

This handout presents an overview of the tautology checking problem. Tautology checking is used in several steps in *espresso*. In addition, some of the techniques will carry over into designing fast algorithms for recursive complementation, prime generation, and identifying all essential prime implicants.

Note: Some of the material below appears in H/S ch. 5.2, or refers to pages of this book, i.e. see Handout #10. (“H/S” refers to “Hachtel/Somenzi book”.)

The Tautology Problem:

Given a cover F of a Boolean function, determine if F is a tautology.

Note: We will focus mainly on *single-output functions*, but also give some indications of how to handle multi-output functions. The H/S readings gives some of the extensions to the multi-output case – see pp. 199-200 as well as solved problems in pp. 206-218.

Definitions.

First, some basic definitions. The **universal cube** or **1 cube** is a cube whose input fields are all ‘-’ (*i.e.*, don’t-cares). That is, the universal cube spans the entire input domain.

A cover is **positive unate in a variable** x if \bar{x} does not appear in any cube in the cover. That is, in each cube, x either is uncomplemented or x does not appear. A cover is **negative unate in a variable** x if x does not appear (uncomplemented) in any cube in the cover. That is, in each cube, x is either complemented or x does not appear. If a cover is positive or negative unate in variable x , the cover is said to be **unate in variable** x . If a cover is unate in every variable, it is called a **unate cover**.

Rules for Detection a Tautology.

A number of rules can be applied, to determine *immediately* if F (i) is, or (ii) is not, a tautology.

A. Basic Rules for Detecting a Tautology.

- B1. **Universal Cube.** For a single-output function, if the cover includes the universal cube, then the function *is a tautology*. (For a multi-output function, if the cover contains the universal cube, then each output function to which it contributes, *i.e.* where the output field is 1, *is a tautology*.)
- B2. **Input Column of All 1’s/All 0’s.** For a single-output function, if an input column contains all 0’s (complemented literal) or all 1’s (uncomplemented literal), then the function *is not a tautology*. (For a multi-output function, if an input column contains all 0’s, or all 1’s, then the multi-output function *is not a tautology*.)
- B3. **Single-Input Dependence.** For a single-output function, if the function *depends* on only one input x (*i.e.*, all other input columns contain only ‘-’), *and* Rules B1 and B2 do not apply (*i.e.*, the x column contains both 1’s and 0’s), then the function *is a tautology*. (For a multi-output function, a similar result holds for each output function, if the above rule holds for the rows which contribute to that output.)

B. Advanced Rules for Detecting a Tautology: Unateness Conditions

Advanced rules can be applied, to determine if F (i) is, or (ii) is not, a tautology, using properties of unate covers.

- U1. **No Universal Cube.** (*H/S Theorem 5.2.3*) For a single-output function, if the cover is unate, and the cover does not include the universal cube, then the function *is not a tautology*. (For a multi-output function, a similar result holds for each output function, if the above rule holds for the rows which contribute to that output.)

C. Miscellaneous Rules for Detecting a Tautology

Miscellaneous rules can be applied, which take advantage of special case conditions. Some of these rules are easier to implement in a CAD program, than to do manually. (We will not focus much on these rules.)

M1. **Small Functions.** (H/S bottom p. 197) If the number of inputs is less than 8, then a truth table can be generated, and tautology checking can be done by inspection. The rationale is that if the number of inputs is small enough, the problem can be answered quickly without recursion (see later in handout).

M2. **Insufficient Vertex Count.** (H/S bottom p. 197) This rule is a fast heuristic to check if the current cover is too small to be a tautology. The *vertex count* of a cube is the number of vertices (i.e., minterms) which it contains. If a cube contains d don't-care inputs ('-'), then that cube covers 2^d minterms. It is very difficult to compute exactly how many minterms are covered by a cover F , so an *approximation* is computed quickly: the *upper bound*. In particular, for each cube c_i in the cover, if it has d_i dashes (don't cares) in the cubical representation, it covers 2^{d_i} minterms. Therefore, if cover F has m cubes, c_1 to c_m , then an *upper bound on the vertex count* (i.e. minterms covered) by F is: $\sum_{i=1 \text{ to } m} (2^{d_i})$.

