Tautology Problem and Two Dimensional Formulas

Deniz Uyar

November 3, 2017

Abstract

Finding whether a boolean formula is a tautology or not in a feasible time is an important problem of computer science. Many algorithms have been developed to solve this problem but none of them is a polynomial time algorithm. Our aim is to develop an algorithm that achieve this in polynomial time.

In this article, we convert boolean functions to some graph forms in polynomial time. They are called two dimensional formulas and similar to AND-OR graphs except arches on them are bidirectional. Then these graphs are investigated to find properties which can be used to differentiate tautological formulas from non tautological ones.

Contents

Part I
From statement formulas to two dimensional ones

1 Statement formulas and tautology problem

1.1 Statement formulas

Boolean functions(or any functions in general) can be represented in various ways. In this section, we consider only the boolean functions given as a string consist of AND, OR, NOT signs, variable signs, constants 0 and 1 and parenthesis written as usual. We will use $\cdot$(point), $\wedge$ and~for the AND, OR, NOT signs and use the letters similar to x,y,z alone or with indices to represent variables.

Suppose the variables in a statement formula be $(x_1, .., x_n)$. If we assign either 0 or 1 value to each of the variables, then we get an interpretation of these variables. For example $(0, 0, 0)$ is an interpretation of the variables (x_1, x_2, x_3). Let represent a function as $f(x_1, .., x_n)$. If (x_1, x_2, x_3) is the list of the variables in a function, then the value of the function $f(x_1, x_2, x_3)$ becomes $f(0, 0, 0)$ for the interpretation $(0, 0, 0)$. If two functions have the same variables and gives the same results for every interpretation of their variables, then these are equal functions. Otherwise they are different functions. For example the functions $f_1(x, y) = x \cdot y$ and $f_2(x, y) = x \vee y$ are different because $f_1(0, 1) = 0$ and $f_2(0, 1) = 1$ true for the interpretation $(0, 1)$. The number of different functions for the n variables would be 2^{2^n}.

A boolean function is said a tautology if it gives the value 1 for every interpretation of its variables. For example 1 or $x \vee \widetilde{x}$ gives 1 for all the interpretation of a variable list. Therefore 1 and $x \vee \widetilde{x}$ functions are tautologies.

There are infinite number of different formulas representing the same function. For example $x \vee y$ is equal to $x \vee y \vee x \vee y..x \vee y$ where $x \vee y$ is duplicated as many times as desired. Then, the tautology checking problem may be expressed as "is a given formula a different writing of the function 1".

The most straightforward way of checking whether a formula is tautology or not is to evaluate the formula for each interpretations of the variables. The number of interpretation for n different variables is 2^n. The number of different variables in a formula is at most $s/2$ where s is the length of the formula. So the function must be evaluated $2s/2$ times. That is the complexity is $O(s2^{s/2})$. This is not polynomial. So we will investigate various ways to achieve this in polynomial time.

1.2 Variable reduction

Proposition 1 *$f(x_1, .., x_i, .., x_n)$ is a tautology if and only if $f(x_1, .., 0, .., x_n) \cdot f(x_1, .., 1, .., x_n)$ is a tautology.*

Proof 1 *Trivial.*

According to this proposition, from a formula, we can obtain another one which has one less variable and still preserves the tautological property of the original one. This is a variable reduction operation and can be illustrated as

$f(x_1, .., x_i, .., x_n) \xrightarrow{x_i} f(x_1, .., 0, .., x_n) \cdot f(x_1..1..x_n)$

Let call this v-reduction.

All the variables of a boolean formula can be eliminated by a series of v-reductions. After simplification, what would be in hand at last, is either a 1 or 0. Since each v-reduction preserves the tautological property of the original formula, a 1 at hand means the original formula is a tautology an 0 means it is not a tautology.

The obvious way of doing reduction is to get two formulas r_1 and r_2 as follows; replace x_i with 0 in the initial formula to obtain r_1 and replace x_i with 1 in the initial formula to obtain r_2, parenthesise them if necessary and put an AND sign between them. Also all the 1's and 0's are removed from the formula by applying the identities

$0 \cdot x = 0, 1 \cdot x = x, 0 \cdot x = x, 1 \cdot x = 1, \widetilde{0} = 1, \widetilde{1} = 0$.

All of these process takes polynomial time. However the length of $(r_1) \cdot (r_2)$ which is the resulting formula would be $2s$ approximately in the worst case. Doing this for $s/2$ variables would give us a complexity of $O(s \cdot 2^{s/2}$, so it seems that this process could not be achieved in polynomial time.

1.3 Simple form and reduction

A variable, the inverse of a variable and the constants 0,1 are called atomic terms.

If a formula is in the form $a_1 \vee a_2 \vee .. \vee a_n$ then it is a disjunction. Each a_i is an element of it. If all the elements of a disjunction is an atomic term then this is a simple sum. For example $x_1 \vee \widetilde{x_1}$ is a simple sum.

If a formula is in the form $a_1 \cdot a_2 \cdot .. \cdot a_n$ then it is a conjunction. Each a_i is an element of it. If all the elements of a conjunction is an atomic term then this is a simple product. For example $x_1 \cdot \widetilde{x_1}$ is a simple product.

If all the elements of a disjunction are simple products, then the formula is said to be in the disjunctive normal form. Each boolean formula has a disjunctive normal form or DNF. In this form, each element of the form is called an elementary product or e-product and either contain x_i or $\widetilde{x_i}$ or neither contain x_i nor $\widetilde{x_i}$ as follows.

$x_i \cdot \alpha_1 \vee .. \vee x_i \cdot \alpha_n \vee \widetilde{x_i} \cdot \beta_1 \vee .. \vee \widetilde{x_i} \cdot \beta_m \vee \delta_1 \vee .. \vee \delta_l$

Here δ_k is the elementary product including neither x_i nor $\widetilde{x_i}$. The elementary products which include both x_i and $\widetilde{x_i}$ are either $x_i \cdot \alpha_i$ or $\widetilde{x_i}.\beta_j$.

This formula can be written simply as

$$x_i \cdot \alpha \vee \widetilde{x_i} \cdot \beta \vee \delta$$

where $\alpha = \alpha_1 \vee .. \vee \alpha_n, \quad \beta = \beta_1 \vee .. \vee \beta_m$ ve $\delta = \delta_1 \vee .. \vee \delta_l$
If α includes x_i and/or $\widetilde{x_i}$, then they can be eliminated by the following equivalences.

$$\begin{aligned} x_i \cdot \alpha(x_i) &= x_i \cdot \alpha(1) \\ x_i \cdot \alpha(\widetilde{x_i}) &= x_i \cdot \alpha(0) \end{aligned}$$

Similarly if any β_j includes x_i or $\widetilde{x_i}$, they are eliminated by

$$\begin{aligned} \widetilde{x_i} \cdot \beta(x_i) &= \widetilde{x_i} \cdot \beta(0) \\ \widetilde{x_i} \cdot \beta(\widetilde{x_i}) &= \widetilde{x_i} \cdot \beta(1) \end{aligned}$$

Thus, the terms α, β, δ do not include x_i or $\widetilde{x_i}$. This is called as "simple form". Now let see the reduction in this form.

$$\begin{aligned} x_i \cdot \alpha \vee \widetilde{x_i} \cdot \beta \vee \delta &\xrightarrow{x_i} (\beta \vee \delta) \cdot (\alpha \vee \delta) \\ &= \beta \cdot \alpha \vee \delta \end{aligned}$$

After this point on, we will assume the NOT operator is applied to the variable symbols only. If a formula is not in this form, then an equivalent formula in this form can be obtained in polynomial time.[2]

The DNF form of a given formula can be produced by applying the rule $x \cdot (y \vee z) = x \cdot y \vee x \cdot z$ to the formula recursively. Then the right hand side of the v-reduction could be obtained by examining each elementary product to mark them as α_i,β_j or δ_k and by doing the necessary operations to get $\alpha \cdot \beta \vee \delta$. However, the number of elementary products is not polynomial wrt n which is the number of different variables. Therefore, obtaining all the elementary products in this way dos not take polynomial time.

