Knowledge Representation and Reasoning 2022

Homework assignment #1

Piyush Bagad (13677640)

Exercise 1. We construct a solution to (k, n)-Queens problem as follows.

Let P denote the set of propositional variables, defined as, $P = \{p_{i,j} : i, j \in \{1, 2, .., n\}\}$ where $p_{i,j}$ denotes whether or not a queen is placed at position (i, j) on the chess board.

1.5/1.5

(a) Given n and k we can construct the desired CNF formula $\psi_{n,k}$ as follows. First, we provide an intuition, then provide our solution and then follow it up with an informal proof.

- *Intuition.* In order to be able to place (any) number of queens on a chess board without any two attacking each other, on placing a queen at a position (i, j) we need to ensure that all positions attacked by the queen at (i, j) cannot be further occupied by another queen. Extending this to every position shall provide us a formula that returns true or false depending on whether or not a given placement (truth assignment) satisfies our criteria.
- *Solution.* Suppose we place a queen at position (i, j) then we cannot place another queen at all positions attacked by the queen at (i, j). Formally, this translates to satisfying the following constraints:

$$\begin{aligned}
\text{Row constraint:} \quad & p_{i,j} \to \neg p_{i',j}, \forall i' \in \{1, .., n\}, i' \neq i \\
\text{Column constraint:} \quad & p_{i,j} \to \neg p_{i,j'}, \forall j' \in \{1, .., n\}, j' \neq j \\
\text{Positive diagonal constraint:} \quad & p_{i,j} \to \neg p_{i+s,j+s}, \max\{1-i, 1-j\} \leq s \leq \min\{n-i, n-j\}, s \neq 0 \\
\text{Negative diagonal constraint:} \quad & p_{i,j} \to \neg p_{i+s,j-s}, \max\{1-i, j-n\} \leq s \leq \min\{n-i, j-1\}, s \neq 0
\end{aligned}$$

For convenience, let the set of positions defining the diagonals for (i, j) be denoted by $D^+_{i,j}$ and $D^-_{i,j}$. It is trivial to prove that, for propositional variables α, β $\alpha \to \beta$ is logically equivalent to $\neg\alpha \vee \beta$. Using this, the above constraints can be combined into a single propositional formula in CNF form as follows:

$$\begin{aligned}
\gamma_{i,j} := (\wedge_{i' \neq i}(\neg p_{i,j} \vee \neg p_{i',j})) \wedge (\wedge_{j' \neq j}(\neg p_{i,j} \vee \neg p_{i,j'})) \wedge \\
\left(\wedge_{s \in D^+_{i,j}}(\neg p_{i,j} \vee \neg p_{i+s,j+s})\right) \wedge \left(\wedge_{s \in D^-_{i,j}}(\neg p_{i,j} \vee \neg p_{i+s,j-s})\right)
\end{aligned}$$

Now, we need this to be satisfied for each position (i, j) on the chess board where we place a queen. Thus, we obtain:

$$\psi_{n,k} := \wedge_{i,j} \gamma_{i,j} \tag{1}$$

Note that this would lead to redundant terms that can be eliminated to simplify the answer. But the above formula shall work nevertheless.

- *Informal proof.* Suppose that we are given a placement of k-queens, i.e, a truth assignment $\alpha : P \to \{0, 1\}$ with $\alpha_{i,j} := \alpha(p_{i,j}) \in \{0, 1\}, \forall i, j$ such that no two queens attack each other. We need to show that $\psi_{n,k}$ as defined in equation (1), is satisfied by α. Suppose that is not the case. Then, $\exists (i, j)$ such that $\gamma_{i,j}$ is false. By definition of $\gamma_{i,j}$ this would imply that atleast one of the following is true (i) there is atleast one more queen in the row i (ii) there is atleast one more queen in column j (iii) there is atleast one more queen on either of the two diagonals. This is in contradiction with our assumption that no two queens attack each other. Thus, our original supposition is wrong. Hence, proved. Conversely, if we assume $\psi_{n,k}$ to be true, we can find a truth assignment α since we can exactly find (i, j) which have a queen by checking if each of the sub-formulas are true and finding if $p_{i,j}$ is true or not.

