AN ABSTRACT OF THE THESIS OF

John D. Gilbert, for the degree of Master of Science in Electrical and Computer Engineering, presented on February 22, 1988.

Title: Acceleration Of A Heuristic PLA Product Term Reduction Program Through Complementation Of The PLA Specification.

Abstract approved: ____________________________

(V. M. Powers)

Minimizing the number of product terms in a PLA implementation is a large step towards saving area on a VLSI chip using PLA logic. Due to the large amounts of computer time necessary to achieve this minimization, a number of heuristic approaches have been developed to provide near-optimal solutions in a smaller amount of time.

One such approach is the Pronto Algorithm. Through the expansion of selected product terms, Pronto produces larger cubes covering a number of the remaining product terms, which then become redundant. A significant amount of time is required to determine the validity of a given product term expansion.

The sharp operation currently used for this test is slow compared to the intersection operation. By replacing a sharp with an intersection, significant time savings are achieved. One way to do this is to intersect the expanded cube with the complement of the original function. Tests show that this produces the same result in less time. The tests were designed to decide whether or not the time savings gained by using this technique would justify the overhead in run time necessary to perform a one-time generation of the complement of the PLA function at the beginning of the program.

Test results show that the time used to generate the complement of the PLA function was more than offset by the time savings gained later. More importantly, the saving in time grows as the size of the PLA grows.
Acceleration Of A
Heuristic PLA Product Term Reduction Program
Through Complementation Of The PLA Specification.

by

John D. Gilbert

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed February 22, 1988
Commencement June 1988
Date thesis is presented February 22, 1988.
ACKNOWLEDGEMENT

I would like to thank both Professor Powers and my wife, Barbara, for much patience.
Table of Contents

Chapter 1 - Introduction ........................................................................ 1
  1.1 Thesis Topic ............................................................................. 1
  1.2 Project Status ......................................................................... 1
  1.3 Thesis Goal ............................................................................. 2

Chapter 2 - Basic Definitions ............................................................. 3
  2.1 PLA ....................................................................................... 3
  2.2 Literal .................................................................................... 3
  2.3 PLA Representation ............................................................... 3
  2.4 PLA Specification ................................................................. 3
  2.5 Product Term ......................................................................... 4
  2.6 PLA Function ......................................................................... 4
  2.7 Cube ..................................................................................... 4
  2.8 + Operator ............................................................................. 4
  2.9 Examples ............................................................................... 5

Chapter 3 - Cube Relations And Operations ....................................... 7
  3.1 Cover ..................................................................................... 7
  3.2 Expansion ............................................................................... 7
  3.3 Adjacency ............................................................................... 7
  3.4 Complete Adjacency .............................................................. 8
  3.5 Cube Operations ..................................................................... 8

Chapter 4 - Modification Of The PRONTO Program ............................ 12
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 The PRONTO Algorithm</td>
<td>12</td>
</tr>
<tr>
<td>4.2 Checking The Validity Of Expanded Terms</td>
<td>13</td>
</tr>
<tr>
<td>4.3 Equivalence Of c # F And c F'</td>
<td>13</td>
</tr>
<tr>
<td>4.4 Previous PRONTO Evaluation Of Expanded Cubes</td>
<td>14</td>
</tr>
<tr>
<td>4.5 Modifications To PRONTO</td>
<td>15</td>
</tr>
<tr>
<td>Chapter 5 - Fast Complementation</td>
<td>16</td>
</tr>
<tr>
<td>5.1 Definitions</td>
<td>16</td>
</tr>
<tr>
<td>5.2 Lemma: The Complement Of A Cube</td>
<td>16</td>
</tr>
<tr>
<td>5.3 A Fast Complementation Algorithm</td>
<td>18</td>
</tr>
<tr>
<td>5.4 Enhancements To The Algorithm</td>
<td>23</td>
</tr>
<tr>
<td>5.5 The Complementation Program</td>
<td>25</td>
</tr>
<tr>
<td>Chapter 6 - Timing Studies</td>
<td>32</td>
</tr>
<tr>
<td>6.1 Timing Considerations</td>
<td>32</td>
</tr>
<tr>
<td>6.2 Initial Implementation</td>
<td>32</td>
</tr>
<tr>
<td>6.3 Sparse PLA Functions</td>
<td>39</td>
</tr>
<tr>
<td>6.4 Addition Of Enhancement Functions To The Complementation Program</td>
<td>43</td>
</tr>
<tr>
<td>Chapter 7 - Conclusion</td>
<td>44</td>
</tr>
<tr>
<td>7.1 Summary Of Thesis Results</td>
<td>44</td>
</tr>
<tr>
<td>7.2 Suggestion For Future Work</td>
<td>45</td>
</tr>
<tr>
<td>Bibliography</td>
<td>47</td>
</tr>
</tbody>
</table>
List of Figures

Figure 2.1 - A PLA representation ......................................................... 5
Figure 3.1 - Input Coordinate Cube Intersection ......................................... 8
Figure 3.2 - Output Coordinate Cube Intersection ....................................... 9
Figure 6.1 - Run time comparison of PRONTO versions for input PLA functions of between 0 and 40 terms. ................................................................. 33
Figure 6.2 - Run time comparison of PRONTO versions for input PLA functions of between 49 and 100 terms. ................................................................. 34
Figure 6.3 - Run time comparison of PRONTO versions for input PLA functions of between 100 and 150 terms. ................................................................. 35
Figure 6.4 - Run time comparison of PRONTO versions for input PLA functions of between 150 and 500 terms. ................................................................. 36
Figure 6.5 - Results of two PRONTO versions for identical input ...................... 37
Figure 6.6 - Differing reduction results ......................................................... 38
Figure 6.7 - Calculation time for sparse PLA functions .................................... 40
Figure 6.8 - Intersection procedure breakdown for PLA functions of varying sparsity ................................................................. 41
Figure 6.9 - Results of enhanced and unenhanced complementation programs for identical input ................................................................. 43
1.1. Thesis Topic

The speed with which heuristically guided PLA product term reduction programs compute solutions is an ongoing area of research. Improvements in the implementations of algorithms are being developed and tested.

One such is the PRONTO program. This thesis discusses an attempt to improve the program through implementation of a fast complementation procedure, allowing replacement of the sharp operation by one quicker to implement, the intersection operation.

1.2. Project Status

The PRONTO algorithm was developed as a one-pass heuristically guided approach to PLA product term reduction by J. R. Martinez-Carballido [MAR]. The PRONTO program was written in the Mainsail (a trademark of Xidak Corporation) programming language through the combined efforts of E. Burns, R. Stettler, and D. Dagit. D. Dagit produced the first running version of PRONTO.

Improvements were made to the program by N. Abweh [ABW] based on his studies of the "dominant time-growth factors" in the program. He tested different procedure implementations, comparing them for speed. One result of his studies was the identification of the sharp operation as the consumer of a significant portion of the program's execution time. He suggested that the replacement of the sharp operation by a technique using the intersection with the complement of the PLA function would be a
good direction of research to further improve the program's run time. His suggestion is the focus of this thesis.

