CS 5100: Foundations of Artificial Intelligence

Agents, Logic, and Reasoning

Prof. Amy Sliva
September 22, 2011

Outline

- Propositional logic
  - Horn clauses
  - Forward-chaining
  - Backward-chaining
- First-order logic
- Knowledge engineering for first-order logic

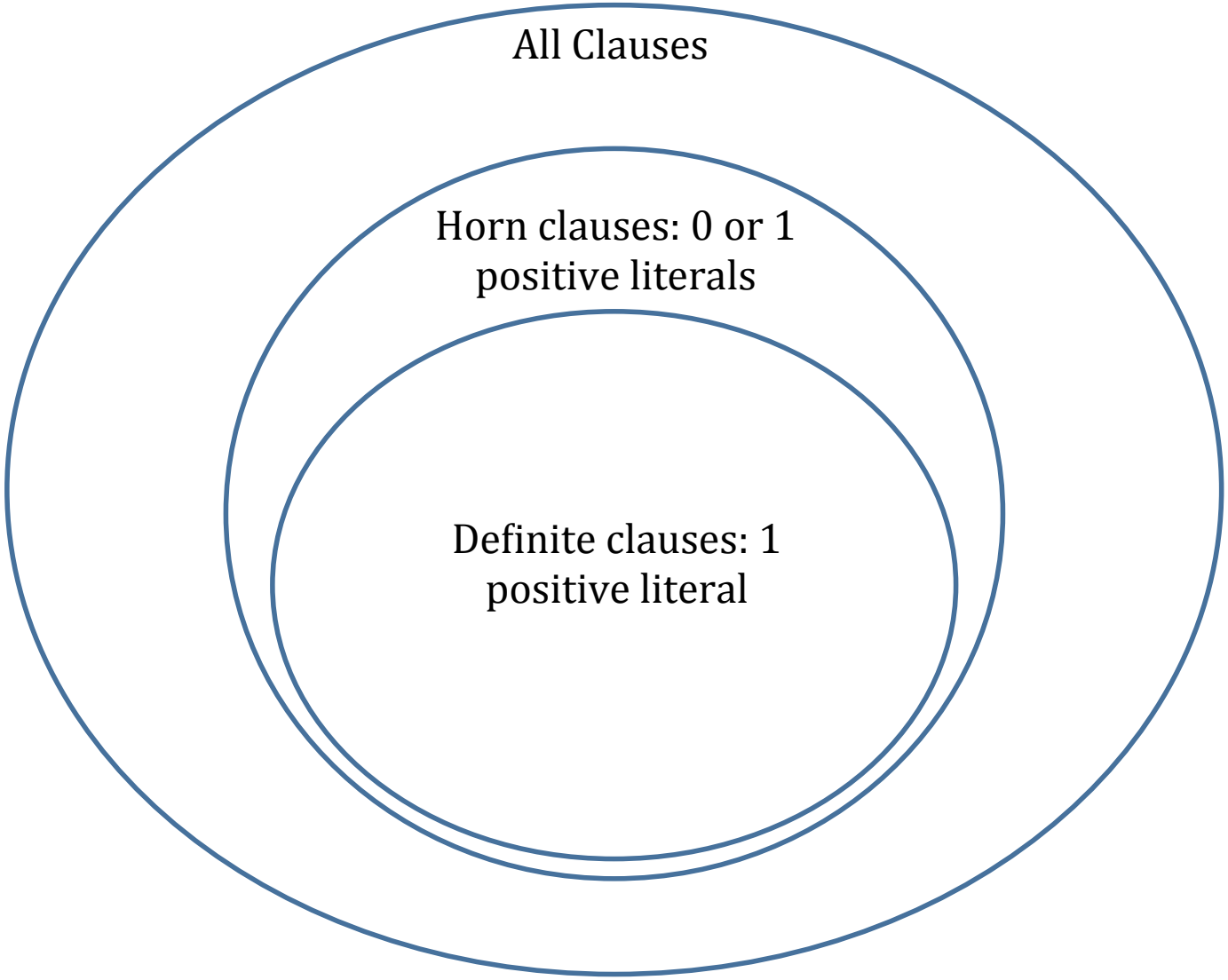
Review: clauses and inference

- **Literal** is an "atomic sentence" (i.e., P, Q, R) or the negation of an atom (i.e., $\neg P$)
- **Clause** is a **disjunction** of literals (i.e., $P \vee \neg Q \vee R$)
- KB is in **Conjunctive Normal Form (CNF)** if represented as a conjunction of disjunctions of literals
  - A set of clauses (AND is implicit) representing the agent's knowledge
- With KB in CNF, resolution is **sound and complete** inference procedure in a single rule!!
- **Theorem**: any set of logic sentences can be transformed into CNF (conjunctive normal form)

Horn clauses

- **Horn clause**—clause with at most one positive literal
  $\neg P_1 \vee \neg P_2 \vee \ldots \vee \neg P_n$
- **Definite clause**—Horn clause with exactly one positive literal
  $\neg P_1 \vee \neg P_2 \vee \ldots \vee \neg P_n \vee R$
- **Goal clause**—Horn clause with no positive literals
  $\neg P_1 \vee \neg P_2 \vee \ldots \vee \neg P_n$
- Closed under resolution (i.e., resolution of Horn clauses will return Horn clause)
- Special properties of KBs with Horn clauses
  1. Definite clauses can be written as implication rules <body> ⇒ <head>
     $(\neg P_1 \vee \neg P_2 \vee \ldots \vee \neg P_n) \Rightarrow R$
  2. Two inference methods that work for Horn clauses
     - **Forward chaining** (data driven)
     - **Backward chaining** (goal driven)
  3. Entailment can be decided in linear time w.r.t. size of KB

Horn clauses and definite clauses



Which of the following are clauses? IF yes, convert to implicative form. Which are Horn clauses? Definite clauses?