So let investigate the other ways of reducing the formula. It can easily be seen that, if a formula does not have x_i but includes $\widetilde{x_i}$, then the simple form would be $\widetilde{x_i} \cdot \beta \vee \delta$ and the reduction becomes as

$$\begin{aligned} \widetilde{x_i} \cdot \beta \vee \delta &\xrightarrow{x_i} (\beta \vee \delta) \cdot \delta \\ &= \delta \end{aligned}$$

But this can be achieved simply by putting 0 instead of $\widetilde{x_i}$ and eliminating 0's from the formula.

$$\begin{aligned} \widetilde{x_i} \cdot \beta \vee \delta &\xrightarrow{x_i} 0 \cdot \beta \vee \delta = \delta \\ \Rightarrow f(\widetilde{x_i}) &\xrightarrow{x_i} f(0) \end{aligned}$$

This process can be applied to the original formula in polynomial time. Similarly for the formulas which does not include $\widetilde{x_i}$ but having x_i, reducing is achieved as

$$\begin{aligned} x_i \cdot \alpha \vee \delta &\xrightarrow{x_i} (\alpha \vee \delta) \cdot \delta = \delta \\ \Rightarrow f(x_i) &\xrightarrow{x_i} f(0) \end{aligned}$$

Again the reduced form can be obtained from the original one by replacing x_i with 0 and eliminating all 0s

Thus any variable which occurs either only alone or only with negation can be eliminated by just replacing them with 0. This takes polynomial time. As a result, a formula is obtained in which all the variables occur at least twice; one as alone and the other with negation. Now if we evaluate the formula by examining every interpretations of them, the complexity would be $O(s.2^{s/2})$ since the number of different variables is $s/4$ at most in such a formula.

In the rest of this section we will assume all the variables are not alone. If they are not in this form, they can be changed to this form in polynomial time as mentioned above.

1.4 AND terms

Consider the v-reduction again.

$f(x_i, \widetilde{x_i}) = x_i \cdot \alpha \vee \widetilde{x_i}.\beta \vee \delta \xrightarrow{x_i} \beta \cdot \alpha \vee \delta$

Here the question is that "are there alternate ways of obtaining the right hand side of the reduction from the left hand side".

The answer is yes and as shown below, the formula can be reduced by replacing x_i with β_i and $\widetilde{x_i}$ with 0.

$$\begin{aligned} x_i \cdot \alpha \vee \widetilde{x_i}.\beta \vee \delta \quad &\xrightarrow{x_i} \quad \underset{\beta}{\not{x_i}} \cdot \alpha \vee \underset{0}{\widetilde{\not{x_i}}}.\beta \vee \delta \\ &= \beta \cdot \alpha \vee \delta \end{aligned}$$

The same result can be obtained by replacing x_i with 0 and $\widetilde{x_i}$ with α,

$$\begin{aligned} x_i \cdot \alpha \vee \widetilde{x_i}.\beta \vee \delta \quad &\xrightarrow{x_i} \quad \underset{0}{\not{x_i}} \cdot \alpha \vee \underset{\alpha}{\widetilde{\not{x_i}}}.\beta \vee \delta \\ &= \beta \cdot \alpha \vee \delta \end{aligned}$$

or by replacing x_i with β and $\widetilde{x_i}$ with α.

$$\begin{aligned} x_i \cdot \alpha \vee \widetilde{x_i}.\beta \vee \delta \quad &\xrightarrow{x_i} \quad \underset{\beta}{\not{x_i}} \cdot \alpha \vee \underset{\alpha}{\widetilde{\not{x_i}}}.\beta \vee \delta \\ &= \beta \cdot \alpha \vee \delta \end{aligned}$$

α and β are just the terms conjuncted with x_i and $\widetilde{x_i}$ respectively. So, this idea can be applied to any formula even it is not in DNF form.

$$\begin{aligned} f(x_i, \widetilde{x_i}) &\xrightarrow{x_i} f(\beta, 0) \\ f(x_i, \widetilde{x_i}) &\xrightarrow{x_i} f(0, \alpha) \\ f(x_i, \widetilde{x_i}) &\xrightarrow{x_i} f(\beta, \alpha) \end{aligned}$$

Definition 1 *If two terms t_1 and t_2 are conjuncted then t_2 is an A-term of t_1 and vice versa*

For example in $(x \vee y) \cdot z$, $(x \vee y)$ is an A-term of z and z is an A-term of $(x \vee y)$.

Then the idea "replace x_i with β" becomes "replace each occurrence of x_i with the A-term of $\widetilde{x_i}$". Similarly, the idea "replace $\widetilde{x_i}$ with α" becomes "replace each occurrence of $\widetilde{x_i}$ with the A-term of x_i"

1.5 AND links and reachability

Let try to find the and-terms of x_i without changing the formula.

In any formula, each occurrence of x_i in a formula is in the following form.

$..l_m \cdot (u_k \vee .. \vee l_2 \cdot (u_i \vee l_1 \cdot x_i \cdot r_1 \vee u_{i-1}) \cdot r_2 \vee .. \vee u_1) \cdot r_n \vee ..$

The term which is directly conjuncted with x_i is $l_1 \cdot r_1$. If the parenthesis are opened then the A-term of x_i for this occurrence would be $l_m..l_2 \cdot l_1 \cdot r_1 \cdot r_2..r_n$. Let this be α_k where k represents the k'th occurrence of x_i. If α_k contains x_i or $\widetilde{x_i}$, then they can be eliminated by replacing them with 1 and 0 respectively, since α_k is an A-term of x_i . That is the equivalences $\alpha(x_i) = \alpha(1)$ and $\alpha(\widetilde{x_i}) = \alpha(0)$ are applied. The entire A-term of x_i then becomes $\alpha = \alpha_1 \vee .. \vee \alpha_n$ where n is the number of occurrences of x_i in the formula.

One algorithm to do this is the following.

Algorithm 1 *Find an occurrence of x_i. Do the followings for this occurrence.*

1. *Let T be an empty set and x_i is the last A-term.*

2. *Do the following steps toward right until the end of the formula*

   (a) *If t_1 is the last A-term and the formula is in the form $..t_1 \cdot t_2..$ then make t_2 the last A-term and put it into T.*

   (b) *If the formula is in the form $..(.. \vee t \cdot t_i \vee .. \vee t_n) \cdot r..$ and t_i is the last A-term, then make r the last A-term and put it into T.*

3. *Starting from x_i, do the following steps toward left until the end of the formula.*

   (a) *If t_2 is the last A-term and the formula is in the form $..t1 \cdot t2..$ then make t_1 the last A-term and put it into T.*

   (b) *$f\ t$ is the last A-term and the formula is in the form $..l \cdot (t1 \vee .. \vee t \cdot t_i \vee .. \vee t_n)..$, then make l is the last A-term and put it into T.*

The A-term of the this occurrence would be $a_1 \cdot .. \cdot a_n$ where $a_i \in T$

In this case, the A-term of x_i, i.e. α, is the disjunction of the A-terms of all the occurrences of x_i. If the number of occurrences of x_i is n and the length of the formula is s, this operation takes ns times. If there are x_i or $\widetilde{x_i}$ terms in α, then they are eliminated by the equations $\alpha(x_i, \widetilde{x_i}) = \alpha(1, 0)$. β is found similarly. The term δ can be found by replacing x_i and $\widetilde{x_i}$ with 0. All of these jobs take polynomial time. So the reduced formula $\alpha \cdot \beta \vee \delta$ can be found in polynomial time. However, the length of the formula found may be larger than the original one. If the length of the formula found is $2s$ in average then the length seems to increase exponential in each reduction. On the other hand, the number of different variables would be reduced by one and thus the length can not increase so much. From here, one may prove that the length can not increase exponentially. However, we will assume that this process can not be realized in polynomial time.

The algorithm above replaces the concept of finding an A-term with the concept of going from one term to another. To visualise this concept, we will use $-$ symbol instead of $\cdot$. By this notation, a statement formula would look like

$..l_m - (u_k \vee .. \vee l_2 - (u_i \vee l_1 - x_i - r_1 \vee u_{i-1}) - r_2 \vee .. \vee u_1) - r_n \vee ..$

Thus $-$ represents both reachability and A-links. So, two terms are conjuncted to each other if they are reachable to each other by the following rules.

1. In $t_1 - t_2$, from t_1 one can reach to t_2 and vice versa.
2. In $(t_1 \vee t_2) - t_3$, t_3 is reached from t_1 or from t_2 and from t_3, t_1 or t_2 can be reached.
3. If t_2 is reachable from t_1 and t_3 is reachable from t_2, then t_3 is reachable from t_1.

So, if t_2 is reachable from t_1 then t_2 is an A-term of t_1.