1.3. Thesis Goal

Fast Complementation has been implemented as an aid to fast PLA product term reduction in other PLA reduction programs [SAS] [BRA1]. Application to the PRONTO program is straightforward. This thesis develops a fast complementation algorithm suitable for use in PRONTO, and incorporates it into the program. Studies are then done on the run time of the new version of PRONTO vs. the old, to determine the relative merits of the addition of the fast complementation procedure.
Chapter 2

Basic Definitions

2.1. PLA

A PLA (Programmable Logic Array) is an integrated circuit taking as input \( m \) variables and their complements, and having as output \( q \) functions, each a function of a subset of the \( m \) input variables. The PLA implements these functions with 2-level logic, usually AND-OR, but often NOR-NOR. A typical use of a PLA is to implement functions described in Sum-Of-Products form.

2.2. Literal

A literal is an input variable to a PLA.

2.3. PLA Representation

A PLA representation consists of a set of input lines (the AND plane) and a set of output lines (the OR plane). The input lines are vertical and represent input variables to a multiple output function. The input lines are set in complement pairs, i.e. \( x \) and \( x' \), \( y \) and \( y' \), \( z \) and \( z' \). The output lines are also vertical, and represent the output functions, one to a line. Horizontal lines run through the AND plane and into the OR plane. These lines compute the AND of all inputs connected to them, hence each represents a term in a SOP form of the functions. If an output line is connected to one of these horizontal lines, then that function has the AND of those input variables connected to the horizontal line as a term of that function. The function is output as the logical OR of all such terms.

2.4. PLA Specification

A PLA specification is a shorthand used to describe the PLA representation. It is a matrix with \( m+q \) columns and \( n \) rows. The matrix has \( m \) 'input' columns,
representing the \( m \) literals of the PLA, and \( q \) 'output' columns, representing the \( q \)
output functions. There are \( n \) rows in the specification, each representing a horizontal
line in the PLA representation. A '1' in an input column of a row means the
variable is connected at that line in the PLA's AND plane. A '0' means the
variable's complement is connected. And a '-' , 'X', or 'd' means there is no connec-
tion for the variable. In an output column of a row, a '1' means there is a connec-
tion in the OR plane at that line for a function, a '0' means there is no connection,
and a '-' , 'X', or 'd' means it does not matter if there is a connection or not.

2.5. Product Term

A product term is a row of the PLA specification.

2.6. PLA Function

A PLA function, \( F \), is a PLA specification that describes a desired multiple-output
function.

2.7. Cube

A cube, \( c \), is a product term (including outputs) of the PLA specification. Each cube
consists of \( m \) input variables, \( x_1, x_2, \ldots, x_m \), and a set of outputs, \( y \). If all outputs are 0
in \( y \), then \( c \) is said to be empty and may be discarded.

2.8. + Operator

The + operator ("OR") is used as a shorthand for replacing the PLA specification.
A PLA function can be described as a set of \( n \) cubes such that

\[
F = c_1 + c_2 + \ldots + c_n.
\]
2.9. Examples

Example 1.1 - Definition Illustration

Consider the following functions:

\[
\begin{align*}
    f_1 &= x_1 + x_2 + x_3, \\
    f_2 &= x_1'x_2, \\
    f_3 &= x_1'x_2 + x_2x_3.
\end{align*}
\]

The PLA representation for these functions is:

Figure 2.1
A PLA representation.

(X indicates a connection)

The PLA representation above has the following PLA specification:

<table>
<thead>
<tr>
<th></th>
<th>x₁</th>
<th>x₂</th>
<th>x₃</th>
<th>f₁</th>
<th>f₂</th>
<th>f₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Note that the above specification is not unique for the PLA representation. Row 5 can also be written as

\[-1 \ 1 \ / \ d \ 0 \ 1\]

since \( f_1 \)'s output will be the same regardless of whether or not there is a connection in row 5.

A product term of the above PLA specification is

\[-1 \ 1 \ / \ 0 \ 0 \ 1\]

The cube

\[0 \ -1 \ - \ / \ 0 \ 0 \ 0 \ 0\]

is empty, ie. it is not a term of any of the output functions of the PLA.

The PLA function shown by the PLA specification in this example can be written

\[F = c_1 + c_2 + c_3 + c_4 + c_5,\]

where

\[
\begin{align*}
c_1 &= 1 \ - \ - \ / \ 1 \ 0 \ 0 \\
c_2 &= - \ 1 \ - \ / \ 1 \ 0 \ 0 \\
c_3 &= - \ - \ 1 \ / \ 1 \ 0 \ 0 \\
c_4 &= 0 \ 1 \ - \ / \ 0 \ 1 \ 1 \\
c_5 &= - \ 1 \ 1 \ / \ 0 \ 0 \ 1
\end{align*}
\]
Chapter 3

Cube Relations And Operations

3.1. Cover

Cube a covers cube b if the following conditions hold -

1. For every $x_i = 1$ in cube a, $x_i = 1$ in cube b.
2. For every $x_i = 0$ in cube a, $x_i = 0$ in cube b.
3. For every 1 in the output of cube b, there exists a 1 in the same output column of cube a.

3.2. Expansion

Cube a is an expansion of cube b if one of the following occurs -

1. Cube b is identical to cube a, except in a single input $x_i$, which is either 0 or 1 in cube a and is a dc in cube b. This is expansion in the input direction $x_i$.
2. Cube b is identical to cube a, except that it has 1's in one or more output columns where cube a has 0's or dc's. This is expansion in the output direction.

3.3. Adjacency

Cubes a and b are adjacent if both of the following occur -

1. Input adjacency. This happens when a and b have exactly one input column where one cube has a 0 entry and the other a 1. The variable of adjacency is that represented by the input column for which the above condition holds.
2. Output adjacency. This happens when a and b have at least one common output column where each has a 1.
3.4. Complete Adjacency

Cube a is completely adjacent to cube b if a and b are adjacent, and, when the variable of adjacency is set to dc in both cubes, b covers a.

3.5. Cube Operations

Def: $a_i$ and $b_i$ denote input variable $x_i$ in cubes a and b, respectively.

$a_o$ and $b_o$ denote output $f_o$ in cubes a and b, respectively.

3.5.1. Intersection

Def: The intersection of 2 cubes, a and b, is calculated as follows -

(1) A cube, c, is created where $c_i = a_i b_i$ and $c_o = a_o b_o$, according to the tables in Figures 3.1 and 3.2.

(2) If $c_i = *$ for some input, or $c_o = 0$ for all outputs, then c is empty. Otherwise, it remains unchanged.

Figure 3.1
Input Coordinate Cube Intersection

<table>
<thead>
<tr>
<th>ab</th>
<th>$b_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i$</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>*</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>


3.5.2. Sharp

Def: The sharp product of two cubes, \( a \# b \) is calculated as follows:

1. if \( a \# b \) is empty, then \( a \# b = a \).
2. if \( b \) covers \( a \), then \( a \# b \) is empty.
3. otherwise, an array of cubes is found for \( a \# b \) in the following manner:
   - for each \( x_i = - \) in cube \( a \), if either \( x_i = 0 \) or \( x_i = 1 \) in cube \( b \) then copy cube \( a \) to a new cube \( c_j \) (i.e. \( c_j = a \)).
   - for each cube \( c_j \) copied, set the corresponding \( x_i \) in \( c_{j'} \) for which the cube was created, to \( x_i' \) of cube \( b \).

Def: Let \( A_i \) represent cube \( i \) of PLA function \( A \), and \( B_i \) represent cube \( i \) of PLA function \( B \). Let \( c \) represent an arbitrary cube. Then
(1) \( c \# A = (\cdots (c \# A_1) \# A_2) \# \cdots \# A_n) \).

