Artificial Intelligence
Logic and Theorem Proving

Professor Hager
http://www.cs.jhu.edu/~hager

Reading: Chapters 6, 7, 9 of R&N

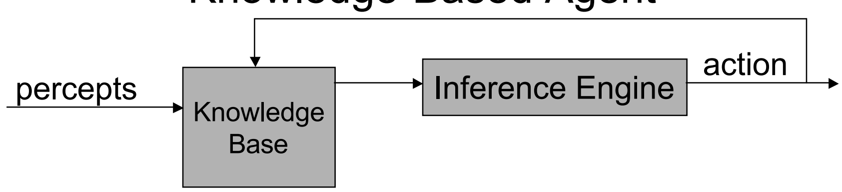
Unit III: Logic

- One of the original problem areas in AI was mathematical theorem proving: logic theorist, GPS
  – first complete inference procedure was computational
- Early on many researchers realized that it was essential to have a "formal" language for talking about knowledge
  – logic seems like the obvious language
- These two facts have led logic and logical inference to be generally viewed as essential to symbolic AI
  – explicit language for expressing "knowledge"
  – formal inference procedures
  – nothing "hidden" in the code --- a fully **declarative** approach

R&N Notes

- R&N tries to avoid some of the formal details by extended “intuitive explanations
  - I prefer to present some ideas a little more formally
- R&N uses the wumpus world to illustrate ideas
  - I will not use the wumpus world, but you may want to read though it
- R&N spends a lot of time on propositional logic
  - I’ll move to first order as quickly as possible

Knowledge-Based Agent



1. Knowledge or epistemological level
   “What the agent can know about”
2. Logical level
   representation of facts as sentences in a formal language
3. Implementation level
   how inference is implemented on a physical representation

This is, broadly taken, the science of *knowledge representation*
This is also often called the “declarative approach.”

Logic: The Landscape

- Logics consists of two basic pieces:
  - **syntax** --- what is written down; the language

    3<= 6, a > b, 4 + 6 = 9, p ^ q, K p => K K p, ...

  - **semantics** --- the meaning of the language; what it says about the world --- the definition of what it means to be **true**

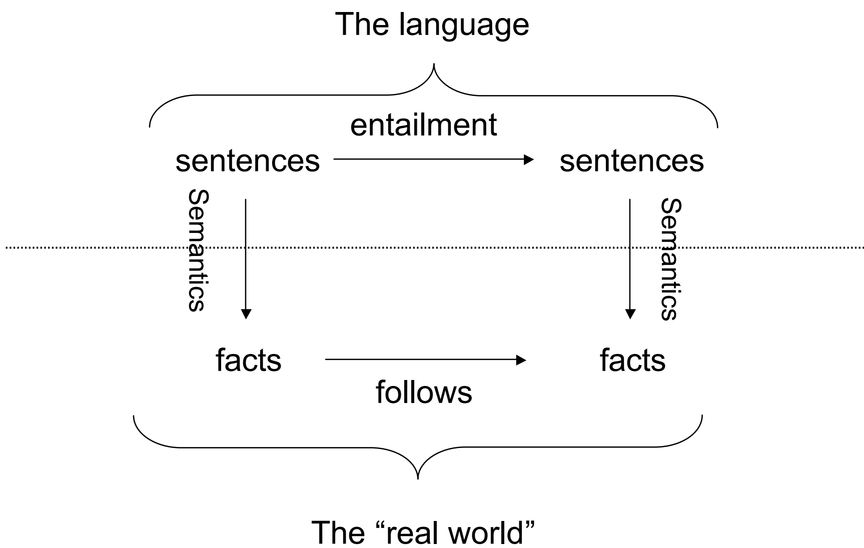
    p, q, ... take on values 0 and 1, p ^ q true iff both p ^ q are 1

- Two notions of determining what follows from what we know
  - KB |= a ◊ KB *entails* a; if KB is true, then so is a
  - KB $|-_i$ a ◊ inference procedure i *derives* a from the sentences in KB

Logic: The Landscape

- We can make up any way of creating new sentences from old
  - i is *sound* if KB $|-_i$ a implies that KB |= a
    - we only derive statements that are true given what we know
    - the record of the derivation is called a *proof*
  - i is *complete* if KB |= a implies that KB $|-_i$ a
    - if a fact is true, we can derive it
- Soundness is essential (and usually easy)
- Completeness is hard (and sometimes impossible!)