1.05/1.5

(b) Next, we want to express $\chi_{n,k}$ to represent if there are atleast k queens on the board. Let us introduce intermediate propositional variables $\Sigma_{(i,j),t}, 1 \leq i, j \leq n, t \in \{1, 2, .., k\}$ which is true if and only if there are exactly t queens in the subset of the chessboard upto the cell (i, j). Formally,

$$\Sigma_{(i,j),t} = \begin{cases} \text{True}, & \text{if } \sum_{i' \leq i, j' \leq j} \mathbb{I}[p_{i',j'}] = t \\ \text{False}, & \text{otherwise} \end{cases} \tag{2}$$

First, we recursively express this cardinality condition using the following propositional clauses.

$$\begin{aligned}
\text{Base conditions:} \quad & \Sigma_{(1,1),0} \leftrightarrow \neg p_{1,1}, \Sigma_{(1,1),1} \leftrightarrow p_{1,1}, \neg\Sigma_{(1,1),t}, \forall t >= 2 \\
\text{First row conditions:} \quad & \Sigma_{(i,1),0} \leftrightarrow \left(\Sigma_{(i-1,1),0} \wedge \neg p_{i-1,1}\right), \forall i \geq 2 \\
\text{First column conditions:} \quad & \Sigma_{(1,j),0} \leftrightarrow \left(\Sigma_{(1,j-1),0} \wedge \neg p_{1,j-1}\right), \forall j \geq 2 \\
\text{Recursive condition:} \quad & \Sigma_{(i,j),t} \leftrightarrow \left(\Sigma_{(i,j-1),t} \wedge \neg p_{i,j}\right) \vee \left(\Sigma_{(i-1,j),t} \wedge \neg p_{i,j}\right) \vee \\
& \left(\Sigma_{(i,j-1),t-1} \wedge p_{i,j}\right) \vee \left(\Sigma_{(i-1,j),t-1} \wedge p_{i,j}\right), \forall i,j,t
\end{aligned}$$

The idea is that, to ensure k queens in a sub-square (i,j), you can have $k-1$ queens in either of the subsets $(i-1,j),(i,j-1)$ and have $p_{i,j}$ true, or, have k queens till the smaller subsets and then have $p_{i,j}$ as false.

Now, given $\alpha \leftrightarrow \beta_1 \wedge \beta_2$ is logically equivalent to $[(\neg\alpha \vee \neg\beta_1) \wedge (\neg\alpha \vee \neg\beta_2)] \wedge [\neg\beta_1 \vee \neg\beta_2 \vee \neg\alpha]$. Further, $\neg(\delta_1 \wedge \delta_2)$ is logically equivalent to $(\neg\delta_1 \vee \neg\delta_2)$. Using the above constrains and these deductions, we can construct a propositional formular in CNF form to be true if and only if there are at least k queens on an $n \times n$ chessboard. This is equivalent to not letting number of queens be $\leq (k-1)$.

$$\chi_{n,k} = \bigwedge_{0 \leq t \leq k-1} \neg\Sigma_{(n,n),t} \tag{3}$$

with $\Sigma_{(n,n),t}$ derived from the previously defined recursive clauses.

Complexity analysis. Note that for each new variable $\Sigma_{(i,j),t}$ that we introduce, we add 4 new clauses (from the "Recursive condition" as defined previously), and $1 \leq i,j \leq n$, $1 \leq t \leq k$, thus, we would have approximately $4 \times n^2 \times k$ clauses for $\Sigma_{(n,n),k}$. Thus, in equation (3), we would have $\leq 4 \times n^2 \times k \times k$ clauses and $k \leq n$, so $\leq 4 \times n^4$ clauses.

1.75/3 **Exercise 2.** Let A be the black-box algorithm that, given a propositional CNF formula ϕ outputs 1 if ϕ is satisfiable and outputs 0 otherwise. Suppose $P = \{p_1, p_2, .., p_n\}$ be the set of propositional variables contained in ϕ. As defined in the lectures, let us denote ϕ_p to be such that we set p to be true in ϕ for $p \in P$. This means that we remove all clauses where p appears, and for all other clauses, if $\neg p$ appears, we remove it.

