**National University of Singapore**
**School of Computing**
**CS3243 Introduction to AI**

# Tutorial 6: First-Order Logic

Issue: March 20, 2017 Due: March 24, 2017

Important Instructions:

- *Your solutions for this tutorial must be TYPE-WRITTEN.*
- *Make TWO copies of your solutions: one for you and one to be SUBMITTED TO THE TUTOR IN CLASS. Your submission in your respective tutorial class will be used to indicate your CLASS ATTENDANCE. Late submission will NOT be entertained.*
- *YOUR SOLUTION TO QUESTION* $2$ *will be GRADED for this tutorial.*
- *You may discuss the content of the questions with your classmates. But everyone should work out and write up ALL the solutions by yourself.*

1. (Modified Question $8.26$ from AIMA 3rd edition) Represent the following sentences in first-order logic, using a consistent vocabulary defined as follows:

   $Took(x, y, z)$ : is true if student $x$ took subject $y$ in semester $z$
   $Score(x, y, z)$ : is true if student $x$ obtains score $z$ in subject $y$
   $Passed(x, y)$ : is true if student $x$ passed subject $y$
   $Buys(x, p)$ : is true if person $x$ buys policy $p$
   $IsSmart(x)$ : is true if person $x$ is smart
   $IsExpensive(x)$ : is true if $x$ is expensive
   $Sells(x, y, p)$: is true if person $x$ sells policy $p$ to person $y$
   $IsInsured(x)$: is true if person $x$ is insured
   $IsBarber(x)$: is true if $x$ is a barber
   $Shaves(x, y)$: is true if person $x$ shaves person $y$

   (a) Some students took French in Spring 2001.
   $\exists x : Took(x, French, Spring2001)$

   (b) Every student who takes French passes it.
   $\forall x, y : Took(x, French, y) \Rightarrow Passed(x, French)$

(c) Only one student took Greek in Spring 2001.
$\exists x : \mathit{Took}(x, \mathit{Greek}, \mathit{Spring}2001) \wedge \forall y : (\mathit{Took}(y, \mathit{Greek}, \mathit{Spring}2001) \Rightarrow y = x)$

(d) The best score in Greek is always higher than the best score in French.
There are many ways of defining this, but here is one:

$$\forall x, s : \mathit{Score}(x, \mathit{French}, s) \Rightarrow \exists y, s' : \mathit{Score}(y, \mathit{Greek}, s') \wedge s' > s.$$

This roughly translates to 'for every student $x$ who took French and got a score $s$ there is some student $y$ who took Greek, got $s'$ and $s' > s$. One can first define the property $\mathit{Best}(t)$ which is the best score of any student who took topic $t$, and then state that $\mathit{Best}(\mathit{Greek}) > \mathit{Best}(\mathit{French})$. One can also insist that there is some student who took Greek and some student who took French (add existential quantifiers appropriately).

(e) Everyone who buys a policy is smart. $\forall x, p : \mathit{Buys}(x, p) \Rightarrow \mathit{IsSmart}(x)$

(f) No person buys an expensive policy.

$$\neg\exists x, p : \mathit{Buys}(x, p) \wedge \mathit{IsExpensive}(p)$$

or equivalently

$$\forall x, p : \mathit{Buys}(x, p) \Rightarrow \neg\mathit{IsExpensive}(p).$$

(g) There is an agent who sells policies only to those people who are not insured.

$$\exists x : \forall y, p : \mathit{Sells}(x, y, p) \Rightarrow \neg\mathit{IsInsured}(y).$$

(h) There is a barber who shaves all men in town who do not shave themselves.

$$\exists x : \mathit{IsBarber}(x) \wedge \forall y : \neg\mathit{Shaves}(y, y) \Rightarrow \mathit{Shaves}(x, y).$$

(i) There is a barber who shaves all men in town who does not shave himself.

$$\exists x : \mathit{IsBarber}(x) \wedge (\forall y : \mathit{Shaves}(x, y)) \wedge \neg\mathit{Shaves}(x, x).$$

2. Recall that a CNF formula $\phi$ is defined over a set of variables $X = \{x_1, \ldots, x_n\}$; a truth assignment assigns true/false (or equivalently 1 or 0) to every variable $x_i \in X$. Thus, it is useful to think of $\phi$ as a mapping from $\{0,1\}^n$ to $\{0,1\}$. We say that an assignment $\vec{a} \in \{0,1\}^n$ satisfies $\phi$ if $\phi(\vec{a}) = 1$. A $k$-CNF formula is one where each clause contains at most $k$ literals. For example,

$$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4)$$

is a 3-CNF formula, since the size of each clause is no more than 3.

