National University of Singapore
School of Computing
CS3243 Introduction to AI

Tutorial 8: Logical Agents II (Solutions)

Please refer to **Appendix A** for notes on Knowledge Bases, and **Appendix B** for Propositional Logic Laws.

1. Consider an instance of the Vertex Cover problem given in Figure 1. In the Vertex Cover problem we are given a graph $G = \langle V, E \rangle$. We say that a vertex v *covers* an edge $e \in E$ if v is incident on the edge e. We are interested in finding a *vertex cover*; this is a set of vertices $V' \subseteq V$ such that every edge is covered by some vertex in V'. In what follows you may **only** use variables of the form x_v where $x_v = 1$ if v is part of the vertex cover, and is 0 otherwise.

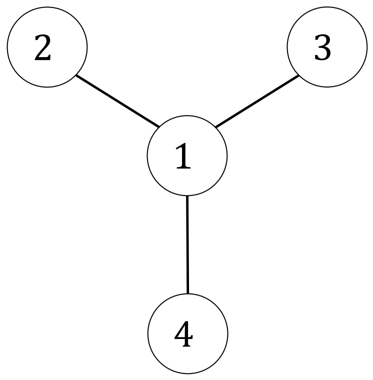

Figure 1: Graph for Vertex Cover CSP part (b)

When writing the constraints you may **only use**

- Standard logical and set operators ($\forall, \exists, \vee, \wedge$ and $x \in X, X \subseteq Y$)

(i) Write down the vertex cover constraints as logical statements, as well as the size constraints in the case that the vertex cover is of size $k = 1$. Express your answers in CNF.

(ii) Apply the resolution algorithm in order to prove that the vertex 1 must be part of the vertex cover; again, assume that the cover in Figure 1 must be of size $k = 1$.

Solution: We can write the edge cover constraints as

$$x_1 \lor x_2; x_1 \lor x_3$$
$$x_1 \lor x_4;$$

and the cardinality constraints as

$$x_1 \Rightarrow \neg x_2; x_1 \Rightarrow \neg x_3$$
$$x_1 \Rightarrow \neg x_4; x_2 \Rightarrow \neg x_3$$
$$x_2 \Rightarrow \neg x_4; x_3 \Rightarrow \neg x_4$$

Which translate to CNF as

$$\neg x_1 \lor \neg x_2; \neg x_1 \lor \neg x_3$$
$$\neg x_1 \lor \neg x_4; \neg x_2 \lor \neg x_3$$
$$\neg x_2 \lor \neg x_4; \neg x_3 \lor \neg x_4$$

For resolution we need to show that $KB \models \alpha = x_1$. Thus we take our $KB \land \neg\alpha \equiv KB \land \neg x_1$. Applying resolution we have

$$\neg x_1 \oplus x_1 \lor x_2 \Rightarrow x_2$$
$$x_2 \oplus \neg x_2 \lor \neg x_3 \Rightarrow \neg x_3$$
$$\neg x_1 \oplus x_1 \lor x_3 \Rightarrow x_3$$
$$x_3 \oplus \neg x_3 \Rightarrow \emptyset$$

Here, $\oplus$ is the resolution operation.

2. Suppose that we are maintaining a knowledge base with propositional logical statements involving Boolean variables $x_1, \ldots, x_n$. Given a logical formula q, let $M(q)$ be the set of all truth assignments to variables for which q is true. Recall that an inference algorithm $\mathcal{A}$ is sound if whenever a statement q is inferred by a knowledge base KB, it must be the case that $M(KB) \subseteq M(q)$. An inference algorithm $\mathcal{A}$ is *complete* if whenever $M(KB) \subseteq M(q)$, then eventually q will be inferred by $\mathcal{A}$. Formally prove that the resolution algorithm is sound (you've seen a sketch in class). You may also try to show completeness, but this is a longer proof (hint: you can use induction).

Solution: Suppose that we obtained α from KB by running a sequence of resolution operations. Our proof is by induction on the number of resolutions we executed before obtaining α. Let $\vec{x} = (x_1, \dots, x_n)$; suppose that KB is in CNF form. For the first resolution step we have that there exist two OR clauses in KB of the form $P(\vec{x}) \vee x$ and $Q(\vec{x}) \vee \neg x$. Applying resolution we get $P(\vec{x}) \vee Q(\vec{x})$. Note that if $\vec{t} \in \{\mathit{True}, \mathit{False}\}^n$ is a satisfying truth assignment for KB then it must be that both $P(\vec{x}) \vee x$ and $Q(\vec{x}) \vee \neg x$ are true. In particular, if $x = \mathit{True}$ under $\vec{t}$, then $Q(\vec{t}) = \mathit{True}$; if $x = \mathit{False}$ then $P(\vec{t}) = \mathit{True}$; in either case the expression $P(\vec{t}) \vee Q(\vec{t}) = \mathit{True}$. Since this is true for any resolvent reachable after 1 resolution step (there was nothing special about the one we picked), we have shown the case for resolvents that are reached after one step. For the inductive step, suppose that any resolvent q that is achievable after r resolution steps satisfies $M(KB) \subseteq M(q)$; we show the claim holds for $r+1$. However, this is simply a repetition of the proof for the one-step case, with the KB being KB^r: the set of all resolvents reachable from KB after r resolution steps.

3. Recall that a CNF formula ϕ is defined over a set of variables $X = \{x_1, \dots, x_n\}$; 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 ϕ as a mapping from $\{0,1\}^n$ to $\{0,1\}$. We say that an assignment $\vec{a} \in \{0,1\}^n$ satisfies ϕ 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 ϕ with $k \geq 3$, over n variables $X = \{x_1, \dots, x_n\}$. You need to show that there exists some 3-CNF formula ϕ', whose domain is X and additional variables $Y = \{y_1, \dots, y_m\}$, such that for every truth assignment $\vec{a} \in \{0,1\}^n$, there exists a truth assignment to $y_1, \dots, y_m$, 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.

