

- Constants *KingJohn, 2, UB, ...*
- Predicates *Brother, >, ...*
- Functions *Sqrt, LeftLegOf, ...*
- Variables *x, y, a, b, ...*
- Connectives  $\wedge \vee \neg \Rightarrow \Leftrightarrow$
- Equality  $=$
- Quantifiers  $\forall \exists$

Atomic sentence = *predicate(term<sub>1</sub>, ..., term<sub>n</sub>)*  
 or *term<sub>1</sub> = term<sub>2</sub>*  
 Term = *function(term<sub>1</sub>, ..., term<sub>n</sub>)*  
 or *constant or variable*

*Brother(KingJohn, RichardTheLionheart)*  
 $> (Length(LeftLegOf(Richard)), Length(LeftLegOf(KingJohn)))$

SEMANTICS OF FOL

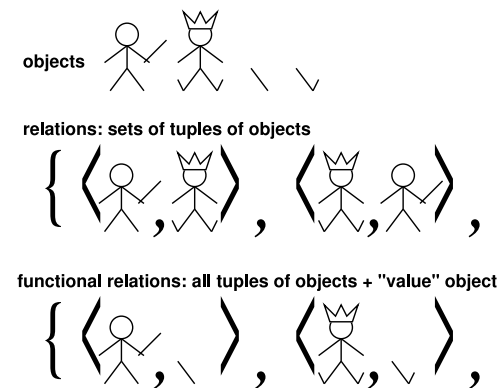
- Sentences are true with respect to a **model** and an **interpretation**.
    - The model contains objects and relations among them
    - An interpretation is a triple  $I = (D, \phi, \pi)$ , where
      - \*  $D$  (the **domain**) is a nonempty set; elements of  $D$  are **individuals**.
      - \*  $\phi$  is a mapping that assigns to each constant an element of  $D$ .
      - \*  $\pi$  is a mapping that assigns to each predicate with  $n$  arguments a function  $p : D^n \rightarrow \{True, False\}$  and to each function of  $k$  arguments a function  $f : D^k \rightarrow D$ .
- The interpretation specifies referents for
- constant symbols  $\rightarrow$  **objects** (individuals)
  - predicate symbols  $\rightarrow$  **relations**
  - function symbols  $\rightarrow$  **functional relations**
- An atomic sentence *predicate(term<sub>1</sub>, ..., term<sub>n</sub>)* is true iff the **objects** referred to by *term<sub>1</sub>, ..., term<sub>n</sub>* are in the **relation** referred to by *predicate*.

- 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)*  
 $>(1, 2) \vee \leq(1, 2)$   
 $>(1, 2) \wedge \neg >(1, 2)$

SEMANTICS OF FOL: EXAMPLE



## UNIVERSAL QUANTIFICATION

$\forall \langle \text{variable} \rangle \langle \text{sentence} \rangle$

- Everyone at Bishop's is smart:  $\forall x \text{ Attends}(x, \text{Bishops}) \Rightarrow \text{Smart}(x)$

$\forall x P$  is equivalent to the **conjunction of instantiations** of  $P$

$$\begin{aligned} & \text{Attends}(\text{KingJohn}, \text{Bishops}) \Rightarrow \text{Smart}(\text{KingJohn}) \\ \wedge & \text{Attends}(\text{Richard}, \text{Bishops}) \Rightarrow \text{Smart}(\text{Richard}) \\ \wedge & \text{Attends}(\text{Bishops}, \text{Bishops}) \Rightarrow \text{Smart}(\text{Bishops}) \\ \wedge & \dots \end{aligned}$$

- Do not use  $\wedge$  as the main connective with  $\forall$ :

$$\forall x \text{ Attends}(x, \text{Bishops}) \wedge \text{Smart}(x)$$

## UNIVERSAL QUANTIFICATION

$\forall \langle \text{variable} \rangle \langle \text{sentence} \rangle$

- Everyone at Bishop's is smart:  $\forall x \text{ Attends}(x, \text{Bishops}) \Rightarrow \text{Smart}(x)$

$\forall x P$  is equivalent to the **conjunction of instantiations** of  $P$

$$\begin{aligned} & \text{Attends}(\text{KingJohn}, \text{Bishops}) \Rightarrow \text{Smart}(\text{KingJohn}) \\ \wedge & \text{Attends}(\text{Richard}, \text{Bishops}) \Rightarrow \text{Smart}(\text{Richard}) \\ \wedge & \text{Attends}(\text{Bishops}, \text{Bishops}) \Rightarrow \text{Smart}(\text{Bishops}) \\ \wedge & \dots \end{aligned}$$

- Do not use  $\wedge$  as the main connective with  $\forall$ :

$$\forall x \text{ Attends}(x, \text{Bishops}) \wedge \text{Smart}(x)$$

means "Everyone attends Bishop's and everyone is smart"!