Show that every CNF formula can be converted to a 3-CNF formula. More formally, suppose that we are given a $k$-CNF formula $\phi$ with $k \geq 3$, over $n$ variables $X = \{x_1, \ldots, x_n\}$. You need to show that there exists some 3-CNF formula $\phi'$, whose domain is $X$ and additional variables $Y = \{y_1, \ldots, y_m\}$, such that for every truth assignment $\vec{a} \in \{0,1\}^n$, there exists a truth assignment to $y_1, \ldots, y_m$, say $\vec{b} \in \{0,1\}^m$ such that $\phi(\vec{a}) = 1$ if and only if $\phi'(\vec{a}; \vec{b}) = 1$.

**Hint:** We have seen that the resolution algorithm can iteratively reduce the size of clauses e.g.

$$\frac{(x_1 \vee a) \wedge (x_2 \vee x_3 \vee \cdots \vee x_m \vee \neg a)}{(x_1 \vee x_2 \vee x_3 \vee \ldots x_m)}.$$

The trick in this question is to run the algorithm 'in the other direction' by adding dummy variables (whose value will be determined later as in the resolution algorithm), in order to reduce the size of the clauses.

**Solution:** We are given a $k$-CNF formula $\phi$ whose clauses are $C_1, \ldots, C_\ell$; if $k \leq 3$ we are done. Suppose that $k > 3$. Let us define the following parameter

$$\alpha(\phi) = \sum_{C_t : |C_t| > 3} |C_t|.$$

$\alpha(\phi)$ simply counts the total size of all clauses in $\phi$ whose size is more than 3; in particular, if $\phi$ is a 3-CNF formula, $\alpha(\phi) = 0$. We denote $\phi_0 = \phi$, and propose a sequence of CNF formulas $\phi_0, \phi_1, \phi_2, \ldots$ such that

$$\alpha(\phi_0) > \alpha(\phi_1) > \alpha(\phi_2) > \ldots$$

Moreover, for every $j$ we let $Y_j$ be a set of additional variables used in $\phi_j$ (i.e. $\phi_j$'s domain is $X \cup Y_j$) with the following properties:

(a) We begin with no additional variables: $Y_0 = \emptyset$

(b) We only add one variable at every iteration: $|Y_j| = |Y_{j-1}| + 1$ for $j \geq 1$; we call the variable we add at time $j$ $y_j$.

Finally, we have the following property: for every truth assignment $\vec{a}$ to $X \cup Y_j$ there is some truth assignment $b_j$ to $y_j$ such that $\phi_{j-1}(\vec{a})$ is true iff $\phi_j(\vec{a}, b_j)$ is true. There are no additional truth assignments: so if $\vec{b}$ is a truth assignment to $\phi_j$ and $b_j$ is the value assigned to $y_j$, then $\vec{b}_{-j}$ (i.e. the truth assignment to all other variables except $b_j$) satisfies $\phi_{j-1}$.

Since $\alpha(\phi_0)$ is a positive integer and the sequence $(\alpha(\phi_j))_{j=0}^{\infty}$ is strictly decreasing, there is some point $j^*$ such that for all $j' \geq j^*$ $\alpha(\phi_{j'}) = 0$. In other words, $\phi_{j^*}$ is a 3-CNF formula, who, by construction, satisfies the property we require.

We proceed as follows: given $\phi_0$, we take some clause $C_t$ in $\phi_0$ whose size is $q > 3$. We write $C_t = \ell_1 \vee \cdots \vee \ell_q$ (here $\ell_r$ is some literal of the form $x$ or $\neg x$) and replace $C_t$ with two clauses:

$$(\ell_1 \vee \cdots \vee \ell_q) \Rightarrow (y_1 \vee \ell_1 \vee \ell_2) \wedge (\neg y_1 \vee \ell_3 \vee \cdots \vee \ell_q).$$

We call the resulting CNF formula $\phi_1$. First, we observe that since $y_1$ appears both as a positive and as a negative literal in the two clauses we create, we can resolve it to the original

clause $C_t$, much like what we did in class; in other words, $\phi_0$ is satisfiable if and only if $\phi_1$ is.

Moreover, the newly created clauses have a size of 3 and $q-1$, respectively, so $\alpha(\phi_1) = \alpha(\phi_0) - 1$. We repeatedly apply this procedure to $\phi_1$ to obtain $\phi_2$ and so on. Let us illustrate what the procedure does to a single clause with 6 literals

$$\begin{aligned}(x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4 \vee x_5 \vee \neg x_6) \Rightarrow \\ (y_1 \vee x_1 \vee \neg x_2) \wedge (\neg y_1 \vee \neg x_3 \vee x_4 \vee x_5 \vee \neg x_6) \Rightarrow \\ (y_1 \vee x_1 \vee \neg x_2) \wedge (y_2 \vee \neg y_1 \vee \neg x_3) \wedge (\neg y_2 \vee x_4 \vee x_5 \vee \neg x_6) \Rightarrow \\ (y_1 \vee x_1 \vee \neg x_2) \wedge (y_2 \vee \neg y_1 \vee \neg x_3) \wedge (y_3 \vee \neg y_2 \vee x_4) \wedge (\neg y_3 \vee x_5 \vee \neg x_6) \Rightarrow\end{aligned}$$

Note that running the resolution procedure on the clauses in the last line results in the original 6-literal clause, and that the original clause is satisfiable if and only if the 3-literal clauses that we obtain are. Another question that is worth asking is - what is the running time of our procedure? s it a poly-time algorithm? If so, think why (hint: think about the number of times $\alpha$ strictly decreases).

