Lecture 7

Logical Agents
Inference in First Order Logic

Marco Chiarandini

Deptartment of Mathematics & Computer Science
University of Southern Denmark

Slides by Stuart Russell and Peter Norvig

Course Overview

- ✔ Introduction
  - ✔ Artificial Intelligence
  - ✔ Intelligent Agents
- ✔ Search
  - ✔ Uninformed Search
  - ✔ Heuristic Search
- ✔ Adversarial Search
  - ✔ Minimax search
  - ✔ Alpha-beta pruning
- Knowledge representation and Reasoning
  - ✔ Propositional logic
  - ✔ First order logic
  - Inference
- Uncertain knowledge and Reasoning
  - Probability and Bayesian approach
  - Bayesian Networks
  - Hidden Markov Chains
  - Kalman Filters
- Learning
  - Decision Trees
  - Maximum Likelihood
  - EM Algorithm
  - Learning Bayesian Networks
  - Neural Networks
  - Support vector machines

Summary

First-order logic:
– objects and relations are semantic primitives
– syntax: constants, functions, predicates, equality, quantifiers
Increased expressive power: sufficient to define wumpus world

Situation calculus:
– conventions for describing actions and change in FOL
– can formulate planning as inference on a situation calculus KB

Outline

- ♢ Reducing first-order inference to propositional inference
- ♢ Unification
- ♢ Generalized Modus Ponens
- ♢ Forward and backward chaining
- ♢ Logic programming
- ♢ Resolution

A brief history of reasoning

450b.c.	Stoics	propositional logic, inference (maybe)
322b.c.	Aristotle	"syllogisms" (inference rules), quantifiers
1565	Cardano	probability theory (propositional logic + uncertainty)
1847	Boole	propositional logic (again)
1879	Frege	first-order logic
1922	Wittgenstein	proof by truth tables
1930	Gödel	$\exists$ complete algorithm for FOL
1930	Herbrand	complete algorithm for FOL (reduce to propositional)
1931	Gödel	$\neg\exists$ complete algorithm for arithmetic
1960	Davis/Putnam	"practical" algorithm for propositional logic
1965	Robinson	"practical" algorithm for FOL—resolution

Definitions

For a predicate calculus expression X and an interpretation I:

- If X has a value of T under I and a particular variable assignment, then I is said to satisfy X.
- If I satisfies X for all variable assignments, then I is a model of X
- X is satisfiable if and only if there exist an interpretation and variable assignment that satisfy it; otherwise, it is unsatisfiable
- If a set of expressions is not satisfiable, it is said to be inconsistent
- If X has a value T for all possible interpretations, X is said to be valid. Eg.: $(p(X) \wedge \neg p(X))$ while $\exists X(P(X) \vee \neg p(X))$

Definition

A Proof Procedure is a combination of an inference rule and an algorithm for applying that rule to a set of logical expressions to generate new sentences.

Eg: Resolution inference rule.

Definition

A predicate calculus expression X logically follows from a set S of predicate calculus expressions if every interpretation and variable assignment that satisfies S also satisfies X.

An inference rule is sound if every predicate calculus expression produced by the rule from a set S of predicate calculus expressions also logically follows from S.

An inference rule is complete if, given a set S of predicate calculus expressions, the rule can infer every expression that logically follows from S.

Rules of Inference for Propositions

Rule of inference	Name	Rule of inference	Name
$\dfrac{p \quad p \rightarrow q}{\therefore\ q}$	Modus Ponens	$\dfrac{p}{\therefore\ p \vee q}$	Addition
$\dfrac{\neg q \quad p \rightarrow q}{\therefore\ \neg p}$	Modus tollens	$\dfrac{p \vee q}{\therefore\ p}$	Simplification
$\dfrac{p \rightarrow q \quad q \rightarrow r}{\therefore\ p \rightarrow r}$	Hypothetical syllogism	$\dfrac{p \quad q}{\therefore\ p \vee q}$	Conjunction
$\dfrac{p \vee q \quad \neg p}{\therefore\ q}$	Disjunctive syllogism	$\dfrac{p \vee q \quad \neg p \vee r}{\therefore\ q \vee r}$	Resolution