Typically,  $\Rightarrow$  is used instead.

## EXISTENTIAL QUANTIFICATION

$\exists \langle \text{variable} \rangle \langle \text{sentence} \rangle$

- Someone at Queen's is smart:  $\exists x \text{ Attends}(x, \text{Queens}) \wedge \text{Smart}(x)$

$\exists x P$  is equivalent to the **disjunction of instantiations** of  $P$

$$\begin{aligned} & \text{Attends}(\text{KingJohn}, \text{Queens}) \wedge \text{Smart}(\text{KingJohn}) \\ \vee & \text{Attends}(\text{Richard}, \text{Queens}) \wedge \text{Smart}(\text{Richard}) \\ \vee & \text{Attends}(\text{Queens}, \text{Queens}) \wedge \text{Smart}(\text{Queens}) \\ \vee & \dots \end{aligned}$$

- Do not use  $\Rightarrow$  as the main connective with  $\exists$ :

$$\exists x \text{ Attends}(x, \text{Queens}) \Rightarrow \text{Smart}(x)$$

## EXISTENTIAL QUANTIFICATION

$\exists \langle \text{variable} \rangle \langle \text{sentence} \rangle$

- Someone at Queen's is smart:  $\exists x \text{ Attends}(x, \text{Queens}) \wedge \text{Smart}(x)$

$\exists x P$  is equivalent to the **disjunction of instantiations** of  $P$

$$\begin{aligned} & \text{Attends}(\text{KingJohn}, \text{Queens}) \wedge \text{Smart}(\text{KingJohn}) \\ \vee & \text{Attends}(\text{Richard}, \text{Queens}) \wedge \text{Smart}(\text{Richard}) \\ \vee & \text{Attends}(\text{Queens}, \text{Queens}) \wedge \text{Smart}(\text{Queens}) \\ \vee & \dots \end{aligned}$$

- Do not use  $\Rightarrow$  as the main connective with  $\exists$ :

$$\exists x \text{ Attends}(x, \text{Queens}) \Rightarrow \text{Smart}(x)$$

is true if there is anyone who is not at Queen's!

Typically,  $\wedge$  is used instead.

## 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  $\forall y \exists x$ 
  - $\exists x \forall y \text{ Loves}(x, y)$  (“There is a person who loves everyone in the world”)
  - $\forall y \exists x \text{ Loves}(x, y)$  (“Everyone in the world is loved by at least one person”)
- Quantifier duality: each can be expressed using the other
  - $\forall x \text{ Likes}(x, \text{IceCream}) \quad \neg(\exists x \neg \text{Likes}(x, \text{IceCream}))$
  - $\exists x \text{ Likes}(x, \text{Broccoli}) \quad \neg(\forall x \neg \text{Likes}(x, \text{Broccoli}))$

## FOL AS A SECOND LANGUAGE

---

- Brothers are siblings.
- All animals eat custard.
- Everyone loves Arcand's movies.
- Jim likes Fred's stuff.
- A first cousin is a child of a parent's sibling

## FOL AS A SECOND LANGUAGE

---

- Brothers are siblings.  
 $\forall x \forall y \text{ Brother}(x, y) \Leftrightarrow \text{Sibling}(x, y)$
- All animals eat custard.
- Everyone loves Arcand's movies.
- Jim likes Fred's stuff.
- A first cousin is a child of a parent's sibling

## FOL AS A SECOND LANGUAGE

---

- Brothers are siblings.  
 $\forall x \forall y \text{ Brother}(x, y) \Leftrightarrow \text{Sibling}(x, y)$
- All animals eat custard.  
 $\forall x \text{ Animal}(x) \Rightarrow \text{Eats}(x, \text{Custard})$
- Everyone loves Arcand's movies.
- Jim likes Fred's stuff.
- A first cousin is a child of a parent's sibling

