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_Fatherly, 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.

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)

* Name changed for privacy reasons.

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: \( P_1 \land P_2 \land \ldots \land P_n \Rightarrow Q \)
      where \( P_i \) and \( Q \) are non-negated atoms.
  – First-Order logic
    • Gödel’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: $E_1 \lor E_2 \lor ... \lor E_k$

and $\neg E \lor E_2 \lor ... \lor E_l$

Then: $E_1 \lor E_2 \lor ... \lor E_k \lor \neg E \lor E_2 \lor ... \lor E_l$

Note: $E$ can be negated.

**Example:**

Assume: $E_1 \lor E_2$ playing tennis or raining

and $\neg E_2 \lor E_3$ not raining or working

Then: $E_1 \lor E_3$ 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: $\neg P$
- Transform Knowledge Base into CNF

  1. $\neg P \lor Q$ Sentence 1
  2. $\neg Q \lor R$ 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:**

1: $\forall x: \neg\text{feathers}(x) \rightarrow \neg\text{bird}(x)$
2: $\text{feathers}(\text{tweety})$
3: $\neg\text{bird}(\text{tweety})$
4: $\neg\neg\text{feathers}(\text{tweety})$

**Resolution Proof**

1. Resolve 3 and 1, specializing (i.e. “unifying”) tweety for x.
   Add $\neg\text{feathers}(\text{tweety})$
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:

¬ Knows (John, x) ∨ 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:

¬Knows(John, Jim) ∨ Hates(John, Jim)

Unification (example)

Who does John hate?

Knowledge base (in clause form):

1. ¬ Knows (John, v) ∨ Hates (John, v)
2. Knows (John, Jim)
3. Knows (y, Leo)
4. Knows (z, Mother(z))
5. ¬ Hates (John, x) (since ¬ ∃ x: Hates (John, x) ∨ ∀ x: ¬ Hates (John, x))

Resolution with 5 and 1:

unify(Hates(John,x),Hates(John,v)) = [v/x]

6. ¬ Knows (John, v)

Resolution with 6 and 2:

unify(Knows(John,y),Knows(John,Jim)) = [v/John]

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 [v/John,v/Leo] (i.e. John hates Leo)
2. Hates(John,x) with [v/John,v/Leo] (i.e. John hates Leo)
3. Hates(John,x) with [v/John,v/Leo] (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, z/Freda}

not {y/John,x/John, z/Freda}

not {y/John,x/John, z/Freda}

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

Converting More Complicated Sentences to CNF

Substitute:

∀x : brick(x) → ((∃y : on(x,y) ∧ ¬ pyramid(y))
∧ (∃y : on(x,y) ∧ on(y,x))
∧ (∀y : ¬ brick(y) ∨ ¬equal(x,y)))

CNF:

¬brick(x) ∨ on(x, support(x))
brick(w) ∨ pyramid(v auprès de(w))
brick(u) ∨ ¬ on(u, w) ∨ ¬ on(u, y)
¬brick(v) ∨ brick(z) ∨ ¬equal(v, z)

1. Eliminate Implications

Substitute, ¬E₁ ∨ E₂ for E₁ → E₂

∀x : brick(x) → ((∃y : on(x,y) ∧ ¬ pyramid(y))
∧ (∃y : on(x,y) ∧ on(y,x))
∧ (∀y : ¬ brick(y) ∨ ¬equal(x,y)))

∀x : brick(x) ∨ ((∃y : on(x,y) ∧ ¬ pyramid(y))
∧ (∃y : on(x,y) ∧ on(y,x))
∧ (∀y : ¬ brick(y) ∨ ¬equal(x,y)))

2. Move negations down to the atomic formulas

Equivalence Transformations:

¬(E₁ ∧ E₂) ⇔ (¬E₁ ∨ ¬E₂)
¬(E₁ ∨ E₂) ⇔ (¬E₁ ∧ ¬E₂)
¬(¬E₁) ⇔ E₁
¬∀x : E₁(x) ⇔ ∃x : ¬E₁(x)
¬∃x : E₁(x) ⇔ ∀x : ¬E₁(x)

Result:

∀x : ¬brick(x) ∨ ((∃y : on(x,y) ∧ ¬ pyramid(y))
∧ (∃y : on(x,y) ∧ on(y,x))
∧ (∀y : ¬ (¬brick(y) ∨ ¬ equal(x,y))))
3. Eliminate Existential Quantifiers:
Skolemization

