CSEE E6861y **Overview of Fast Complementation** Handout 15
Prof. Steven Nowick February 11, 2016

This handout presents an overview of a fast algorithm for **recursive complementation** of a Boolean function.

Introduction

As with the previous tautology-checking algorithm, the complementation algorithm uses a recursive "divide-and-conquer" approach. Given an initial cover F of a Boolean function f, the algorithm divides the cover into smaller pieces until termination conditions are met. Results are then returned and reassembled into a final solution. For the complementation algorithm, the result is the complement of the initial cover.

An important aspect the complementation algorithm, as in tautology checking, is that properties of *unate functions* are exploited to simplify or terminate the recursion. Unate functions have special properties which make them especially useful. While most logic functions are not unate, the recursive decomposition often leads to cofactors which *are* unate.

The name for this general approach is the **unate recursive paradigm**. This paradigm was proposed by Brayton *et al.*, and is used in *tautology*, *complementation, prime generation*, and several other algorithms. The unate recursive paradigm consists of exploiting special properties of unate functions, while performing recursive decomposition.

The Complementation Problem:

Given a cover F corresponding to a Boolean function f, return a cover for the *complement of* f.

The Fast Complementation Algorithm.

We now describe the algorithm used in *espresso*. (The algorithm is also described in Hachtel/Somenzi, "H/S" Ch. 5.3.1; see Handout #10.)

Fast complementation is based on a general theorem, called *Shannon Decomposition* or *Boole's Expansion Theorem* (see also Handout #11). The following *Recursive Complementation Theorem* is a special case (see H/S p. 201, Theorem 5.3.1, Handout #10):

Recursive Complementation Theorem: on Function f. Given a Boolean function f and splitting variable x, then $f' = x \cdot f'_x + x' \cdot f'_{x'}$.

The term f'_x is read as "the complement of f_x", and the term $f'_{x'}$ is read as "the complement of $f_{x'}$". This theorem defines how a Boolean function f can be complemented recursively. The problem of complementing function f is transformed into the problem of finding the complements of two simpler cofactors, f_x and f'_x. Once these two results are returned, they are combined to form the final complement, f'.

The above formulation is in terms of a Boolean function f. Similarly, there is an analogous Recursive Complementation Theorem on a *cover* F of a Boolean function f:

Recursive Complementation Theorem: on Cover F. Given a cover F of a Boolean function f and splitting variable x, then a cover F' of the f' is given by: $F' = x \cdot F'_x + x' \cdot F'_{x'}$.

In practice, this algorithm is often used to generate the OFF-set of a function, when given both ON-set and

DC-set. In this case, the given f and F represent the *combined ON-set and DC-set* of the function. As a result, the returned cover, F', defines the function's OFF-set.

The basic structure of the complementation algorithm is quite similar to recursive tautology checking algorithm described in Handout #11. In each case, termination conditions are checked; if they are not satisfied, recursion is performed. However, *there are 3 key differences:* (i) different rules for termination; (ii) different rules for simplifying the recursion step; and (iii) a different method for returning results.

Basic Rules for Termination.

Basic rules are used to terminate recursion, as in tautology. However, there is a fundamental difference: while the tautology algorithm returns a *True/False* result (*i.e.,* is/is not a tautology), the complementation algorithm returns an *actual cover* (the complement of the initial cover).

B1. **Cover F is Empty.** A cover is *empty* if it contains no cube. In this case, the function is all 0. Therefore, the complement of the function is all 1 (*i.e.,* a tautology). Hence, a cover containing *the universal cube* is returned (*i.e.,* the set containing one cube).

B2. **Cover F includes the Universal Cube.** Here, F is a tautology, so its complement is all 0. Therefore, *an empty cover* is returned (a set containing no cubes).

B3. **Cover F contains a Single Cube.** Here, the complement of F can be computed directly using DeMorgan's Law. That is, *the cube is complemented*, and the resulting cover is returned. For example, if F contains the single cube abc, then $(abc)' = a' + b' + c'$; therefore, the cover containing 3 cubes $\{a', b', c'\}$ is returned.

B4. **Single-Input Dependence.** If the function depends on only one input x (*i.e.,* all other input columns contain only '-'), and the x column contains *both* 1's and 0's, then the function is a tautology, and its complement is all 0. Therefore, *an empty cover* is returned (a set containing no cubes).

Rules for Simplifying the Recursion Step.