(2) \( A \# c = (A_1 \# c) \cup (A_2 \# c) \cup \cdots \cup (A_n \# c) \).

(3) \( A \# B = (A_1 \# B) \cup (A_2 \# B) \cup \cdots \cup (A_n \# B) \).

\[
= (\cdots (A \# B_1) \# B_2) \# \cdots \# B_n) .
\]

Conceptually, \( A \# B \) returns that portion of \( A \) not covered by \( B \).

3.5.3. Examples

Example 3.1 - Cover

Let

\[
\begin{align*}
cube b &= \begin{array}{cccc}
1 & 0 & - & - \\
- & - & 1 & - \\
/ & 1 & 1 & 0 \\
0 & 0 & 1 & 1
\end{array} \\
cube a &= \begin{array}{cccc}
1 & 0 & - & - \\
1 & 1 & 0 & - \\
/ & 0 & 0 & 1
\end{array}
\end{align*}
\]

Then cube \( b \) covers cube \( a \).

Example 3.2 - Input Expansion

Let

\[
\begin{align*}
cube b &= \begin{array}{cccc}
1 & 0 & - & - \\
- & - & 1 & - \\
/ & 1 & 0 & 0
\end{array} \\
cube a &= \begin{array}{cccc}
1 & 0 & 1 & - \\
- & - & 1 & 0 \\
/ & 0 & 0 & 0
\end{array}
\end{align*}
\]

Then \( b \) is an expansion of \( a \) in the \( x_3 \) direction.

Example 3.3 - Output Expansion

Let

\[
\begin{align*}
cube b &= \begin{array}{cccc}
1 & - & - & - \\
& & 1 & 1 \\
& / & 1 & 0 \\
& & 0 & 1
\end{array} \\
cube a &= \begin{array}{cccc}
1 & - & - & - \\
& & 1 & - \\
& / & 0 & 0 \\
& & d & 0
\end{array}
\end{align*}
\]

Then \( b \) is an expansion of \( a \) in the output direction.
Example 3.4 - Adjacency

Let

\[
\text{cube } b = \begin{bmatrix} -1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}
\]

\[
\text{cube } a = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}
\]

Then \(a\) and \(b\) are adjacent, in the variable \(x_3\).

Example 3.5 - Complete Adjacency

Let

\[
\text{cube } b = \begin{bmatrix} 1 & -1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}
\]

\[
\text{cube } a = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}
\]

Then cube \(a\) is completely adjacent to cube \(b\).

Example 3.6 - Intersection

The intersection of the two cubes

\[
\begin{bmatrix} 0 & 1 & -1 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 & d & d & d & 1 \end{bmatrix}
\]

and

\[
\begin{bmatrix} 0 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 1 & d & 1 \end{bmatrix}
\]

is

\[
\begin{bmatrix} 0 & * & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & d & d & 1 \end{bmatrix}
\]

which is discarded due to the '*' in the 2nd column, indicating an invalid cube.

Example 3.7 - Sharp

The sharp of the two cubes

\[
\begin{bmatrix} - & - & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}
\]

\[
\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}
\]

is the pair of cubes

\[
\begin{bmatrix} 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix}
\]

\[
\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \end{bmatrix}
\]
Chapter 4
Modification Of The Pronto Program

4.1. The PRONTO Algorithm

The PRONTO algorithm is heuristically guided to reduce the number of product terms in a PLA specification to a near-optimal number, in one pass. It uses a loop of four main actions, repeatedly applying these actions until a solution specification is built and the original specification is emptied. The actions taken in the loop are as follows:

1. Select a base product term. This is a product term or cube of the PLA specification that is most likely to cover other terms if expanded. The base product term is chosen as being "most nearly essential to a solution" [MAR].

2. Find a set of directions in which expansion of the base product term is considered likely to cover other terms in the specification.

3. Expand the base product term in those directions determined to be valid and useful. It is the determination of the validity of these expanded terms that is the basis for the research done in this thesis.

4. Update the solution and original specifications. Here, chosen expansions of the base product term are included in the solution. Terms in the original specification covered by these expanded terms are removed and discarded. Partially covered terms are modified to reflect that part remaining that is uncovered, and are returned to the original specification.

A more detailed explanation of the PRONTO algorithm can be found in [MAR] and an explanation of the implementation as a program can be found in [ABW].
4.2. Checking the Validity of Expanded Terms

As stated above, the principal calculation addressed by this thesis is that of determining the validity of expansion directions of base product terms. To date, the implementation of the PRONTO algorithm checked the validity of the expanded terms by a sharp operation, \( c \# F \), where \( c \) is the expanded cube, and \( F \) is the original PLA function. It was found [ABW] that the time needed to perform the many sharp operations required was a significant portion of the run time of the program.

Modifications were made to the program lessening the number of times the sharp operation had to be performed [ABW]. However, it was suggested that a way to eliminate the sharp operation altogether could be found by using a technique that intersected the expanded cubes with the complement of the original function. It will be shown in the next section that intersection of a cube with the complement of a PLA function is equivalent to performing the sharp operation between that cube and the uncomplemented PLA function.

4.3. Equivalence Of \( c \# F \) And \( c F' \)

Let \( c \) be a possible cube contained in a PLA function \( F + F' \). Then \( c \# F \) and \( c F' \) are equivalent.

Proof -

Following the definition for the sharp operation given in section 3.5.3, \( c \# F = (...) (c \# F_1) \# F_2 \) \( ... \# F_n) \). It is obvious from the definition of the sharp of two cubes that \( c \# F_1 \) produces a set of cubes describing that part of \( c \) not covered by \( F_1 \). This result is then sharped, cube by cube, with \( F_2 \). Since each resulting cube of the first sharp was a subcube of \( c \), the second sharp must produce a set of cubes that are that part of the first set not covered by \( F_2 \). Hence they represent that part of \( c \) not covered by the cubes \( F_1 \) or \( F_2 \) . Continuing in this manner, \( c \# F \) represents that part of \( c \) not covered by any of the
cubes in $F$. This is the same as that part of $c$ covered by the cubes of $F'$ (since all cubes must be in $F$ or $F'$, or partially in both). This is the same as $cF'$.

4.4. Previous PRONTO Validation Of Expanded Cubes

After a set of possible expansion directions is found, the PRONTO program calls a procedure $\text{Valid}$, which expands the base product term in all the chosen directions, and has them checked for validity. A valid product term is one which is completely covered by the original PLA function. The original PLA function may have don't cares in the outputs, in which case the expanded term is valid if it is covered by the PLA function obtained by setting all output don't cares to 1 in the original specification. The change of output don't cares to 1's is done by the PRONTO procedure $\text{Larger}$.

The checking of the expanded cube against the original specification is performed by one of two procedures, $\text{Donts}$ or $\text{Donts2}$. The latter is a modification of the former to take advantage of certain characteristics of the result of sharp operations to lessen the frequency with which they need be performed. It is therefore the actions of the procedure $\text{Donts}$ which is significant here.