Rules of Inference for Quantified Statements

Rule of inference	Name
$\dfrac{\forall x\, P(x)}{\therefore\ P(c)}$	Universal instantiation
$\dfrac{P(c) \text{ for an arbitrary } c}{\therefore\ \forall x P(x)}$	Universal generalization
$\dfrac{\exists x\, P(x)}{\therefore\ P(c) \text{ for some element } c}$	Existential instantiation
$\dfrac{P(c) \text{ for some element } c}{\therefore\ \exists x P(x)}$	Existential generalization

Universal instantiation (UI)

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

$$\frac{\forall\, v \quad \alpha}{\therefore\ \text{Subst}(\{v/c\}, \alpha)}$$

for any variable v and ground term c. (Note, here we used prolog notation.)

E.g., $\forall\, x \quad King(x) \wedge Greedy(x) \implies Evil(x)$ yields

$$\begin{array}{l} King(John) \wedge Greedy(John) \implies Evil(John) \\ King(Richard) \wedge Greedy(Richard) \implies Evil(Richard) \\ King(Father(John)) \wedge Greedy(Father(John)) \implies Evil(Father(John)) \\ \vdots \end{array}$$

Existential instantiation (EI)

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

$$\frac{\exists\, v \quad \alpha}{\therefore\ \text{Subst}(\{v/k\}, \alpha)}$$

E.g., $\exists\, x \quad Crown(x) \wedge OnHead(x, John)$ yields

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

provided C_1 is a new constant symbol, called a Skolem constant

Another example: from $\exists\, x \quad d(x^y)/dy = x^y$ we obtain

$$d(e^y)/dy = e^y$$

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:

$$\forall x \;\; King(x) \land Greedy(x) \implies Evil(x)$$
$$King(John)$$
$$Greedy(John)$$
$$Brother(Richard, John)$$

Instantiating the universal sentence in **all possible** ways, we have

$$King(John) \land Greedy(John) \implies Evil(John)$$
$$King(Richard) \land Greedy(Richard) \implies Evil(Richard)$$
$$King(John)$$
$$Greedy(John)$$
$$Brother(Richard, John)$$

The new KB is propositionalized: proposition symbols are

$King(John),\ Greedy(John),\ Evil(John), King(Richard),$ etc.

and can therefore be solved by the methods seen with propositional logic

Reduction to propositional inference (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 variables and function symbols, there are infinitely many ground terms,
  e.g., $Father(Father(Father(John)))$
- Theorem: Herbrand (1930). If a sentence α is entailed by an FOL KB,
  it is entailed by a **finite** subset of the propositional KB
- Idea: For $n = 0$ to ∞ do
  create a propositional KB by instantiating with depth-n terms
  see if α is entailed by this KB
- Problem: works if α is entailed, loops if α is not entailed
- Theorem: Turing (1936), Church (1936), entailment in FOL is semidecidable

Problems with propositionalization

Propositionalization seems to generate lots of irrelevant sentences.
E.g., from

$$\forall x \;\; King(x) \land Greedy(x) \implies Evil(x)$$
$$King(John)$$
$$\forall y \;\; Greedy(y)$$
$$Brother(Richard, John)$$

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 σ such that $King(x)$ and $Greedy(x)$ match $King(John)$ and $Greedy(y)$

$\sigma = \{x/John, y/John\}$ works

Unify$(\alpha, \beta) = \sigma$ if $\alpha\sigma = \beta\sigma$

p	q	σ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x/Jane\}$
$Knows(John, x)$	$Knows(y, OJ)$	$\{x/OJ, y/John\}$
$Knows(John, x)$	$Knows(y, Mother(y))$	$\{y/John, x/Mother(John)\}$
$Knows(John, x)$	$Knows(x, OJ)$	$fail$

Standardizing apart: rename variables to eliminate name overlap, e.g., $Knows(z, OJ)$

Generalized Modus Ponens (GMP)

Any inference in FOL has to use unification
Here is an inference rule with the use of unification

$$\frac{p_1{}',\ p_2{}',\ \ldots,\ p_n{}' \qquad (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{\therefore\ q\sigma} \qquad \text{where } p_i{}'\sigma = p_i\sigma \text{ for all } i$$