1. $A \vee B$
2. $A \wedge B$
3. $\neg A \vee \neg B$
4. $\neg A \wedge \neg B$
5. $\neg A \vee \neg B \vee C \vee D \vee E$
6. $(A \wedge B) \vee C$
7. $\neg(A \wedge \neg B) \vee C$

Which of the following are clauses? IF yes, convert to implicative form. Which are Horn clauses? Definite clauses?

1. $A \vee B$ — **Yes. $\neg A \Rightarrow B$**
2. $A \wedge B$ — **No.**
3. $\neg A \vee \neg B$ — **Yes. $A \Rightarrow \neg B$. Horn clause.**
4. $\neg A \wedge \neg B$ — **No.**
5. $\neg A \vee \neg B \vee C \vee D \vee E$ — **Yes. $A \wedge B \rightarrow C \vee D \vee E$**
6. $(A \wedge B) \vee C$ — **No.**
7. $\neg(A \wedge \neg B) \vee C$ — ***No.***

Forward-chaining

- Determines if query q is entailed by KB of definite clauses
  - Starts with known facts and derives new knowledge

- **Horn clauses:**
  C1. $\neg P_1 \vee \neg P_2 \vee P_4$
  C2. $\neg P_4 \vee P_5$

  Rules:
  $P_1 \wedge P_2 \Rightarrow P_4$
  $P_4 \Rightarrow P_5$

- **Facts:** P_1 , P_2

- Step 1: Percepts P_1 and P_2 resolve with C1 to get P_4 (Add P_4 to KB)
- Step 2: Resolve P_4 with C2 to get P_5
  - This is called **rule chaining**
- Agent can derive conclusions from incoming **percepts**

Forward-chaining algorithm

- **Algorithm** (recursive):

```
PLForwardChain()
      # uses KBase -- a knowledge base of Horn clauses for each new
      # percept p

      PLFC1(p)  #use a recursive "helper function"

      PLFC1(percept)
            if percept is already in KBase, return
            else
                  add percept to Kbase
                  for r in rules s.t. conclusion is not in Kbase
                        if percept is a premise of r and all
                        other premises of r are known
                              PLFC1(conclusion of r)
```

- Note: 1) Efficient implementation requires indexing rules by LHS
  2) How are infinite loops prevented by this algorithm?

