Foundations of Artificial Intelligence

First-Order Logic

CS472 – Fall 2007
Thorsten Joachims

First-Order Logic

- **Idea:**
  - Don't treat propositions as "atomic" entities.
- **First-Order Logic:**
  - Objects: cs472, fred, ph219, emptylist …
  - Relations/Predicates: is_Man(fred), Located(cs472, ph219) …
    - Note: Relations typically correspond to verbs
  - Functions: Pair(search,Pair(learning,Pair(kbsystems, emptylist)))
  - Connectives: ∧, ∨ , ¬, ⇒, ⇔
  - Quantifiers:
    - Universal: ∀ x: (is_Man(x) ⇒ is_Mortal(x))
    - Existential: ∃ y: (is_Father(y, fred))

Example: Representing Facts in First-Order Logic

1. **Lucy* is a professor**
2. **All professors are people.**
3. **Fuchs is the dean.**
4. **Deans are professors.**
5. **All professors consider the dean a friend or don't know him.**
6. **Everyone is a friend of someone.**
7. **People only criticize people that are not their friends.**
8. **Lucy criticized Fuchs.**

* Name changed for privacy reasons.

Example: Proof

Knowledge base:

- is-prof(lucy)
- ∀ x (is-prof(x) → is-person(x))
- is-dean(fuchs)
- ∀ x (is-dean(x) → is-prof(x))
- ∀ x (∀ y (is-prof(x) ∧ is-dean(y) → is-friend-of(y,x) ∨ ¬ knows(x, y)))
- ∀ x (∃ y (is-friend-of (y, x)))
- ∀ x (∀y (is-person(x) ∧ is-person(y) ∧ criticize (x,y) → ¬ is-friend-of (y,x)))
- criticize(lucy,fuchs)

Question: Is Fuchs no friend of Lucy?

¬ is-friend-of(fuchs,lucy)

Knowledge Engineering

1. **Identify the task.**
2. **Assemble the relevant knowledge.**
3. **Decide on a vocabulary of predicates, functions, and constants.**
4. **Encode general knowledge about the domain.**
5. **Encode a description of the specific problem instance.**
6. **Pose queries to the inference procedure and get answers.**
7. **Debug the knowledge base.**

Inference Procedures: Theoretical Results

- **There exist complete and sound proof procedures for propositional and FOL.**
  - Propositional logic
    - Use the definition of entailment directly. Proof procedure is exponential in *n*, the number of symbols.
    - In practice, can be much faster…
    - Polynomial-time inference procedure exists when KB is expressed as **Horn clauses:** $P_1 \wedge P_2 \wedge \ldots \wedge P_n \Rightarrow Q$ where the P_i and Q are non-negated atoms.
  - First-Order logic
    - Godel's completeness theorem showed that a proof procedure exists…
    - But none was demonstrated until Robinson's 1965 *resolution algorithm*.
    - Entailment in first-order logic is *semidecidable*.

Resolution Rule of Inference

General Rule:

Assume:	$E \vee E_{12} \vee \ldots \vee E_{1k}$
and	$\neg E \vee E_{22} \vee \ldots \vee E_{2l}$
Then:	$E_{12} \vee \ldots \vee E_{1k} \vee E_{22} \vee \ldots \vee E_{2l}$

Note: E_{ij} can be negated.

Example:

Assume:	$E_1 \vee E_2$	playing tennis or raining
and	$\neg E_2 \vee E_3$	not raining or working
Then:	$E_1 \vee E_3$	playing tennis or working

Algorithm: Resolution Proof

- **Negate the theorem to be proved, and add the result to the knowledge base.**
- **Bring knowledge base into conjunctive normal form (CNF)**
  - CNF: conjunctions of disjunctions
  - Each disjunction is called a clause.
- **Until there is no resolvable pair of clauses,**
  - Find resolvable clauses and resolve them.
  - Add the results of resolution to the knowledge base.
  - If NIL (empty clause) is produced, stop and report that the (original) theorem is true.
- **Report that the (original) theorem is false.**

Resolution Example: Propositional Logic

- **To prove: ¬ P**
- **Transform Knowledge Base into CNF**

	Regular	CNF
Sentence 1:	$P \rightarrow Q$	$\neg P \vee Q$
Sentence 2:	$Q \rightarrow R$	$\neg Q \vee R$
Sentence 3:	$\neg R$	$\neg R$

