Lecture 11, Mar 26, 2024

First-Order Logic

- We generalize propositional logic to have the notion of variables
- *First-order logic* consists of the following components:
  - A set of *variables*, V
    - These can take values from a domain D
  - A set of *predicate/relation symbols* $P^k : D^k \mapsto \{ 0, 1 \}$ where k is the number of arguments
    - These take a set of arguments (variables) and can be true or false, depending on the value of the variables
    - P^0 is the set of predicates that don't take any arguments, which is the set of propositions
    - These define relations among variables
  - A set of *function symbols* $f^k : D^k \mapsto D$ where k is the number of arguments
    - These define functions based on the variables, returning another variable
    - A special case of the relations
  - The *quantifiers* $\forall$ and $\exists$
- Define the set of all terms:
  - $\mathrm{TERM}_{i+1} = \mathrm{TERM}_i \cup \{ f_n^k(t_1, \dots, t_k) \mid t_1, \dots, t_k \in \mathrm{TERM}_i, \forall n, k \}$
  - $\mathrm{TERM}_0 = V$
- Define the set of all well-formed formulas:
  - $$\begin{aligned} \mathrm{FORM}_{i+1} = \quad & \mathrm{FORM}_i \\ & \cup \{ (\alpha \circ \beta) \mid \alpha, \beta \in \mathrm{FORM}_i \} \\ & \cup \{ (\neg \alpha) \mid \alpha \in \mathrm{FORM}_i \} \\ & \cup \{ \forall x \varphi \mid x \in V, \varphi \in \mathrm{FORM}_i \} \\ & \cup \{ \exists x \varphi \mid x \in V, \varphi \in \mathrm{FORM}_i \} \end{aligned}$$
    - We augment our definition from propositional logic with the new quantifiers $\forall$ and $\exists$
  - $\mathrm{FORM}_0 = \{ P_n^k(t_1, \dots, t_k) \mid t_1, \dots, t_k \in \mathrm{TERM}, \forall n, k \}$
    - This is the set of all predicates over all terms
- Consider the expression $\forall x \exists y (x + y > 3)$
  - Formally we express this as $\forall x \exists y (> (+(x, y), 3))$
  - $+$ is a function and $>$ is a predicate
  - We need to define the domain of all variables, and define the meaning of $+$ and $>$
- A *context* or *structure* consists of $\mathcal{A} = (D, f_i^{k,\mathcal{A}}, \dots, P_i^{k,\mathcal{A}}, \dots)$, which is a domain and definition of all the functions and predicates
  - $f_i^{k,\mathcal{A}}$ defines the meaning of function f_i^k in the context of $\mathcal{A}$
  - $P_i^{k,\mathcal{A}}$ defines the meaning of predicate P_i^k in the context $\mathcal{A}$
  - The definitions assign $D^k \mapsto \{ 0, 1 \}$ for every combination of the values of the variables
- A k-ary function can be converted into a predicate by adding an extra argument; so functions are syntactic sugar that's not needed to define first-order logic
- An *assignment* is $\sigma : V \mapsto D$ which gives a value to all variables
  - We need to assign values to variables before evaluating some expressions, e.g. $\forall x (x + y > 3)$
  - $\sigma(x \mapsto m)$ or equivalently $\sigma(x/m)$ denotes the value m being assigned to x
- Similar to propositional logic $\mathcal{A}, \sigma \models \varphi$ if φ is true under the structure $\mathcal{A}$ and assignment σ
  - Note that in ϕ we might have variables appearing in quantifiers that have been assigned a value by σ; in this case we don't care about the assignment in σ
  - $\exists x \psi$ iff there exists $m \in D$ such that $\mathcal{A}, \sigma(x \mapsto m) \models \psi$
  - $\forall x \psi$ iff for all $m \in D$ we have $\mathcal{A}, \sigma(x \mapsto m) \models \psi$
- The *extension* of σ is $\bar{\sigma} : \mathrm{TERM} \mapsto D$ which assigns a value to all terms
  - This can be defined recursively, since a term is either a variable or a function of terms
  - $\bar{\sigma}(t) = f_i^k(\bar{\sigma}(t_1), \dots, \bar{\sigma}(t_k))$ for $t \in \mathrm{TERM}$
  - Base case is $\bar{\sigma}(t) = \sigma(t)$ for $t \in V$
- We are now interested in the analog of the relevance lemma from propositional logic