$\text{Donts}$ takes an expanded cube produced by $\text{Valid}$, and sharps it with the original PLA specification (modified if don't cares are present in the output, as discussed above). If the result is anything other than an empty cube, then an empty cube is returned to $\text{Valid}$, signifying an invalid expanded cube. Otherwise, $\text{Donts}$ iteratively sets the don't cares in the outputs of the expanded cube to 1, checking each time that the cube is still valid through use of the sharp operation. The cube returned to $\text{Valid}$ is the expanded cube with as many output don't cares set to 1 as possible, while still remaining a valid expanded cube. The reason that a maximum number of don't cares are set to 1 in the expanded cube is that this enlarges the cube, making it more likely to cover other terms in the specification.
4.5. Modifications To PRONTO

Because the original procedure *Donts* sharped each expanded cube with the original specification (with all output don't cares set to 1), it was replaced for the tests described in this thesis by a version of *Donts* which computed the intersection of the expanded cube with the complement or the original specification. The complement of the original function (again, with all output don't cares set to 1) is calculated at the beginning of the PRONTO program. This complement is then passed as a parameter through the necessary levels of procedures to the procedure *Donts*. 
Chapter 5

Fast Complementation

The following algorithm is derived as a modified form of the complementation algorithm for multiple-valued input functions given by [SAS].

First, some necessary definitions, and a lemma:

5.1. Definitions

The following hold for a PLA function \( F \) with \( m \) literals, and \( n \) cubes.

1. \( y_i \) = outputs of cube \( c_i \), \( i = 1, \ldots, n \).
2. \( y_i' = \) complement of \( y_i \). Each output bit in \( c_i \) is complemented.
3. \( F = 0 \) or \( F = 1 \) \( \implies \) \( F \) is either composed of all 0 functions or all 1 functions, respectively. \( y_i = 0 \) or \( y_i = 1 \) \( \implies \) \( y_i \) is all 0's or all 1's, respectively, in the cubes of \( F \).
4. \( F_k = F \), with variables \( x_1, x_2, \ldots, x_k \) set to don't care.
5. \( c_L = \) one of the largest cubes in \( F \) (arbitrarily chosen if several cubes of the same size exist as the largest).
6. \( F/c_i = F \), with cube \( c_i \) removed.
7. \( F = c = \) \( \implies \) \( F \) is the PLA function consisting of only the cube \( c \).

5.2. Lemma: The Complement Of A Cube.

For cube \( c \), whose outputs are \( y \) and whose non-dc input variables (ie. \( x_j = 0 \) or \( x_j = 1 \)) are arranged in the order \( x_1, x_2, \ldots, x_k \), the complement is constructed as follows:

- Form \( k \) cubes \( c_i' \), \( i = 1, \ldots, k \). Each \( c_i \) has only one non-dc input, \( x_i' \), and all outputs are set to 1.
• Form a cube with all input variables set to don't care. Set the outputs to the complement of the outputs in the original cube, i.e. \( y' \).

• The complement of the cube is the PLA function consisting of the \( k + 1 \) new cubes generated as above.

**Example 5.1 - Complement of a Cube**

Consider the cube

\[
\begin{array}{cccc}
1 & 0 & 1 & - \\
- & - & - & 0 \\
1 & 1 & 1 & 0
\end{array}
\]

The complement of this cube is

\[
\begin{array}{cccc}
0 & - & - & - \\
- & 1 & - & - \\
- & - & 0 & - \\
- & - & - & 1
\end{array}
\]

Proof of Lemma:

Cube \( c \) can be represented by a product term with inputs \( x_1x_2...x_k \) and output set \( y \).

By DeMorgan's Rule, the complement of a product term's input \( x_1x_2...x_k \) is \( x_1' + x_2' + ... + x_k' \).

Let \( F \) be the PLA function consisting of the cube \( c \) only. Application of DeMorgan's rule to the inputs of \( F \) generates the complement for each \( f_j \) in \( F \) whose \( y_j = 1 \) in \( c \). The cubes thus generated are also part of the complement of those \( f_j \) in \( F \) whose \( y_j = 0 \) in \( c \), since the complement of these \( f_j = 1 \). Hence all \( y_j = 1 \) for these cubes.

For each \( f_j \) in \( F \) such that \( y_j = 0 \) in \( c \), the complement is, as stated, \( f_j = 1 \). This is added to \( F' \) by setting all corresponding \( y_j = 1 \) in a cube for which all inputs are set to don't care.
5.3. A Fast Complementation Algorithm

5.3.1. Rule 1

Definition -

(1) If $F = 0$ then $F' = 1$.

(2) a.) If all inputs in $F$ are don't care, then $F$ is equivalent to a single cube $c$, whose inputs are all set to don't care, and whose outputs are determined as follows:
- If any cube $c_i$ of $F$ has a 1 in an output position, then that output position has a 1 in the new cube, $c$. Otherwise, it is 0.

b.) $F' = c'$.

Note that Rule 1 is a termination rule, i.e. no further PLA function complements need be generated.

Proof -

(1) trivial

(2) If every $c_i$ in $F$ is a cube such that all inputs are dc, then each $f_j$ is a constant 0 or 1. If $f_j$ has a 1 for any $y_j$, then $f_j = 1$. Otherwise, $f_j = 0$. So, $F$ can be represented by the cube whose inputs are all don't care, and whose outputs indicate which $f_j = 0$, and which $f_j = 1$. $F'$ is the complement of this cube.

5.3.2. Rule 2

Definition -

If $c_L$ covers all $c_i$, $i = 1, \ldots, n$, then $F' = c'_L$.

Note that Rule 2 is also a termination rule.
Proof -

If $c_L$ covers all $c_i$, then $F = c_L$. By the lemma previously stated, $F' = c_L'$.

5.3.3. Rule 3

Definition -

If $k$ inputs can be rearranged such that $x_j$ is 0 or 1 for $j = 1, \ldots, k$, then $x_1x_2\ldots x_k$ is a common factor to all $c_i$ in $F$, and $F'$ can be constructed as follows:

- Form $k$ cubes, $c_i$, $i = 1, \ldots, k$. Each cube has only one non-don't care input, $x_i'$, and all outputs are set to 1.
- Let $H$ be the PLA function formed by the $k$ cubes generated above. Then

$$F' = H + F_k'$$

(as stated previously, $F_k = F$, with variables $x_1, x_2, \ldots, x_k$ set to don't care).

Note that Rule 3 causes the complementation algorithm to recurse, since the complement of $F_k$ must be generated.

Proof -

If $F$ has a common factor $x_1x_2\ldots x_k$, then $F$ can be rewritten as $x_1x_2\ldots x_k F_k$. By DeMorgan's Rule,

$$F' = x_1' + x_2' + \ldots + x_k' + F_k'.$$

5.3.4. Rule 4

Definition -

If for every $c_i$ in $F$ either $x_j = 1$ or $x_j = 0$, then $F$ is decomposable on $x_j$ ($F = x_jG_{1_j}$ + $x_j'G_{0_j}$) and $F' = x_jG_{1_j}' + x_j'G_{0_j}'$, where $G_0$ contains all $c_i$ of $F$ whose $x_j = 0$, and $G_1$ contains all $c_i$ of $F$ whose $x_j = 1$. In both $G_0$ and $G_1$, $x_j$ is set to don't care.
Note that Rule 4 causes the complementation algorithm to recurse, since the complements of \( G0 \) and \( G1 \) must be generated.

**Proof -**

For each function \( f \) in \( F \), \( f \) can be rewritten as \( x_i'g0 + x_ig1 \), where \( g0 \) and \( g1 \) are as defined in Rule 4 above. The complement of this is \( x_i'g0' + x_ig1' \). If all \( f \) in \( F \) are decomposable on the same \( x \), then (for \( x=x_j \))

\[
F = (x_j'g0_1 + x_ig1_{11}) + \ldots + (x_j'g0_n + x_ig1_{n1}) = x'G0 + xG1.
\]

The complement follows.