- **Proof**

1.	$\neg P \vee Q$	Sentence 1
2.	$\neg Q \vee R$	Sentence 2
3.	$\neg R$	Sentence 3
4.	*P*	Assume opposite
5.	*Q*	Resolve 4 and 1
6.	*R*	Resolve 5 and 2
7.	nil	Resolve 6 with 3

Resolution Example: FOL

Example: Prove bird (tweety)

Axioms:	**Regular**	**CNF**
1:	$\forall x: feathers(x) \rightarrow bird(x)$	$\neg feathers(x) \vee bird(x)$
2:	$feathers(tweety)$	$feathers(tweety)$
3:	$\neg bird(tweety)$	$\neg bird(tweety)$
4:		$\neg feathers(tweety)$

Resolution Proof

1. Resolve 3 and 1, specializing (i.e. "unifying") tweety for x. Add ¬feathers(tweety)
2. Resolve 4 and 2. Add NIL.

Resolution Theorem Proving

Properties of Resolution Theorem Proving:

- sound (for propositional and FOL)
- (refutation) complete (for propositional and FOL)

Procedure may seem cumbersome but note that can be easily automated. Just "smash" clauses until empty clause or no more new clauses.

Unification

Unify procedure: Unify(P,Q) takes two atomic (i.e. single predicates) sentences P and Q and returns a substitution that makes P and Q identical.

Rules for substitutions:

- Can replace a variable by a constant.
- Can replace a variable by a variable.
- Can replace a variable by a function expression, as long as the function expression does not contain the variable.

Unifier: a substitution that makes two clauses resolvable.

$$v_1/C; v_2/v_3; v_4/f(\ldots)$$

Unification - Purpose

Given:

$\neg$ *Knows* (*John*, *x*) $\vee$ *Hates* (*John*, *x*)

Knows (*John*, *Jim*)

Derive:

Hates (*John*, *Jim*)

Unification:

$unify(Knows(John,x), Knows(John,Jim)) = \{x/Jim\}$

Need unifier {*x/Jim*} for resolution to work.

Add to knowledge base:

$\neg Knows(John, Jim) \vee Hates(John, Jim)$

Unification (example)

Who does John hate?

$\exists x$: *Hates* (*John*, *x*)

Knowledge base (in clause form):

1. $\neg$ *Knows* (*John*, *v*) $\vee$ *Hates* (*John*, *v*)
2. *Knows* (*John*, *Jim*)
3. *Knows* (*y*, *Leo*)
4. *Knows* (z, *Mother*(z))
5. $\neg$ *Hates* (*John*, *x*) (since $\neg \exists x$: *Hates* (*John*, *x*) $\Leftrightarrow \forall$ x: $\neg$Hates(John,x))
   Resolution with 5 and 1:
   unify(*Hates*(*John*, *x*), *Hates*(*John*, *v*)) = {x/v}
6. $\neg$ *Knows* (*John*, v)
   Resolution with 6 and 2:
   unify(*Knows*(*John*, v), *Knows*(*John*, *Jim*))= {v/Jim}
   or resolution with 6 and 3:
   unify(*Knows* (*John*, v), *Knows* (y, Leo)) = {y/John, v/Leo}
   or Resolution with 6 and 4:
   unify(*Knows* (*John*, v), *Knows* (z, Mother(z))) = {z/John, v/Mother(z)}

Answers:

1. Hates(John,x) with {x/v, v/Jim} (i.e. John hates Jim)
2. Hates(John,x) with {x/v, y/John, v/Leo} (i.e. John hates Leo)
3. Hates(John,x) with {x/v, v/Mother(z), z/John} (i.e. John hates his mother)

Most General Unifier

In cases where there is more than one substitution choose the one that makes the least commitment (most general) about the bindings.

