Sum-of-products Minimization, Shannon Expansion etc.

Madhav P. Desai

January 25, 2018

Most of the material in this handout is covered in great detail in the book by Kohavi. Find it and read it!

1 The tabular method for the minimization of a function

We start with the min-terms, identify the prime implicants, and generate a cover for the function using the smallest number of literals.

Suppose we are given the following function of 3 variables:

index	x2	x1	x0	f
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

The min-terms of the function are then $\{2, 3, 4, 5, 6, 7\}$.

We can generate the 1-cubes by combining pairs of min-terms (note that two terms can be combined only if their representations differ in exactly one bit position):

```
(2,3)     0 1 _      1
(2,6)     _ 1 0      1
(3,7)     _ 1 1      1
(4,5)     1 0 _      1
(4,6)     1 _ 0      1
(5,7)     1 _ 1      1
```

Next, we generate 2-cubes by combining 1-cubes:

```
(2,6,3,7)      _ 1 _       1 0
(4,5,6,7)      1 _ _       1 0
```

Thus, there are two prime implicants, both of which are essential. The function f can be represented by the formula $x_1 + x_2$.

Remark: This approach to the minimization of a sum-of-products formula has a problem, in that one needs to explicitly list the min-terms of the function before beginning the minimization. There are many functions which have a large number of minterms but very small formulas. Thus, the tabular approach is typically applicable only for functions with a small number of variables (about 10 or so).

2 Adapting the tabular method towards the joint minimization of two functions

The tabular method can be adapted to the joint minimization of two (or more) functions. The hope here is that if we find common implicants for the two functions, then the total number of gates needed to implement the two functions together may be reduced.

I will illustrate this extension by an example: Suppose that you have to implement the two functions f and g of three variables (x_2, x_1, x_0) described as follows: f evaluates to 1 if the integer represented by the three bits is ≥ 2, and g evaluates to 1 if the integer represented by the bits is ≤ 5. The truth-table is then:

```
x2 x1 x0   f g
0  0  0    0 1
0  0  1    0 1
0  1  0    1 1
```

```
0  1  1     1 1
1  0  0     1 1
1  0  1     1 1
1  1  0     1 0
1  1  1     1 0
```

We start the tabular method in the usual way, except that each cube has an output-part as well. The 0-terms are then (the — is used to separate the input and output pars):

```
0  0  0  0 |   0   1
1  0  0  1 |   0   1
2  0  1  0 |   1   1
3  0  1  1 |   1   1
4  1  0  0 |   1   1
5  1  0  1 |   1   1
6  1  1  0 |   1   0
7  1  1  1 |   1   0
```

Now we combine two 0-terms to form a 1-term if they disagree on the input-part in exactly one-bit and if they have a common 1 in either the output-part for f or for g.

```
(0,1)    0 0 _  |   0 1
(0,2)    0 _ 0  |   0 1
(0,4)    _ 0 0  |   0 1
(1,3)    0 _ 1  |   0 1
(1,5)    _ 0 1  |   0 1
(2,3)    0 1 _  |   1 1
(2,6)    _ 1 0  |   1 0
(3,7)    _ 1 1  |   1 0
(4,5)    1 0 _  |   1 1
(4,6)    1 _ 0  |   1 0
(5,7)    1 _ 1  |   1 0
```

In combining the 0-terms to get these 1-terms, we mark those 0-terms which were combined into something larger in f_{ON} and g_{ON} respectively (marked by a "Y", the "_" means that that minterm is not needed for the function.).

```
0-term             f g
```

```
----------------------
0                   _ Y
1                   _ Y
2                   Y Y
3                   Y Y
4                   Y Y
5                   Y Y
6                   Y _
7                   Y _
----------------------
```

So in this case, all the 0-terms have been covered by 1-terms.

Continuing from the 1-terms, we can combine them to form two-terms as follows:

```
(0,1,4,5)     _ 0 _    |  0 1
(0,2,1,3)     0 _ _    |  0 1  (also (2,3,0,1))
(2,6,3,7)     _ 1 _    |  1 0
(4,5,6,7)     1 _ _    |  1 0
```

Thats it.. No further combinations are possible. So the final candidate-implicants are

```
(0,1,4,5)        _ 0 _    0 1
(0,2,1,3)        0 _ _    0 1
(2,6,3,7)        _ 1 _    1 0
(4,5,6,7)        1 _ 1    1 0
(2,3)            0 1 _    1 1   <- needed to cover g.
(4,5)            1 0 _    1 1   <- needed to cover f.
```

Note that $(2, 3)$ and $(4, 5)$ are included because they are not covered for both f and g while forming the 2-terms from the 1-terms.

Now the PI table. The PI's must cover all the min-terms in f and all the min-terms in g. Only those PI's with a 1 in the f-column should be considered for f, and only those with a 1 in the g-column should be considered for g.

```
                      f          |          g
               2  3  4  5  6  7  |   0  1  2  3  4  5
-------------------------------------------------------------------
(0,1,4,5)                        |   x  x        x  x
```

```
(0,2,1,3)                          x  x  x  x
(2,6,3,7)       x  x        x  x
(4,5,6,7)             x  x  x  x
(2,3)           x  x                       x  x
(4,5)                 x  x                          x  x
```

One covering is $(2,3,6,7),(4,5,6,7)$ for f and $(0,1,4,5),(0,1,2,3)$ for g (this uses four product terms and four literals). An alternative solution would be $(2,3),(4,5),(4,5,6,7)$ for f and $(2,3),(4,5),(0,2,1,3)$ for g. This also uses four products but needs six literals. The first solution is preferable in this case.