- Brothers are siblings.  
 $\forall x \forall y \text{ Brother}(x, y) \Leftrightarrow \text{Sibling}(x, y)$
- All animals eat custard.  
 $\forall x \text{ Animal}(x) \Rightarrow \text{Eats}(x, \text{Custard})$
- Everyone loves Arcand's movies.  
 $\forall x \forall y \text{ Person}(x) \wedge \text{DirectedBy}(y, \text{Arcand}) \Rightarrow \text{Likes}(x, y)$
- Jim likes Fred's stuff.  
 $\forall x \text{ Has}(\text{Fred}, x) \Rightarrow \text{Likes}(\text{Jim}, x)$
- A first cousin is a child of a parent's sibling

- Brothers are siblings.  
 $\forall x \forall y \text{ Brother}(x, y) \Leftrightarrow \text{Sibling}(x, y)$
- All animals eat custard.  
 $\forall x \text{ Animal}(x) \Rightarrow \text{Eats}(x, \text{Custard})$
- Everyone loves Arcand's movies.  
 $\forall x \forall y \text{ Person}(x) \wedge \text{DirectedBy}(y, \text{Arcand}) \Rightarrow \text{Likes}(x, y)$
- Jim likes Fred's stuff.  
 $\forall x \text{ Has}(\text{Fred}, x) \Rightarrow \text{Likes}(\text{Jim}, x)$
- A first cousin is a child of a parent's sibling  
$$\forall x \forall y \text{ FirstCousin}(x, y) \Leftrightarrow$$
  
$$\exists p \exists ps \text{ Parent}(p, x) \wedge \text{Sibling}(ps, p) \wedge \text{Parent}(ps, y)$$

- Brothers are siblings.  
 $\forall x \forall y \text{ Brother}(x, y) \Leftrightarrow \text{Sibling}(x, y)$
- All animals eat custard.  
 $\forall x \text{ Animal}(x) \Rightarrow \text{Eats}(x, \text{Custard})$
- Everyone loves Arcand's movies.  
 $\forall x \forall y \text{ Person}(x) \wedge \text{DirectedBy}(y, \text{Arcand}) \Rightarrow \text{Likes}(x, y)$
- Jim likes Fred's stuff.  
 $\forall x \text{ Has}(\text{Fred}, x) \Rightarrow \text{Likes}(\text{Jim}, x)$
- A first cousin is a child of a parent's sibling

Any sentence (or KB) can be transformed into a set of clauses (**clausal form**).

$$\neg((a \Leftrightarrow b) \vee (c \Rightarrow \neg(d \wedge (f \Rightarrow e))))$$

1. Eliminate  $\Leftrightarrow$  and  $\Rightarrow$ :  $\alpha \Rightarrow \beta$  is changed to  $\neg\alpha \vee \beta$ , and  $\alpha \Leftrightarrow \beta$  is equivalent to  $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$ .  
$$\neg(((\neg a \vee b) \wedge (\neg b \vee a)) \vee (\neg c \vee (\neg(d \wedge (\neg f \vee e))))))$$
2. Apply De Morgan rules to move all the negations in, and remove double negations.  
$$\neg(\neg(\neg a \vee b) \wedge \neg(\neg b \vee a)) \wedge \neg(\neg c \vee (\neg(d \wedge (\neg f \vee e))))$$
  
$$(\neg(\neg a \vee b) \vee \neg(\neg b \vee a)) \wedge (\neg\neg c \wedge (\neg\neg(d \wedge (\neg f \vee e))))$$
  
$$((a \wedge \neg b) \vee (b \wedge \neg a)) \wedge (c \wedge (d \wedge (\neg f \vee e)))$$
3. Use the distributiveness, associativity and commutativity to move the  $\wedge$ 's out:  $\alpha \vee (\beta \wedge \gamma)$  becomes  $(\alpha \vee \beta) \wedge (\alpha \vee \gamma)$ .  
$$((a \vee (b \wedge \neg a)) \wedge (\neg b \vee (b \wedge \neg a))) \wedge c \wedge d \wedge (\neg f \vee e)$$
  
$$(a \vee b) \wedge (a \vee \neg a) \wedge (\neg b \vee b) \wedge (\neg b \vee \neg a) \wedge c \wedge d \wedge (\neg f \vee e)$$
  
$$(a \vee b) \wedge (\neg b \vee \neg a) \wedge c \wedge d \wedge (\neg f \vee e)$$
4. Clausal form is more conveniently represented as a set of clauses:  
$$\{(a \vee b), (\neg b \vee \neg a), c, d, (\neg f \vee e)\}$$