**5.3.5. Rule 5**

**Definition -**

If none of the above Rules 1 - 4 apply, then

\[
F' = c_i'(F/c_i)' \text{ for any } i \text{ in } \{1, \ldots, n\}.
\]

Note that Rule 5 causes the complementation algorithm to recurse, since the complement of \( F/c_i \) must be generated.

**Proof -**

\[
F = F/c_i + c_i. \text{ Hence, } F' = (F/c_i)'c_i'.
\]

**5.3.6. Proof of Correctness of Complementation Algorithm**

It is evident that any valid \( F \) will have one of Rules 1-5 applied, since Rule 5 covers all cases not already covered by Rules 1-4, by definition. To prove the correctness of this algorithm, the correctness of the individual Rules must and has been shown. It remains to prove that application of these Rules always results in the complement of any valid PLA function.
5.3.8.1. Proof of Recursion Termination

It remains to prove that the recursion of Rules 3, 4, and 5 will always terminate.

In each of Rules 3, 4, and 5 the complement of the PLA function is determined through computation with the complements of one or two simpler PLA functions (fewer non-dc inputs or fewer product terms). The complements of these functions are generated through recursion of the complementation algorithm. The progression of this recursion is always towards a PLA function that will satisfy the conditions of a termination Rule (Rule 1 or Rule 2).

1. In Rule 3, the complement to be generated is $F_k'$. $F_k$ has its first $k$ inputs set to don’t care. This brings the representation closer to that accepted by a termination Rule than is the case with $F$, since when all inputs are don’t cared, a PLA function satisfies the conditions of Rule 1.

2. In Rule 4, the complements to be generated are $G0'$ and $G1'$. Since $G0$ and $G1$ both have input $x_j$ set to don’t care, their representation is closer to the conditions that will satisfy Rule 1 than that of $F$. In addition, $G0$ and $G1$ are likely to have fewer cubes than $F$, bringing their representations closer to the conditions which will satisfy Rule 2, also a termination Rule.

3. In Rule 5, the complement to be generated is $(F/c_i)'$. Since $F/c_i$ has fewer cubes than $F$, its representation is closer to the conditions which will satisfy Rule 2.

5.3.7. Examples of the use of complementation algorithm Rules

Example 5.2 - Use of Rule 1

The following is an example of the use of Rule 1:
Let $F = \begin{array}{ll}
- & \cdot \cdot \cdot / 010 \\
- & \cdot \cdot \cdot / 100
\end{array}$

Then form $c = \begin{array}{ll}
- & \cdot \cdot \cdot / 110
\end{array}$

Finally, $F' = \begin{array}{ll}
- & \cdot \cdot \cdot / 001
\end{array}$

Example 5.3 - Use of Rule 2

The following is an example of the use of Rule 2:

Let $F = \begin{array}{ll}
- & \cdot \cdot \cdot / 011 \\
0 & \cdot \cdot \cdot / 001
\end{array}$

Then $c_L = c_1 =$

\begin{array}{ll}
-1 & / 011 \\
01 & / 001
\end{array}

and $c_L$ covers $c_2$.

Finally, $F' = c_L' =$

\begin{array}{ll}
0 & / 111 \\
- & / 100
\end{array}

Example 5.4 - Use of Rule 3

The following is an example of the use of Rule 3:

Let $F = \begin{array}{ll}
1 & / 011 \\
1 & / 100
\end{array}$

Then $x_1$ is a common factor, and $F' =$

\begin{array}{ll}
0 & / 111 \\
- & / 010
\end{array}

(Note that the complement of $F_1$ is found by applying Rule 1).

Example 5.5 - Use of Rule 4

The following is an example of the use of Rule 4:
Let $F =$

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
</tbody>
</table>

Then $G_0 =$

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

and $G_1 =$

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
</tbody>
</table>

Finally, $F' =$

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
</tbody>
</table>

**Example 5.6 - Use of Rule 5**

The following is an example of the use of Rule 5:

Let $F =$

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Then $c_1' =$

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

and $(F/c_1)' =$

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
</tbody>
</table>

Finally, $F' =$

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

(Note that $c_4$ and $c_5$ are redundant, since they are covered by $c_6$).

**5.4. Enhancements To The Algorithm**

Enhancements to the complementation algorithm were designed with the intent of minimizing the number of product terms in the complement of the given PLA specification. The goal of such a minimization was to reduce the time spent in the Pronto program doing intersections between the expanded cubes and the complement, as this intersection is done on a cube by cube basis.
5.4.1. Heuristics

One heuristic was used to improve the complementation program. For the case where none of the Rules 1-4 of the complementation algorithm apply, the algorithm says to apply Rule 5, $F' = c_i^r(F/c_i)$. An enhancement to this Rule is to modify it to read $F' = c_L^r(F/c_L)$, where $c_L$ is the largest cube in $F$. The reason for this change is that the complement of $c_L$ produces a minimum number of cubes, since it has the least number of literals (see Lemma). This in turn means that the intersection of $c_L^r$ with $(F/c_L)^r$ will likely produce a minimal number of cubes.

A note: Another heuristic, possibly more effective, is given in [SAS].

5.4.2. Simple Reduction Techniques.

- The removal of covered terms from the PLA function complement was implemented as an enhancement.

- The raising of terms was implemented. This is the expansion of a cube that is completely adjacent to another. The expansion is done in the direction of the variable of adjacency.

Example 5.7 - Cube Reduction

For the cubes of Example 3.5, ie.

\[
\begin{align*}
cube b &= 1 - 1 - / 1 0 1 \\
cube a &= 1 0 0 - / 1 0 0
\end{align*}
\]

cube a is completely adjacent to cube b. Therefore, cube a can be raised in the $x_3$ direction. This is accomplished by setting the variable of adjacency in the smaller cube to don't care, as has been done in cube $a^*$ below. Cube a is completely adjacent to b across the variable $x_3$,

\[
a^* = 1 0 - / 1 0 0
\]
and cube $a^*$ can replace cube $a$ in the complement. This is desirable because cube $a^*$ is larger than cube $a$.

### 5.4.2.1. Application of reduction techniques

The above two reduction techniques can be applied between steps of a complementation, or after a complement is computed. Reduction in the partial complement was not attempted due to the complexity of the problem. Reduction of the final result was implemented and the results were compared to those of a complementation program without reduction.

### 5.5. The Complementation Program

The following is a description of the program used to implement the fast complementation algorithm.

#### 5.5.1. Primary Functions and Procedures.

- **Complement.**
  
  This function accepts a PLA specification as input and returns the complement of that specification by applying the Rules of the complementation algorithm.

- **Rule1.**
  
  This function accepts as input a PLA specification, and checks whether or not the conditions of Rule 1 in the complementation algorithm are satisfied, returning an appropriate boolean value. In addition, if the conditions of Rule 1 are satisfied, the function passes back a parameter stating which condition was satisfied.

- **Rule2.**
  
  This function accepts as input a PLA specification, and checks whether or not the conditions of Rule 2 in the complementation algorithm are satisfied, returning an appropriate boolean value. In addition, if the conditions of Rule 2 are satisfied, the
function passes back a parameter containing a copy of the largest cube in the PLA specification.

- **Rule3.**
  This function accepts as input a PLA specification, and checks whether or not the conditions of Rule 3 in the complementation algorithm are satisfied, returning an appropriate boolean value. In addition, if the conditions of Rule 3 are satisfied, the function passes back a parameter containing a copy of the factor common to all cubes in the PLA specification.