Logic: The Landscape



Semantics

- Semantics is the link between what we write (syntax) and what it means (semantics)
  - Suppose that *S* is the set of all sentences in the language (we have to define this)
  - Suppose that I is a function that maps every sentence of *S* to some *interpretation* of that sentence: M is a *model* that determines the basic truth values of the primitives:
    e.g I[M]: *S* ◊ {True,False}
  - We will assume all of our languages of *compositional*: we can recursively define the meaning of sentences in terms of their components

Semantics

- A sentence s is *true under an interpretation* (alternatively, true in a model *M)* if I[M](s) = True
  - I will sometimes write M |= s
  - Think about KB |= s as shorthand for
    forall M such that M |= KB, M |= a

- A sentence s is *valid* if it is true in all interpretations (true in all models)
  - I will sometimes write |= s
  - This is really just like the above; we just don't constrain the models we consider
- A sentence s is *satisfiable* if it is true in some interpretation (or model)
- A sentence s is *unsatisfiable* if it true in no interpretation

Propositional Logic: An Example

- Atoms: True, False, p, q, dave_is_here, students_sleep_in_AI
- Language: Sent = Atom |
  ~ Sent | Sent ^ Sent | Sent v Sent | ....

- Model: set of pairs: { (p,1), (q,0), (students_sleeping 1), ..}

- Atoms:
  - I[M](True) = True
  - I[M](False) = False
  - I[M](a) = True if (a,1) in M; False otherwise
- Connectives:
  - I[M](~s) = True if I[M](s) is False; False otherwise
  - I[M](s1 v s2) = True if either I[M](s1) is true or I[M](s2) is true; False otherwise
  - I[M](s1 ^ s2) = True if both I[M](s1) is true and I[M](s2) is true; False otherwise

Propositional Logic: An Example

- Atoms: True, False, p, q, dave_is_here, students_wake_up, ...
- Language: Sent = Atom |
  ~ Sent | Sent ^ Sent | Sent v Sent

- Model: a universe U; each element p associated with S_p, a subset of U

- Atoms:
  - I[M](a) = S_a
  - I[M](True) = U
  - I[M](False) = $\varnothing$
- Connectives:
  - I[M](~s) = U - I[M](s)
  - I[M](s1 v s2) = I[M](s1) $\cup$ I[M](s2)
  - I[M](s1 ^ s2) = I[M](s1) $\cap$ I[M](s2)

Propositional Logic: Inference

- We could do inference for propositional logic just by checking models
  - 2^n potential true values for a sentence with n atoms
- Normally, we (well, computer scientists) manipulate syntax
  - many logics (e.g. first order) don't have enumerable models!
- Proof theory is the way that define how we manipulate symbols

  - e.g. modus ponens (defining a => b as ~ a v b)
    - given a and a => b, we conclude b

  - modus ponens is sound
    - hint: assume this is not the case and work it through ...

  - Is modus ponens complete
    - yes --- if we provide enough axioms!

Forward-Chaining

- A commonly used paradigm in logic is forward-chaining: that is, repeated use of modus ponens:

1. {a^b,r=>q^t,q=>s,s^q=>done,r^m} -->{a,b,r=>q,q=>s,s^q=>done,r,m}
2. {a,b,r=>q,q=>s,s^q=>done,r,m,q}
3. {a,b,r=>q,q=>s,s^q=>done,r,m,q,s}
4. {a,b,r=>q,q=>s,s^q=>done,r,m,q,s,done}

Note that we can put sentences into a “normal form” using only ^, => and ~. As a result, we can make forward chaining complete

Similarly, We Can Backward Chain

- Given {a,b,r=>q,q=>s,s^q=>done,r,m}, prove “done”:

1. done<=s^q ====> prove s, prove q
2. s<=q ===> prove q
3. q<=r ===> prove r <----- “a given”

In general, this creates a tree of desired subproofs which must eventually terminate with a fringe of “givens”