## CLAUSAL FORM IN FOL

1. Eliminate  $\Leftrightarrow$  and  $\Rightarrow$ .
2. Apply De Morgan rules to move all the negations in, and remove double negations. Also move negations inside quantifiers:  $\neg(\forall x w)$  becomes  $(\exists x \neg w)$ , and  $\neg(\exists x w)$  becomes  $(\forall x \neg w)$ .
3. Standardize variables: rename variables such that no two different variables have the same name.

$$(\forall x P(x)) \vee (\exists x Q(x)) \rightsquigarrow (\forall x P(x)) \vee (\exists y Q(y))$$

4. Move all the quantifiers to the left.

$$(\forall x P(x)) \vee (\exists y Q(y)) \rightsquigarrow \forall x \exists y P(x) \vee Q(y)$$

## EQUALITY AND SUBSTITUTION

- $=$  is a predicate with the predefined meaning of **identity**:  $term_1 = term_2$  is true under a given interpretation iff  $term_1$  and  $term_2$  refer to the same object.
- Suppose a wumpus-world agent is using an FOL KB and perceives a smell and a breeze (but no glitter):

$Tell(KB, Percept([Smell, Breeze, None]))$   
 $Ask(KB, \exists a Action(a))$

I.e., does the KB entail any particular actions?

Possible answer: *Yes*,  $\{a/Shoot\}$  ← **substitution** (binding list)

- Given a sentence  $S$  and a substitution  $\sigma$ ,  $S_\sigma$  denotes the result of plugging  $\sigma$  into  $S$ ; e.g.,

$$S = Smarter(x, y) \\ \sigma = \{x/Hillary, y/Bill\} \\ S_\sigma = Smarter(Hillary, Bill)$$

- $Ask(KB, S)$  returns some/all  $\sigma$  such that  $KB \models S_\sigma$

## CLAUSAL FORM IN FOL (CONT'D)

5. Skolemization: Eliminate existential quantifiers in sentences such as

$$\forall x_1 \forall x_2 \dots \forall x_n \exists y w[x_1, x_2, \dots, x_n, y]$$

- if  $n = 0$ , invent a new constant  $C$  (Skolem constant) and replace  $y$  with  $C$  obtaining

$$\forall x_1 \forall x_2 \dots \forall x_n w[x_1, x_2, \dots, x_n, C]$$

- otherwise (i.e.,  $n \neq 0$ ), invent a new function symbol  $F$  (Skolem function) and replace  $y$  with  $F(x_1, x_2, \dots, x_n)$  obtaining

$$\forall x_1 \forall x_2 \dots \forall x_n w[x_1, x_2, \dots, x_n, F(x_1, x_2, \dots, x_n)]$$

$$\forall x \exists y P(x, y) \rightsquigarrow \forall x P(x, f(x)) \quad \exists y \forall x P(x, y) \rightsquigarrow \forall x P(x, c) \\ \exists v \forall w \exists x \forall y \exists z P(v, w, x, y, z) \rightsquigarrow \forall w \forall y P(C, w, F_2(w), y, F_1(w, y))$$

6. Erase all universal quantifiers, given that all the variables are introduced by such.
7. Use the distributiveness, associativity and commutativity to move the  $\wedge$ 's out.
8. (If you can) convert all the clauses to the Horn form  $\alpha_1 \wedge \dots \wedge \alpha_n \Rightarrow \beta$ .

## FOL PROOFS

**Model checking** completely out of question!

**Application of inference rules** sound generation of new sentences from old

**Proof** = a sequence of inference rule applications

Can use inference rules as operators in a standard search algorithm.