- **Rule4.**
  This function accepts as input a PLA specification, and checks whether or not the conditions of Rule 4 in the complementation algorithm are satisfied, returning an appropriate boolean value. In addition, if the conditions of Rule 4 are satisfied, the function passes back a parameter containing the column number of the input variable on which the PLA specification can be decomposed.

- **DoRule1.**
  This function performs the action required by the complementation algorithm in the event that Rule 1 is satisfied. It takes as input the PLA specification, and the parameter passed by **Rule1** indicating which condition of Rule 1 was satisfied. It returns a PLA specification that is the complement of the input PLA specification.

- **DoRule2.**
  This function performs the action required by the complementation algorithm in the event that Rule 2 is satisfied. It takes as input a parameter holding a copy of a cube. It returns a PLA specification that is the complement of this cube.

- **DoRule3.**
  This function performs the action required by the complementation algorithm in the event that Rule 3 is satisfied. It takes as input the PLA specification, and the
parameter passed by Rule 3 holding a copy of the common factor to all cubes. It removes the common factor from all cubes of the PLA specification, then complements recursively this new PLA specification. It also complements the common factor. It then recombines these two complements according to the method explained in Rule 3 of the complementation algorithm. Finally, it returns a PLA specification that is the complement of the original input PLA specification.

- **DoRule4.**

  This function performs the action required by the complementation algorithm in the event that Rule 4 is satisfied. It takes as input the PLA specification, and the parameter passed by Rule 4 indicating the variable on which the PLA specification can be decomposed. It removes this variable from all cubes of the PLA specification, meanwhile splitting the PLA specification into two new PLA specifications according to the method described by Rule 4 of the complementation algorithm. It then complements these new PLA specifications recursively. Finally, it recombines the complements of the two new PLA specifications, again as per Rule 4 of the complementation algorithm. It returns a PLA specification that is the complement of the original input PLA specification.

- **DoRule5.**

  This function performs the action required by the complementation algorithm in the event that Rules 1-4 are not satisfied. It takes as input the PLA specification, and removes the largest cube. It then complements both this new PLA specification and the removed cube, recursively for the new PLA specification, and by applying DoRule2 to the removed cube. Finally, it recombines these complements according to the method described in Rule 5 of the complementation algorithm. It returns a PLA specification that is the complement of the original input PLA specification.
5.5.2. Helper Functions and Procedures

- **AppendCube.**
  
  This procedure appends a cube onto a PLA specification, copying it from another PLA specification. It accepts as input the PLA specification to which the cube is to be appended, the PLA specification containing the cube, and the location of the desired cube in the second PLA specification.

- **RemoveCube.**
  
  This procedure removes a cube from a PLA specification. It takes as input the PLA specification and the position of the cube to be removed. It passes back the modified PLA specification.

- **AppendFunction.**
  
  This function returns a PLA specification that is two other PLA specifications appended together. It accepts as input the two PLA specifications, which are disposed of during the course of the function.

- **LargestCube.**
  
  This function takes as input a PLA specification and returns the position of the largest cube.

- **EqualOutputs.**
  
  This function takes as input a PLA specification and returns TRUE if the outputs of all the cubes are the same.

- **ZeroOutputs.**
  
  This function takes as input a PLA specification and an integer indicating the location of a cube in the specification. It checks to see if the cube is empty, i.e. all outputs are 0. It returns an appropriate boolean value.

- **SetOutputs.**
  
  This procedure takes as input a PLA specification and two boolean 'flags'. It
modifies the outputs of the PLA specification according to the following 'flag' combinations:

(i) FALSE, FALSE - All outputs in the specification are set to 0. (ii) TRUE, FALSE - All outputs are set to 1. (iii) FALSE, TRUE or TRUE, TRUE - All outputs are complemented.

- OutCubed.
  This function takes as input two PLA specifications and two integers. The first integer indicates the location of a cube in the first specification, and the second integer the location of a cube in the second specification. The function checks whether or not the cube of the first specification covers that of the second, returning an appropriate boolean value.

5.5.3. Enhancement Functions and Procedures

- IdentCubes.
  This function takes as input two PLA specifications and two integers. The first integer indicates the location of a cube in the first specification, and the second integer the location of a cube in the second specification. The function checks whether or not the inputs of the two cubes are identical, returning an appropriate boolean value.

- DeleteOutputs.
  This procedure takes as input a PLA specification and two integer inputs. The integers specify the locations of cubes in the PLA specification. Anywhere in the output of the cube indicated by the second integer that there is a 1, the function places a 0 in the corresponding output of the cube indicated by the first integer. Essentially, all connected outputs of the second cube are removed from the first.

- OneSpot.
  This function takes as input a PLA specification and two integers. Each integer
indicates the location of a cube in the specification. The function checks whether or not the two cubes have opposing input values for exactly one literal (ie. \( x_j = 0 \) for one cube, and \( x_j = 1 \) for the other). It returns an appropriate boolean value, and passes a parameter indicating the variable of adjacency.

- **CubeMerge.**

  This function takes as input two PLA specifications and an integer. The integer indicates the location of a cube in the second specification that is to be added to the first specification according to the following:
  
  (i) If the cube to be added is covered by a cube already in the first specification, then it is not added. If the cube to be added is covered in the inputs, but not all outputs, by a cube already in the specification, then it is appended with the covered outputs set to 0.
  
  (ii) If the cube to be added is appended to the first specification, and it covers any cubes in the first specification, then these cubes are removed. If it covers any cubes already in the first specification in the inputs, but not all outputs, then those outputs that it does cover are set to 0 in these cubes.

- **Simplify.**

  This procedure takes as input a PLA specification and removes all covered cubes.

- **Reduce.**

  This procedure takes as input a PLA specification, recursively expands all completely adjacent cubes, and removes all covered cubes, until no further changes are made in the PLA specification.

### 5.5.4. Pronto Functions and Procedures Used.

The following is a list of the Pronto functions called by the fast complementation algorithm. An explanation of the workings of these functions can be found in [ABW].
- **NewLogicArray.**
  Creates memory space for a new PLA specification.

- **DisposeLogicArray.**
  Releases to memory a PLA specification.

- **CopyLogicArray.**
  Makes a copy of a PLA specification.

- **Intersection.**
  Takes the intersection of two PLA specifications.

- **InArray.**
  Inputs a PLA specification from a file.

- **SPAMOutArray.**
  Outputs a PLA specification to a file.
Chapter 6
Timing Studies

6.1. Timing Considerations

All program runs and timing were done on the Oregon State University research VAX/750. The clock used was the system clock.

A major problem was the separation of load factors from actual program run time. Depending on the load factor, certain system functions, such as swapping processes in and out, varied in time, and produced a corresponding variation in the run times obtained for the PRONTO programs, even for identical programs run with identical inputs at different times. Variations ran to 10% and more of total run time.

Due to these variations in time, versions of the PRONTO program to be compared were run with the same inputs as close to simultaneously as possible, i.e. in the background of a Berkeley UNIX (tm Bell Labs) operating system. So, while the results obtained are good for comparison purposes, they should not be taken as absolutes.

6.2. Initial Implementation

The graphs on the next four pages show the results of running a new PRONTO program using the technique of intersection with complement (as discussed, without any enhancement functions) against the old version of PRONTO. A full table of the results is also given in Figure 6.5. The following conclusions can be drawn from these results:
Figure 6.1

Run time comparison of PRONTO versions for input PLA functions of between 0 and 40 product terms.