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

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 ϕ whose clauses are $C_1, \dots, C_\ell$; 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 ϕ whose size is more than 3; in particular, if ϕ is a 3-CNF formula, $\alpha(\phi) = 0$. We denote $\phi_0 = \phi$, and propose a sequence of CNF formulas $\phi_0, \phi_1, \phi_2, \dots$ such that

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

Moreover, for every j we let Y_j be a set of additional variables used in ϕ_j (i.e. ϕ_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 ϕ_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 ϕ_{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, ϕ_{j^*} is a 3-CNF formula, who, by construction, satisfies the property we require.

We proceed as follows: given ϕ_0, we take some clause C_t in ϕ_0 whose size is $q > 3$. We write $C_t = \ell_1 \vee \dots \vee \ell_q$ (here ℓ_r is some literal of the form x or $\neg x$) and replace C_t with two clauses:

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

We call the resulting CNF formula ϕ_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, ϕ_0 is satisfiable if and only if ϕ_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 ϕ_1 to obtain ϕ_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 α strictly decreases).

4. 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.

Hint: Note that any clause containing exactly two variables can be written in terms of a conditional, i.e.:

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

Next, look at the directed graph whose nodes are variables (and their negations), and think what happens if there is some cycle containing a variable x and its negation $\neg x$.

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 (ℓ_1, ℓ_2) and $(\neg \ell_2, \neg \ell_1)$. We call the resulting graph the *implication graph* of ϕ.

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 $a_1, \dots, a_q$ and $b_1, \dots, b_t$ are literals (either in the form y or $\neg y$); suppose that we can satisfy ϕ 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. □

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 ϕ in this case. We run the following algorithm.

First, find a literal ℓ that has not been assigned yet, such that there is no path from ℓ to its negation. Such a literal must exist: if there is a path from ℓ to $\neg\ell$ then there can't be a path from $\neg\ell$ to ℓ 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

$$\begin{aligned} x &\Rightarrow a_1 \Rightarrow \cdots \Rightarrow a_q \Rightarrow y \\ x &\Rightarrow b_1 \Rightarrow \cdots \Rightarrow b_t \Rightarrow \neg y \end{aligned}$$

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.

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 2.

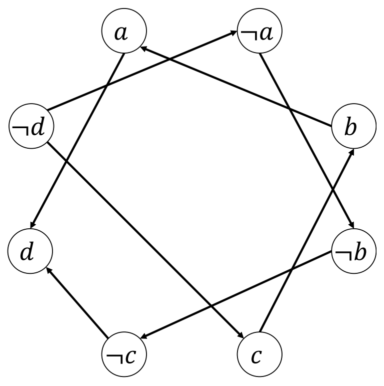

Figure 2: The implication graph for the 2-CNF formula

Appendix A: Notes on Knowledge Bases

A knowledge base KB is a set of logical rules that model what the agent knows. These rules are written using a certain language (or *syntax*) and use a certain truth model (or *semantics* which say when a certain statement is true or false). In propositional logic sentences are defined as follows

1. Atomic Boolean variables are sentences.
2. If S is a sentence, then so is $\neg S$.
3. If S_1 and S_2 are sentences, then so is:
   (a) $S_1 \wedge S_2$ "S_1 and S_2"
   (b) $S_1 \vee S_2$ "S_1 or S_2"
   (c) $S_1 \Rightarrow S_2$ "S_1 implies S_2"
   (d) $S_1 \Leftrightarrow S_2$ "S_1 holds if and only if S_2 holds"

We say that a logical statement a models b ($a \models b$) if b holds whenever a holds. In other words, if $M(q)$ is the set of all value assignments to variables in a for which a holds true, then $M(a) \subseteq M(b)$.

An inference algorithm $\mathcal{A}$ is one that takes as input a knowledge base KB and a query α and decides whether α is derived from KB, written as $KB \vdash_{\mathcal{A}} \alpha$. $\mathcal{A}$ is sound if $KB \vdash_{\mathcal{A}} \alpha$ implies that $KB \models \alpha$; $\mathcal{A}$ is complete if $KB \models \alpha$ implies that $KB \vdash_{\mathcal{A}} \alpha$.

Appendix B: Propositional Logic Laws

De Morgan's Laws	$\neg(p \vee q) \equiv \neg p \wedge \neg q$	$\neg(p \wedge q) \equiv \neg p \vee \neg q$
Idempotent laws	$p \vee p \equiv p$	$p \wedge p \equiv p$
Associative laws	$(p \vee q) \vee r \equiv p \vee (q \vee r)$	$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$
Commutative laws	$p \vee q \equiv q \vee p$	$p \wedge q \equiv q \wedge p$
Distributive laws	$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$	$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
Identity laws	$p \vee False \equiv p$	$p \wedge True \equiv p$
Domination laws	$p \wedge False \equiv False$	$p \vee True \equiv True$
Double negation law	$\neg\neg p \equiv p$	
Complement laws	$p \wedge \neg p \equiv False \wedge \neg True \equiv False$	$p \vee \neg p \equiv True \vee \neg False \equiv True$
Absorption laws	$p \vee (p \wedge q) \equiv p$	$p \wedge (p \vee q) \equiv p$
Conditional identities	$p \Rightarrow q \equiv \neg p \vee q$	$p \Leftrightarrow q \equiv (p \Rightarrow q) \wedge (q \Rightarrow p)$