CS 316: First-order logic

Stefan D. Bruda

Winter 2023

Syntax of FOL

- Basic ingredients:
  - Constants — $KingJohn,\ 2,\ UB, \ldots$
  - Predicates — $Brother,\ >, \ldots$
  - Functions — $Sqrt,\ LeftLegOf, \ldots$
  - Variables — $x,\ y,\ a,\ b, \ldots$
  - Connectives — $\wedge\ \vee\ \neg\ \Rightarrow \Leftrightarrow$
  - Equality — $=$
  - Quantifiers — $\forall\ \exists$
- Complex constructs:
  - Atomic sentence — $predicate(term_1, \ldots, term_n)$ or $term_1 = term_2$
  - Term — $function(term_1, \ldots, term_n)$ or $constant$ or $variable$

  $Brother(KingJohn, RichardTheLionheart)$

  $> (Length(LeftLegOf(Richard)), Length(LeftLegOf(KingJohn)))$

  - Complex sentences are made from atomic sentences using connectives

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

  $Sibling(KingJohn, Richard) \Rightarrow Sibling(Richard, KingJohn)$

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

Semantics of FOL

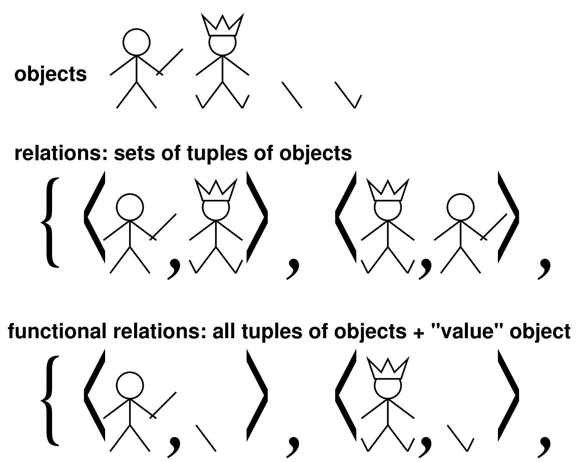
- Sentences are true with respect to a model and an interpretation
  - The model contains objects and relations among them
  - An interpretation is a triple $I = (D, \phi, \pi)$, where
    - D (the domain) is a nonempty set; elements of D are individuals
    - ϕ is a mapping that assigns to each constant an element of D
    - π is a mapping that assigns to each predicate with n arguments a function $p : D^n \to \{True, False\}$ and to each function of k arguments a function $f : D^k \to D$
- The interpretation specifies referents for

  constant symbols $\to$ objects (individuals)

  predicate symbols $\to$ relations

  function symbols $\to$ functional relations
- An atomic sentence $predicate(term_1, \ldots, term_n)$ is true iff the objects referred to by $term_1, \ldots, term_n$ are in the relation referred to by $predicate$

Semantics of FOL: Example



UNIVERSAL QUANTIFICATION

$\forall \langle variable \rangle \ \langle sentence \rangle$

- Everyone at Bishop's is smart: $\forall x \ \ Attends(x, Bishops) \Rightarrow Smart(x)$
  $\forall x \ \ P$ is equivalent to the conjunction of instantiations of P

$$\begin{array}{lll} & Attends(KingJohn, Bishops) & \Rightarrow & Smart(KingJohn) \\ \wedge & Attends(Richard, Bishops) & \Rightarrow & Smart(Richard) \\ \wedge & Attends(Bishops, Bishops) & \Rightarrow & Smart(Bishops) \\ \wedge & \ldots \end{array}$$

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

$$\forall x \ \ Attends(x, Bishops) \wedge Smart(x)$$

  "Everyone attends Bishop's and everyone is smart"!
  Typically, $\Rightarrow$ is used instead

EXISTENTIAL QUANTIFICATION

$\exists \langle variable \rangle \ \langle sentence \rangle$

- Someone at Queen's is smart: $\exists x \ \ Attends(x, Queens) \wedge Smart(x)$
  $\exists x \ \ P$ is equivalent to the disjunction of instantiations of P

$$\begin{array}{llll} & Attends(KingJohn, Queens) & \wedge & Smart(KingJohn) \\ \vee & Attends(Richard, Queens) & \wedge & Smart(Richard) \\ \vee & Attends(Queens, Queens) & \wedge & Smart(Queens) \\ \vee & \ldots \end{array}$$