Forward-chaining algorithm

- **Algorithm** (recursive):

```
PLForwardChain()
      # uses KBase -- a knowledge base of Horn clauses for each new
      # percept p

      PLFC1(p)  #use a recursive "helper function"

      PLFC1(percept)
            if percept is already in KBase, return
            else
                  add percept to Kbase
                  for r in rules s.t. conclusion is not in Kbase
                        if percept is a premise of r and all
                        other premises of r are known
                              PLFC1(conclusion of r)
```

- Note: 1) Efficient implementation requires indexing rules by LHS
  2) How are infinite loops prevented by this algorithm?

Backward-chaining

- Works backward to determine if the query q is true

- **Horn clauses:**
  C1. $\neg P_1 \vee \neg P_2 \vee P_4$
  C2. $\neg P_4 \vee P_5$

  Rules:
  $P_1 \wedge P_2 \Rightarrow P_4$
  $P_4 \Rightarrow P_5$

- **Facts:** P_1 , P_2

  Goal: P_5

- Subgoal: prove P_4
  - Sub-sub goal: prove P_2
  - Sub-sub goal: prove P_1
- Very efficient—only touches relevant facts/rules

Backward-chaining

- Goal-driven reasoning triggered by a **new percept** (fact)
- Basis of backward-chaining

  $P \wedge R \Rightarrow Q$ is an assertion in the KB

  Q is a query we want to prove (or disprove)

  Set up P and R as sub-queries, if true then Q is proved

- What if we cannot find Q or a rule that proves Q?
  - Answer *False*—**negation by failure**
    (not the same as a real proof of $\neg Q$)
- Note: $P \Rightarrow \neg Q$ is not a Horn Clause
  - Normalizes to $P \vee Q$, which has two positive literals

Backward-chaining

- Goal-driven reasoning triggered by a **question** being asked
- KB:
  fruit $\Rightarrow$ edible
  vegetable $\Rightarrow$ edible
  edible $\wedge$ green $\Rightarrow$ healthy
  apple $\Rightarrow$ fruit
  banana $\Rightarrow$ fruit
  spinach $\Rightarrow$ vegetable
  spinach $\Rightarrow$ green
  edible $\wedge$ healthy $\Rightarrow$ recommended
  $\Rightarrow$ apple
- Consider some queries:
  ?apple ?fruit ?banana ?edible ?healthy

Sketch of backward-chaining algorithm

- **Algorithm** (recursive):

```
backwardChain(KB, query) returns Boolean
      if query is in KB, return True
      for each rule r in KB such that RHS(r) == query
            testing = True
            for each element e of the LHS(r)
                  if backwardChain(KB, e) = False
                        testing = False
                        break
            if testing = True return True
      return False
```

- NOTE: backward-chaining does **not update** the KB

Review of the wumpus world in PL

- What do we need to represent?

I. **Static knowledge**

  - Relevant **ontology** of possible world configurations:
    - locations on a 4x4 grid and their properties
      (e.g., $P_{x,y}$ means a pit in $[x,y]$)
  - Player's current **state** ($L_{x,y}$, *has-arrow*)
  - The **axioms** of the world configuration
    - $L_{2,1} \wedge Breeze \Rightarrow P_{2,2} \vee P_{3,2}$
    - There is exactly one wumpus:
      $W_{2,1} \vee W_{3,1} \vee \ldots \vee W_{4,4}$ == there is at least one
      $\neg(W_{2,1} \wedge W_{2,2})$—one axiom like this for each pair == there is at most one
  - Player's current **percepts**
    Breeze, *Stench*

The wumpus world in PL (cont.)

II. Dynamic Knowledge

- Possible actions: up, down, left, right, grab, shoot
- **Effects** of actions (requires temporal indexing)

  $L_{1,1,0} \wedge up_0 \Rightarrow L_{2,1,1}$ — one for each location $[x,y]$ at time t

  $L_{1,1,0} \wedge has\text{-}arrow_0 \wedge shoot_0 \Rightarrow L_{1,1,1} \wedge \neg has\text{-}arrow_1$

  - The **frame problem** requires exhaustive representation of effects (and non-effects)

    $L_{1,1,0} \wedge up_0 \Rightarrow L_{2,1,1} \wedge \neg L_{1,1,1}$

    But it gets even worse!