+ - old version of PRONTO
x - new version of PRONTO
Figure 8.2

Run time comparison of PRONTO versions for input PLA functions of between 49 and 100 product terms.

+ = old version of PRONTO
x = new version of PRONTO
Figure 6.3

Run time comparison of PRONTO versions for input PLA functions of between 100 and 150 product terms.

+ - old version of PRONTO
x - new version of PRONTO
Figure 6.4

Run time comparison of PRONTO versions for input PLA functions of between 150 and 500 product terms.

+ - old version of PRONTO
x - new version of PRONTO
Figure 6.5. Results of two PRONTO versions for identical input.

<table>
<thead>
<tr>
<th>no. of input terms</th>
<th>old PRONTO</th>
<th>new PRONTO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>calculation time</td>
<td>result terms</td>
</tr>
<tr>
<td>22</td>
<td>966</td>
<td>12</td>
</tr>
<tr>
<td>24</td>
<td>1113</td>
<td>14</td>
</tr>
<tr>
<td>30</td>
<td>1927</td>
<td>13</td>
</tr>
<tr>
<td>33</td>
<td>1695</td>
<td>18</td>
</tr>
<tr>
<td>33</td>
<td>1128</td>
<td>19</td>
</tr>
<tr>
<td>35</td>
<td>1183</td>
<td>13</td>
</tr>
<tr>
<td>35</td>
<td>1373</td>
<td>30</td>
</tr>
<tr>
<td>37</td>
<td>1879</td>
<td>32</td>
</tr>
<tr>
<td>38</td>
<td>2987</td>
<td>23</td>
</tr>
<tr>
<td>38</td>
<td>427</td>
<td>14</td>
</tr>
<tr>
<td>49</td>
<td>6784</td>
<td>32</td>
</tr>
<tr>
<td>56</td>
<td>11315</td>
<td>39</td>
</tr>
<tr>
<td>64</td>
<td>7800</td>
<td>48</td>
</tr>
<tr>
<td>64</td>
<td>1674</td>
<td>28</td>
</tr>
<tr>
<td>68</td>
<td>1577</td>
<td>32</td>
</tr>
<tr>
<td>76</td>
<td>11748</td>
<td>75</td>
</tr>
<tr>
<td>91</td>
<td>12353</td>
<td>35</td>
</tr>
<tr>
<td>106</td>
<td>15860</td>
<td>105</td>
</tr>
<tr>
<td>137</td>
<td>19954</td>
<td>105</td>
</tr>
<tr>
<td>173</td>
<td>53531</td>
<td>56</td>
</tr>
<tr>
<td>236</td>
<td>29390</td>
<td>59</td>
</tr>
<tr>
<td>481</td>
<td>167445</td>
<td>253</td>
</tr>
</tbody>
</table>

all times in 1/60 CPU sec.
For small PLA functions, ie. < 40 terms, the time difference in calculation time is small. For several cases this difference was much less than 1 CPU second. Either implementation is therefore adequate.

In general, as the number of terms in the PLA function grew, the new version of PRONTO performed correspondingly better than the old. The savings in time cannot be meaningfully mapped to a curve, however, due to the dependence of the new version on characteristics of the PLA function.

As stated above, the new version was highly dependent on the characteristics of the PLA function. Specifically, PLA functions with many input connections in the AND plane were reduced with a savings in time. However, PLA functions with sparse connections in the AND plane gave extremely poor results. The problem behind these results lies not with the nature of the complementation algorithm used, but with the utility intersection procedure provided with the old PRONTO program. This will be discussed in section 6.3.

The new version of PRONTO and the old did not always provide the same reduction. In particular, when differing numbers of product terms were obtained, the results were as given in Figure 6.6.

The reason the results were not always the same is due to the specific version of PRONTO modified to use the complementation algorithm.

### Figure 6.6
Differing reduction results.

<table>
<thead>
<tr>
<th># terms input</th>
<th># terms in reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>new PRONTO</td>
</tr>
<tr>
<td>38</td>
<td>24</td>
</tr>
<tr>
<td>64</td>
<td>27</td>
</tr>
<tr>
<td>481</td>
<td>232</td>
</tr>
</tbody>
</table>
As mentioned in chapter 4, the latest version of PRONTO (Abweh) had used a procedure, *Donts2*, to check the validity of expanded cubes. This procedure depended in part on characteristics of the sharp operation. It had been written in order to decrease the number of times the sharp operation needed to be performed. All comparisons providing data produced by the "old PRONTO" apply to this version of PRONTO (Abweh).

When the new version of PRONTO (Gilbert) was created, using the intersect with complement technique, a still older version of PRONTO (Dagit) was modified, since the code taking advantage of sharp operation characteristics was unnecessary. This older version, in conjunction with the modifications added to use the complementation algorithm, is referred to herein as the "new PRONTO" (Gilbert).

The "new" version of PRONTO (Gilbert) produces the same reduction as a version previous to the "old PRONTO" (Dagit, previous to Abweh. Addition of the complementation algorithm produces only time differences between PRONTO (Dagit) and PRONTO (Gilbert)). Therefore there is still some question as to whether the "old" version of PRONTO (Abweh) is better than previous versions (Dagit and earlier). In particular, were the savings in time when PRONTO was last updated enough to justify the large disparity in the third result in the above table? However, successful use of an intersect with complement technique would avoid this question.

**6.3. Sparse PLA Functions**

Sparse PLA functions are those functions characterized by a sparsity of connections in the AND plane. For the purposes of this thesis, the following guidelines were applied:

- Sparse PLA function - A PLA function with an average of less than 20% of the inputs connected in the AND plane per product term.
• Moderately sparse PLA function - A PLA function with an average of 21-40% of the inputs connected in the AND plane per product term.

• Average PLA function - A PLA function with an average of 41-59% of the inputs connected in the AND plane per product term.

• Moderately dense PLA function - A PLA function with an average of 60-79% of the inputs connected in the AND plane per product term.

• Dense PLA function - A PLA function with an average of 80% or more of the inputs connected in the AND plane per product term.

The calculation of the complement of sparse PLA functions caused a major downgrading in performance of the new PRONTO program. As can be seen in the table of Figure 6.7, this problem is significant enough to question the use of the entire intersect with complement technique. However, the problem appears to lie in the utility intersection procedure provided with the old PRONTO, and good results with non-sparse PLA functions suggests that a new utility intersection procedure should be added for use with the complementation algorithm.

The difficulty with sparse PLA functions lies in the fact that the cubes are large, with few input connections, and hence Rules 1-4 of the complementation algorithm, which depend highly on input patterns in the PLA function, will usually not be satisfied. This means that as the algorithm progresses, Rule 5 will be most frequently applied (see Figure 6.8). This greatly extends the run time of the complementation program for two

<table>
<thead>
<tr>
<th># terms input (sparse function)</th>
<th>time to perform reduction</th>
<th>new PRONTO</th>
<th>old PRONTO</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>2497</td>
<td>427</td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>24880</td>
<td>1577</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>270755</td>
<td>11748</td>
<td></td>
</tr>
</tbody>
</table>

Figure 6.7
Calculation time for sparse PLA functions
times given in 1/60 CPU sec.
reasons:

(1) The current utility intersection procedure provided with PRONTO automatically performs more than just an intersection. It also checks the result and reduces it by removing any unnecessary (covered) cubes. While this is necessary during PLA reduction, it is not at all necessary for PLA complementation. In fact, as will be shown in Section 6.4, reductions done during complementation greatly degrade the program’s performance (in some cases, as much as 70%).