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

$$\exists x \ \ Attends(x, Queens) \Rightarrow Smart(x)$$

  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 $\forall y \ \exists x$
  - $\exists x \ \forall y \ \ Loves(x, y)$ ("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
  - $\forall x \ \ P(x)$ is equivalent to $\neg(\exists x \ \ \neg P(x))$
  - $\exists x \ \ P(x)$ is equivalent to $\neg(\forall x \ \ \neg P(x))$

$$\begin{array}{rcl} \forall x \ \ Likes(x, IceCream) & \equiv & \neg(\exists x \ \ \neg Likes(x, IceCream)) \\ \exists x \ \ Likes(x, Broccoli) & \equiv & \neg(\forall x \ \ \neg Likes(x, Broccoli)) \end{array}$$

FOL AS A SECOND LANGUAGE

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

$$\begin{array}{c} \forall x \ \forall y \ \ FirstCousin(x, y) \Leftrightarrow \\ \exists p \ \exists ps \ \ Parent(p, x) \wedge Sibling(ps, p) \wedge Parent(ps, y) \end{array}$$

Clausal form in propositional logic

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.
$$\neg((\neg a \vee b) \wedge (\neg b \vee a)) \wedge \neg(\neg c \vee (\neg(d \wedge (\neg f \vee e))))$$
$$(\neg(\neg a \vee b) \vee \neg(\neg b \vee a)) \wedge (\neg\neg c \wedge (\neg\neg(d \wedge (\neg f \vee e))))$$
$$((a \wedge \neg b) \vee (b \wedge \neg a)) \wedge (c \wedge (d \wedge (\neg f \vee e)))$$
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
$$(\forall x\ \ P(x)) \vee (\exists y\ \ Q(y)) \ \rightsquigarrow\ \forall x\ \exists y\ \ P(x) \vee Q(y)$$

Clausal form in FOL (cont'd)

5. Skolemization: Eliminate existential quantifiers in sentences having the following form:
$$\forall x_1\ \forall x_2\ \ldots \forall x_n\ \exists y\ \ w[x_1, x_2, \ldots, x_n, y]$$
   - If $n = 0$ then invent a new constant C (Skolem constant) and replace y with C obtaining
   $$\forall x_1\ \forall x_2\ \ldots \forall x_n\ \ w[x_1, x_2, \ldots, x_n, C]$$
   - Otherwise (i.e., $n \neq 0$), invent a new function symbol F (Skolem function) and replace y with $F(x_1, x_2, \ldots, x_n)$ obtaining
   $$\forall x_1\ \forall x_2\ \ldots \forall x_n\ \ w[x_1, x_2, \ldots, x_n, F(x_1, x_2, \ldots, x_n)]$$

$$\forall x\ \exists y\ \ P(x,y) \ \rightsquigarrow\ \forall x\ \ P(x, F(x)) \qquad \exists y\ \forall x\ \ P(x,y) \ \rightsquigarrow\ \forall x\ \ P(x, C)$$
$$\exists v\ \forall w\ \exists x\ \forall y\ \exists z\ \ P(v,w,x,y,z) \ \rightsquigarrow\ \forall w\ \forall y\ \ P(C, w, F_2(w), y, F_1(w,y))$$

6. Erase all universal quantifiers (all the variables are introduced by them)
7. Use the distributiveness, associativity and commutativity to move the $\wedge$'s out, thus obtaining the clausal form
8. (If possible) convert all the clauses to the Horn form $\alpha_1 \wedge \cdots \wedge \alpha_n \Rightarrow \beta$

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]))
- Does the KB entail any particular actions?
  $$Ask(KB, \exists a\ \ Action(a))$$
- Possible answer: *Yes*, $\{a/Shoot\} \leftarrow$ substitution (binding list)
  - Given a sentence S and a substitution σ, S_σ denotes the result of plugging σ into S
  - Example:
    $S = Smarter(x,y)$
    $\sigma = \{x/Hillary, y/Bill\}$
    $S_\sigma = Smarter(Hillary, Bill)$
  - $Ask(KB, S)$ returns some/all σ such that $KB \models S_\sigma$

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

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

  - Generalized modus ponens

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

Proof by contradiction

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

- 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

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

- We need to determine a suitable substitutions and there are many ways to do it, how do we go about it?