    $L_{1,1,0} \wedge has\text{-}arrow_0 \wedge up_0 \Rightarrow L_{2,1,1} \wedge \neg L_{1,1,1} \wedge has\text{-}arrow_0$

    $L_{1,1,0} \wedge \neg has\text{-}arrow_0 \wedge up_0 \Rightarrow L_{2,1,1} \wedge \neg L_{1,1,1} \wedge \neg has\text{-}arrow_0$

    The frame problem arises when we use **temporal indexing**—causes axioms to multiply almost without limit!!!

Pros and cons of propositional logic

- ☺ Propositional logic is **declarative**
- ☺ PL allows **partial/disjunctive/negated** information
  - Unlike most data structures and databases
  - Horn clauses are a nice intermediate form
- ☺ 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 PL 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 (FOL)

- Why FOL?
- Syntax and semantics
- Using FOL
- Wumpus world in FOL!
- Knowledge engineering in FOL

Two parallel tracks in AI

- **Track 1:** Study important formalisms for representing what an agent knows and perceives, and the algorithms for reasoning, understanding, problem solving and learning that make use of these formalisms.
  - E.g., FOL syntax and semantics, and algorithms for logical deduction

- **Track 2:** Consider the knowledge and reasoning abilities underlying various kinds of intelligent behavior, learn to apply the important formalisms and algorithms to these tasks, and also understand their limitations.
  - E.g., representing common sense knowledge in FOL; ontology design

First-order logic

- Propositional logic limits world models to **atomic facts**
  E.g., $P_{1,2} \Rightarrow B_{2,2}$
- First-order logic (like natural language) can manipulate world models that include
  - **Objects:** people, houses, numbers, colors, baseball games, wars, …
  - **Relations:** red, round, prime, brother of, bigger than, part of, comes between, …
  - **Functions:** father, nationality, one more than, plus, …

  and **structured facts** such as
  - $Adjacent([x,y], [z,w]) \wedge Pit([x,y]) \Leftrightarrow Breeze([z,w])$

Basic syntax of FOL

• **Constant symbols**	***KingJohn, 2, NU, …***
• **Predicate symbols**	***IsHappy, Likes, >, …***
• **Function symbols**	***Sqrt, Nationality, …***
• **Variables**	***x, y, a, b, …***
• Connectives	¬, ∧, ∨, ⇒, ⇔
• Equality	=
• Quantifiers	∀, ∃

- Constant, predicate, and function symbols called a **"logical language"**
- Given LL we can define all logical sentences that can be expressed

Atomic sentences in FOL

- Atomic sentence = $predicate(term_1,\ldots,term_n)$
  or $term_1 = term_2$
- Term = $function(term_1,\ldots,term_n)$
  or constant or variable
- Examples
  - Brother(KingJohn, RichardTheLionheart)
  - >(AgeOf(Richard), AgeOf(John))
  - Brother(AgeOf(Richard), AgeOf(John))

Complex sentences

- Complex sentences are made from atomic sentences using connectives
  - Connectives have same semantics as propositional logic
  - $\neg S, S_1 \wedge S_2, S_1 \vee S_2, S_1 \Rightarrow S_2, S_1 \Leftrightarrow S_2$
- Examples
  - $Sibling(John,Richard) \Leftrightarrow Sibling(Richard,John)$
  - $>(1,2) \wedge \leq(1,2)$
  - $>(1,2) \wedge \neg>(1,2)$

Complex sentences (cont.)

- Additional complex sentences may include quantifiers: ∀ and ∃
- Syntax
  - ∀<*var*> [*S*]
  - ∃<*var*> [*S*]
- Short-hand notation
  - Abbreviate ∀*x* ∀*y* ∀*z* [*S*] as ∀*x,y,z* [*S*]
  - Abbreviate ∃*x* ∃*y* ∃*z* [*S*] as ∃*x,y,z* [*S*]

Break time!

Sign up for presentation teams.