1.6 Reachability and reduction

Now consider the v-reduction again.

$$\begin{aligned} x_i - \alpha \vee \widetilde{x_i} - \beta \vee \delta &\xrightarrow{x_i} (\beta \vee \delta) - (\alpha \vee \delta) \\ &= \beta - \alpha \vee \delta \end{aligned}$$

Note that after replacement α becomes conjuncted to β and vice versa. To make two term conjuncted, it is sufficient to made them reachable from each other. So the idea can be realized by putting an A-link between α and β as shown below.

$$x_i - \alpha \vee \widetilde{x_i} - \beta \vee \delta \xrightarrow{x_i} 1 - \alpha \vee 1 - \beta \vee \delta$$

This path make α and β reachable to each other and thus they become A-terms of each other. A better realization of this is as follows

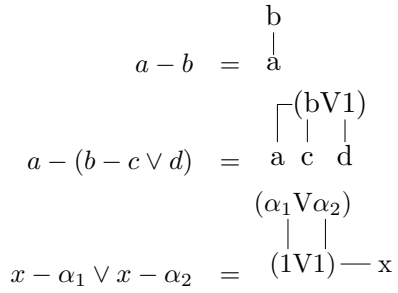
$$x_i - \alpha \vee \widetilde{x_i} - \beta \vee \delta \xrightarrow{x_i} 1 - \alpha \vee 1 - \beta \vee \delta$$

α and β become A-term of each other via 1's linked to them.

2 Two Dimensional Formulas

2.1 Two dimensional formulas

Once we escape from the restrictions of one dimension and feel free to draw links in two dimensions, then the amount of choices to express a function will be increased. Consider the following examples.



Two dimensional formulas can easily be obtained from statement formulas as shown in the figure.

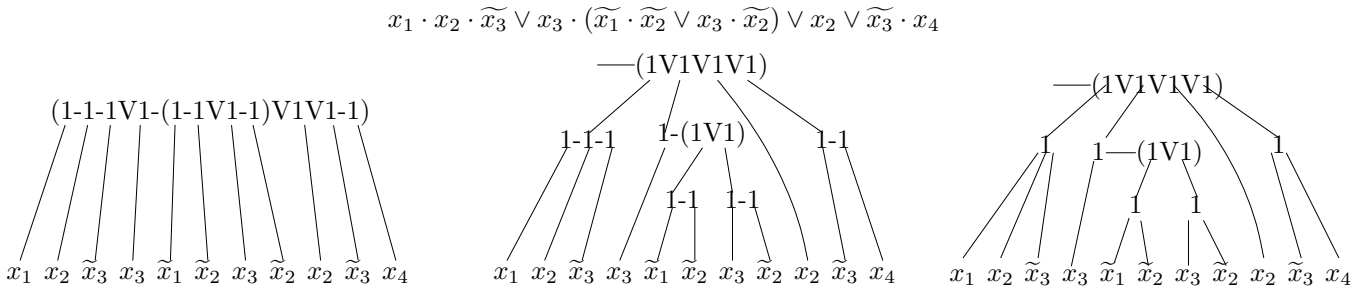


Figure 1: Two dimensional equivalences of a statement formula

The third form is a tree structure composed of arches, the nodes $-1-$, $(1V..V1)$ and atomic terms . The first node is an AND-node and the second one is an OR-node. The OR-node has a main arc and several base arches. Every formula has a root arc.

To show that the above equivalences are true, it is enough to show that the statement formula and the corresponding two dimensional forms takes the same value at every interpretation of the variables.

In the following example, an interpretation of the formula and the value of each arc for this interpretation is shown. Naturally the arches has directions. The value of each arc is evaluated as follows: If an arc is connected to a variable then the value of arc becomes the value of that variable. For an OR-node, if one of the base arches is 1 then the main arc becomes 1. Otherwise it becomes 0. For an AND-node if one of the outgoing arches is 0, then the incoming arc becomes 0. Otherwise it becomes 1. For this interpretation the root arc takes the value 0, so this means that the value of this formula is 0 for this interpretation.

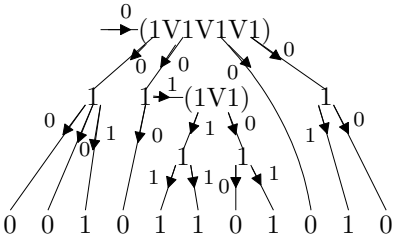


Figure 2: The values of the arches for the interpretation $x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 0$

Arces, nodes and formulas Consider the formula given in the figure. In it, all the arches are labelled to differentiate them.

The arches are denoted by small letters. A label near an arc normally represents the arc in the downward direction. If an arc is in the left-right direction, then the label near it denotes the arc directed to right. The reverse of an arc is shown

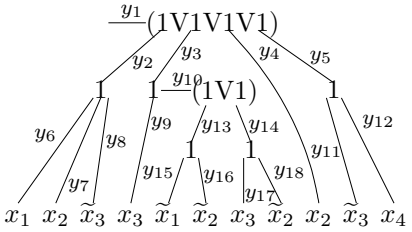


Figure 3: Labeling arches

by putting an $'$ sign above the label of it. For example the reverse of the arc y is written as y'. The reverse of the reverse of an arc is the arc itself.

Each arc is either tied to an atomics or an AND-node or an OR-node. If an arc p is connected to an atomic term, show this as px where x is the atomic term, if it is connected to an AND-node, $p-(r_1,..,r_n)$ where $r_1,..,r_n$ are outgoing arches, if it is tied to an OR-node, show this as $p-(p_1 \vee .. \vee p_n)$ where $r_1,..,r_n$ are the outgoing arches.

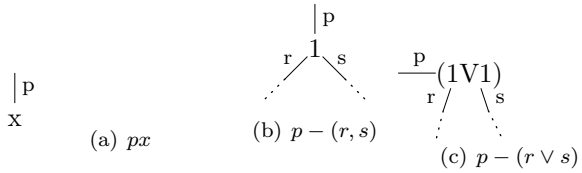


Figure 4: Connection types

For any arc p, let $|p|$ be represent the value of the arc in any interpretation. In this case, the relationship among the values of the arches be as the following.

1. For px $|p| = |x|$
2. For pr $|p| = |r|$
3. For $p-(r_1,..,r_n)$, $|p| = |r_1| \cdot .. \cdot |r_n|$ is true.
4. For $p-(r_1 \vee .. \vee r_n)$, $|p| = |r_1| \vee .. \vee |r_n|$ is true.

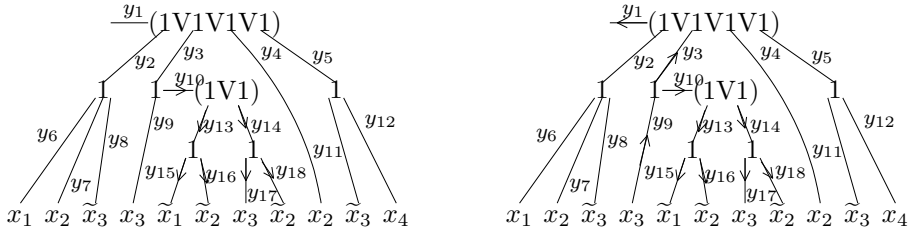
Formula of an arc Each arc represents a formula and this is the formula which affects the value of this arc only. If an atomic term can affect the value of an arc, then this means that there is a path from the arc to that atomic term. I.e. the formula of an arc is the subformula which is found by following the links beginning from that arc.

To find the formula of an arc the following rules are applied.

Algorithm 2 *This algorithm finds the formula for the arc p.*

1. *push the arc p to the stack.*
2. *Pop an element from the stack. Let this be p.*
   (a) *If p and beyond is in the form $p-(p_1,..,p_n)$, then push the arches $p_1,..,p_n$ to the stack.*
   (b) *If p and beyond is in the form $p-(p_1 \vee .. \vee p_n)$, then push the arches $p_1,..,p_n$ to the stack.*
   (c) *If p and beyond is in the form px, then do nothing.*
3. *The scanned subgraph is the formula of the arc p.*

For example in the figure, the subformula beginning from the arc y_{10} and y'_9 is shown. However, as seen, there are some problems here. For example, in the subformula of y'_9, the values of the arches y'_1 and y'_3 are undetermined and therefore the value of the arc y'_9 is unknown.