Next, compute how many minterms are in the entire input space (domain). If the function has n dimensions (inputs), then there are 2^n minterms in its input space.

Finally, check if the *upper bound* on the number of vertices covered by F is insufficient: is $\sum_{i=1 \text{ to } m} (2^{d_i})$ less than 2^n ? If so, cover F cannot be a tautology.

The Tautology Checking Algorithm.

Tautology checking of a cover F is performed by a recursive algorithm. The basic idea is as follows.

Step #1. Apply the above rules, to determine if F (i) is a tautology, or (ii) is not a tautology. (We will usually ignore rules M1 and M2.) If the result is (i) or (ii), the algorithm is done.

Step #2. If the algorithm is not done, the cover is recursively split, in two halves, and the same algorithm is now repeated on each half.

More details on the recursion: Recursion is based on a fundamental theorem, called *Shannon Decomposition* or *Boole's Expansion Theorem*. Given a splitting variable x_1 , the theorem states (see H/S p. 192):

$$f = x_1 \cdot f_{x_1} + x'_1 \cdot f_{x'_1}$$

In words, the theorem indicates how a function can be examined, by cofactoring with respect to both x_1 and x'_1 , and combining the results.

We will explore Shannon decomposition in more detail shortly. For tautology, Shannon decomposition reduces to a very simple form: A function f is a tautology *if and only if* f_{x_1} and $f_{x'_1}$ are both tautologies (see H/S p. 194). More formally:

$$(f = 1) \equiv ((f_{x_1} = 1) \text{ and } (f_{x'_1} = 1)).$$

Summary of Step #2: If Step #1 cannot determine if the cover is, or is not a tautology, then recursion is performed. A splitting (or branching) variable, x_1 , is selected. Check if (i) f_{x_1} is a tautology, and (ii) $f_{x'_1}$ is a tautology. If both are tautologies, f is a tautology, and the algorithm is done. If at least one is not a tautology, then f is not a tautology, and the algorithm is done.

Choice of Splitting Variable. The choice of splitting variable is important. A key goal is to select a variable that is likely to create unate subproblems. The heuristic of Brayton *et al.* is therefore to select (i) a *binate variable*, which (ii) *has the most implicants dependent on it*. A binate variable x is one which appears in both complemented and uncomplemented form. An implicant depends on the variable, if the variable appears in the implicant (complemented or uncomplemented). In case of a tie, the heuristic choose a variable minimizing the difference between number of occurrences with positive polarity and the number of occurrences with negative polarity. The rationale is to keep the recursion tree as balanced as possible.

Rules for Pruning/Simplifying the Recursion Step.

A number of rules can be applied, to avoid unnecessary recursion. The first (**U2**) is a key rule, which uses properties of recursive functions. The second (**M3**) is a special-case speedup (we will not usually focus on M3).

- U2. **Unate Variable.** (*H/S pp. 196-197*) For a single-output function, the recursion can be simplified if the cover is unate in some variable x . In this case, if the cover is positive unate in x , then it is sufficient to check if $f_{x'}$ is a tautology. Alternatively, if the cover is negative unate in x , then it is sufficient to check if f_x is a tautology. (This is a pruning step: in each case, one of the two recursive calls is not necessary. See H/S pp. 196-197 for the justification.)
- M3. **Disjoint Variables.** (*H/S bottom p. 198-199*) The final technique is partitioning. Suppose the cover F can be divided into two subsets of cubes, G and H , where G and H cubes depend on disjoint inputs. That is, if an input x appears as 0 or 1 in a cube of G , then x is '-' for every cube of H . In this case, F is a tautology if and only if either G or H is a tautology. (We will not focus on this rule; if you are interested in details, see H/S bottom p. 198-199.)