(2) A major characteristic of PLA functions is that there are generally many more product terms than inputs. Rules 3 and 4 of the complementation algorithm reduce

---

**Figure 6.8**

**Intersection procedure breakdown for PLA functions of varying sparsity.**

The following figure shows the amount of time performing intersections as a percentage of total program time. The significance of this figure lies in the fact that intersection is only used in two places, the PLA function input procedure, and Rule 5 of the complementation program. Since the time spent in the intersection procedure during PLA function input is fairly constant for PLA functions of comparable sizes, the total time spent in the intersection procedure provides a good measure of how frequently Rule 5 of the complementation program must be invoked.

Below is a sample for a sparse PLA function of 38 input terms (S), a moderately sparse function of 38 input terms (MS), and a dense function of 37 input terms (D). Note that the time spent in the intersection procedure does not include helper functions invoked.

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>MS</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>12.78%</td>
<td>4.20%</td>
<td>1.73%</td>
</tr>
</tbody>
</table>

Percentages are a percentage of total run time.
the complexity of the problem by reducing the number of inputs that need be considered at each level of recursion. Since the number of inputs is generally few, a termination condition is quickly reached when these Rules are applied. Good results are therefore obtained for PLA functions which satisfy the conditions of these Rules. The major characteristic of these functions is a high density of connections in the AND plane.

Rule 5, on the other hand, reduces the complexity of the problem one product term at a time. For a large number of product terms, this is quite inefficient if cover checking and other non-complementation related reduction techniques are to be applied each time a product term is removed. As mentioned above, these reductions can degrade the program's performance by as much as 70%.

The solution to this problem in future work seems to be to eliminate all cover checking and other non-complementation related reduction techniques from the complementation section of the program when performing intersections. Straight intersection can done quickly, which is the basis of this entire thesis. Furthermore, intersection time can be make proportional to the number of product terms to be intersected. This allows two possible solutions to the problem of time spent in Rule 5. One, the intersection procedure can be speeded up sufficiently to no longer use such a significant portion of the program's run time. Two, the program could determine in advance if the amount of time spent in intersection will be too great (from the number of product terms in the PLA), and avoid complementation altogether. The former solution is better than the latter, and should be attempted first.

6.4. Addition Of Enhancement Functions To The Complementation Program

The enhancement techniques and functions discussed in sections 5.4 and 5.5.3 were added in the hope that by generating fewer terms in the complement the intersection in the PRONTO program would be faster, since intersection is done on a cube by cube
basis. The functions were added at the end of the complementation program in an attempt to reduce the size of the overall result.

Results were poor. As can be seen from the table of Figure 6.9, the additional overhead in reducing the complement of the PLA function to fewer product terms was so great as to override any future savings in the intersection procedure. These initial results strongly discourage any attempt to reduce the complement at the end, except for a one time removal of covered terms, which is reasonably fast. An alternative approach would be to attempt a reduction of terms at strategic points in the complementation program, using the characteristics known from application of the Rules of the complementation algorithm. One technique that is to be avoided at all costs is reduction of terms during the intersection evaluation procedure.

Figure 6.9.

<table>
<thead>
<tr>
<th>no. of input terms in F</th>
<th>unenhanced</th>
<th>enhanced</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>calculation time</td>
<td>terms in F'</td>
</tr>
<tr>
<td>22</td>
<td>149</td>
<td>14</td>
</tr>
<tr>
<td>24</td>
<td>147</td>
<td>19</td>
</tr>
<tr>
<td>30</td>
<td>160</td>
<td>30</td>
</tr>
<tr>
<td>33</td>
<td>188</td>
<td>31</td>
</tr>
<tr>
<td>37</td>
<td>218</td>
<td>44</td>
</tr>
<tr>
<td>49</td>
<td>262</td>
<td>57</td>
</tr>
</tbody>
</table>

all times in 1/60 CPU sec.
Chapter 7
Conclusion

7.1. Summary Of Thesis Results

Heuristic PLA reduction programs have been developed with the goal of achieving near optimal solutions in a reasonable amount of computer time. When time is a major concern, implementation techniques become important to the success of an algorithm put to practical use.

The PRONTO algorithm, by nature of its single pass reduction characteristic, has been shown to generate good results quickly [ABW]. Time-consuming portions of the program have been identified, and possible courses of improvement suggested. It has been shown in this thesis that addition of a complementation procedure to the program can achieve significant time savings. Furthermore, these savings grow as the size of the PLA function grows.

One major obstacle remains, however, before the use of the complement procedure can be expected to consistently generate better results. This is the strong dependence of the complement procedure on the characteristics of the PLA function. Sparseness in the AND plane of a PLA function can drive the time cost of computing the complement of a PLA function past the cost of the reduction itself. Two possible solutions to this problem were proposed in the previous chapter:

1. The utility function behind the large run time cost increase has been identified. It seems likely that modifications can be made to provide much faster results.

2. In the event that the utility intersection procedure run time cannot be improved enough to overcome the large number of intersection calls made during complementation of sparse PLA's, the PRONTO program can be altered to take a measure of the sparseness of the PLA at the beginning of the program. From this, a decision
on whether or not to complement can be made.

Further modifications to the PRONTO algorithm using the complement of the input PLA function can gain more time saving, and some possibilities will be discussed in the next section. The program can become more tolerant to the necessary overhead of generating the complement.

In sum, while the use of the complement of the input PLA function as a viable alternative in the PRONTO program is not yet profitable, the problems to be overcome have been identified, and the potential of the technique demonstrated.

7.2. Suggestions For Future Work

Now that it has been shown that a complement procedure can provide time savings in the PLA reduction program, the next step is to make the procedure less vulnerable to the characteristics of the PLA function. The major modification that needs to be made is to the utility intersection procedure. Elimination of cover checking during intersection (when generating the PLA complement) would greatly increase the efficiency of the program.

A second area of possible improvement is in the PRONTO validation process for expanded cubes. As stated in section 4.4, a valid expanded cube is found originally with all outputs set to 1. Then the outputs are iteratively set to don't care, and the cube is again tested during each such iteration. Setting an output to don't care enlarges a cube, and, if the new cube is still valid, this larger cube is preferable. The test performed is to intersect with the complement of the PLA function.

Instead of iteratively setting each don't care in the output of the expanded cube to 1 and intersecting with the complement of the PLA function, an analysis could be made of the result of a first intersection with all output don't cares in the expanded cube set to 1. If the result is non-empty, but only has 1's in output columns that correspond to
columns in which the outputs of the expanded cube were originally don't care, then the largest valid expanded cube is the one with these outputs set to don't care and the rest to 1.

One last improvement to be made concerns the manner in which the intersection itself is performed. Since intersection is done on a cube by cube basis, the fewer cubes, the faster the intersection. Because the complementation procedure breaks the PLA function into sub-problems of known characteristics (recursively complementing smaller and smaller PLA functions) it might be possible to optimize the overall complement generated by optimizing the complement of each smaller PLA function with respect to number of cubes at strategic points in the complementation procedure (for example, when the recursion termination rules are invoked). If this leaves fewer cubes in the complement of the original PLA, then intersection between it and expanded cubes later in the PRONTO program will be faster.
Bibliography