(a) the formula of y_{10} (b) the formula of y'_9

Figure 5: Each arc represents a formula

2.2 And terms

To find the A-term of a variable, we can adapt the rules used in statement formulas. Suppose $(p_1 \vee .. \vee p_n) - r$ represents an OR-node where $p_1, .., p_n$ are incoming and r is outgoing arches. That is $p - (p_1, .., p_n)$ and $(p'_1, .., p'_n) - p'$ represent the same OR-node. However, in the first, the direction is from the main arc to the base arches while in the second the direction is from base arches to the main arc. The following algorithm finds the A-term of x_i.

Algorithm 3 *1. Let T is an empty set and x_i is the last a-term.*

2. Put the reverse of the arc connected to x_i to T.

3. Do the followings until stack is empty.

(a) Pop an arc from the stack. Let this be p.

(b) If p and beyond is in the form $p - (r_1, ..r_n)$ then push $r_1, .., r_n$ to the stack.

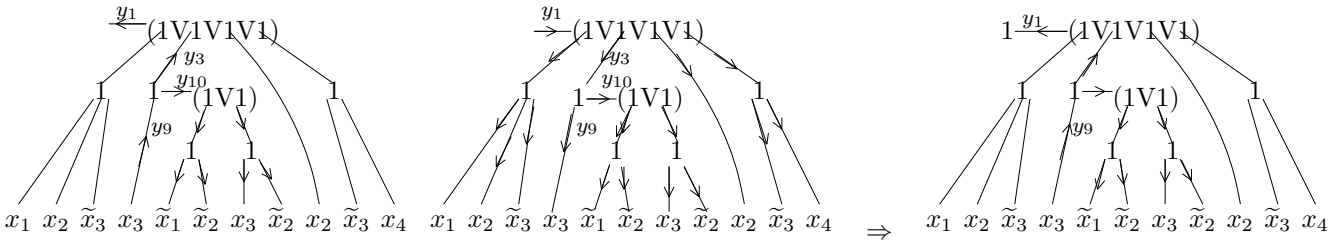
(c) If p and beyond is in the form $p - (r_1 \vee .. \vee r_n)$ then the subformula of p is an a-term. Traverse this subformula.

(d) If p and beyond is in the form $(p \vee p_1 \vee .. \vee p_n) - r$ then put r into T.

(e) If p and beyond is in the form px, do nothing.

The a-term of this occurrence of x_i is the subformula traversed.

In fact, the formula of the reverse arc of the arc tied to a variable is an A-term of that variable. I.e. The subformula of y'_9 which is shown in the figure is in fact the A-term of an occurrence of x_3. Thus the problem of whether the value of the arc p in the connection $(p \vee p_1 \vee .. \vee p_n) - r$ is resolved: $|p|$ must be equal to $|r|$. I.e the value of y'_3 must be equal to the value of y'_1. On the other hand deciding whether the value of y'_1 is can be resolved as follows. We know that the value of the root arc, i.e. y_1, is equal the value of the formula and the arc y'_1 is not involved in the formula starting from y_1. Therefore the value of the reverse root arc has no affect on the value of the formula. On the other hand, y'_1 is a member of the A-term of x_3 and this A-term is a part of the formula. Any thing that has no affect on the value the formula can not affect the value of A-term also. Then the reverse root arc must be terminated with a 1.



(a) The subformula of y'_9 is an a-term of an occurrence of x_3

(b) The value of y_1 is the value of the formula. y'_1 is not a part of the formula traversed starting from the root arc

(c) To not affect the value of a-term, $|y'_1| = 1$ must be true

Figure 6: The arc y'_1 does not affect the value of the formula. Therefore it must not affect the value of any A-term

We will start the root arc with 1 after here on and call it root term. Therefore all the terminals of a formula would contain a atomics.

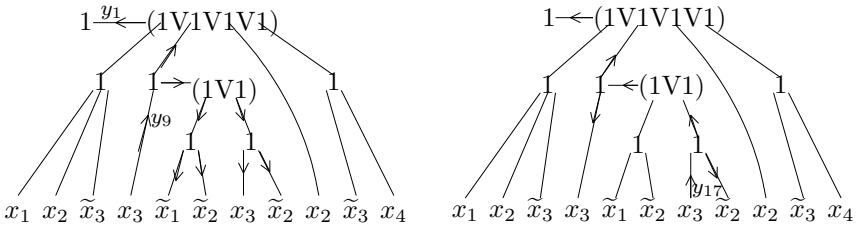
2.3 Reduction

The simple form of a statement formula is in the form $\alpha \cdot x_i \vee \beta \cdot \widetilde{x}_i \vee \delta$ where α is the A-term of x_i, β is the A-term of $\widetilde{x}_i$ and δ is the rest. In a statement formula, we can get the reduced form without converting it into simple form as follows:

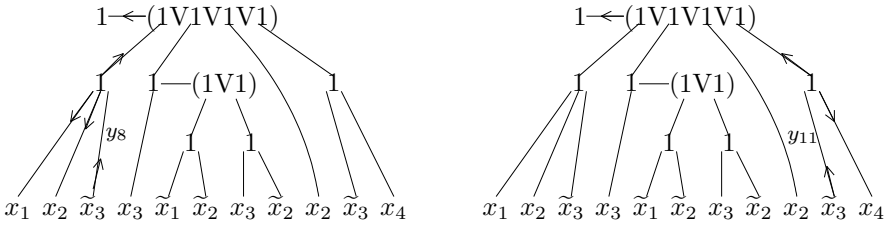
$f(x_i, \widetilde{x_i}) \xrightarrow{x_i} f(\beta, \alpha)$

So the reduction for x_3 is simple. replace every occurrence of x_3 with the and-term of $\widetilde{x_3}$ and replace every occurrence of $\widetilde{x_3}$ with the and-term of x_3.

The A-term of a variable then is the disjunction of all the and-terms of it. In the figure, the A-terms of each occurrences of x_3 and $\widetilde{x_3}$ are shown.



(a) the a-terms of the variable x_3



(b) The a-terms of the variable $\widetilde{x}_3$

Figure 7: The a-terms of x_3 and $\widetilde{x_3}$

So there are two ways to realise the reduction as shown. Thus in the reduced formula, the value of x_3 is replaced by the value of the A-term of $\widetilde{x_3}$ and the value of $\widetilde{x_3}$ is replaced by the value of the A-term of x_3.

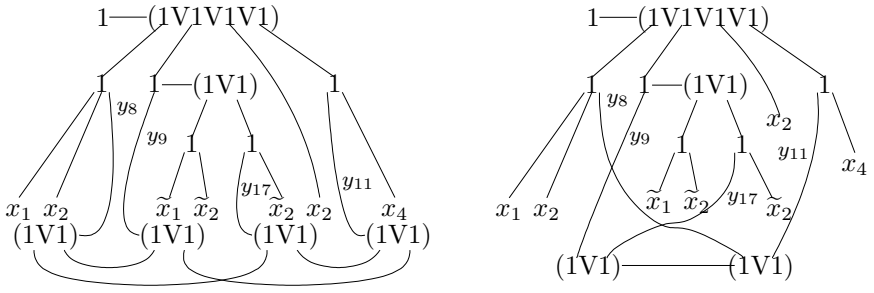
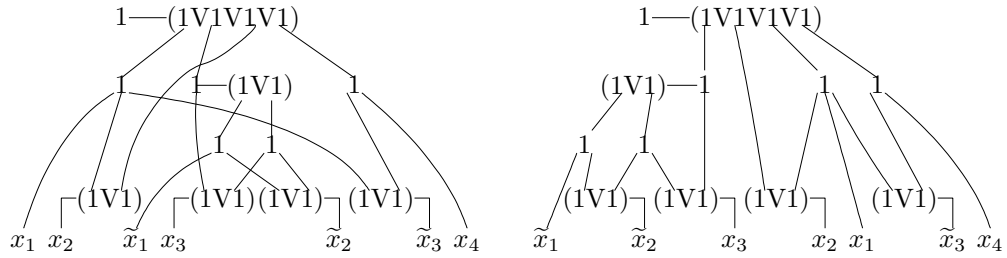


Figure 8: Two ways of reduction by x_3

The second method of reduction is more simple and it seems that there is no advantage of using the first method. The second method also lead another way of representing the two dimensional formulas. It is shown in the figure.

In such graphs, the "alone" variables written only once and the others written twice: one for itself and another for its negation. Then the reduction by x_i is simple: If there exist both itself and its negation then the symbols $\widetilde{x_i}$ and x_i are erased and the arches tied to them are connected to each other. If it is an "alone" variable then it is replaced by 0. In the rest of this article, we will always use this way for v-reduction.