Harder cases:
\( \forall x : \exists y : \text{father}(y, x) \) becomes \( \forall x : \text{father}(..., x) \)

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

Easy case:
\( \exists x : \text{President}(x) \) becomes \( \exists x : \text{President}(S1(x)) \)
\( \forall x : \text{brick}(x) \lor (\exists y : \text{on}(x, y) \land \neg \text{pyramid}(y)) \lor ... \)

4. Rename variables as necessary

We want no two variables of the same name.

\[ \forall x : \neg \text{brick}(x) \lor (\text{on}(x, S1(x)) \land \neg \text{pyramid}(S1(x))) \]
\[ \lor \forall y : (\neg \text{on}(x, y) \lor \neg \text{on}(y, x)) \]
\[ \lor \forall z : (\text{brick}(z) \land \neg \text{equal}(x, z)) \]

5. Move the universal quantifiers to the left

This works because each quantifier uses a unique variable name.

\[ \forall x : \text{brick}(x) \lor (\text{on}(x, S1(x)) \land \neg \text{pyramid}(S1(x))) \]
\[ \lor (\exists y : \neg \text{on}(x, y) \lor \neg \text{on}(y, x)) \]
\[ \lor (\forall z : (\text{brick}(z) \land \neg \text{equal}(x, z))) \]

\[ \forall \forall \forall \forall : \neg \text{brick}(x) \lor (\text{on}(x, S1(x)) \land \neg \text{pyramid}(S1(x))) \]
\[ \lor (\neg \text{brick}(x) \lor \neg \text{on}(x, y) \lor \neg \text{on}(y, x)) \]
\[ \lor (\text{brick}(z) \lor \neg \text{equal}(x, z)) \]

6. Move disjunctions down to the literals

\[ E_1 \lor (E_2 \land E_3) \iff (E_1 \lor E_3) \lor (E_1 \lor E_2) \]

\[ \forall \forall \forall : (\neg \text{brick}(x) \lor \text{on}(x, S1(x)) \land \neg \text{pyramid}(S1(x))) \]
\[ \lor (\neg \text{brick}(x) \lor \neg \text{on}(x, y) \lor \neg \text{on}(y, x)) \]
\[ \lor (\text{brick}(z) \lor \neg \text{equal}(x, z)) \]

7. Eliminate the conjunctions

\[ \forall \forall \forall : (\neg \text{brick}(x) \lor \text{on}(x, S1(x))) \]
\[ \land (\neg \text{brick}(x) \lor \neg \text{pyramid}(S1(x))) \]
\[ \land (\neg \text{brick}(x) \lor \neg \text{on}(x, y) \lor \neg \text{on}(y, x)) \]
\[ \land (\text{brick}(z) \lor \neg \text{equal}(x, z)) \]

\[ \forall x : \text{brick}(x) \lor \text{on}(x, S1(x)) \]
\[ \forall x : \neg \text{brick}(x) \lor \neg \text{pyramid}(S1(x)) \]
\[ \forall x : \neg \text{brick}(x) \lor \neg \text{on}(x, y) \lor \neg \text{on}(y, x) \]
\[ \forall x \forall z : \neg \text{brick}(x) \lor \text{brick}(z) \lor \neg \text{equal}(x, z) \]

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

\[ \forall x : \neg \text{brick}(x) \lor \text{on}(x, S1(x)) \]
\[ \forall x : \neg \text{brick}(x) \lor \neg \text{pyramid}(S1(x)) \]
\[ \forall x : \neg \text{brick}(x) \lor \neg \text{on}(x, y) \lor \neg \text{on}(y, x) \]
\[ \forall x \forall z : \neg \text{brick}(x) \lor \text{brick}(z) \lor \neg \text{equal}(x, z) \]
\[ \forall w : \neg \text{brick}(w) \lor \neg \text{pyramid}(S1(w)) \]
\[ \forall w : \neg \text{brick}(u) \lor \neg \text{on}(u, y) \lor \neg \text{on}(u, y) \]
\[ \forall u \forall z : \neg \text{brick}(u) \lor \text{brick}(z) \lor \neg \text{equal}(u, z) \]
9. Eliminate the universal quantifiers
\[-\text{brick}(x) \lor \text{on}(x, S1(x))\]
\[-\text{brick}(w) \lor \neg \text{pyramid}(S1(w))\]
\[\text{brick}(u) \lor \text{on}(u, y) \lor \text{on}(y, u)\]
\[-\text{brick}(v) \lor \text{brick}(z) \lor \neg \text{equal}(v, z)\]