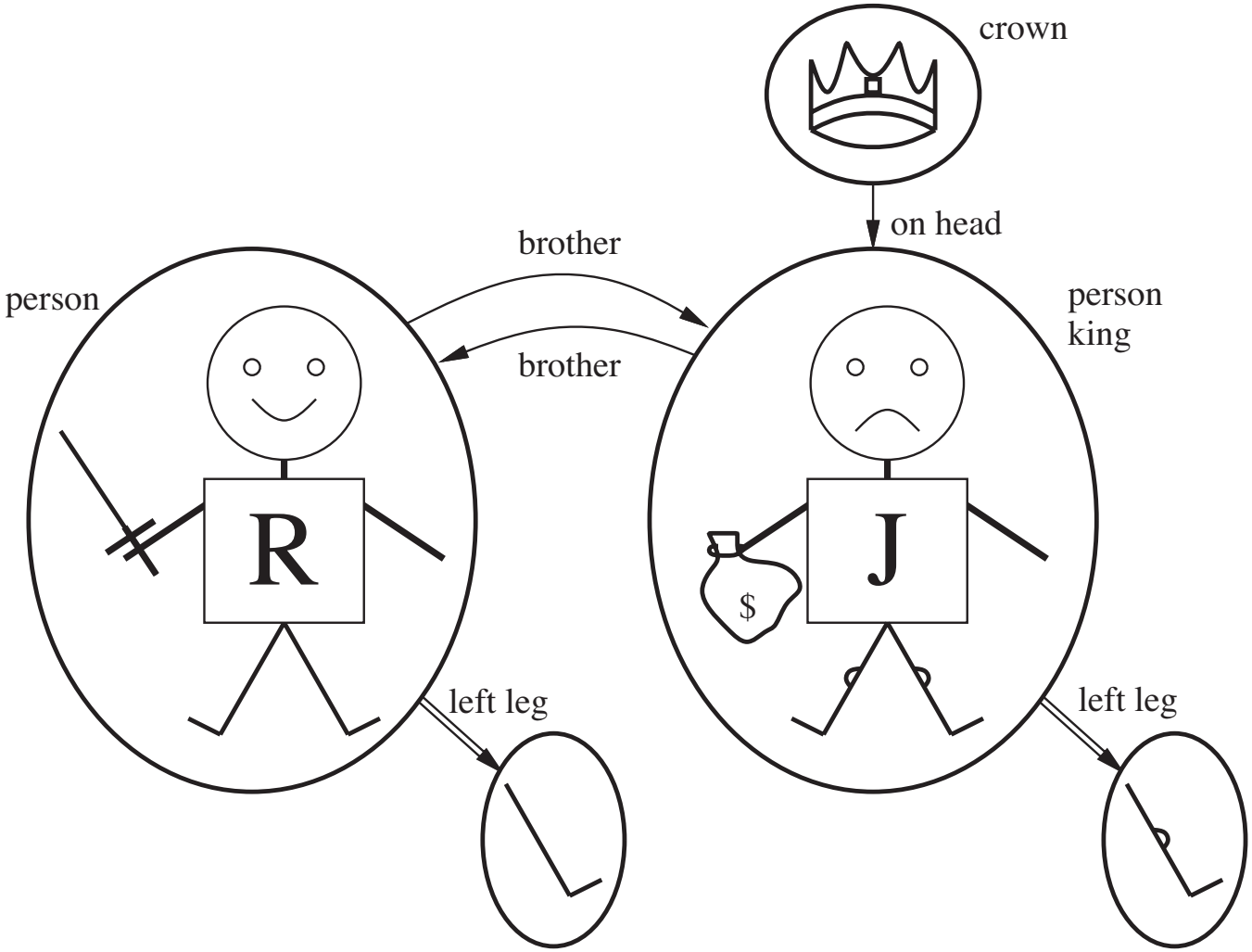
Meaning and truth in FOL

- Sentences are true w.r.t. a **model** and an **interpretation**
- FOL **model** contains objects (**domain elements**) and relations
  - PL model only had truth assignments to proposition symbol
- **Interpretation *I*** specifies referents for
  - Constant symbols → objects
  - Predicate symbols → relations
  - Function symbols → functions
- Atomic sentence $P(term_1,...,term_n)$ is true iff **objects** referred to by $term_1,...,term_n$ are in the **relation** $I(P)$

Meaning and truth in FOL (cont.)