$p_1{}'$ is $King(John)$ — p_1 is $King(x)$
$p_2{}'$ is $Greedy(y)$ — p_2 is $Greedy(x)$
σ is $\{x/John, y/John\}$ — q is $Evil(x)$
$q\sigma$ is $Evil(John)$

GMP used with KB of definite clauses (**exactly** one positive literal)
All variables assumed universally quantified

Soundness of GMP

Need to show that

$$p_1{}',\ \ldots,\ p_n{}',\ (p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models q\sigma$$

provided that $p_i{}'\sigma = p_i\sigma$ for all i

Lemma: For any definite clause p, we have $p \models p\sigma$ by UI
1. $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models (p_1 \wedge \ldots \wedge p_n \Rightarrow q)\sigma = (p_1\sigma \wedge \ldots \wedge p_n\sigma \Rightarrow q\sigma)$
2. $p_1{}',\ \ldots,\ p_n{}' \models p_1{}' \wedge \ldots \wedge p_n{}' \models p_1{}'\sigma \wedge \ldots \wedge p_n{}'\sigma$
3. From 1 and 2, $q\sigma$ follows by ordinary Modus Ponens

Unification

Unification: search substitution that match two expressions

- constants (ground instances) cannot be substituted
- only variables can be substituted
- cannot substitute x by $p(x)$ $\rightsquigarrow$ creates infinite regression
  occur check
- a variable can be substituted with another variable
- future substitutions must be consistent (substitution sequence)

Composition of substitutions:

$$\{Y/X, Z/W\}; \{X/V\}; \{V/a, W/f(b)\}$$

Unification

Unifiers must be as general as possible otherwise eliminate possibility for future solutions:
Eg: $p(X), p(Y)$ and $\{X/fred, Y/fred\}$

Definition

If μ is any unifier of expressions E and σ is a most general unifier then for μ applied to E there exists μ' such that $E\sigma = E\sigma\mu'$ where $E\mu$ and $E\sigma\mu'$ is the composition of unifiers.

mgu is unique (except for relabelling)

An Unification Algorithm

```
function unify(E1, E2);
  begin
    case
      both E1 and E2 are constants or the empty list:          %recursion stops
        if E1 = E2 then return {}
          else return FAIL;
      E1 is a variable:
        if E1 occurs in E2 then return FAIL
          else return {E2/E1};
      E2 is a variable:
        if E2 occurs in E1 then return FAIL
          else return {E1/E2}
      either E1 or E2 are empty then return FAIL       %the lists are of different sizes
      otherwise:                                          %both E1 and E2 are lists
        begin
            HE1 := first element of E1;
            HE2 := first element of E2;
            SUBS1 := unify(HE1,HE2);
            if SUBS1 : = FAIL then return FAIL;
            TE1 := apply(SUBS1, rest of E1);
            TE2 : = apply (SUBS1, rest of E2);
            SUBS2 : = unify(TE1, TE2);
            if SUBS2 = FAIL then return FAIL;
                else return composition(SUBS1,SUBS2)
        end
  end                                                            %end case
end
```


Conversion to Clausal Form

Definition

We call clausal form any formula where all variables are universally quantified and the quantifier-free part is in CNF.

Any formula can be transformed into an equisatisfiable clausal form.
We obtain it by a number of transformations.

Forward Chaining

Definition

Definite Clauses: Disjunction clauses of literals of which at most one is positive.
(Eg. $\neg p \vee \neg q \vee r$)
They are are equivalent to implications whose premise is a conjunction of positive literals and conclusion is a single positive literal
(Eg: $(p \wedge q) \implies r$)

It is advisable building systems that only definite clauses so that reasoning is done by forward chaining rather than resolution that is much more costly.

Example knowledge base

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

Prove that Col. Thor is a criminal

Example knowledge base contd.