Hard to make this complete in general, although we will see a case where this is a complete inference procedure

Propositional Logic: Inference

- Natural deduction systems require only one axiom, but more rules
  - Generally written in the form of trees
  - Example: show that r ^ (p => q) ^ p |- q v t

- Rules:
  - modus ponens
  - and-elim
  - or-intro
  - and-intro
  - double negation elimination
  - unit resolution
  - general resolution

The Resolution Rule

- Think of a=>b and a as givens
- Suppose we want to prove b
- Proceed by contradiction: assert ~b and show it leads to a contradiction
- Use a special normal form: CNF --> a conjunct of disjuncts
  - ~a v b, a, ~b

- Note that if we have x v y, ~x in the set, then it must be the case that y is true

- Thus, ~a v b,a --> b; but b, ~b ---> empty clause <<<contradiction!!

Resolution is Enough

- The idea of resolution is to use a canonical form:
  - r ^ (p => q) ^ p |- q v t is the same as |- (r ^ (p => q) ^ p) => q v t
  - Put into CNF (a conjunct of disjuncts)
  - Apply the resolution rule (a lot)
- Other facts about proposition (and the other logics we care about)
  - monotonicity: adding facts doesn't make things that were true untrue
  - locality: we don't need to inspect the entire KB to infer something
- Imagine trying to formalize family relationships using PL
  - john_sonof_dave, mary_motherof_john
  - can we describe an infer the notion of grandmother (or, more generally, ancester)?

First Order Logic

- Propositional logic is often referred to as "0^{th}-order" logic
  - you can only talk about facts, not objects in the world
  - first order talks about objects
  - second order talks about sets of objects (predicates)
  - third order talks about sets of sets of objects (predicates on predicates)
  - ....
- Invented by Frege, Peano, 1880's
- Even with first order logics, there are lots of questions as to what to include in the language
  - equality
  - arithmetic
  - sets

FOL: Syntax

- Term: constant | variable | fn ($term_1$, $term_2$, .. $term_n$)
  - ground terms contain no variables
- AtomicSentence: pred ($term_1$, $term_2$, .. $term_n$)

- Sentence: AtomicSentence | ~ Sentence |
  Sentence connective Sentence
  quantifier Sentence
  (Sentence)

- quantifier: ∀ | ∃ (by convention, scope as far as possible)
- connective: ^ | v | ◊ | ⇓◊ | ~

- W.L.G, we will assume that all variables are quantified: a well-formed formula (wff -- pronounced “woof”)

An Example

- ∀ x male(x) v female(x) ^ ∀ x ~(male(x) ^ female(x))
- ∀ x y z parentof(x,y) ^ ancesterof(y,z) ◊ ancesterof(x,z)
- ∀ x y parentof(x,y) ◊ ancesterof(x,y)
- ∀ x parentof(fatherof(x),x) ^ parentof(motherof(x),x) ^ male(fatherof(x)) ^ female(motherof(x))
- ∀ x ∃ y1, y2 parentof(y1,x) ^ parentof(y2,x) ^ ~(y1 = y2)
- ∀x,y childof(y,x) ⇓◊ parentof(x,y)
- male(john), female(annika), parentof(annika,john) .....

- E.g. show ∀ x ∃ y ancesterof(y,x)

A Few Facts

- $\forall$ x ~ P(x) == ~ $\exists$ x P(x)
- $\exists$ x ~ P(x) == ~ $\forall$ x P(x)

- Equality is semantic --- objects *denote* the same thing
  - equivalent to a two place predicate of identical pairs

- Other extensions
  - lambda operators
  - unique existence $\exists!$

How Can We Interpret FOL?