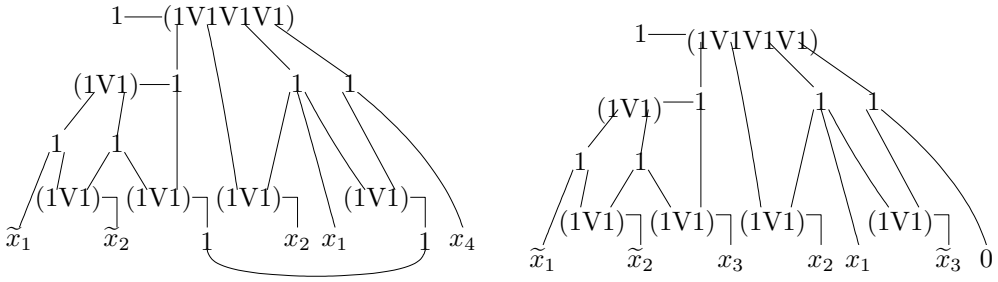
For the reduction be valid, the reduced two dimensional formula must be equal to its statement counterpart in every interpretation. The statement counterpart of the sample formula is the statement formula from which it is derived. Let reduce the statement formula by x_3.



(a) Another form. Each variable and its negation is written only once. The a-term of a variable includes all the a-terms of the variable

(b) A more simple drawing

Figure 9: Another form of the sample formula.



(a) v-reduction by x_3

(b) v-reduction by x_4

Figure 10: v-reduction of "alone" variables and the others

$$
\begin{aligned}
& x_1 \cdot x_2 \cdot \widetilde{x_3} \vee x_3 \cdot (\widetilde{x_1} \cdot \widetilde{x_2} \vee x_3 \cdot \widetilde{x_2}) \vee x_2 \vee \widetilde{x_3} \cdot x_4 \\
& \qquad \overset{x_3}{\rightarrow} \quad (x_1 \cdot x_2 \cdot 1 \vee 0 \cdot (\widetilde{x_1} \cdot \widetilde{x_2} \vee 0 \cdot \widetilde{x_2}) \vee x_2 \vee 1 \cdot x_4) \\
& \qquad\qquad \cdot (x_1 \cdot x_2 \cdot 0 \vee 1 \cdot (\widetilde{x_1} \cdot \widetilde{x_2} \vee 1 \cdot \widetilde{x_2}) \vee x_2 \vee 0 \cdot x_4) \\
& \qquad\qquad = (x_1 \cdot x_2 \vee x_2 \vee x_4) \cdot (\widetilde{x_1} \cdot \widetilde{x_2} \vee \widetilde{x_2} \vee x_2) \\
& \qquad\qquad = (x_2 \vee x_4) \cdot 1 \\
& \qquad\qquad = x_2 \vee x_4
\end{aligned}
$$

I.e. the reduced form of the statement formula becomes $x_2 \vee x_4$. So the reduced two dimensional formula must be equal to this.

In the figure, the values of the reduced two dimensional formula for the interpretations (x2 = 1),(x2 = 0, x4 = 1) and (x2 = 0, x4 = 0). The reduced two dimensional formula and the reduced statement formula have the same value for every interpretation, thus the reduction is valid.

The formula obtained directly from the statement formula is called initial formula. The other formulas derived from this by reduction is called reduced formulas.

Obtaining the two dimensional form of a statement formula as mentioned above takes polynomial time. For each different variables, one or two OR-nodes are formed. For each atomics, an arc is tied to this nodes. So the total number of arches becomes the number of parenthesis + the number of atomic terms + 2*number of different variables. It takes $O(s)$ times to form the graph. The reduction also takes polynomial time.

The last form of the formula after reducing by all variables are shown in the figure.

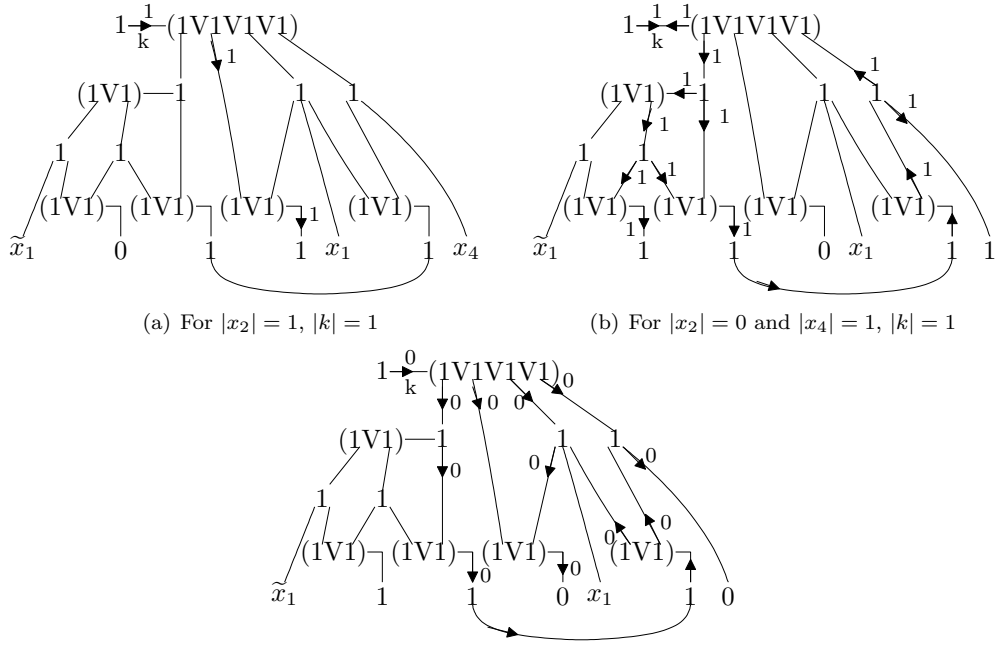
As seen, what at hand is a pure graph and we should decide whether a formula is a tautology or not by investigating this graph. The question here is: "which properties of such a graph differentiate a tautology from a non tautology". Before investigating the answer for this question, let change the notation and introduce some definitions.

2.4 A notation

Let use $\frown$ instead of $(1V..V1)$ and $\bullet$ instead of $-1-$.

The formulas given in the figure is redrawn with this notation and shown in the figure.

We want to express a two dimensional formula by means of its arches and nodes. Due to this new notation, we will write the nodes as follows. $p \bullet (p_1, .., p_n)$ represents an AND-node where p is the incoming arc and $p_1, .., p_n$ are the outgoing arches. $p \models (p_1, .., p_n)$ represents an OR-node where p is the incoming arc and $p_1, .., p_n$ are the outgoing arches. Naturally, the node $p \bullet (r, s)$ can be written as $r' \bullet (p', s)$ or $s' \bullet (p', r)$. Similarly, the node $p \models (p_1, .., p_n)$ can be written as $(p'_1, .., p'_n) \dashv p'$.



(a) For $|x_2| = 1$, $|k| = 1$

(b) For $|x_2| = 0$ and $|x_4| = 1$, $|k| = 1$

(c) For $|x_2| = 0$ and $|x_4| = 0$, $|k| = 0$

Figure 11: The interpretations for the reduced formula. The values of arches which does not affect the result are not shown

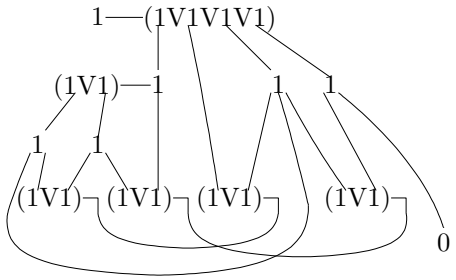


Figure 12: The totally reduced form of the example formula

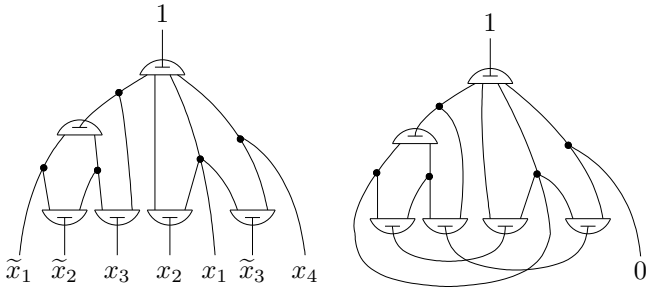


Figure 13: The sample formula and its last reduced form redrawn with new notation

(a) Denoted as $p \bullet (r, s)$ or $r' \bullet (p', s)$ or $s' \bullet (p', r)$

(b) Denoted as $p| = (r, s)$ or $(r', s') \dashv p'$

Figure 14: Node illustrations

Formulas of arches . Let the formula of an arc p be represented by $p()$. Each formula can be thought as composed of other formulas. Each formula is either a atomic term or an disjunction or an conjunction. An AND-node represents a

conjunction and an OR-node represents an disjunction. If p is tied to an AND-node then $p()$ is a conjunction, if it is the main arc of an OR-node then it is a disjunction. For example y_3 is tied to an AND-node whose arches directed to outside is y_9 and y_{10}. For this reason $y_3()$ is a conjunction and the elements of this conjunction is $y_9()$ and $y_{10}()$.