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. ∃x: Dog(x) ∧ Owns(Jack, x)
2. ∀y: (Dog(y) ∨ Owns(x, y)) → AnimalLover(x).
3. ∀y: AnimalLover(x) → (\forall y Animal(y) → ~Kills(x, y))
4. Kills(Jack, Tuna) ∨ Kills(Curiosity, Tuna)
5. Cat(Tuna)
6. ∀x: Cat(x) → Animal(x)
Conjunctive Normal Form

\( \neg \text{kills}(\text{Curiosity}, \text{Tuna}) \)
\( \neg \text{kills}(\text{Jack}, \text{Tuna}) \)
\( \text{AnimalLover}(w) \lor \neg \text{Animal}(y) \lor \neg \text{kills}(w, y) \)
\( \text{AnimalLover}(\text{Jack}) \lor \neg \text{Cat}(\text{Tuna}) \lor \neg \text{Animal}(\text{Tuna}) \)
\( \neg \text{AnimalLover}(\text{Jack}) \lor \text{Cat}(\text{Tuna}) \lor \text{Dog}(\text{Tuna}) \)
\( \text{Dog}(D) \lor \text{Owns}(x, y) \lor \text{AnimalLover}(x) \)
\( \text{Owns}(\text{Jack}, D) \lor \text{Dog}(\text{D}) \)

Proof by Resolution

\( \neg \text{kills}(\text{Curiosity}, \text{Tuna}) \)
\( \text{kills}(\text{Jack}, \text{Tuna}) \lor \text{kills}(\text{Curiosity}, \text{Tuna}) \)
\( \text{kills}(\text{Jack}, \text{Tuna}) \lor \text{AnimalLover}(w) \lor \neg \text{Animal}(y) \lor \neg \text{kills}(w, y) \)
\( \text{AnimalLover}(\text{Jack}) \lor \neg \text{Cat}(\text{Tuna}) \lor \neg \text{Animal}(\text{Tuna}) \)
\( \text{AnimalLover}(\text{Jack}) \lor \text{Cat}(\text{Tuna}) \lor \text{Dog}(\text{Tuna}) \)
\( \text{Dog}(D) \lor \text{Owns}(x, y) \lor \text{AnimalLover}(x) \)
\( \text{Owns}(\text{Jack}, D) \lor \text{Dog}(\text{D}) \)

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

- **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 + 20
    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

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

- **PROSPECTOR (Duda et al., 1979)**

- **R1 (McDermott, 1982)**

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 coccius, and
(3) the growth confirmation of the organism is clumped,
then there is suggestive evidence (0.7) that the identity of the organism is staphylococcus.
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

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

Prospector (Programming in Logic)

- What is Prolog?
  - Full-featured programming language
  - Running a program means proving a theorem
- Syntax of Prolog
  - Predicates, objects, and functions:
    - cat(name, append(pair(y))
    - Variables: X, Y, List (capitalized)
  - Facts:
    - university(cornell).
    - populate(pair(X,Y)).
  - 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,phi).
  - edge(phi,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|List],List,List) :- append(List,List).
- Query:
  - length([a,b,c],3).
  - length([a,b,c],X).
  - member([a,b,c],List).
  - member([a,b,c],List).

Programming in Prolog

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

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% deriv(Polynomial, variable, derivative) %
% deriv(C,X,0) :- number(C).
% deriv(X,X,1).
% deriv(C*U,X,C*DU) :- number(C), deriv(U,X,DU).
% deriv(U*V,X,U*DV + V*DU) :- deriv(U,X,DU), deriv(V,X,DU).
% deriv(U+V, X,DU+DV) :- deriv(U,X,DU), deriv(V,X,DU).
% deriv(U-V, X,DU-DV) :- deriv(U,X,DU), deriv(V,X,DU).
% deriv(U^N, X,NU^{N-1}DU) :- N is M-1, deriv(U,X,DU).
Programming in Prolog

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

```prolog
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.
```

- 8-Queens:

```prolog
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 come from???