  - Not all variables in formulas are *free variables*, e.g. for $\exists x(x + y > 3)$, x is not a free variable because of the $\exists$, and it doesn't matter what value σ assigns to it
- FreeVars: $\text{FORM} \mapsto 2^V$, a mapping from formulas to sets of variables
  - $\text{FreeVars}(\forall x\varphi) = \text{FreeVars}(\varphi) \setminus \{ x \}$
  - $\text{FreeVars}(\exists x\varphi) = \text{FreeVars}(\varphi) \setminus \{ x \}$
  - $\text{FreeVars}(\neg\varphi) = \text{FreeVars}(\varphi)$
  - $\text{FreeVars}(\varphi \circ \psi) = \text{FreeVars}(\varphi) \cup \text{FreeVars}(\psi)$
  - $\text{FreeVars}(P_i^k(t_1, \ldots, t_k)) = \text{FreeVars}(t_1) \cup \ldots \cup \text{FreeVars}(t_k)$ for $t_n \in \text{TERM}$
  - $\text{FreeVars}(f_i^k(t_1, \ldots, t_k)) = \text{FreeVars}(t_1) \cup \ldots \cup \text{FreeVars}(t_k)$ for $t_n \in \text{TERM}$
  - $\text{FreeVars}(x) = \{ x \}$ for $x \in V$
- *Relevance lemma*: if $\forall x \in \text{FreeVars}(x)$, $\sigma_1(x) = \sigma_2(x)$, then $\mathcal{A}, \sigma_1 \models \varphi$ iff $\mathcal{A}, \sigma_2 \models \varphi$
  - This has the same interpretation as the relevance lemma for propositional logic
- Define $\mathcal{A} \models \varphi$ iff $\forall\sigma(\mathcal{A}, \sigma \models \varphi)$, i.e. φ is always satisfied in structure $\mathcal{A}$ (analog of valid formulas)
  - Likewise $\mathcal{A} \not\models \varphi$ iff $\forall\sigma(\mathcal{A}, \sigma \not\models \varphi)$ (analog of unsatisfiable formulas)
  - We only need to care about the free variables, since the non-free ones don't affect whether φ is modelled
- If $\text{FreeVars}(\varphi) = \varnothing$, then φ is a *sentence*
  - We can use sentences to store our knowledge in a knowledge base
- Define $\varphi \models \psi$ if the set of all assignments that model φ is a subset of all assignments that model ψ (so if φ is modelled by an assignment, ψ will also be)
  - $\text{models}(\varphi) \subseteq \text{models}(\psi)$
  - Alternatively $\text{models}(\varphi) \cap \overline{\text{models}(\psi)} = \varnothing$
  - Equivalently $\text{models}(\varphi \wedge \neg\psi) = \varnothing$
  - If φ is a knowledge base of sentences, we use this to check if a formula is true
- $p \wedge (\neg p)$ is an *empty clause*, denoted $()$, which is a contradiction
- $(\alpha \vee p) \wedge (\neg p \vee \beta)$ gives $(\alpha \vee \beta)$
  - This is known as *resolution*
  - This is the transitivity of implication, since $\neg x \vee y$ means $x \to y$
- Given some logical statement we can keep applying resolution, and eventually if we end up with an empty clause, we know the original statement was false because it leads to a contradiction
- Therefore if we want to check if our knowledge base models some formula α, we can check if $KB \wedge (\neg\alpha)$ leads to an empty clause