**Inference rules** **Generalized resolution**

$$\frac{\alpha \vee \beta', \quad \neg\beta'' \vee \gamma, \quad \exists \sigma \beta = \beta'_\sigma \wedge \beta = \beta''_\sigma}{\alpha_\sigma \vee \gamma_\sigma}$$

and **Generalized modus ponens**

$$\frac{\alpha_1, \dots, \alpha_n, \quad \alpha'_1 \wedge \dots \wedge \alpha'_n \Rightarrow \beta, \quad \exists \sigma (\alpha_1)_\sigma = (\alpha'_1)_\sigma \wedge \dots \wedge (\alpha_n)_\sigma = (\alpha'_n)_\sigma}{\beta_\sigma}$$

## PROOF BY CONTRADICTION

<b>KB</b> Bob is a buffalo Pat is a pig Buffaloes outrun pigs	1. $Buffalo(Bob)$ 2. $Pig(Pat)$ 3. $Buffalo(x) \wedge Pig(y) \Rightarrow Faster(x, y)$
<b>Query</b> Is something outrun by something else? <b>Negated query:</b>	4. $Faster(u, v) \Rightarrow \square$
(1), (2), and (3), $\sigma = \{x/Bob, y/Pat\}$ (4) and (5), $\sigma = \{u/Bob, v/Pat\}$	5. $Faster(Bob, Pat)$ $\square$

- All the techniques presented with respect to propositional logic work (inference rules, control strategies), except that in FOL each application of the inference rule generates a substitution.
- All the substitutions regarding variables appearing in the query are typically reported (why?).

## UNIFICATION (CONT'D)

Unify:	With:	Substitution:
$Dog$	$Dog$	$\emptyset$
$x$	$y$	$\{x/y\}$
$x$	$A$	$\{x/A\}$
$F(x, G(T))$	$F(M(H), G(m))$	$\{x/M(H), m/T\}$
$F(x, G(T))$	$F(M(H), t(m))$	Failure!
$F(x)$	$F(M(H), T(m))$	Failure!
$F(x, x)$	$F(y, L(y))$	Failure!

- Equality, revised:**  $=$  is a predicate with the predefined meaning of **identity**:  $term_1 = term_2$  is true under a given interpretation iff  $term_1$  and  $term_2$  **unify with each other**.

## UNIFICATION

$$\frac{\alpha \vee \beta', \quad \neg\beta'' \vee \gamma, \quad \exists \sigma \beta = \beta'_\sigma \wedge \beta = \beta''_\sigma}{\alpha_\sigma \vee \gamma_\sigma}$$

- OK, we have to determine substitutions, but how do we do it?
  - We look for the **most general substitution**.

**KB**  $Short(LeftLegOf(Richard))$

**Queries**

$Short(x)$   $\sigma = \{x/???\}$   
 $Short(LeftLegOf(x))$   $\sigma = \{x/???\}$

- The process of determining the most general substitution is called **unification**.
  - The substitution produced by such an algorithm is often referred to as the **most general unifier**.

## THE UNIFICATION ALGORITHM

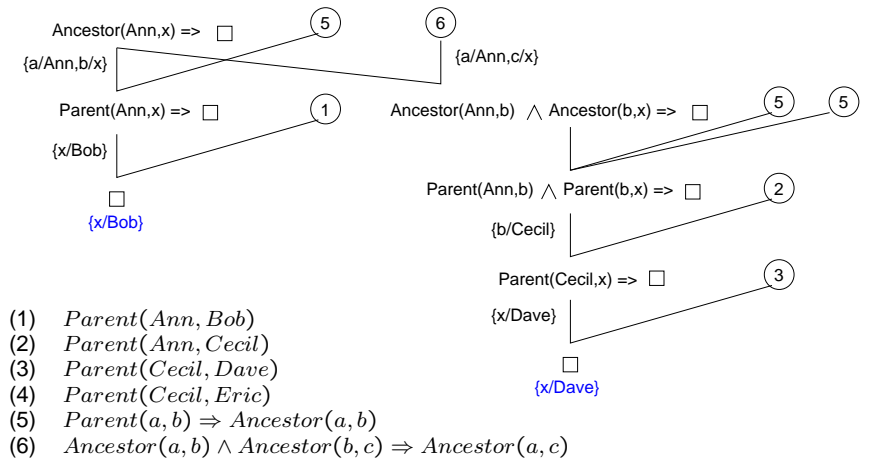
**function** UNIFY( $A, B$ : terms) **returns** failure or the most general unifier

