CS 208 : Automata Theory and Logic Spring 2024

Homework 1

Due: 19th Jan, 2024 *Max Marks:* 40 (+bonus 5)

Instructions:

- *Please start writing your solution to each homework problem on a fresh page.*
- *For each homework problem, you must scan your solution and upload a separate PDF file on Moodle. Please check Moodle for detailed instructions on file naming and uploading instructions.*
- *Be brief, complete, and stick to what has been asked.*
- *Untidy presentation of answers, and random ramblings will be penalized by negative marks.*
- *Unless asked for explicitly, you may cite results/proofs covered in class without reproducing them.*
- ***If you need to make any assumptions, state them clearly.***
- ***Do not copy solutions from others. All detected cases of copying will be reported to DADAC with names and roll nos. of all involved. The stakes are high if you get reported to DADAC, so you are strongly advised not to risk this.***

Question 1 : Implication ”Equations” 15 points

In this question, we will consider "implications" in the same spirit as "equations" between unspecified propositional formulas. You can think of these as "equations" where the unknowns are propositional formulas themselves, and the = symbol has been replaced by $\rightarrow$.

Let φ_1 and φ_2 be unknown propositional formulas over $x_1, x_2, \ldots x_n$. Consider the following implications labelled (1a), (1b) through (na), (nb). A solution to this set of simultaneous implications is a pair of specific propositional formulas (φ_1, φ_2) such that all implications are **valid**, i.e. evaluate to true for all assignments of variables.

$$\begin{array}{ll|ll}
\text{(1a)} & (x_1 \wedge \varphi_1) \rightarrow \varphi_2 & \text{(1b)} & (x_1 \wedge \varphi_2) \rightarrow \varphi_1 \\
\text{(2a)} & (x_2 \wedge \varphi_1) \rightarrow \varphi_2 & \text{(2b)} & (x_2 \wedge \varphi_2) \rightarrow \varphi_1 \\
 & \vdots & & \vdots \\
\text{(na)} & (x_n \wedge \varphi_1) \rightarrow \varphi_2 & \text{(nb)} & (x_n \wedge \varphi_2) \rightarrow \varphi_1
\end{array}$$

In each of the following questions, you must provide complete reasoning behind your answer. Answers without reasoning will fetch 0 marks.

1. *[5 marks]* Suppose we are told that φ_1 is $\bot$. How many semantically distinct formulas φ_2 exist such that implications (1a) through (nb) are valid?
2. *[2 marks]* Answer the above question, assuming now that φ_1 is $\top$.
3. *[5 marks]* If we do not assume anything about φ_1, how many semantically distinct pairs of formulas (φ_1, φ_2) exist such that all the implications are valid?
4. *[3 marks]* Does there exist a formula φ_1 such that there is exactly one formula φ_2 (modulo semantic equivalence) that can be paired with it to make (φ_1, φ_2) a solution of the above implications?

Question 2 : Balanced Parentheses 25 points

A *decision problem* is a computational problem that has a ”yes” or ”no” answer for every given input. An input to a decision problem is often encoded simply as a string of 0’s and 1’s. Not surprisingly, we can encode *some* decision problems P in propositional logic. Specifically, we construct a propositional logic formula φ_P over as many propositional variables as the count of letters (0s and 1s) in the binary string encoding the input, such that if the binary string is interpreted as an assignment to the propositional variables, then the ”yes”/”no” answer to P is obtained from the value given by the semantics of φ_P. If the semantics of φ_P evaluates to 0 (or ”false”) for the given input, then the output of P for that input is “no”; else, the output is “yes”.

Consider the following decision problem of checking if a given string of parentheses is balanced. Given a binary string of length $n, n \geq 1$, where 0 encodes ‘(’ and 1 encodes ‘)’, we say that the string has balanced parentheses if and only if:

- The number of open parentheses, represented by 0s, in the entire string equals the number of closing parentheses, represented by 1s.
- In every proper prefix of the string, the number of open parentheses is at least as much as the number of closing parentheses.

We wish to encode the above problem in propositional logic. Recall from your data structures and algorithms course that checking balanced parentheses can be solved in polynomial time (in fact, with linear time complexity). We want this to be reflected in some aspects of your solution to this problem.

In class, we saw that every propositional formula φ can be represented by a parse tree whose internal nodes are labelled by connectives and whose leaves are labelled by propositional variables. Sometimes, two different sub-trees in a parse tree may be identical. In such cases, it makes sense to represent the formula as a directed acyclic graph (DAG), where syntactically common sub-formulas are represented exactly once. As an example, consider the parse tree and corresponding DAG in Fig. 1, both representing the formula $(r \vee (p \vee q)) \wedge (p \vee q)$. Note

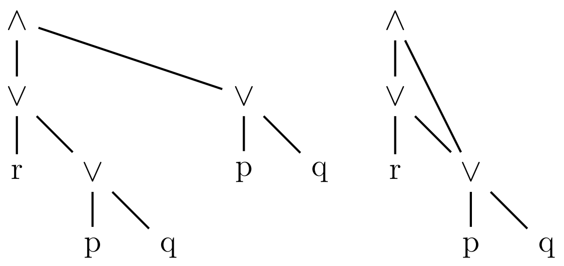

Figure 1: A parse tree and corresponding DAG

that a DAG representation of a formula can be exponentially smaller than a parse tree representation. Furthermore, a DAG representation suffices to evaluate the semantics of a formula, since the semantics of a shared sub-formula needs to be evaluated only once. Thus, if the DAG representation of a formula is small, its semantics can be evaluated efficiently. We define the *DAG size* of a formula to be the number of nodes in its DAG representation. This is also the number of syntactically distinct sub-formulas in the formula.

(a) *[10 marks]* Encode the problem of checking balanced parentheses in a binary string of length n as a propositional logic formula whose DAG size is at most $\mathcal{O}(n^3)$. You must use only n propositional variables corresponding to the n bits in the input string, and the only connectives allowed in your formula are two-input $\vee, \wedge$ and $\neg$. Express the DAG size of your formula as a function of n using big-$\mathcal{O}$ notation, with a clear justification of how you obtained the size expression.

(b) *[5 marks]* Prove that your formula is unsatisfiable for all odd values of n.

(c) *[10 (+ bonus 5) marks]* Suppose your formula is represented as a DAG, as discussed above. Given an input string of 0s and 1s, there is a simple way to evaluate the semantics of the formula. Specifically, we evaluate DAG nodes bottom-up, starting from the leaves (these represent propositional variables whose values are given by the input string) and moving upwards until we reach the root. The value of the root gives the semantics of the formula. Normally, a DAG node (labelled $\wedge$, $\vee$ or $\neg$) can be evaluated only after all its children have been evaluated. However, if a $\vee$ (resp. $\wedge$) labelled node has a child that has evaluated to 1 (resp. 0), then the node itself can be evaluated to 1 (resp. 0) without evaluating its other child. This can allow us to find the value of the root without evaluating all nodes in the DAG.

What is the worst-case number of DAG nodes (as a function of n in big-$\mathcal{O}$ notation) that need to be evaluated for your formula in part (a), in order to find the value at the root node for any input string of length n? You must give justification for why you really need these many nodes to be evaluated in the worst-case. Answers without justification will fetch 0 marks.

You will be awarded bonus 5 marks if you can show that your formula requires evaluating only $\mathcal{O}(n)$) DAG nodes.