Then we can apply the following rules for the statement equivalences

1. The formula of p be $p()$.
2. If p and beyond is in the form px then $p() = px$,
3. If p and beyond is in the form pr then $p() = pr()$.
4. p and beyond is in the form $p \bullet (p_1, .., p_n)$ then $p() = p \bullet \{p_1(), .., p_n()\}$.
5. p and beyond is in the form $p \models (p_1, .., p_n)$ then $p() = p \models \{p_1(), .., p_n()\}$.
6. p and beyond is in the form $p =\!| r$ then $p() = p =\!| r()$.

For example for the subformula given in the figure and begins from y_3, some equivalences would be valid.

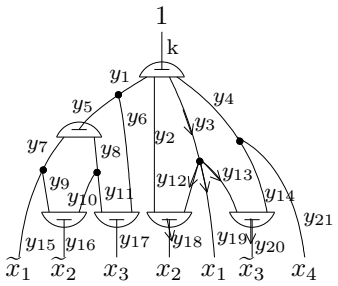


Figure 15: subformula for y_3

From here

$$\begin{aligned} y_3() &= y_3 \bullet \{y_{12}(), y_{19}(), y_{13}()\} \\ &= y_3 \bullet \{y_{12} =\!| y_{18}(), y_{19}x_1, y_{13} =\!| y_{20}()\} \\ &= y_3 \bullet \{y_{12} =\!| y_{18}x_2, y_{19}x_1, y_{13} =\!| y_{20}\widetilde{x_3}\} \end{aligned}$$

is found. The reason to choose such a notation is that when the parenthesis are opened, then we get the paths of the formula. For example the path set of y_3 would be found as

$y_3\{\} = \{y_3 \bullet y_{12} =\!| y_{18}x_2, y_3 \bullet y_{19}x_1, y_3 \bullet y_{13} =\!| y_{20}\widetilde{x_3}\}$

Paths The ways used to go from one element to another one is known as paths. A path is shown by writing arches,nodes and atomicss in sequence. For example, the path begin with p_1 and end with x_2 is written as $p_1 \models p_2 \bullet y_9 =\!| y_{17}x_2$. Here $\models$ represents a head-to-base or-node between the arches p_1 and p_2. Similarly $\bullet$ and $=\!|$ represents an and node and a base-to-head or-node. The direction of path is from left to right. If we do not need to describe the types of nodes , we use the notation $p.r$. Here it is indicated that there is a node between p and r. Two subsequent arches are tied to each other with an and-node. So if r immediately comes after p we will write this as $p \bullet r$ or pr. Naturally several subsequent arches can be thought as one arc and one arc can be thought as composed of several subsequent arches. We will use both accordingly.

For each path there is a reverse path. For example the reverse of $p_1 \models p_2 \bullet y_9 =\!| y_{17}x_2$ is $x_2y'_{17} \models y'_9 \bullet p'_2 =\!| y'_1$. $\models$ and $=\!|$ are the reverse of each other. The reverse of a variable such as x is itself. the reverse of $\bullet$ is $\bullet$ again.

The notation pAr shows a path which begins with p and end with r. The paths are shown with capital letters. The reverse of the path pAr is written as $r'A'p'$. On the other hand, the notation $p..r$ shows any path begins with p and end with r. The reverse of them are shown as $r'..p'$. $p..$ and $..r$ shows the paths beginning with p and the paths end with r respectively.

The path relation is transitive. I.e if $p..r$ and $r..s$ exist then a path $p..r..s$ exists.

If a path begins with a atomic term and end with an atomic term then it is a complete path. For example in a non-reduced formula any path begins with root 1 and end with a variable is a complete path. The paths not complete can be called subpaths.

2.5 Equality and equivalences

In the standard statement formulas, if two formulas give the same results for the same interpretation, they are said to be equal. The same thing can be defined for the two dimensional formulas. Now define equality and equivalence relations as follows.

Definition 2
- *The formulas give the same results for the same interpretations are equivalent.*
- *If two statement formulas give the same results for the same interpretations, they are equal.*
- *If two 2d formulas give the same results for the same interpretations, they are equal.*

We show the equality with = and equivalence with $\cong$.

Each formula and therefore each arc has a statement equivalence. The statement equivalence of an initial formula is the statement formula which it has been formed. However,the statement equivalence of any two dimensional formula can be found by the following recursive procedure regardless of how to construct it.

1. $D(p)$ represents the statement correspondence of the formula beginning from p.
2. If p and beyond is in the form px then $D(p) = x$,
3. If p and beyond is in the form pr then $D(p) = D(r)$.
4. If p and beyond is in the form $p \bullet (p_1, .., p_n)$ then $D(p) = D(p_1) \cdot .. \cdot D(p_n)$.
5. If p and beyond is in the form $p \models (p_1, .., p_n)$ then $D(p) = D(p_1) \vee .. \vee D(p_n)$.
6. If p and beyond is in the form $(p, ..) \dashv r$, then $D(p) = D(r)$.

If a 2d formula and a statement formula give the same results for the same interpretations then they are equivalent. Therefore a 2d formula and its statement correspondence are equivalent. I.e. $p\{\} \cong D(p)$ becomes true.

If a 2d formula differs from the others by some formal chances we can call them matched formulas. We show matching relation with $\equiv$. For example an OR-node with more than one base arches can be shown with several OR-nodes with only two base arches. Also the AND-nodes with more than three arches can be represented by AND-nodes with three arches only. Then these are matched formulas.

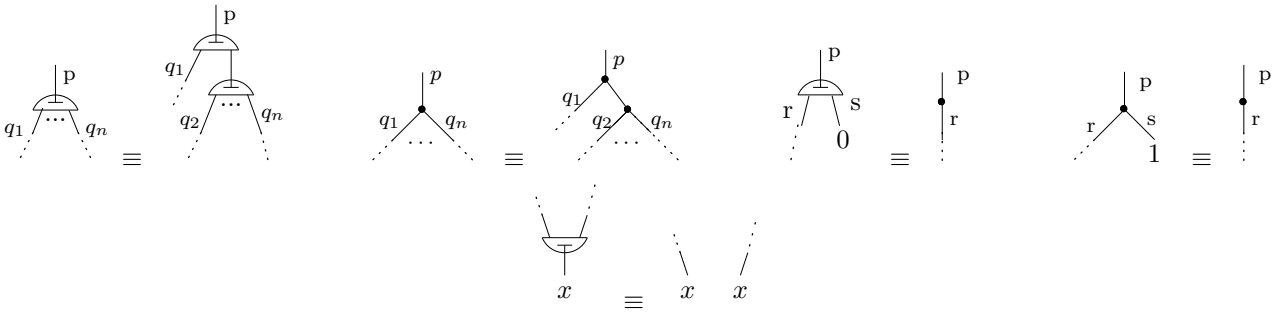


Figure 16: Matching relations

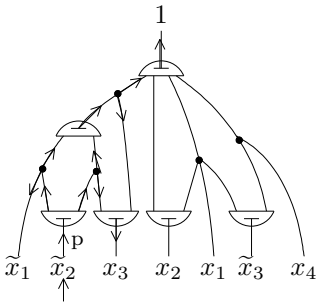
An equivalent formula We can also mention the formulas starting from a variable. For example the formula found beginning from $\widetilde{x}_2$ is shown in the figure.