(a) We provide the required algorithm B as follows:

- *Solution.* We provide the algorithm B in Algorithm 1.

Algorithm 1: Algorithm that returns a single truth assignment for a CNF propositional formula.

```
function B (φ);
Input  : Propositional formula in CNF φ
Output: α : p → {0,1} such that α ⊢ φ
i ← 1
α ← {}
if A(φ) = 0 then
    return "unsat"
end
while i ≤ n do
    if A(φ_{p_i}) = 1 then
        α(p_i) = 1        /* Assign truth value to p_i as true        */
        φ ← φ_{p_i}
    else
        α(p_i) = 0        /* Assign truth value to p_i as false       */
        φ ← φ_{¬p_i}
    end
    i ← i + 1
end
return α
```

- *Informal proof.* The proof of correctness directly follows from the fact that we set $\alpha(p_i) = 1$ only when $A(\varphi_{p_i})$ returns True which implies that only assignment where p_i is set to be true shall satisfy φ. The same argument holds for the case when $A_{\varphi_{p_i}} = 0$ implying we need to set p_i to be False.

- *Complexity analysis.* Assuming each call to A as a single step in the running time, we loop over all the propositional variables amounting to a complexity of $\mathcal{O}(n)$. Note that I am ignoring the length of φ which may affect how long A takes to return an answer.

(b) We provide the required algorithm C as follows:

- *Solution.*

Algorithm 2: Algorithm takes in a CNF formula φ and n and returns n truth assignments for φ.

```
function C (φ, n);
Input  : Propositional formula in CNF φ positive integer n
Output: S = [α_1, α_2, ..α_n] such that α_t ⊢ φ, ∀t
i ← 1
S ←
if A(φ) = 0 then
 | return "unsat"
end
while i ≤ n do
 | if |S| = n then
 |  | return S
 | end
 | if A(φ_{p_i}) = 1 then
 |  | β ← B(φ_{p_i}) ∪ {p_i}        /* Assign value to p_i as true       */
 |  | if β not in S then
 |  |  | S ← S ∪ {β}
 |  | end
 |  | φ ← φ_{p_i}
 | else
 |  | β ← B(φ_{¬p_i}) ∪ {¬p_i}      /* Assign value to p_i as false      */
 |  | if β not in S then
 |  |  | S ← S ∪ {β}
 |  | end
 |  | φ ← φ_{p_i} φ ← φ_{¬p_i}
 | end
 | i ← i + 1
end
return S
```

- *Informal proof.* In essense, we loop over all the propositional variables and keep eliminating literals depending on which truth assignment satisfies them and keep adding to our solution set S using the previously defined algorithm B.
- *Complexity analysis.* Like in the previous part, we loop over all propositional variables and for each iteration, we call B which takes $\mathcal{O}(n)$ time. So overall, our complexity shall be $\mathcal{O}(n^2)$.

3/4

Exercise 3. In this problem, we provide solution for inter-conversion between a propositional formula and an answer set of an ASP.

- Translation of φ into an answer set program P. Suppose we are given a propositional formula φ (not necessarily in CNF), we define subformulas of φ like in the lectures. We will define a recursive algorithm using the subformulas. We assume the propositional variables in φ are $\{p_1, p_2, .., p_n\}$. Let $D(\varphi)$ denote the desired algorithm that returns an ASP. We also define two helper functions that operate on pairs of ASPs. For example, let `simple-combine`(P, Q) be a function that takes in two ASPs with rules without any heads and combines them by simply appending rules together. Likewise, let `cross-combine`(P, Q) be another function which takes in ASPs containing rules with no heads, combines them in a cross-product fasion, i.e., for every rule in P is combined with every rule in Q.

  For example, let P and Q be defined as follows

```
1 :- p11 p12 .., p1k.
2 :- p21 p22 .., p2k.
```

```
1 :- q11 q12 .., q1k.
2 :- q21 q22 .., q2k.
```

Then, `simple-combine`(P, Q) will be

```
:- p11 p12 .., p1k.
:- p21 p22 .., p2k.
:- q11 q12 .., q1k.
:- q21 q22 .., q2k.
```

And `cross-combine`(P, Q) will be

```
:- p11 p12 .., p1k, q11 q12 .., q1k.
:- p21 p22 .., p2k, q11 q12 .., q1k.
:- p11 p12 .., p1k, q21 q22 .., q2k.
:- p21 p22 .., p2k, q21 q22 .., q2k.
```

Now, we shall describe the steps in our recursive algorithm.