UNIFY (*Knows* (*John*,*x*), *Knows* (*y*,*z*))
= {*y / John, x / z*}
not {*y / John, x / z, z / Freda*}
not {*y / John, x / John, z / John*}
…

See R&N for general unification algorithm. O(n^2) with Refutation

Converting More Complicated Sentences to CNF

Sentence:

$$\forall x : brick(x) \rightarrow ((\exists y : on(x,y) \wedge \neg pyramid(y)) \wedge (\neg \exists y : on(x,y) \wedge on(y,x)) \wedge (\forall y : \neg brick(y) \rightarrow \neg equal(x,y)))$$

CNF:

$\neg brick(x) \vee on(x, support(x))$

$\neg brick(w) \vee \neg pyramid(support(w))$

$\neg brick(u) \vee \neg on(u,y) \vee \neg on(y,u)$

$\neg brick(v) \vee brick(z) \vee \neg equal(v,z)$

1. Eliminate Implications

Substitute $\neg E_1 \vee E_2$ for $E_1 \rightarrow E_2$

$$\forall x : brick(x) \rightarrow ((\exists y : on(x,y) \wedge \neg pyramid(y)) \wedge (\neg \exists y : on(x,y) \wedge on(y,x)) \wedge (\forall y : \neg brick(y) \rightarrow \neg equal(x,y))$$

$$\forall x : \neg brick(x) \vee ((\exists y : on(x,y) \wedge \neg pyramid(y)) \wedge (\neg \exists y : on(x,y) \wedge on(y,x)) \wedge (\forall y : \neg(\neg brick(y)) \vee \neg equal(x,y))$$

2. Move negations down to the atomic formulas

Equivalence Transformations:

$$\neg(E_1 \wedge E_2) \iff (\neg E_1) \vee (\neg E_2)$$
$$\neg(E_1 \vee E_2) \iff (\neg E_1) \wedge (\neg E_2)$$
$$\neg(\neg E_1) \iff E_1$$
$$\neg \forall x : E_1(x) \iff \exists x : \neg E_1(x)$$
$$\neg \exists x : E_1(x) \iff \forall x : \neg E_1(x)$$

Result:

$$\forall x : \neg brick(x) \vee ((\exists y : on(x,y) \wedge \neg pyramid(y)) \wedge (\neg \exists y : on(x,y) \wedge on(y,x)) \wedge (\forall y : \neg(\neg brick(y)) \vee \neg equal(x,y)))$$

3. Eliminate Existential Quantifiers: Skolemization

Harder cases:

$\forall x : \exists y : father(y, x)$ becomes $\forall x : father(S1(x), x)$

There is one argument for each universally quantified variable whose scope contains the Skolem function.

Easy case:

$\exists x : President(x)$ becomes $President(S2)$

$\forall x : \neg brick(x) \vee ((\exists y : on(x, y) \wedge \neg pyramid(y)) \wedge \ldots$

4. Rename variables as necessary

We want no two variables of the same name.

$$\begin{aligned}\forall x : \neg brick(x) \vee \quad & (on(x, S1(x)) \wedge \neg pyramid(S1(x))) \\ & \wedge (\forall y : (\neg on(x, y) \vee \neg on(y, x))) \\ & \wedge (\forall y : (brick(y) \vee \neg equal(x, y)))\end{aligned}$$

$$\begin{aligned}\forall x : \neg brick(x) \vee \quad & (on(x, S1(x)) \wedge \neg pyramid(S1(x))) \\ & \wedge (\forall y : (\neg on(x, y) \vee \neg on(y, x))) \\ & \wedge (\forall z : (brick(z) \vee \neg equal(x, z)))\end{aligned}$$

5. Move the universal quantifiers to the left

This works because each quantifier uses a unique variable name.

$$\begin{aligned}\forall x : \neg brick(x) \vee \quad & (on(x, S1(x)) \wedge \neg pyramid(S1(x))) \\ & \wedge (\forall y : (\neg on(x, y) \vee \neg on(y, x))) \\ & \wedge (\forall z : (brick(z) \vee \neg equal(x, z)))\end{aligned}$$

$$\begin{aligned}\forall x \forall y \forall z : \neg brick(x) \vee \quad & (on(x, S1(x)) \wedge \neg pyramid(S1(x))) \\ & \wedge (\neg on(x, y) \vee \neg on(y, x)) \\ & \wedge (brick(z) \vee \neg equal(x, z))\end{aligned}$$