This formula is in fact the conjunction of the formula of arc p' which is connected to the variable $\widetilde{x}_2$ and the variable $\widetilde{x}_2$ itself. I.e. the statement correspondence of this formula becomes $\widetilde{x}_2 \cdot D(p')$. The subformula starting from p' is indeed the A-term of $\widetilde{x}_2$. Similarly the subformula found beginning from x_2 is the conjunction of x_2 and it's A-term.

In the statement formulas, we have shown the simple form as $x.\alpha \vee \widetilde{x} \cdot \beta \vee \delta$ where α,β and δ are statement formulas. Here α, β are the a-terms of x_2 and $\widetilde{x}_2$ respectively and δ represents the rest of the statement formula. The simple form is equal to the formula. Therefore, the following form must also be another equivalent form of the formula in 2d representations.

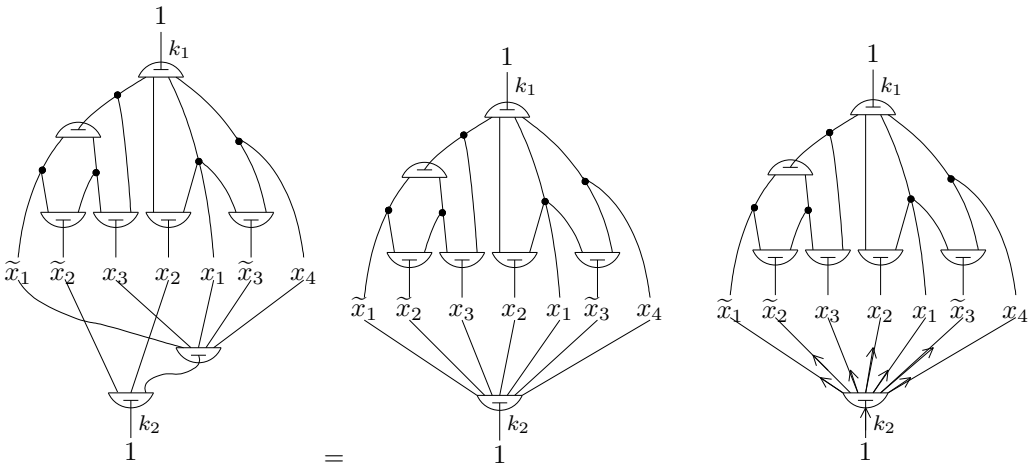
In this representation, we accept that the atomic terms connected with two arches are conjuncted with the arches. For example, for $p - \widetilde{x}_2 - r$, $D(p) = \widetilde{x}_2 \cdot D(r)$ and $D(r') = \widetilde{x}_2 \cdot D(p')$ becomes true.Thus the simple form of the formula is the formula starting from k_2'. However, the simple form is equal to the formula. Then both k_1 and k_2' gives the same value, therefore both arches can be seen as the root arch

The formulas can start not only a variable but from any terminals of the formula. So, the scans starting from the root, can be started from the root term also since the root term is a terminal of the graph. From here on, we will start the root scans from the root term also.



$\cong \widetilde{x}_2 \cdot D(p')$

Figure 17: A formula starting from a variable



(a) The formula

(b) The simple form is the formula starting from k_2'

Figure 18: An equivalent formula

2.6 Validating the reductions and invalid terms

Validating the reductions We mentioned that for the reduction in 2d formulas to be valid, the statement equivalence of a reduced formula obtained when a 2d formula is reduced, must be equal to the statement formula obtained when the statement equivalence of that formula is reduced. In other words, the following must be true.

Proposition 2 *Let $2D(x_i)$ and $S(x_i)$ represent a 2d formula and its statement equivalence respectively. I.e. let $2D(x_i) \cong S(x_i)$ be true. If $2D(x_i) \xrightarrow{x_i} 2D()$ and $S(x_i) \xrightarrow{x_i} S()$ then $2D() \cong S()$.*

The statement equivalence of an initial formula is the formula from which it is formed. However, it is not determined what the statement equivalence of the reduced formulas and how to find it. For example in the figure, the 2d correspondence of the statement formula $x \vee x.\widetilde{x} \vee \widetilde{x} \cdot y$ and it's reduced form wrt x are given. If we reduce the statement formula with x, we obtain y. Therefore, for the 2d reduction to be true, the statement correspondence of the reduced 2d formula must be y. We don't know yet whether it is indeed equivalent to y or the properties which make it equivalent to y

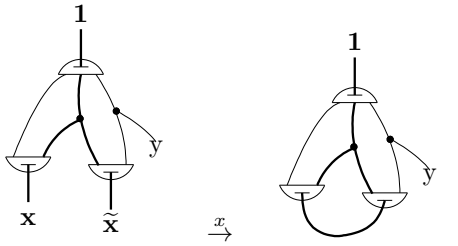
A solution to this problem might be to divide the formula into its elementary products and define the value of the formula in terms of the values of its elementary products.

Invalid terms In a statement formula the value of the formula is equal to the conjuncted value of the elementary products. The same thing must be valid for the 2d formulas. In the disjunctive normal form of a formula each elementary product either includes x_i or $\widetilde{x}_i$ or both x_i and $\widetilde{x}_i$ or does not include them. Then the formula can be written as

$x_i \cdot (a_1 \vee .. \vee a_k) \vee \widetilde{x}_i \cdot (b_1 \vee .. \vee b_l) \vee x_i \cdot \widetilde{x}_i \cdot (c_1 \vee .. \vee c_m) \vee (d_1 \vee .. \vee d_n) = x_i \cdot \alpha \vee \widetilde{x}_i \cdot \beta \vee x_i \cdot \widetilde{x}_i \cdot \delta \vee \gamma$

Here the term $x_i \cdot \widetilde{x}_i \cdot \delta$ is invalid and eliminated directly or indirectly.

$$x_i \cdot \alpha \vee \widetilde{x}_i \cdot \beta \vee x_i \cdot \widetilde{x}_i \cdot \delta \vee \gamma \xrightarrow{x_i} (0 \cdot \alpha \vee 1 \cdot \beta \vee 0 \cdot 1 \cdot \delta \vee \gamma) \cdot (1 \cdot \alpha \vee 0 \cdot \beta \vee 1 \cdot 0 \cdot \delta \vee \gamma)$$
$$= \alpha \cdot \beta \vee \gamma$$



(a) A formula whose statement equivalance is $x \vee x.\widetilde{x} \vee \widetilde{x} \cdot y$

(b) Is this formula equivalent to y?

Figure 19: A formula and its reduced form

However, if we reduce a 2d formula, there is no term or part eliminated. I.e. invalid terms are preserved after reduction.

In any formula, the invalid elementary products does not affect the truth of the formula. Then after any reduction. the value of the formula is determined by the value of elementary products not invalid. If in a formula the invalid products are not eliminated then in the last reduced form of any invalid formula, all the elementary products must be invalid.

Thus if in a last reduced form, it is found that every elementary product is inconsistent, then we can decide that the original formula is invalid, otherwise it is valid. However, before applying this idea, we must find answers to the questions what is an elementary product and how they can be found in a two dimensional formula.

Part II
Elementary products and scans

3 Initial formulas and elementary products

3.1 Finding elementary products

To find the elementary products of a statement formula, it is enough to apply the equivalence $x \cdot (y \vee z) = x \cdot y \vee x \cdot z$ until can not be applied. We could use a similar method in two dimension, but then the original formula would be changed. However, there are ways to obtain the elementary products without changing the original formula. Now let investigate these ways.

The following is an algorithm to do this.

Algorithm 4 *1. Put the formula to the stack.*

2. Do the followings until there is no more element in the stack.

(a) Pop an element from the stack.

(b) If this is a disjunction, i.e. the current element is in the form $(a_1 \vee .. \vee a_n)$ then choose one of a_i and push to the stack.

(c) If the current element is in the form $(a_1 \cdot .. \cdot a_n)$ then push all the a_i's to the stack.

(d) If the current element is an atomic term then put it to VAR.

3. All the atomic terms in VAR form an elementary product. Each different selection in a disjunction leads to different elementary products.

For example let find the elementary products of $x_1 \cdot x_2 \cdot \widetilde{x_3} \vee x_3 \cdot (\widetilde{x_1} \cdot \widetilde{x_2} \vee x_3 \cdot \widetilde{x_2}) \vee x_2 \vee \widetilde{x_3} \cdot x_4$ by this way. The following figure shows how to find an elementary product by this way. The elements marked by an underline shows the selection made in each disjunction.