3. Show that the resolution procedure for CNF formulas described in class yields a polynomial time algorithm for deciding whether a 2-CNF formula is satisfiable.
**Solution:** Consider the following algorithm. We begin by noting that any clause in a 2-CNF formula can be written either as

$$\begin{aligned}x &\Rightarrow y \\ x &\Rightarrow \neg y \\ \neg x &\Rightarrow y \\ \neg x &\Rightarrow \neg y\end{aligned}$$

(verify to yourself why this is correct!). We create a graph whose nodes are variables and their negations (so each variable $x$ generates two nodes, $x$ and $\neg x$). Next, if the clause $\ell_1 \Rightarrow \ell_2$ appears in the formula, we add the edges $(\ell_1, \ell_2)$ and $(\neg\ell_2, \neg\ell_1)$. We call the resulting graph the *implication graph* of $\phi$.

**Lemma 1.** *If there exists a cycle that connects a node $x$ with a node $\neg x$ then the CNF formula is not satisfiable.*

*Proof.* Suppose that there is a cycle of the form

$$x \Rightarrow a_1 \Rightarrow a_2 \Rightarrow \cdots \Rightarrow a_q \Rightarrow \neg x \Rightarrow b_1 \cdots \Rightarrow \cdots \Rightarrow b_t \Rightarrow x$$

Where $a_1, \ldots, a_q$ and $b_1, \ldots, b_t$ are literals (either in the form $y$ or $\neg y$); suppose that we can satisfy $\phi$ by setting $x = 1$. Then This implies that $a_1$ must hold, which implies $a_2$,..., which implies that $\neg x$ must hold, a contradiction. Suppose that we set $x = 0$, then this means that $b_1$ must hold, which implies $b_2$, ..., which implies that $x$ must hold, again a contradiction. $\square$

Suppose that the implication graph has no cycles containing both $x$ and $\neg x$. We will show that it is possible to generate a truth assignment for $\phi$ in this case. We run the following algorithm First, find a literal $\ell$ that has not been assigned yet, such that there is no path from $\ell$ to its negation. Such a literal must exist: if there is a path from $\ell$ to $\neg\ell$ then there can't be a path from $\neg\ell$ to $\ell$ by the no-cycles assumption. For simplicity assume that the literal is positive (i.e. $\ell = x$ for some variable $x$). Set $x$ to be true (if the literal is of the form $\neg x$ set $x$ to be false). Next, set all literals of the form $x \Rightarrow \ell$ to true (for example, if $x \Rightarrow \neg y$ set $y = 0$) and continue assigning along the paths from $x$ to every literal reachable by $x$ via a path in the implication graph until all reachable vertices are assigned a truth value. This procedure is valid: we run the risk of having an issue like

$$x \Rightarrow a_1 \Rightarrow \cdots \Rightarrow a_q \Rightarrow y$$
$$x \Rightarrow b_1 \Rightarrow \cdots \Rightarrow b_t \Rightarrow \neg y$$

in which case we will be assigning $y$ both $1$ and $0$. But this cannot happen or there is a cycle containing both $x$ and $\neg x$ (why??).

Repeat this procedure until all variables have been assigned a value.

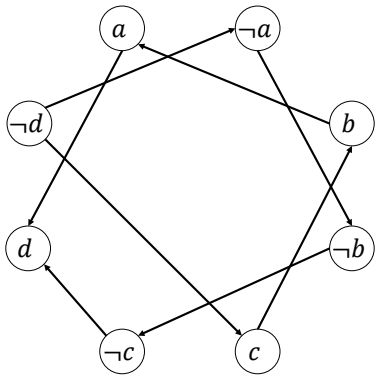


Figure 1: The implication graph for the 2-CNF formula

For example, consider the 2-CNF formula

$$(a \vee \neg b) \wedge (c \vee d) \wedge (b \vee \neg c) \wedge (\neg a \vee d)$$

This yields the implications

$$b \Rightarrow a; \neg c \Rightarrow d; c \Rightarrow b; a \Rightarrow d$$

and their negations (e.g. $\neg d \Rightarrow \neg a$ etc.). The implication graph is given in Figure 1.