CSCI 5582 Artificial Intelligence

Lecture 11

Jim Martin

Today 10/5

- First Order Logic
  - Also called First Order Predicate Calculus
- Break
- New HW

Clarification

Implies TT

A	B	A->B
T	T	T
T	F	F
F	T	T
F	F	T

Clarification

Implies TT ----> Rewrite

A	B	A->B
T	T	T
T	F	F
F	T	T
F	F	T

A	B	~A or B
T	T	T
T	F	F
F	T	T
F	F	T

Clarification

Implies TT ----> Rewrite

A	B	A->B
T	T	T
T	F	F
F	T	T
F	F	T

A	B	A or B
T	T	T
T	F	T
F	T	T
F	F	F

Pros and Cons of Propositional Logic

- ☺ Propositional logic is declarative
- ☺ Propositional logic allows partial/disjunctive/negated information
  - (unlike most data structures and databases)
- ☺ Propositional logic is compositional:
  - meaning of $B_{1,1} \wedge P_{1,2}$ is derived from meaning of $B_{1,1}$ and of $P_{1,2}$
- ☺ Meaning in propositional logic is context-independent
  - (unlike natural language, where meaning depends on context)
- ☹ Propositional logic has very limited expressive power
  - (unlike natural language)
  - E.g., cannot say "pits cause breezes in adjacent squares"
    - except by writing one sentence for each square

First Order Logic

- At a high level…
  - FOL allows you to represent objects, properties of objects, and relations among objects
  - Specific domains are modeled by developing knowledge-bases that capture the important parts of the domain (change, auto repair, medicine, time, set theory, etc)

First-order logic

- Whereas propositional logic assumes the world contains facts (that are true or false)
- First-order logic (like natural language) assumes the world contains
  - Objects: people, houses, numbers, colors, baseball games, wars, …
  - Relations: red, round, prime, brother of, bigger than, part of, comes between, …
  - Functions: father of, best friend, one more than, plus, …

Syntax of FOL

• Constants	KingJohn, TheEmpireStateBldg,...
• Predicates	Brother, Near, Loves,...
• Functions	Sqrt, LeftLegOf,...
• Variables	x, y, a, b,...
• Connectives	$\neg$, $\Rightarrow$, $\wedge$, $\vee$, $\Leftrightarrow$
• Equality	=
• Quantifiers	$\forall$, $\exists$

Atomic sentences

Atomic sentence = *predicate* ($term_1$,...,$term_n$)
or $term_1$ = $term_2$

Term = *function* ($term_1$,...,$term_n$)
or *constant* or *variable*

- E.g.,
  - *Brother(KingJohn, RichardTheLionheart)*
  - *> (Length(LeftLegOf(Richard)), Length(LeftLegOf(KingJohn)))*

Complex sentences

- Complex sentences are made from atomic sentences using connectives

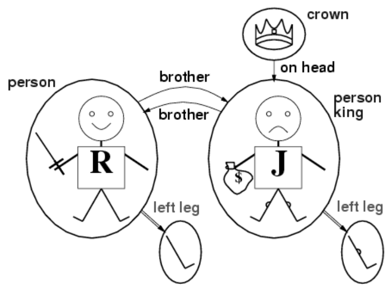
$\neg S,\ S_1 \wedge S_2,\ S_1 \vee S_2,\ S_1 \Rightarrow S_2,\ S_1 \Leftrightarrow S_2,$

E.g.
Sibling(KingJohn,Richard) $\Rightarrow$
Sibling(Richard,KingJohn)

Truth in first-order logic

- Sentences are true with respect to a model and an interpretation
- Models contain objects (domain elements) and relations among them
- Interpretation specifies referents for
  constant symbols → objects
  predicate symbols → relations
  function symbols → functional relations
- An atomic sentence *predicate*($term_1$,...,$term_n$) is true iff the objects referred to by $term_1$,...,$term_n$ are in the relation referred to by *predicate*.

Models for FOL: Example



CSCI 5582 Fall 2006

Models as Sets

- Let's populate a domain:
  - {R, J, RLL, JLL, C}
- Property Predicates
  - Person = {R, J}
  - Crown = {C}
  - King = {J}
- Relational Predicates
  - Brother = { <R,J>, <J,R>}
  - OnHead = {<C,J>}
- Functional Predicates
  - LeftLeg = {<R, RLL>, <J, JLL>}

CSCI 5582 Fall 2006

Quantifiers

- Allow us to express properties of collections of objects instead of enumerating objects by name
- Universal: "for all" ∀
- Existential: "there exists" ∃

CSCI 5582 Fall 2006

Universal quantification

∀<*variables*> <*sentence*>

Everyone at CU is smart:
$\forall x\ At(x, CU) \Rightarrow Smart(x)$

$\forall x$ *P* is true in a model *m* iff *P* is true with *x* being each possible object in the model

Roughly speaking, equivalent to the conjunction of instantiations of *P*

$At(KingJohn,CU) \Rightarrow Smart(KingJohn)$
$\wedge\ At(Richard,CU) \Rightarrow Smart(Richard)$
$\wedge\ At(Ralphie,CU) \Rightarrow Smart(Ralphie)$
$\wedge\ \ldots$

Existential quantification

∃<*variables*> <*sentence*>

Someone at CU is smart:
$\exists x\ At(x, CU) \wedge Smart(x)$

$\exists x$ *P* is true in a model *m* iff *P* is true with *x* being some possible object in the model