  KB $Short(LeftLegOf(Richard))$
  Queries $Short(x)$ $\sigma = \{x/???\}$
  $Short(LeftLegOf(x))$ $\sigma = \{x/???\}$
- We look for the most general substitution
  - $\sigma = \{x/norvig, y/AIMA, z/AIMA\}$ is a substitution that makes $book(x, y)$ and $book(norvig, z)$ agree, but it is not the most general
- 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

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 algorithm

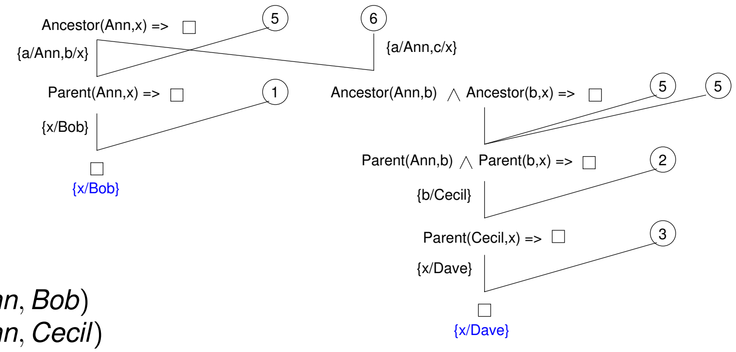
function UNIFY(A, B: terms, σ: substitution) **returns** failure or substitution

- Initial call: UNIFY(A, B, $\emptyset$)
- A is bound to X in σ whenever $A/X \in \sigma$, otherwise A is free

1. **if** A and B are both atoms **and** $A = B$ **then return** σ
2. **if** A is a variable that occurs in B **or** B is a variable that occurs in A **then return** failure
3. **if** A is a free variable **then return** $\sigma \cup \{A/B\}$
4. **if** B is a free variable **then return** $\sigma \cup \{B/A\}$
5. **if** $A/X \in \sigma$ **then return** UNIFY(X, B, σ)
6. **if** $B/X \in \sigma$ **then return** UNIFY(A, X, σ)
7. **if** $A = p(a_1, a_2, \ldots, a_n)$ and $B = p(b_1, b_2, \ldots, b_n)$
   1. **for** $i \leftarrow 1$ **to** n **do**
      1. $\alpha \leftarrow$ UNIFY(a_i, b_i, σ)
      2. **if** $\alpha =$ failure **then return** failure
      3. $\sigma \leftarrow \sigma \cup \alpha$
   2. **return** σ
8. **return** failure

Multiple solutions

Is there such thing as multiple solutions? Yes!



(1) $Parent(Ann, Bob)$
(2) $Parent(Ann, Cecil)$
(3) $Parent(Cecil, Dave)$
(4) $Parent(Cecil, Eric)$
(5) $Parent(a, b) \Rightarrow Ancestor(a, b)$
(6) $Ancestor(a, b) \wedge Ancestor(b, c) \Rightarrow Ancestor(a, c)$

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 □
- 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 assumtion that the conclusion is true and we prove that this holds only if □ belongs to the KB
  - The big advantage of backward chaining is that it often expands a much smaller portion of the AND/OR graph than forward chaining

Fun with lists

- A singly linked list is either empty (`NIL`) or a pointer to a cons cell `cons(a,b)` where `a` is the value at the head of the list and `b` is (recursively) a list
- A logical representation would use a function to represent a cons cell, e.g.

$$\texttt{cons(a,b)} \quad \rightsquigarrow \quad .(a, b)$$

- We also choose a constant to represent the empty list, e.g.,

$$\texttt{NIL} \quad \rightsquigarrow \quad []$$

- We can now write a predicate on lists like this:
  $\neg member(a, [])$
  $member(a, .(a, b))$
  $member(a, c) \Rightarrow member(a, .(b, c))$
- Check out the result of the following queries:
  $member(Joe, [])$
  $member(Jack, .(Joe, .(Jack, .(Jill, []))))$
  $member(x, .(Joe, .(Jack, .(Jill, []))))$

FOL inference summary

- 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
  - More control strategies are also possible, see some more in Section 9.5.6 (p. 308)

FOL completeness

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

    can find a proof of α if KB $\models \alpha$, but cannot always prove that KB $\not\models \alpha$