(i) Setup the basic rules to define propositional variables and ensure they are either true or false.

```
var(1..n).
true(X):- not false(X), var(X)
false(X):- not true(X), var(X)
```

(ii) Base case 1: If $\varphi = p$, where p is a propositional variable, return

```
:- false(p)
```

(iii) Base base 2: If $\varphi = \neg p$, return

```
:- true(p)
```

(iv) Recursion calls

- If $\varphi = \varphi_1 \wedge \varphi_2$, for some subformulas φ_1, φ_2, compute $D(\varphi_1)$ and $D(\varphi_2)$, return

$$\texttt{simple-combine}(D(\varphi_1), D(\varphi_2))$$

- If $\varphi = \varphi_1 \vee \varphi_2$, for some subformulas φ_1, φ_2, compute $D(\varphi_1)$ and $D(\varphi_2)$, return

$$\texttt{cross-combine}(D(\varphi_1), D(\varphi_2))$$

- If $\varphi = \neg\varphi'$, then it can be either $\varphi = \neg(\varphi_1 \wedge \varphi_2)$ or $\varphi = \neg(\varphi_1 \vee \varphi_2)$. We can propagate the negation inside and call $D(\neg\varphi_1 \vee \neg\varphi_2)$ or $D(\neg\varphi_1 \wedge \neg\varphi_2)$ respectively.
- If $\varphi = \varphi_1 \to \varphi_2$, we can reduce it as $\varphi = \neg\varphi_1 \vee \varphi_2$ and call D on it
- If $\varphi = \varphi_1 \leftrightarrow \varphi_2$, we can reduce it as $\varphi = (\neg\varphi_1 \vee \varphi_2) \wedge (\neg\varphi_2 \vee \varphi_1)$ and call D on it

The routines for combining ASPs are cheap operations. The main compute cost shall come from the recursive calls which means traversing a binary tree of height $\mathcal{O}(\log n)$. However, since we convert every step into a logically equivalent form, in the worst-case, the overall computation may blow-up if we have too many operations like $\leftarrow$ or $\leftrightarrow$. As alternate solution using Tseytin's transformation is provided below.

- Translation of obtained answer set A into α a satisfying assignment for φ. By construction of our ASPs, we will only end up with rules that do not have a head. Once we obtain an answer set, it is trivial to derive a truth assignment for φ for that. Thus, algorithm E is as follows:
  - Each element in the answer set is of the form `true(i)` or `false`(i), for $i \in \{1, 2, .., n\}$. If `true(i)`, then assign $\alpha(p_i) = 1$, else $\alpha(p_i) = 0$.
  - For i not occuring in the answer set, once can choose any truth assignments for such p_i.

Aliter. The alternative is to convert φ into a CNF using **Tseytin's transformation** which provides an efficient way of obtaining an *equisatisfiable* formula. Converting a CNF forumla into ASP can be easily done.

(i) Assuming $\varphi = \wedge_i c_i$, where c_i is a clause $c_i = \vee_{j=1}^{k_i} l_j$ where $l_j = p$ or $\neg p$, for some $p \in P, \forall j$.

(ii) For each c_i, let $L_i^+ := \{j \in \{1, .., k_i\} \mid \exists p \in Pl_j = p\}$ be the indices with literals that are directly propositional variables likewise define L_i^-. Then, add the following rule into the ASP

```
:- false(j)_{j∈L_i^+}, true(j)_{j∈L_i^-},
```

(iii) Do this for each clause and we have number of rules equal to the number of clauses.

The algorithm to convert from an answer set to a truth assignment remains similar as in our original solution. Note that, with Tseytin's methods, we shall have additional variables that we introduced to create CNF formula. However, the CNF forumla still consists of all original propositional variables. Thus, we can obtain the truth values using only the corresponding elements in the answer set.