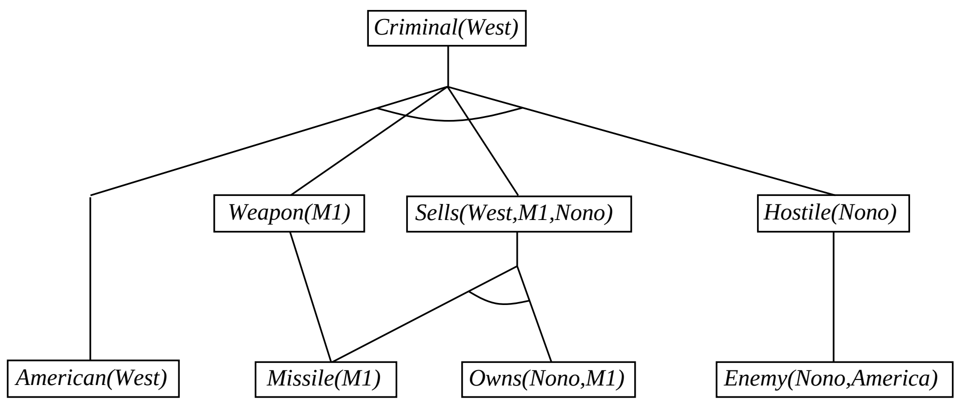
... it is a crime for an Dane to sell weapons to hostile nations:
$Dane(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \implies Criminal(x)$
Nono ... has some missiles, i.e., $\exists\, x\; Owns(Nono,x) \wedge Missile(x)$:
$Owns(Nono, M_1)$ and $Missile(M_1)$
... all of its missiles were sold to it by Colonel Thor
$\forall\, x\;\; Missile(x) \wedge Owns(Nono,x) \implies Sells(Thor,x,Nono)$
Missiles are weapons:
$Missile(x) \Rightarrow Weapon(x)$
An enemy of Denmark counts as "hostile":
$Enemy(x, Denmark) \implies Hostile(x)$
Thor, who is Dane ...
$Dane(Thor)$
The country Nono, an enemy of Denmark ...
$Enemy(Nono, Denmark)$

Forward chaining algorithm

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

repeat until *new* is empty
  new $\leftarrow \{\}$
  for each sentence *r* **in** *KB* **do**
    $(\, p_1 \wedge \ldots \wedge\; p_n \implies \; q) \leftarrow$ Standardize-Apart(*r*)
    for each δ such that $(p_1 \;\wedge\; \ldots \;\wedge\; p_n)\delta \;=\; (p'_1 \;\wedge\; \ldots \;\wedge\; p'_n)\delta$
      for some $p'_1, \ldots, p'_n$ in *KB*
      $q' \leftarrow$ Subst(δ, q)
      if q' is not a renaming of a sentence already in *KB* or *new*
then do
        add q' to *new*
        $\sigma \leftarrow$ Unify(q', α)
        if σ is not *fail* **then return** σ
  add *new* to *KB*
return *false*

Forward chaining proof



Properties of forward chaining

For first-order definite clauses the algorithm is:
- Sound because application of generalized modus ponens
- Complete (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

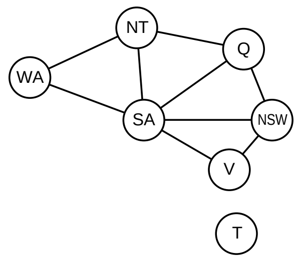
May not terminate in general (with functions) if α is not entailed
This is unavoidable: entailment with definite clauses is semidecidable

Efficiency of forward chaining

- Simple observation: no need to match a rule on iteration k if a premise wasn't added on iteration $k-1$
  $\Longrightarrow$ 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 $Missile(x)$ retrieves $Missile(M_1)$
  - Matching conjunctive premises against known facts is NP-hard

Forward chaining is widely used in deductive databases

Hard matching example



$$\begin{aligned}&Diff(wa,nt) \wedge Diff(wa,sa) \wedge \\ &\quad Diff(nt,q) Diff(nt,sa) \wedge \\ &\quad Diff(q,nsw) \wedge Diff(q,sa) \wedge \\ &\quad Diff(nsw,v) \wedge Diff(nsw,sa) \wedge \\ &\quad Diff(v,sa) \implies Colorable()\end{aligned}$$

$Diff(Red, Blue)$ $Diff(Red, Green)$
$Diff(Green, Red)$ $Diff(Green, Blue)$
$Diff(Blue, Red)$ $Diff(Blue, Green)$