6. Move disjunctions down to the literals

$$E_1 \vee (E_2 \wedge E_3) \iff (E_1 \vee E_2) \wedge (E_1 \vee E_3)$$

$$\begin{aligned}\forall x \forall y \forall z : \quad & (\neg brick(x) \vee (on(x, S1(x)) \wedge \neg pyramid(S1(x)))) \\ & \wedge (\neg brick(x) \vee \neg on(x, y) \vee \neg on(y, x)) \\ & \wedge (\neg brick(x) \vee brick(z) \vee \neg equal(x, z))\end{aligned}$$

$$\begin{aligned}\forall x \forall y \forall z : \quad & (\neg brick(x) \vee on(x, S1(x))) \\ & \wedge (\neg brick(x) \vee \neg pyramid(S1(x))) \\ & \wedge (\neg brick(x) \vee \neg on(x, y) \vee \neg on(y, x)) \\ & \wedge (\neg brick(x) \vee brick(z) \vee \neg equal(x, z))\end{aligned}$$

7. Eliminate the conjunctions

$$\begin{aligned}\forall x \forall y \forall z : \quad & (\neg brick(x) \vee on(x, S1(x))) \\ & \wedge (\neg brick(x) \vee \neg pyramid(S1(x))) \\ & \wedge (\neg brick(x) \vee \neg on(x, y) \vee \neg on(y, x)) \\ & \wedge (\neg brick(x) \vee brick(z) \vee \neg equal(x, z))\end{aligned}$$

$$\begin{aligned}& \forall x : \neg brick(x) \vee on(x, S1(x)) \\ & \forall x : \neg brick(x) \vee \neg pyramid(S1(x)) \\ & \forall x \forall y : \neg brick(x) \vee \neg on(x, y) \vee \neg on(y, x) \\ & \forall x \forall z : \neg brick(x) \vee brick(z) \vee \neg equal(x, z)\end{aligned}$$

8. Rename all variables, as necessary, so no two have the same name

$$\begin{aligned}& \forall x : \neg brick(x) \vee on(x, S1(x)) \\ & \forall x : \neg brick(x) \vee \neg pyramid(S1(x)) \\ & \forall x \forall y : \neg brick(x) \vee \neg on(x, y) \vee \neg on(y, x) \\ & \forall x \forall z : \neg brick(x) \vee brick(z) \vee \neg equal(x, z)\end{aligned}$$

$$\begin{aligned}& \forall x : \neg brick(x) \vee on(x, S1(x)) \\ & \forall w : \neg brick(w) \vee \neg pyramid(S1(w)) \\ & \forall u \forall y : \neg brick(u) \vee \neg on(u, y) \vee \neg on(y, u) \\ & \forall v \forall z : \neg brick(v) \vee brick(z) \vee \neg equal(v, z)\end{aligned}$$

9. Eliminate the universal quantifiers

$$\neg brick(x) \vee on(x, S1(x))$$
$$\neg brick(w) \vee \neg pyramid(S1(w))$$
$$\neg brick(u) \vee \neg on(u, y) \vee \neg on(y, u)$$
$$\neg brick(v) \vee brick(z) \vee \neg equal(v, z)$$

Algorithm: Putting Axioms into Clausal Form

1. **Eliminate the implications.**
2. **Move the negations down to the atomic formulas.**
3. **Eliminate the existential quantifiers.**
4. **Rename the variables, if necessary.**
5. **Move the universal quantifiers to the left.**
6. **Move the disjunctions down to the literals.**
7. **Eliminate the conjunctions.**
8. **Rename the variables, if necessary.**
9. **Eliminate the universal quantifiers.**

Resolution Proofs as Search

- **Search Problem**
  - States: Content of knowledge base in CNF
  - Initial state: Knowledge base with negated theorem to prove
  - Successor function: Resolution inference rule with unify
  - Goal test: Does knowledge base contain the empty clause 'nil'
- **Search Algorithm**
  - Depth first search (used in PROLOG)
    - Note: Possibly infinite state space
    - Example:
      - IsPerson(Fred)
      - IsPerson(y) → IsPerson(mother(y))
      - Goal: ∃ x: IsPerson(x)
      - Answers: {x/Fred} and {x/mother(Fred)} and {x/mother(mother(Fred))} and …

Strategies for Selecting Clauses

unit-preference strategy: Give preference to resolutions involving the clauses with the smallest number of literals.

set-of-support strategy: Try to resolve with the negated theorem or a clause generated by resolution from that clause.

subsumption: Eliminates all sentences that are subsumed (i.e., more specific than) an existing sentence in the KB.