3 Iterated consensus

Often we are given an initial sum of products formula for a function (not the min-terms, but a sum of some products). Can we find the prime implicants of this function without listing the min-terms?

The method of iterated consensus is a purely algebraic technique for finding the prime-implicants of a sum-of-products formula. In this method, we use the equality:

$$x.A + \overline{x}.B = x.A + \overline{x}.B + A.B$$

where x, A, B are formulas. This introduces the additional term $A.B$ (called the consensus term) which may be a prime implicant or may combine with other terms to form prime implicants. The algorithm starts with a list of cubes L.

```
do {
  L = L union {consensus terms generated from L}
  Remove each element of L which is contained
    in some other element of L
} while (there-is-some-change)
```

Lets say we are given the following expression:

$$u.v.w + \overline{u}.w + \overline{v}.w + \overline{w}.\overline{u} + \overline{u}.x$$

The consensus terms generated out of this list of products are $v.w$, $u.w$, $v.w.x$, $\overline{u}$, $\overline{v}.\overline{u}$. Now several cubes can be dropped, and we are left with

$$v.w + u.w + \overline{u} + \overline{v}.w$$

From this list, we can generate the consensus cube w. Some more cubes can be dropped and we are left with

$$w + \overline{u}$$

No further change is possible. These are the prime-implicants.

Remark: This approach can also break down sometimes. There are functions with a small sum-of-products formula which can have a very large number of prime implicants. See [1] which has been uploaded to the website.

4 Shannon's expansion

Given a formula f in n variables $x_1, x_2, \ldots x_n$, Shannon's expansion of the formula around x_1 is:

$$f = x_1.f(1, x_2, \ldots, x_n) + \overline{x_1}.f(0, x_2, \ldots, x_n)$$

where $f(1, x_2, \ldots, x_n)$ is denoted by f_{x_1} and $f(0, x_2, \ldots, x_n)$ is denoted by $f_{\overline{x}_1}$.

Another way of writing this is

$$f = (x_1 \ ? \ f_{x_1} \ : \ f_{\overline{x}_1})$$

A multi-plexor is then a natural logic gate to be associated with Shannon's expansion.

To implement a formula, say $x_1.x_2 + x_3.x_4$ using Shannon's expansion, we can proceed as follows:

$$f = (x_1 \ ? \ u \ : \ v)$$

where $u = x_2 + x_3.x_4$ and $v = x_3.x_4$. Now

$$u = (x_2 \ ? \ 1 \ : \ v)$$

and

$$v = (x_3 \ ? \ x_4 \ : \ 0)$$

Thus, the function can be implemented using three multiplexors (note that we have reused v while implementing u).

5 Factoring and multi-level expressions

A formula f can be factored into an expression involving simpler formulas. For example

$$x_1.x_3 + x_1.x_4 + x_2.x_3 + x_2.x_4 + x_5.x_6 = (x_1 + x_2).(x_3 + x_4) + x_5.x6$$

Here we have written $f = g.h + r$ for suitable g, h and r.

There are two ways of factoring a formula. In Boolean factoring, we are given a formula f and a formula g, and we wish to express f in terms of g and the original variables on which f depends. For example, let $f = x_1.\overline{x_2} + \overline{x_1}.x_2$ and let $g = x_1 + x_2$. To express f in terms of g, x_1, x_2, we form a map of the function f:

```
  x1x2  00 01 11 10
g
0          d  d  d
1       d  1     1
```

The don't cares correspond to situations which are impossible. For example, if $x_1 = x_2 = 0$, then g cannot be 1. Minimize this (note that you should try to use g in the final expression) to obtain $f = g.(\overline{x_1} + \overline{x_2})$.

The second way of factoring a formula is to treat it as a polynomial. In this manner $x_1.x_2 + x_1.\overline{x_3} = x_1.(x_2 + \overline{x_3})$, but $x_1.\overline{x_2} + \overline{x_1}.x_2$ cannot be factorized. This is weaker, but much easier.

Given a set of formulas to be minimized, we try to identify sub-formulas which are common to several formulas (either as factors or as parts of a sum), implement these sub-formulas individually, and then use them wherever necessary.

6 Problem Set

1. The XNOR function is defined as $(a == b) = \overline{a}.\overline{b} + a.b$. Show, using only Boolean identities, that it can be implemented using just four two-input NOR gates. (Recall that $\overline{a + b} = \overline{a}.\overline{b}$).

2. Using 2-input NAND gates, design a full-adder (with three inputs a,b,cin, and two outputs s, cout) with the following truth-table:

a	b	cin	s	cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

3. Given $f = x_1 \oplus x_2 \oplus x_3$, and $g = x_1 + x_2 + x_3$, find an expression for f in terms of g, x_1, x_2 and x_3.

4. Using the method of iterated consensus, find the prime implicants of the formula

$$\overline{a}.\overline{c} + b.d + a.b.\overline{c} + a.b.d + b.\overline{c} + a.b.\overline{c}$$

5. Using 2-1 multiplexors, design a circuit that implements the formula

$$x_1.x_2 \oplus x_3.x_4 \oplus x_5.x_6 \oplus x_7.x_8$$

References

[1] A. Chandra, G. Markowsky, "On the number of prime implicants", Discrete Mathematics 24, 1978 pp. 9-11.