Sum of Products Algorithm

These notes describe the sum-of-products algorithm.

An example problem

Suppose we want an alarm signal to turn on in a car if the key is in the ignition and the door is open, or if the key is in the ignition and the seatbelt is not fastened. Suppose we introduce Boolean variables a, k, d, b that are 1 when the alarm is on, the key is in the ignition, the door is closed, and the seatbelt is fastened, respectively. Then a Boolean expression for a is

$$a = k \cdot (b' + d').$$

This states that a should be true when k is true and either or both of b and d are false.

A truth table for this function is as follows.

```
k  d  b  |  a
--------------
0  0  0  |  0
0  0  1  |  0
0  1  0  |  0
0  1  1  |  0
1  0  0  |  1
1  0  1  |  1
1  1  0  |  1
1  1  1  |  0
```

The line for `1 0 1` specifies that when the key is in the ignition ($k = 1$) and the door is not closed ($d = 0$) and the seatbelt is on ($b = 1$), the alarm should be on ($a = 1$).

Converting a truth table into a Boolean expression

The sum-of-products, or disjunctive normal form, algorithm converts any truth table for a Boolean function into a Boolean expression that represents the same function. We explain it using the example of the alarm function. Recall that the truth table for that function is as follows:

```
k  d  b  |  a
--------------
0  0  0  |  0
0  0  1  |  0
0  1  0  |  0
0  1  1  |  0
1  0  0  |  1
1  0  1  |  1
1  1  0  |  1
1  1  1  |  0
```

We consider a related function, f_1, that has a single 1 in its values column:

```
k  d  b  |  f1
-------------
0  0  0  |  0
0  0  1  |  0
0  1  0  |  0
0  1  1  |  0
1  0  0  |  1
1  0  1  |  0
1  1  0  |  0
1  1  1  |  0
```

Note that $f_1 = 1$ exactly when $k = 1$ and $d = 0$ and $b = 0$. Thus a Boolean expression for f_1 is,

$$k \cdot d' \cdot b'.$$

For each of the other two rows where a has a 1 in its value column, we can construct a Boolean expression that gives a 1 for just that row:

```
k  d  b  |  f2
-------------
0  0  0  |  0
0  0  1  |  0
0  1  0  |  0
0  1  1  |  0
1  0  0  |  0
1  0  1  |  1
1  1  0  |  0
1  1  1  |  0
```

For f_2 we have the Boolean expression: $k \cdot d' \cdot b$.

```
k  d  b  |  f3
-------------
0  0  0  |  0
0  0  1  |  0
0  1  0  |  0
0  1  1  |  0
1  0  0  |  0
1  0  1  |  0
1  1  0  |  1
1  1  1  |  0
```

For f_3 we have the Boolean expression: $k \cdot d \cdot b'$. If we now construct the "or" of these three expressions, we get an expression that has 1's in exactly the rows that a does:

```
k  d  b  |  f1 + f2 + f3
------------------------
0  0  0  |       0
0  0  1  |       0
0  1  0  |       0
0  1  1  |       0
1  0  0  |       1
1  0  1  |       1
1  1  0  |       1
1  1  1  |       0
```

Thus, we get another Boolean expression for the alarm signal:

$$(k \cdot d' \cdot b') + (k \cdot d' \cdot b) + (k \cdot d \cdot b').$$

Note that if we build a circuit directly from this expression, we end up using 10 gates instead of the 4 that we used for the more concise expression $k \cdot (b' + d')$. One of the uses of Boolean algebra is to try to derive equivalent and more concise expressions.

The sum-of-products algorithm generalizes this method in a straightforward way; for each row of the truth table that contains a 1 in the value column, form an and-expression (product) that depends on the values assigned to the variables in that row, and join all of those products in an or-expression (sum). The and-expression for a row contains each variable or its complement, where it contains the variable if its value in that row is 1, and contains the complement of the variable if its value in that row is 0. The resulting expression is the disjunctive normal form for the Boolean function; there is also a conjunctive normal form, which is an AND of OR's of variables and their negations.