crooked are well-educated
- ¥x : judge(x) À ¬ crooked (x) □ educated(x)
- In the clause form it will take the following shape
- ¬ judge (x) V crooked(x) V educated(x)

Natural Deduction

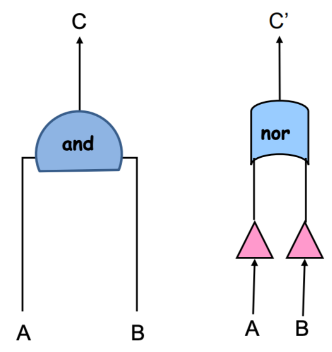
- Another problem with the use of resolution is that people do not think in resolution.
- Computers are still poor at proving very hard things, hence we need a practical standpoint. (focus is on interaction)
- To facilitate it we led to Natural Deduction.
- It describes a blend of techniques, used in combination to solve problems that are not traceable by any one method alone.
- One common technique is to talk about objects involved in the predicate and not the predicate itself.

Reduction to satisfiability problem

- Boolean satisfiability problem (SAT) : mid-1990's
  - given a formula, to check whether it is satisfiable.
  - importance in mantheoretical computer science, complexity theory, algorithmics, cryptography and artificial intelligence.
- SAT: Model Finding
- Find assignments to variables that makes a formula true
- The problem of determining the satisfiability of sentences in propositional logic

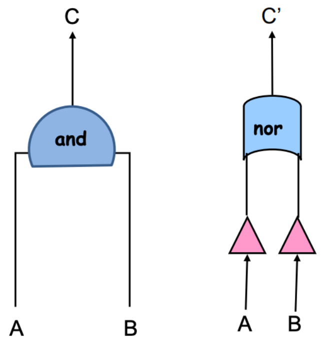
- Why study Satisfiability?
  - Canonical NP complete problem.
    - several hard problems modeled as SAT
  - Tonne of applications
  - State-of-the-art solvers superfast

Testing Circuit Equivalence



- Do two circuits compute the same function?
- Circuit optimization
- Is there input for which the two circuits compute different values?

Testing Circuit Equivalence



- Do two circuits compute the same function?

  $C \equiv A \wedge B$

  $C' \equiv \neg\ D \vee E$

  $D \equiv \neg A$

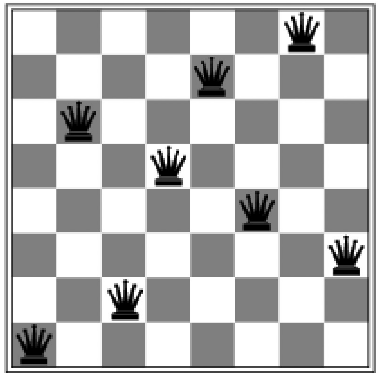
  $E \equiv \neg B$

  Resolution :

  $\neg(C \equiv C')$

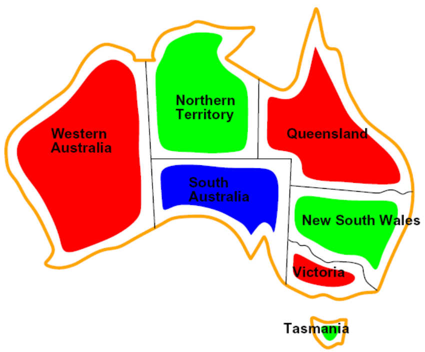
SAT Translation of N-Queens

- At least one queen each column:
  (Q11 v Q12 v Q13 v ... v Q18)
  (Q21 v Q22 v Q23 v ... v Q28)
  …
- No attacks:
  (~Q11 v ~Q12)
  (~Q11 v ~Q22)
  (~Q11 v ~Q21)
  …

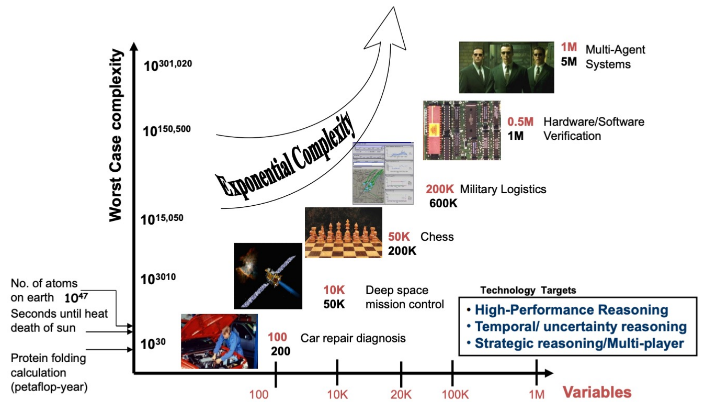


SAT Translation of Graph Coloring

- A new SAT Variable for var-val pair
  - $X_{WA\text{-}r}$, $X_{WA\text{-}g}$, $X_{WA\text{-}b}$, $X_{NT\text{-}r}$…
- Each var has at least 1 value
  - $X_{WA\text{-}r}$ v $X_{WA\text{-}g}$ v $X_{WA\text{-}b}$
- No var has two values
  - $\sim X_{WA\text{-}r}$ v $\sim X_{WA\text{-}g}$
  - $\sim X_{WA\text{-}r}$ v $\sim X_{WA\text{-}b}$
- Constraints
  - $\sim X_{WA\text{-}r}$ v $\sim X_{NT\text{-}r}$



Real-World Reasoning
Tackling inherent computational complexity



Example domains cast in propositional reasoning system (variables, rules).

Summary

- Logical agents apply inference to a knowledge base to derive new information and make decisions
- Basic concepts of logic:
  - Syntax: formal structure of sentences
  - Semantics: truth of sentences wrt models
  - Entailment: necessary truth of one sentence given another
  - Inference: deriving sentences from other sentences
  - Soundness: derivations produce only entailed sentences
  - Completeness: derivations can produce all entailed sentences
- FC and BC are linear time, complete for Horn clauses
- Resolution is a sound and complete inference method for propositional and first-order logic
- SAT: Find assignments to variables that makes a formula true

Next :

- Module 5: AI Planning

References

- ***Artificial Intelligence*** *by Elaine Rich & Kevin Knight*, Third Ed, Tata McGraw Hill
- ***Artificial Intelligence and Expert System*** *by Patterson*
- http://www.cs.rmit.edu.au/AI-Search/Product/
- http://aima.cs.berkeley.edu/demos.html (for more demos)
- ***Artificial Intelligence and Expert System*** *by Patterson*
- Slides adapted from CS188 Instructor: Anca Dragan, University of California, Berkeley
- Slides adapted from CS60045 ARTIFICIAL INTELLIGENCE