- Model:
  - Universe *U of* objects
  - A mapping c that relates constants to elements of *U*
    - e.g. c(John) = ...
  - For each function symbol f of n arguments, a function $f^U: U^n \lozenge U$
    - denote I[M](f)
  - For each predicate P of n arguments, a subset P^U of U^n
  - denote I[M](P)

- For interpretation, assume wff's that are *uniquely named*

- Consider a substitution s to be a list of pairs x/u where x is a variable in the language and u is an element of U
- a [s] is the sentence a with all of the variable assignments given in s

A Sketch of Interpretation

- Terms:
  - I[M,s](constant) = c(constant)
  - I[M,s](variable) = u where x/u is in s
  - I[M,s](f(t1,t2, ... tn)) = I[M](f)(I[M,s](t1), I[M,s](t2) ... I[M,s](tn))

- Atomic Sentences
  - I[M,s](P(t1, t2, ... tn) = true if <I[M,s](t1),... I[M,s](tn)>$\in$ I[M](P)
    false otherwise

- Sentences
  - I[M,s](P1 v P2) = true if one of I[M,s](P1) or I[M,s](P2) is true
  - I[M,s]($\forall$ x P) = true if for every u $\in$ U, I[M,s $\cup$ {x/u}](P) is true
  - I[M,s]($\exists$ x P) = true there is some u $\in$ U such that I[M,s $\cup$ {x/u}](P) is true

Proofs in FOL

- Note that now, except in special cases, our models are infinite in number --- enumeration no longer works

- We can use the same general system as with propositional logic, except we need some extra rules (page 266 R&N or http://plato.stanford.edu/entries/logic-classical/)

  - Universal Elimination
  - Existential Elimination (there is a bug here)
  - Existential Introduction
    - occurs check!
  - Universal Introduction (if a |- v and x does not occur free in V and
  a |- $\forall$ x V would be the normal rule)

- Consider two statements
  - $\forall$ x $\exists$ y p(x,y) => $\exists$ y $\forall$ x p(x,y)
  - $\exists$ y $\forall$ x p(x,y) => $\forall$ x $\exists$ y p(x,y)
  - which is true?
  - how could we prove it?

Example

- From our previous axiomitization
  1. ∀ x male(x) v female(x) ^ ~(male(x) ^ female(x))
  2. ∀ x y z parentof(x,y) ^ ancesterof(y,z) ◊ ancesterof(x,z)
  3. ∀ x y parentof(x,y) ◊ ancesterof(x,y)
  4. ∀ x parentof(fatherof(x),x) ^ parentof(motherof(x),x) ^ male(fatherof(x)) ^ female(motherof(x))
  5. ∀ x ∃ y1, y2 parentof(y1,x) ^ parentof(y2,x) ^ ~(y1 = y2)
  6. ∀x,y childof(y,x) ⇓◊ parentof(x,y)
  7. male(john), female(annika), parentof(annika,john) .....

- show
  – ∀ x ∃ y ancesterof(y,x) ^ female(y)
  – childof(john, annika)
  – ~childof(annika,john)

Resolution in FOL

- Recall we talked briefly about the resolution principle in propositional logic:
  – convert to clausal form
  – use the resolution rule
  – try to derive an empty clause

- For first order logic, it’s about the same thing, but we now have to deal with quantifiers:
  – ∃ y ∀ x p(x,y) => ∀ x p(x,A) where A is a new symbol

  – ∀ x ∃ y p(x,y) => ???
    - intuitively, ∀ x p(x,A) is wrong
    - we want to capture the idea that the existential quantifier is somehow dependent on the universal scoped outside of it

Skolemization

- Skolemization (named after the Polish logician Skolem)
  - replace each existentially quantified variable with a new function with arguments that are any universally quantified variable scoped outside of it
  - $\exists$ y $\forall$ x p(x,y) => $\forall$ x p(x,sk1)
  - $\forall$ x $\exists$ y p(x,y) => $\forall$ x p(x,sk2(x))
  - $\forall$ x $\exists$ y $\exists$ z p(x,y) ^ q(x,z) ◊ $\forall$ x p(x,sk3(x)) ^ q(x,sk4(x))
  - $\forall$ x $\forall$ y $\exists$ z p(x,y) ^ q(x,z) ◊ $\forall$ x y p(x,y) ^ q(x,sk5(x,y))

- Note this is a way to deal with the limitations of the natural deduction system in the book