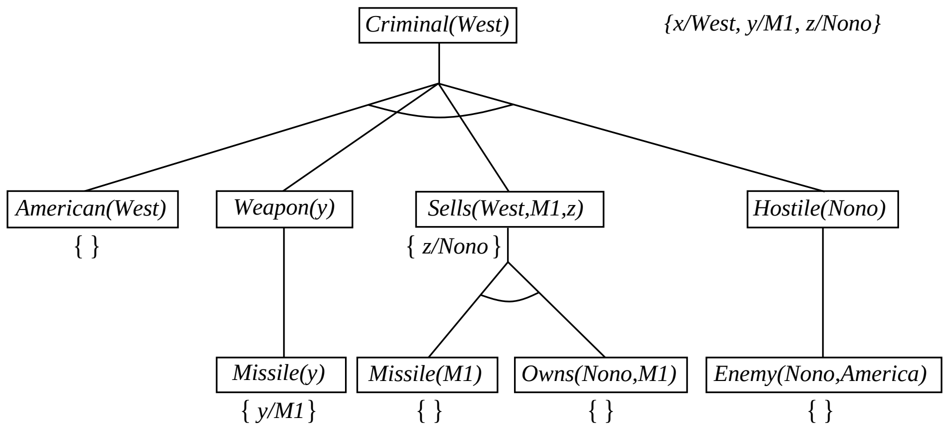
$Colorable()$ is inferred iff the CSP has a solution
CSPs include 3SAT as a special case, hence matching is NP-hard

Backward chaining algorithm

function FOL-BC-Ask(*KB*, *goals*, σ) **returns** a set of substitutions
inputs: *KB*, a knowledge base
goals, a list of conjuncts forming a query (σ already applied)
σ, the current substitution, initially the empty substitution $\{\,\}$
local variables: *answers*, a set of substitutions, initially empty

if *goals* is empty **then return** $\{\sigma\}$
$q' \leftarrow$ Subst(σ, First(*goals*))
for each sentence *r* **in** *KB*
where Standardize-Apart(*r*) = $(\, p_1 \;\wedge\; \ldots \;\wedge\; p_n \;\Rightarrow\; q)$
and $\sigma' \leftarrow$ Unify(q, q') succeeds
new_goals $\leftarrow [\, p_1, \ldots, p_n |$Rest(*goals*)$]$
answers $\leftarrow$ FOL-BC-Ask(*KB*, *new_goals*, Compose(σ', σ)) $\cup$ *answers*
return *answers*

Backward chaining example



Properties of backward chaining

Depth-first recursive proof search: space is linear in size of proof

Incomplete due to infinite loops
$\Longrightarrow$ fix by checking current goal against every goal on stack

Inefficient due to repeated subgoals (both success and failure)
$\Longrightarrow$ 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

	Logic programming	Ordinary programming
1.	Identify problem	Identify problem
2.	Assemble information	Assemble information
3.	Tea break	Figure out solution
4.	Encode information in KB	Program solution
5.	Encode problem instance as facts	Encode problem instance as data
6.	Ask queries	Apply program to data
7.	Find false facts	Debug procedural errors

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

Prolog systems

Basis: backward chaining with Horn clauses + bells & whistles

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

```
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

Resolution: brief summary

Full first-order version:

$$\frac{\ell_1 \vee \cdots \vee \ell_k \qquad m_1 \vee \cdots \vee m_n}{(\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\sigma}$$

where $\text{Unify}(\ell_i, \neg m_j) = \sigma$.

For example,

$$\frac{\neg Rich(x) \vee Unhappy(x) \qquad Rich(Ken)}{Unhappy(Ken)}$$

with $\sigma = \{x/Ken\}$

$\rightsquigarrow$Apply resolution steps to $CNF(KB \wedge \neg\alpha)$; complete for FOL

Conversion to CNF

Everyone who loves all animals is loved by someone:

$\forall x \;[\forall y \;\; Animal(y) \implies Loves(x,y)] \implies [\exists y \;\; Loves(y,x)]$

1. Eliminate biconditionals and implications

$$\forall x \;\; [\neg \forall y \;\; \neg Animal(y) \lor Loves(x,y)] \lor [\exists y \;\; Loves(y,x)]$$

2. Move $\neg$ inwards: $\neg \forall x, p \;\; \equiv \exists x \;\; \neg p$, $\;\; \neg \exists x, p \;\; \equiv \forall x \;\; \neg p$:

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

Conversion to CNF contd.

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