May still require exponential time.

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?

Original Sentences (Plus Background Knowledge)

1. $\exists x : Dog(x) \wedge Owns(Jack, x)$
2. $\forall x;\ (\exists y\ \ Dog(y) \wedge Owns(x, y)) \rightarrow AnimalLover(x)$
3. $\forall x;\ \ AnimalLover(x) \rightarrow (\forall y\ \ Animal(y) \rightarrow \neg Kills(x, y))$
4. $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$
5. $Cat(Tuna)$
6. $\forall x :\ \ Cat(x) \rightarrow Animal(x)$

Conjunctive Normal Form

$Dog(D)$ (D is a placeholder for the dogs unknown name (i.e. Skolem symbol/function). Think of D like “JohnDoe”)

$Owns(Jack, D)$

$\neg Dog(y) \vee \neg Owns(x, y) \vee AnimalLover(x)$

$\neg AnimalLover(w) \vee \neg Animal(y) \vee \neg Kills(w, y)$

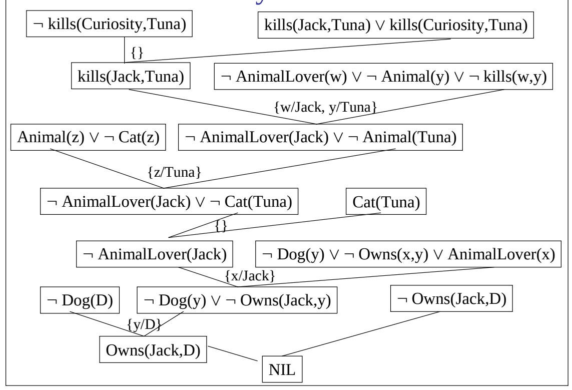
$Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$

$Cat(Tuna)$

$\neg Cat(z) \vee Animal(z)$

$\neg Kills(Curiosity, Tuna)$

Proof by Resolution



Proofs can be Lengthy

A relatively straightforward KB can quickly overwhelm general resolution methods.

Resolution strategies reduce the problem somewhat, but not completely.

As a consequence, many practical Knowledge Representation formalisms in AI use a restricted form and specialized inference.

- Logic programming (Prolog)
- Production systems
- Frame systems and semantic networks
- Description logics

Successes in Rule-Based Reasoning

Expert systems

- **DENDRAL (Buchanan *et al.*, 1969)**
- **MYCIN (Feigenbaum, Buchanan, Shortliffe)**
- **PROSPECTOR (Duda *et al.*, 1979)**
- **R1 (McDermott, 1982)**

Successes in Rule-Based Reasoning

- **DENDRAL (Buchanan *et al.*, 1969)**
  - Infers molecular structure from the information provided by a mass spectrometer
  - Generate-and-test method

```
if  there are peaks at x_1 and x_2 s.t.
    x_1 + x_2 = M + 28
    x_1 - 28 is a high peak
    x_2 - 28 is a high peak
    At least one of x_1 and x_2 is high
then there is a ketone subgroup
```

Successes in Rule-Based Reasoning

- **MYCIN (Feigenbaum, Buchanan, Shortliffe)**
  - Diagnosis of blood infections
  - 450 rules; performs as well as experts
  - Incorporated **certainty factors**

```
If: (1) the strain of the organism is
        gram-positive, and
    (2) the morphology of the organism is
        coccus, and
    (3) the growth conformation of the organism
        is clumps,
then there is suggestive evidence (0.7) that the
  identity of the organism is staphylococcus.
```

Successes in Rule-Based Reasoning

- **PROSPECTOR (Duda *et al.*, 1979)**
  - Correctly recommended exploratory drilling at geological site
  - Rule-based system founded on probability theory
- **R1 (McDermott, 1982)**
  - Designs configurations of computer components
  - About 10,000 rules
  - Uses meta-rules to change context

```
If: current context is ?x
then: deactivate ?x context
      and activate ?y context
```

Cognitive Modeling with Rule-Based Systems

SOAR **is a general architecture for building intelligent systems.**

- Long term memory consists of rules
- Working memory describes current state
- All problem solving, including deciding what rule to execute, is state space search
- Successful rule sequences are *chunked* into new rules
- Control strategy embodied in terms of meta-rules

Prolog (Programming in Logic)

- **What is Prolog?**
  - Full-featured programming language
  - Programs consist of logical formulas
  - Running a program means proving a theorem
- **Syntax of Prolog**
  - Predicates, objects, and functions:
    - cat(tuna), append(a,pair(b))
  - Variables: X, Y, List (capitalized)
  - Facts:
    - university(cornell).
    - prepend(a,pair(a,X)).
  - Rules:
    - animal(X) :- cat(X).
    - student(X) :- person(X), enrolled(X,Y), university(Y).