Conversion to Normal Form

- Negate the formulate to be proven
  - ~ ($\exists$ y $\forall$ x p(x,y) => $\forall$ x $\exists$ y p(x,y))
- Replace A => B by ~ A v B
  - ~(~ $\exists$ y $\forall$ x p(x,y) v $\forall$ x $\exists$ y p(x,y))
- Move negation inward
  - $\exists$ y $\forall$ x p(x,y) ^ $\exists$ x $\forall$ y ~ p(x,y)
- Standardize apart
  - $\exists$ y $\forall$ x p(x,y) ^ $\exists$ w $\forall$ z ~ p(w,z)
- Move quantifiers left
  - $\exists$ y $\forall$ x $\exists$ w $\forall$ z p(x,y) ^ ~ p(w,z)
- Skolemize
  - p(x,sk1) ^ ~ p(sk2(x),z)

Conversion to Normal Form

- Distribute ^ over v
  - we don't have any, but in general (a ^ b) v c ◊ (a v c) ^ (b v c)
- Drop the ^ and v and write sets of clauses
  - {p(x,sk1)} { ~ p(sk2(x),z)}
- As a result, we have sets of clauses that we can now apply the resolution rule to
  - find a set with a positive term that *matches* a negative term in another set
  - but to do so, we need to understand how to match things

Unification

- The process of matching is called unification:
  - p(x) matches p(Jack) with x = Jack
  - q(fatherof(x),y) matches q(y,z) with y=fatherof(x) and z = y
    - note the result of the match is q(fatherof(x),fatherof(x))
  - p(x) matches p(y) with
    - x = Jack and y = Jack
    - x = John and y = John
    - or x = y
  - The match that makes the least commitment is called the *most general unifier (MGU)*
  - We can phrase unification in terms of the parse tree of the expression
  - We use the notation subst(t,s) to denote the application of the substitution s = {v1/t1,v2/t2 ... vn/tn) to t.

Unification Algorithm

- Given two formulas A and B, first standardize them apart
  – parentof(fatherof(John),y) parentof(y,z) ◊
  – parentof(fatherof(John),v1) parentof(v2,v3)

- Unify(t1,t2): Determine if the head matches and the same arity
  – for each pair of terms , call Unify-term(t1,t2) which returns a substitution S
  – Apply S to the remaining terms and repeat the process for the remaining terms
  – Return the total substitution (theunion) so computed
- Unify-term(t1,t2)
  – a constant matches the same constant
  – a variable v matches a term t (with substitution v = t) provided v *does not occur in t; return the substitution (*note how this works with skolemization
  – Otherwise, we must have two terms, t1 and t2.
    - Call Unify on t1 and t2 and return the resulting substitution

Resolution Examples

- ∀ x ∃ y p(x,y) ◊ ∃ y ∀ x p(x,y)
- ∃ y ∀ x p(x,y) ◊ ∀ x ∃ y p(x,y)
- From our previous axiomitization
  – ∀ x male(x) v female(x) ^ ~(male(x) ^ female(x))
  – ∀ x y z parentof(x,y) ^ ancesterof(y,z) ◊ ancesterof(x,z)
  – ∀ x y parentof(x,y) ◊ ancesterof(x,y)
  – ∀ x parentof(fatherof(x),x) ^ parentof(motherof(x),x) ^ male(fatherof(x)) ^ female(motherof(x))
  – ∀ x ∃ y1, y2 parentof(y1,x) ^ parentof(y2,x) ^ ~(y1 = y2)
  – ∀x,y childof(y,x) ⇓◊ parentof(x,y)
  – male(john), female(annika), parentof(annika,john) .....

- show
  – ∀ x ∃ y ancesterof(y,x) ^ female(y)
  – childof(john, annika)
  – ~childof(annika,john)

Example

- Relevant clausal forms from KB
  - {~parentof(x,y), ancesterof(x,y)}
  - {parentof(motherof(x),x)}
  - {female(motherof(x))}

- negated goal
  - ~∀ x ∃ y ancesterof(y,x) ^ female(y)
  - {~ancesterof(y,A),~female(y)}

- Proof .... (done in class)

Resolution Strategies

- Unit Preference
  - always try to resolve with single literals

- Set of support
  - Start with negated query and only resolve against descendents of that query

- Input Resolution
  - Every resolution combines an input sentence (KB or query) with some other sentence
  - linear resolution is a slight generalization

- Subsumption
  - only keep the most general set of sentences around