- Complex sentences—truth is defined using same **truth tables**
  - E.g., $S_1 \wedge S_2$ is true iff S_1 is true and S_2 is true
- Semantics of quantifiers
  - $\forall x\ [S]$ is true iff, for any object C in the model
    $S[x/C]$ is true
  - $\exists x\ [S]$ is true iff, for at least one object C in the model
    $S[x/C]$ is true

FOL models example



Universal quantification examples

- ∀<*variables*> <*sentence*>
  - Everyone at NU is smart:
    $\forall x\ [\ At(x,NU) \Rightarrow Smart(x)\ ]$

- $\forall x$ 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 all possible **instantiations** of P

$$At(KingJohn,NU) \Rightarrow Smart(KingJohn) \quad \wedge$$
$$At(Richard,NU) \Rightarrow Smart(Richard) \quad \wedge$$
$$\mathrm{At}(NU,NU) \Rightarrow Smart(NU)$$

A common mistake to avoid with ∀

- Typically, ⇒ is the main connective with ∀

- **Common mistake:** using ∧ as the main connective with ∀

  E.g., $\forall x\ At(x,NU) \wedge Smart(x)$

  means "Everyone is at NU and everyone is smart"

Existential quantification examples

- ∃<*variables*> <*sentence*>
  - Someone at NU is smart:
    $\exists x \; [\, At(x,NU) \wedge Smart(x) \,]$

- $\exists x \; P$ 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 all possible **instantiations** of P

  $At(KingJohn,NU) \wedge Smart(KingJohn) \quad \vee$
  $At(Richard,NU) \wedge Smart(Richard) \quad \vee$
  $At(NU,NU) \wedge Smart(NU) \quad \vee \ldots$

A common mistake to avoid with ∃

- Typically, ∧ is the main connective with ∃

- **Common mistake:** using ⇒ as the main connective with ∃
  E.g., $\exists x\ At(x,NU) \Rightarrow Smart(x)$
  is true if there is no one who is at NU!

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
  - $\forall x\ Likes(x,IceCream) == \neg \exists x\ \neg Likes(x,IceCream)$
  - $\exists x\ Likes(x,Broccoli) == \neg \forall x\ \neg Likes(x,Broccoli)$

Equality in FOL

- $term_1 = term_2$ is true under a given interpretation iff $term_1$ and $term_2$ refer to the **same object**
  - E.g., definition of *Sibling* in terms of *Parent*
    $\forall x,y\ Sibling(x,y) \Leftrightarrow [\neg(x = y) \wedge \exists m,f \neg(m = f) \wedge Parent(m,x) \wedge Parent(f,x) \wedge Parent(m,y) \wedge Parent(f,y)]$

- We will use a different notation for equality: $=(x, y)$
  - Makes programming simpler

A model *M* for the kinship domain

- Individuals: *J K L M N O P Q R*
- Functions: *mom*[*1*] to *mom*(*N*) → *M*
- Relations[arity]
  - fem[*1*] = {*M, Q*}
  - par[*2*] = {[*M, N*], [*N, R*] . . }
  - sib[*2*] = {[*M, O*], [*P, J*], [*J, P*]}

----------**Interpretation *I*** ----------

- Constants: John, Mary, Sue, Tom .. ..
  - *I*(*Mary*) = *M*, *I*(*Sue*) = *Q*, . . .
- Function symbol: *Mother*, *I*(*Mother*) = *mom*
- Relation symbols: *Female*, *Parent*, *Sibling*

  I(*Female*) = *fem*, *I*(*Parent*) = *par*, *I*(*Sibling*) = *sib*

Using first-order logic

The kinship domain:

- Brothers are siblings

  $\forall x,y\ Brother(x,y) \Rightarrow Sibling(x,y)$

- "Sibling" is symmetric

  $\forall x,y\ Sibling(x,y) \Leftrightarrow Sibling(y,x)$

- One's mother is one's female parent

  $\forall m,c\ =(Mother(c)\ ,\ m) \Leftrightarrow (Female(m) \wedge Parent(m,c))$

- Some mothers are over 40 years old

  $\exists m,x\ =(\text{Mother}(x), m) \wedge\ >(Age(m), 40)$

Use of FOL to represent "common sense" knowledge

- All apples are red
- Some apples are red ("some" means at least one)
- All apples contain (some) worms
- Some apples contain (some) worms
- Every person is mortal
- Every person is male or female (but not both)

Use of FOL to represent "common sense" knowledge

- All apples are red

  $\forall x\ Apple(x) \Rightarrow Red(x)$

- Some apples are red ("some" means at least one)

  $\exists x\ Apple(x) \wedge Red(x)$

- All apples contain (some) worms

  $\forall x\ Apple(x) \Rightarrow \exists y\ Worm(y) \wedge Contains(x,y)$

- Some apples contain (some) worms

  $\exists x,y\ Apple(x) \wedge Worm(y) \wedge Contains(x,y)$

- Every person is mortal

  $\forall x\ Person(x) \Rightarrow Mortal(x)$

- Every person is male or female (but not both)

  $\forall x\ Person(x) \Rightarrow (Male(x) \wedge \neg Female(x)) \vee (Female(x) \wedge \neg Male(x))$

Wumpus world in FOL

- First step—define constants, function symbols, predicate symbols to express the facts

- ***Percept*(*data*, *t*)** means at time *t*, the agent perceived the *data* where *data* is a 5 element vector [*Stench*, *Breeze*, *Glitter*, *Bump*, *Scream*]
  - E.g., *Percept*([*None*, *Breeze*, *None*, *None*, *None*),2)

- ***At*(*Agent*, *s*, *t*)** means agent is at square *s* at time *t*
  - E.g., *At*(*Agent*, [2,1], 2)

Some Wumpus axioms

- Axiom for interpreting percepts in **context**
  $\forall x,t\ At(Agent, x, t) \wedge Breeze(t) \Rightarrow Breezy(x)$
- **Definitional axiom**
  $\forall a,b,c,d,t\ Percept([a, Breeze, b, c, d], t) \Rightarrow Breeze(t)$
- **Diagnostic Axiom**
  $\forall x\ Breezy(x) \Rightarrow \exists z\ Adjacent(z, x) \wedge Pit(z)$
- **Causal Axiom**
  $\forall z\ Pit(z) \Rightarrow (\forall x\ Adjacent(z, x) \Rightarrow Breezy(x))$
- World **model axioms**
  $Adjacent([1,1],[2,1])$ etc.
  $\forall x,y\ Adjacent(x, y) \Leftrightarrow Adjacent(y,x)$

Interacting with FOL KBs

- Wumpus world agent using FOL KB and perceives a stench and a breeze (but no glitter) at t = 5:
  Tell(KB, Percept[Stench, Breeze, None], 5]
  Ask(KB, $\exists a$ *BestAction*(a,5))
  - I.e., Does the KB entail some best action at time 5
  - Answer: {*a/Shoot*} ← **substitution** (binding list)
- Given a sentence *S* and a **substitution** σ
  - *S*σ denotes the result of plugging σ into *S*; e.g.,
    S = *Smarter*(*x*,*y*)
    σ = {*x*/*Hillary*,*y*/*Bill*}
    *S*σ = S {*x*/*Hillary*,*y*/*Bill*} = *Smarter*(*Hillary*,*Bill*)
- Ask(KB,*S*) returns some/all σ such that KB $\models$ *S*σ

Knowledge engineering—choosing representations

- Choice affects the **generality** at which concepts can be expressed
  - $Human(Bob)$ vs. $ISA(Bob, Human)$
  - $Green(B21)$ vs. $Color(B21, Green)$

- Inheritance rule:
  $\forall x,y,z\ ISA(x, y) \wedge ISA(y, z) \Rightarrow ISA(x, z)$

- Two blocks are the same color:
  $\exists x\ Color(B21, x) \wedge Color(B22, x)$

Knowledge engineering in FOL

1. Identify the task
2. Assemble the relevant knowledge
3. Decide on a vocabulary of predicates, functions, and constants (a logical language L)
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