

EE382m — Formal Verification
Midterm

Adnan Aziz
The University of Texas at Austin
`adnan@ece.utexas.edu`

9 October 2000

Name:

1. Introduction

(a) Let f be a Boolean function. Prove that $\hat{f} = \hat{x} \cdot (f_x) + x \cdot (f_{\hat{x}})$.

5 marks

- (b) Describe the three components of the verification triangle that we discussed in class. Use the CTL model checking paradigm to illustrate of the triangle.

8 marks

2. BDDs:

- (a) Build the ROBDD for the Boolean function $abd' + ab'd + a'c + a'c'd$, both without and with use of complement pointers. Use the variable ordering $a \prec b \prec c \prec d$.

7 marks

- (b) Describe how you would implement a procedure for efficiently converting a BDD from one variable ordering to another. You may use the operations we described in class as subroutines, i.e., feel free to use the ITE, compose, and cofactor operators. Hint: this can be solved very easily using one of these operators.

6 marks

(c) A *factored form* is an expression built out of Boolean variables using AND, OR, and NOT, e.g., $(a' + (b \cdot (c + (d \cdot e'))))$. Technically, a factored form is given by the following rules:

- i. a_i is a factored form, where a_i is a Boolean variable
- ii. if f and g are factored forms, then so are f' , $(f + g)$, and $(f \cdot \dots \cdot g)$.

Clearly a factored form represents a Boolean function over the Boolean variables appearing in the factored form.

Through class and the homework, you should have gotten a clear idea of how to implement the ITE operator. Suppose you wanted to write an operator that took a factored form and built a BDD for it (assume the variable ordering is given). One way to implement the new operator would be to recursively perform a series of ITE operations, corresponding to the AND, OR, and NOT operations.

Describe how you would perform the BDD building **directly** from the Shannon expansion of the factored form. It should be analogous to the implementation of ITE, i.e., build BDD's for the left and right cofactors of the factored form.

Discuss how implementing the equivalent of the ITE cache makes your code slow. In what situations is it better to perform the BDD build directly, rather than by calling ITE?

Warning: do not use ITE code, i.e., don't just build your routine by inlining ITE.

11 marks

3. Equivalence Checking:

- (a) Let D_0 and D_1 be two designs. Suppose that **every** pair of states from D_0 and D_1 are inequivalent. Argue that there exists a **single** input sequence that will differentiate every pair of states from D_0 and D_1 .

Hint: if s_i and t_j are inequivalent, there is an input sequence π_{ij} which differentiates them. Consider the set of all state pairs, $S_{D_0} \times S_{D_1}$; think about building an input sequence that will differentiate any pair in this set by concatenating input sequences of the form π_{ij} .

14 marks

4. CTL Model Checking

- (a) Experience shows that many designs behaviors are repetitive, where a basic transaction occurs again and again, and the properties are interesting only within the boundaries of a single transaction. The *within* operator has been proposed to address this issue. For example $\mathbf{AGW}(start, end)(AG(request \rightarrow AF(ack)))$ means: “in all time intervals delimited by *start* and *end*, a request must be followed by an acknowledge.” Write a regular CTL formula corresponding to this property; assume *start* and *end* are regular CTL formula.

6 marks

- (b) Give a convincing argument that the property “there exists a path on which x is finally 1” cannot be expressed by a CTL formula which uses as its only temporal operators AX and EX .

8 marks

- (c) Let D be a finite state machine. One way to compute the set of all equivalent state pairs in D is to start from the identity equivalence relation, and then “grow it up” as shown below:

$$\begin{aligned} E_0 &= \{(s, s) \mid s \in S_D\} \\ E_1 &= E_0 \cup \{(s, t) \mid \text{for every input } i, (\delta(s, i), \delta(t, i)) \in E_0\} \\ &\vdots \\ E_{k+1} &= E_k \cup \{(s, t) \mid \text{for every input } i, (\delta(s, i), \delta(t, i)) \in E_k\} \\ &\vdots \end{aligned}$$

(Here $\delta(s, i)$ is the next state function for D .)

Clearly, since the set of states is finite, the iteration will hit a fixed point, say at n .

Argue that pairs of states that are in E_n are equivalent. Give an example that shows E_n is not necessarily the set of all equivalent state pairs.

10 marks