Unateness properties are used to simplify the recursion step. The rule (**U1**) below is discussed in H/S, pp. 201-202 (Handout #10), and is based on the following theorem:

Unate Complementation Theorem: on Function f. If f is a positive unate function (*i.e.,* monotonically increasing) in variable x, then:

$$f' = f'_x + x' \cdot f'_{x'}$$

If f is a negative unate function (*i.e.,* monotonically decreasing) in variable x, then:

$$f' = x \cdot f'_x + f'_{x'}.$$

Proof. A proof is given in De Micheli, pp. 300-301 (Theorem 7.3.4), as follows. By Shannon decomposition, $f = x \cdot f_x + x' \cdot f_{x'}$. Suppose f is positive unate in x (there is a similar proof if f is negative unate in x). Then $f_{x'} \leq f_x$, and so $x \cdot f_{x'} \leq x \cdot f_x$. That is, $x \cdot f_{x'}$ is covered by $x \cdot f_x$, so we can add the former to the cover without changing the function: $f = x \cdot f_x + x \cdot f_{x'} + x' \cdot f_{x'}$. Simplifying (using the absorption law of Boolean algebra), we get: $f = x \cdot f_x + f_{x'}$. Complementing the result, we get $f' = (x' + f'_x) \cdot f'_{x'}$, which is $x' \cdot f'_{x'} + f'_x \cdot f'_{x'}$. This expression can still be simplified. By positive unateness, $f_x \geq f_{x'}$, so the

opposite is true after complementation: $f'_x \leq f'_{x'}$. Therefore, $f'_x \cdot f'_{x'} = f'_x$. Therefore, $f' = f'_x + x' \cdot f'_{x'}$, as in the theorem statement.

Similarly, we get a corresponding useful theorem given a cover F of a function f:

Unate Complementation Theorem: on Cover F. If F is a positive unate cover (*i.e.,* monotonically increasing) in variable x, then:

$$F' = F'_x + x' \cdot F'_{x'}$$

If F is a negative unate cover (*i.e.,* monotonically decreasing) in variable x, then:

$$F' = x \cdot F'_x + F'_{x'}.$$

This theorem is a powerful technique used to simplify recursion, whenever there is a unate splitting variable. For an application, see Handout #12 ("Handout of Examples"). We formalize it as a rule:

U1. **Unate Variable.** The recursion can be simplified if the cover F is unate in some variable x. In this case, the Unate Complementation Theorem is used.

Returning the Resulting Complemented Cover.

The result of the algorithm is a complemented cover. The *Recursive Complementation Theorem* describes how the cover is constructed. If a basic termination rule applies (B1-B4), then the appropriate cover is returned immediately. If no basic termination rule applies, the initial cover F is cofactored by both x and x', and the algorithm is recursively called. Each recursive call returns a complemented cover: F'_x and $F'_{x'}$, respectively. To assemble the final result, each cube in F'_x is ANDed with the literal x, and each cube in $F'_{x'}$ is ANDed with the literal x'. The final complemented cover, F', is the union of all of these cubes.

During recursion, if the *Unate Complementation Theorem* can be applied, there is a small modification. In this case, if no basic termination rule applies, and a *unate* splitting variable x is selected, then (following the appropriate part of the Unate Complementation Theorem) both F'_x and $F'_{x'}$ are again returned, but *only one of them* is ANDed with a literal. If F is positive unate in x, then only $F'_{x'}$ is ANDed with a literal ($F' = F'_x + x' \cdot F'_{x'}$). If F is negative unate in x, then only F'_x is ANDed with a literal ($F' = x \cdot F'_x + F'_{x'}$). The result is somewhat "larger" cubes in the final complemented cover, which avoids some fragmentation.

Optimization: Cube Merging

The above method is correct, but is suboptimal: it often results in a large number of cubes in the final cover, F'. The reason is that many small complement cubes are collected together, from different cofactors.

A better approach, used in *espresso*, is to try to *merge* some of the returned cubes. As an example, suppose the returned cover F'_x contains the cube wz', and suppose the returned cover $F'_{x'}$ also contains cube wz'. In this case, the Recursive Complementation Theorem indicates to AND cube wz' (from F'_x) with x, to obtain cube wxz'; and to AND cube wz' (from $F'_{x'}$) with x', to obtain cube $wx'z'$; the two cubes, wxz' and $wx'z'$ are returned.

A simple optimization is to note that the *same cube*, wz', was returned by both branches of the recursion. The result is 2 cubes, wxz' and $wx'z'$, which can be combined into a single cube wz' : $wxz' + wx'z' = wz'$ (using the Absorption Law of Boolean algebra). Therefore, only a single (larger) cube, wz', should be returned. (See also H/S pp. 201-202, Handout #10.)