- $\sigma \leftarrow \emptyset$
- repeat while**  $A_\sigma \neq B_\sigma$ :
  - if**  $A_\sigma$  and  $B_\sigma$  have different arities **or**  $A_\sigma$  and  $B_\sigma$  have different functors, **then fail**.
  - repeat for each** argument  $t_A$  of  $A_\sigma$  (and thus for each argument  $t_B$  of  $B_\sigma$ )
    - if**  $t_A$  and  $t_B$  are the same constant **then go to (b)**
    - if** neither  $t_A$  nor  $t_B$  are variables or functions, **or** one is a variable occurring in the other **then fail**.
    - if**  $t_A$  is a variable, **then**  $\sigma \leftarrow \sigma \cup \{t_A/t_B\}$ .
    - else if**  $t_B$  is a variable **then**  $\sigma \leftarrow \sigma \cup \{t_B/t_A\}$ .
    - else if** both  $t_A$  and  $t_B$  are functions **and** if UNIFY( $t_B, t_A$ ) does not fail **then**  $\sigma \leftarrow \sigma \cup \text{UNIFY}(t_B, t_A)$ .
    - else fail**.
- return**  $\sigma$ , the most general unifier of  $A$  and  $B$ .

Is there such thing as multiple solutions???

FORWARD AND BACKWARD CHAINING

- Modus ponens: If  $a$  is true and  $a \Rightarrow b$  then  $b$  is true.
  - We use it in **forward chaining**: we start with the set of clauses (the KB plus the negated conclusion) and we keep inferring clauses until we infer  $\square$ .
- But we can use modus ponens the other way around too: If  $b$  is false and  $a \Rightarrow b$  then  $a$  must be false.
  - This is another way of saying basically the same thing, but with a twist: we use **backward chaining**.
  - We start with the assumption that the conclusion is true and we prove that this holds only if  $\square$  pertains to the KB.
  - Why would one bother with this twisted way of thinking? It is easier for a machine to organize the proof in this way.



FUN WITH LISTS

- We could choose a function to represent a cons cell, e.g.
 
$$(cons\ a\ b) \rightsquigarrow .(a, b)$$
- We could also choose a constant to represent the empty list, e.g.,
 
$$NIL \rightsquigarrow []$$
- Let's write a predicate on lists:
 
$$\neg member(a, [])$$

$$member(a, .(a, b))$$

$$member(a, c) \Rightarrow member(a, .(b, c))$$
- What is the result of the following queries?
 
$$member(Joe, [])$$

$$member(Jack, .(Joe, .(Jack, .(Jill, []))))$$

$$member(x, .(Joe, .(Jack, .(Jill, []))))$$

- The inference rules (resolution, modus ponens) are the same as in propositional logic
  - except that, unification is used instead of identity.
- All the control of the inference process from propositional logic (unit resolution, input resolution, heuristics/preferences) apply, including the discussed completeness considerations.
  - See more control strategies on p. 285.

FOL COMPLETENESS (CONT'D)

- There exist problems that cannot be solved by a computer no matter how many megahertz or gigabytes (cf. Alan Turing, circa 1935).
- You can write a program that does inference using resolution and a general control strategy (e.g., breadth-first search).
- You can express any problem using FOL (Church thesis).

- Modus ponens is **not refutation-complete**, but it is so for Horn KBs.

$$\left. \begin{array}{l} PhD(x) \Rightarrow HighlyQualified(x) \\ \neg PhD(x) \Rightarrow EarlyEarnings(x) \\ HighlyQualified(x) \Rightarrow Rich(x) \\ EarlyEarnings(x) \Rightarrow Rich(x) \end{array} \right\} \models Rich(Me)$$

- Resolution is **refutation-complete** for FOL.
- How about completeness (as opposed to refutation-completeness)?

FOL COMPLETENESS (CONT'D)

- There exist problems that cannot be solved by a computer no matter how many megahertz or gigabytes (cf. Alan Turing, circa 1935).
- You can write a program that does inference using resolution and a general control strategy (e.g., breadth-first search).
- You can express any problem using FOL (Church thesis).
- Ergo, no inference method is complete, not even resolution!  
In other words, entailment in FOL is only **semidecidable**:

can find a proof of  $\alpha$  if  $KB \models \alpha$   
cannot always prove that  $KB \models \alpha$

... or computation as inference on logical KBs

<b>Logic programming</b>	<b>Ordinary programming</b>
1. Identify problem	Identify problem
2. Assemble information	Assemble information
3. Coffee break	Figure out solution
4. Encode information in KB	Program solution
5. Encode problem instance as facts	Encode problem instance as data
6. Ask queries	Apply program to data
7. Find false facts	Debug procedural errors