- Roughly speaking, equivalent to the disjunction of instantiations of *P*

$At(KingJohn,CU) \wedge Smart(KingJohn)$
$\vee\ At(Richard,CU) \wedge Smart(Richard)$
$\vee\ At(Ralphie, CU) \wedge Smart(VUB)$
$\vee\ \ldots$

Properties of quantifiers

$\forall x\ \forall y$ is the same as $\forall y\ \forall x$
$\exists x\ \exists y$ is the same as $\exists y\ \exists x$

$\exists x\ \forall y$ is not the same as $\forall y\ \exists x$
$\exists x\ \forall y\ Loves(x,y)$
– "There is a person who loves everyone in the world"
$\forall y\ \exists x\ Loves(x,y)$
– "Everyone in the world is loved by at least one person"

- Quantifier duality: each can be expressed using the other
  $\forall x\ Likes(x,IceCream)$ $\quad \neg\exists x\ \neg Likes(x,IceCream)$
  $\exists x\ Likes(x,Broccoli)$ $\quad \neg\forall x\ \neg Likes(x,Broccoli)$

Reasoning

- We can do all the same reasoning with FOL that we did with Prop logic
  - Compositional Semantics
  - Model-Based Reasoning
  - Chaining (Forward/Backward)
  - Resolution
- But the presence of variables and quantifiers makes things more complicated

Variables

- A big part of reasoning with FOL involves keeping track of all the variables while reasoning.
- Substitution lists are the means used to track the value, or binding, of variables as processing proceeds.

[illegible]

Examples

[illegible]

[illegible]

[illegible]

[illegible]

Examples

[redacted]

Inference

- Inference in FOL involves showing that some sentence is true, given a current knowledge-base, by exploiting the semantics of FOL to create a new knowledge-base that contains the sentence in which we are interested.

Inference Methods

- Proof as Generic Search
- Proof by Modus Ponens
  - Forward Chaining
  - Backward Chaining
- Resolution
- Model Checking

Generic Search

- States are snapshots of the KB
- Operators are the rules of inference
- Goal test is finding the sentence you're seeking
  - I.e. Goal states are KBs that contain the sentence (or sentences) you're seeking

Example

- Harry is a hare [redacted]
- Tom is a tortoise [redacted]
- Hares outrun tortoises
  [redacted]
- Harry outruns Tom?

Tom and Harry

- And introduction
  [redacted]
- Universal elimination
  [redacted]
- Modus ponens
  [redacted]

What's wrong?

- The branching factor caused by the number of operators is huge
- It's a blind (undirected) search

So…

- So a reasonable method needs to control the branching factor and find a way to guide the search…
- Focus on the first one first

Forward Chaining

- When a new fact *p* is added to the KB
  - For each rule such that p unifies with part of the premise
    - If all the other premises are known
    - Then add consequent to the KB

This is a data-driven method.

Backward Chaining

- When a query q is asked
  - If a matching q' is found return substitution list
  - Else For each rule q' whose consequent matches q, attempt to prove each antecedent by backward chaining

This is a goal-directed method. And it's the basis for Prolog.

Backward Chaining

[redacted]

Is Tom faster than someone?

Notes

- Backward chaining is not abduction; we are not inferring antecedents from consequents.
- The fact that you can't prove something by these methods doesn't mean its false. It just means you can't prove it.

Resolution

- Modus ponens is not complete. I.e. there are things we should be able to prove true that we can't by using Modus ponens alone.
- Used appropriately, resolution is complete.

Resolution Example

[redacted]

Resolution Example

Convert to Normal Form

[redacted]

Resolve 1 and 3

[redacted]

Resolve 2 and 5

[redacted]

Resolve 4 and 6

[redacted]

Break

- New HW (Due 10/17)
  1. Download and install python code for the logic chapters from aima.cs.berkeley.edu
  2. Encode the rules of Wumpus world in prop logic
  3. Debug and complete the WalkSat code in logic.py
  4. Apply WalkSat to answer satisfiability questions that I pose about game situations

Break

- Office Hours changed for today
  - I'll be in my office after class 1:00
  - I'll be back at 3:15 or so until 5.

HW

- I'll give you situations that look like this….
  - ~S11, ~B11, B21, ~S21, P31
- This means that you know there's no stench in 1,1 and no breeze in 1,1 and a breeze in 2,1 and no stench in 2,1
- And I'm asking you if P31 is satisfiable.
  - I'm asking if there could be a pit in 3,1
- You should return a satisfying model if there is one, otherwise return false.

HW

- The tricky part of this HW is that you have to build a correct KB and get the WalkSat code running at the same time.
  - In debugging you may have a hard time determining if your code is wrong or your KB is wrong (or incomplete)
  - You can use any of the other prop logic inference routines in logic.py to help debug your KB.

The `WalkSAT` algorithm

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

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

WalkSat

```
def WalkSAT(clauses, p=0.5, max_flips=10000):
   model = dict([(s, random.choice([True, False]))
                 for s in prop_symbols(clauses)])
   for i in range(max_flips):
      satisfied, unsatisfied = [], []
      for clause in clauses:
          if_(pl_true(clause, model), satisfied, unsatisfied).append(clause)
      if not unsatisfied:
         return model
      clause = random.choice(unsatisfied)
      if probability(p):
          sym = random.choice(prop_symbols(clause))
      else:
          raise NotImplementedError
      model[sym] = not model[sym]
```

Moving On…

- We'll wrap up logic material on Tuesday
- And then start on Chapter 13

CSCI 5582 Fall 2006