CS 208 : Automata Theory and Logic Spring 2024

Homework 1 Solutions

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 You can think of these as "equations" where the unknowns are propositional formulas themselves, and the = symbol has been replaced by $\rightarrow$.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 $\perp$. 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?

Solution: First note that the entire space of assignments of $x_1, \ldots x_n$ can be partitioned into sub-spaces where x_1 is true, $\neg x_1 \wedge x_2$ is true, $\neg x_1 \wedge \neg x_2 \wedge x_3$ is true, and so on until $\neg x_1 \wedge \cdots \neg x_n$ is true.

1. All implications in the left column reduce to $\perp \rightarrow \varphi_2$. Every such implication is trivially valid (since $\neg\perp \vee \varphi_2$ is true for all φ_2 for all assignments of variables). All implications in the right column reduce to $(x_i \wedge \varphi_2) \rightarrow \perp$. This implication is valid iff $(x_i \wedge \varphi_2)$ evaluates to false for all assignments of the variables. Using the partitioning of variable assignments mentioned above, φ_2 must evaluate to false whenever any of the variables $x_1, \ldots x_n$ evaluates to true. Hence, the semantics of φ_2 is determined for all assignments other than $x_1 = \cdots x_n = 0$. Since φ_2 can have two different truth values (0 or 1) for this assignment, only 2 distinct formulas φ_2 are possible.

2. The right column of implications are trivially valid if $\phi_1 \leftrightarrow \top$ is valid. Using reasoning similar to that above for the left column, once again only 2 distinct formulas φ_2 are possible.

3. Note that $(x_i \wedge \phi_j) \rightarrow \phi_k$ is semantically equivalent to $x_i \rightarrow (\phi_j \rightarrow \phi_k)$. This can be easily checked through truth tables. Therefore, if implications (1a) and (1b) are both valid, then $x_1 \rightarrow (\varphi_1 \leftrightarrow \varphi_2)$ must also be valid. Similarly, for implications (2a) and

(2b), and so on until (na) and (nb). Thus, if all the implications are to be valid, then $\varphi_1 \leftrightarrow \varphi_2$ must evaluate to true for all assignments of variables where at least one of $x_1, \ldots x_n$ is true. In other words, for every φ_1, the semantics of φ_2 can potentially differ from that of φ_1 only for the assignment $x_1 = \cdots x_n = 0$. Therefore, there are only two possible φ_2 for every φ_1. Since the total number of semantically distinct formulas φ_1 on n variables is 2^{2^n} (why?), the total count of semantically distinct pairs (φ_1, φ_2) is $2 \times 2^{2^n} = 2^{2^n+1}$.

4. The above implications allow φ_2 to evaluate to any value in $\{0, 1\}$ when $x_1 = x_2 = \cdots x_n = 0$, regardless of what φ_1 is. Therefore, there are always at least two solutions to the given 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

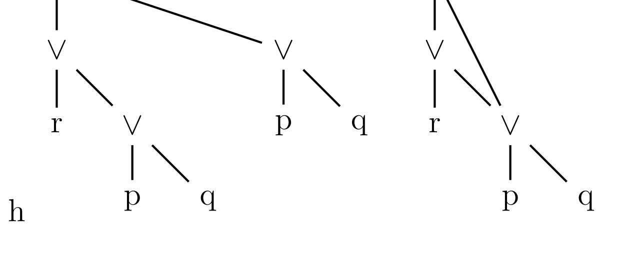

Figure 1: A parse tree and corresponding DAG

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.

Solution: Let $T_{i,j}$ represent the subformula, which evaluates to $\top$ if the number of open brackets exceeds the number of closed brackets by j in the first i characters of the string. Our formula for balanced parenthesis becomes $T_{n,0}$.

To make this formula well-defined, we must define the complete set of $T_{i,j}$ for all values of i, j. For all values of $i > 0, j < 0$, we have $T_{i,j} = \perp$ and for all values of $j \neq 0$, we have $T_{0,j}$ as $\perp$ and $T_{0,0} = \top$ For all other values of i, j, we have $T_{i,j} = (x_i \wedge T_{i-1,j+1}) \vee (\neg x_i \wedge T_{i-1,j-1})$. This makes the formula well-defined.

Lemma (well-defined): The complete set of propositional variables in the above formula is contained in $\{x_i\}$. We prove this by induction. (Base Cases are $T_{0,j}$ and induction over i)

Lemma (Size): The number of distinct subformulae is in $\mathcal{O}(n^3)$. We prove this by explicitly upper bounding the number of distinct subformulae.

Lemma (Correctness): If $T_{n,0}$ evaluates to $\top$ if and only if the string is balanced. We prove this using induction over even values of n.

(b) In the same way, for part b, we do induction over odd values of n and show that the formula is unsat.

(c) Claim: This formula can be evaluated in $O(n)$ time.

Evaluation Algorithm: We evaluate x_n and, based on whether it is $\top$ or $\perp$, evaluate the appropriate branch in the formula.

We prove the above evaluation is correct using induction and has $O(n)$ time complexity.