    → implication ":-" with single predicate on left and only non-negated predicates on the right. All variables implicitly "forall" quantified.
  - Queries:
    - student(X).

    → All variables implicitly "exists" quantified.

Programming in Prolog

- **Path Finding**

```
path(Node1,Node2) :- edge(Node1,Node2).
path(Node1,Node2) :-
     edge(Node1,SomeNode),
     path(SomeNode,Node2).
edge(ith,lga).
edge(ith,phl).
edge(phl,sfo).
edge(lga,ord).
```

- **Query**
  - path(ith,ord).
  - path(ith,X).

Programming in Prolog

- **Data structures: Lists**

```
length([],0).
length([H|T],N) :- length(T,M), N is M+1.

member(X,[X|List]).
member(X,[Element|List]) :- member(X,List).

append([],List,List).
append([Element|L1],L2,[Element|L1L2]) :-
   append(L1,L2,L1L2).
```

- **Query:**
  - length([a,b,c],3).
  - length([a,b,c],X).
  - member(b,[a,b,c]).
  - member(X,[a,b,c]).

Programming in Prolog

Example: Symbolic derivatives (http://cs.wwc.edu/~cs_dept/KU/PR/Prolog.html)

```
% deriv(Polynomial, variable, derivative)
% dc/dx = 0
deriv(C,X,0) :- number(C).
% dx/dx} = 1
deriv(X,X,1).
% d(cv)/dx = c(dv/dx)
deriv(C*U,X,C*DU) :- number(C), deriv(U,X,DU).
% d(u v)/dx = u(dv/dx) + v(du/dx)
deriv(U*V,X,U*DV + V*DU) :- deriv(U,X,DU), deriv(V,X,DV).
% d(u ± v)/dx = du/dx ± dv/dx
deriv(U+V,X,DU+DV) :- deriv(U,X,DU), deriv(V,X,DV).
deriv(U-V,X,DU-DV) :- deriv(U,X,DU), deriv(V,X,DV).
% du^n/dx = nu^{n-1}(du/dx)
deriv(U^+N,X,N*U^+N1*DU) :- N1 is N-1, deriv(U,X,DU).
```

Programming in Prolog

- **Towers of Hanoi: move N disks from pin a to pin b using pin c.**

```
hanoi(N):-hanoi(N, a, b, c).
hanoi(0,A,B,C).
hanoi(N,FromPin,ToPin,UsingPin):-
      M is N-1,
      hanoi(M,FromPin,UsingPin,ToPin),
      move(FromPin,ToPin),
      hanoi(M,UsingPin,ToPin,FromPin).
move(From,To):-
      write([move, disk from, pin, From, to, pin,
      ToPin]),nl.
```

Programming in Prolog

- **8-Queens:**

```
solve(P) :-
      perm([1,2,3,4,5,6,7,8],P),
      combine([1,2,3,4,5,6,7,8],P,S,D),
      all_diff(S),
      all_diff(D).
combine([X1|X],[Y1|Y],[S1|S],[D1|D]) :-
      S1 is X1 +Y1,
      D1 is X1 - Y1,
      combine(X,Y,S,D).
combine([],[],[],[]).
all_diff([X|Y]) :-  \+member(X,Y), all_diff(Y).
all_diff([X]).
```

Properties of Knowledge-Based Systems

Advantages

1. Expressibility*
2. Simplicity of inference procedures*
3. Modifiability*
4. Explainability
5. Machine readability
6. Parallelism*

Disadvantages

1. Difficulties in expressibility
2. Undesirable interactions among rules
3. Non-transparent behavior
4. Difficult debugging
5. Slow
6. Where does the knowledge base come from???