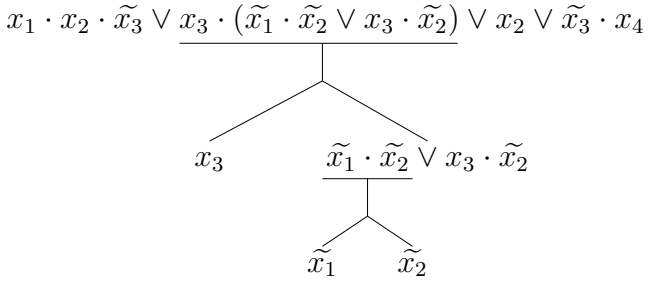


Figure 20: Finding elementary products in a statement formula

Now the corresponding algorithm to find an elementary product in a two dimensional formula is given below.

Algorithm 5 *1. Begin from the root of the formula*

2. Push the current element to the stack.

3. Do the followings until the stack is empty.

(a) Pop an arc from the stack. Let this be p.

(b) If p and beyond is an OR-node, i.e. is in the form $p \models (r_1, .., r_n)$ then push r_i for any i to the stack.

(c) If p and beyond is an AND-node, i.e. is in the form $p \bullet (r_1, .., r_n)$, for all $i = 1, .., n$, push r_i to the stack.

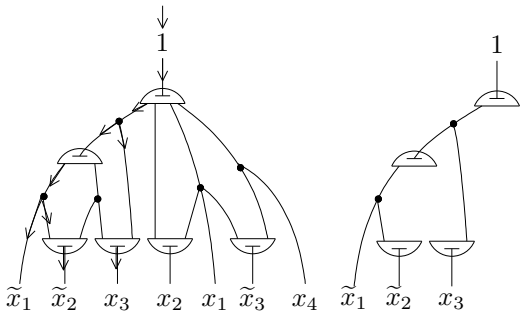
(d) If p and beyond is in the form $p \dashv r$ or pr then push the arc r to the stack.

(e) If p is tied to a variable or atomic term, i.e. is in the form px then put the atomic term to VAR.

4. The subformula scanned is an elementary product. Its statement correspondence is found by production of the atomic terms in VAR.

We call this the algorithm of finding elementary products or the algorithm of elementary product scanning. In the figure, finding an elementary product with this algorithm is shown. The arrows shows the selection and the numbers near the arrows shows the sequence.

Here, we feel it is necessary to make a distinction between the scanning of an elementary product and the elementary product itself. In the scanning, the arches have a direction, but in the e-products, arches have no direction and independent from scanning. An elementary product itself is seen as a tree from all the leafs and there is no difference between the leafs in that respect. So we can say that an elementary product has an existence independent of which terminals its scanning begins. On the other hand an e-product can be found by scanning only. We will investigate this subject later.



(a) The scanning (b) the e-product found

Figure 21: Scanning of an e-product in a two dimensional formula

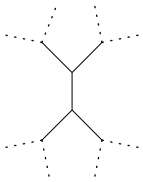


Figure 22: An e-product is seems the same from every terminals and can be found with a single scanning

3.2 Simple and elementary product scans

Each e-product scan can be written as a set of its paths. For example the paths of the elementary product scan in the figure can be written as

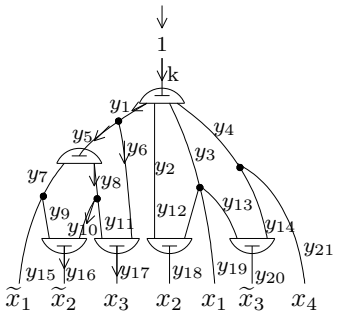


Figure 23:

$$\{\Rightarrow 1k.y_1.y_5.y_8.y_{10}.y_{16}\widetilde{x_2},\ \Rightarrow 1k.y_1.y_5.y_8.y_{10}.y_{11}.y_{17}x_3,\ \Rightarrow 1k.y_1.y_6.y_{17}x_3\}$$

We call such a set as the path set of an e-product scan. Here $\Rightarrow$ indicates both the starting point and the direction.

Every set of paths does not correspond to an elementary product scanning. Let the structure where a path forked from the others by an and node be simple product scans. So, naturally, all the e-product scans are simple product scans. However, not every simple product scans are e-product scans.

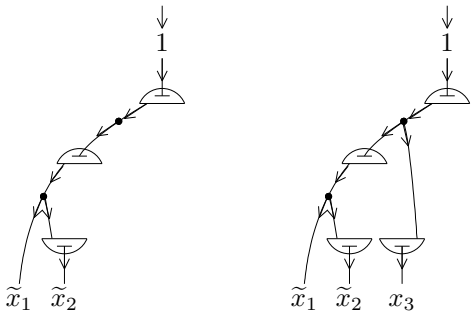
So an simple product scan would have the following properties.

- All the paths starts from the same element. The element can be an arc or atomic term.
- All the paths are separated from an and node.

If a simple product scan has the following additional property, then it is an e-product scan.

- All the arches out from an and-node which belongs to the scan take place in the scan.

Let $\curlywedge$ determine a simple product scan and $\boldsymbol{\lambda}$ determine e-product scans. In that case, $\curlywedge a$ and $\boldsymbol{\lambda} b$, indicate a simple product scan and an e-product scan respectively.



(a) A simple product scan (b) An e-product scan

Figure 24: Simple and elementary product scans

3.3 The value of the scans

The value of a simple product scan is the conjunction of the values of atomic terms found. Each atomic are reached via a path. So it is possible to define a value for paths. Let the statement correspondence of a path is equal to the element terminating it. In this case, for

$$\boldsymbol{\lambda} a = \Rightarrow\{1k.y_1.y_5.y_8.y_{10}.y_{16}\widetilde{x_2},\ 1k.y_1.y_5.y_8.y_{10}.y_{11}.y_{17}x_3,\ 1k.y_1.y_6.y_{17}x_3\}$$

it become

$$\begin{aligned} D(\boldsymbol{\lambda} a) &= D(\Rightarrow 1k.y_1.y_5.y_8.y_{10}.y_{16}\widetilde{x_2}) \cdot D(\Rightarrow 1k.y_1.y_5.y_8.y_{10}.y_{11}.y_{17}x_3) \cdot D(\Rightarrow 1k.y_1.y_6.y_{17}x_3) \\ &= \widetilde{x}_2 \cdot x_3 \cdot x_3 \\ &= \widetilde{x}_2 \cdot x_3 \end{aligned}$$

Thus the following definitions hold.

Definition 3 *1. The statement equivalence of an atomic term is equal to themselves. I.e. $D(x) = x$.*

2. The value of a path is equal to the value of its terminating atomic term. I.e. $D(p..x) = D(x) = x$.

3. The value of a simple product scan is the conjunction of the values of its paths.

On the other hand, since the value of a path is equal to the value of its terminal, the following would always be true.

$$D(p.A) = D(A)$$

4 Initial formulas and the value of arches

4.1 Scan starting from an arc

Not only we can mention the elementary products of a formula, but also the elementary products of subformulas beginning from an arc. For example in the following, a subformula beginning from y_1 and the elementary products of it are shown.

Then we should change the scanning algorithm to cover this new situation. The following is such an algorithm.

Algorithm 6 *1. This algorithm finds an elementary product beginning from the element X. X can be an atomic term or arc.*

2. Put x to the stack.

3. Do the followings until the stack is empty.

(a) pop an element from the stack.

(b) If the element is an atomic term, i.e. is in the form xp then put the term x to VAR and put the arc p to the stack.

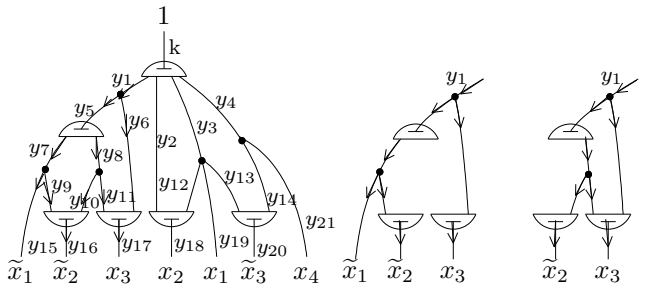


Figure 25: Elementary products beginning from the arc y_1

(c) If the element is an arc, do the followings.

i. Let the arc be p.

ii. If p and beyond is in the form $p\models (r_1, .., r_n)$ *then choose an i and push* r_i *to the stack.*

iii. If p and beyond is in the form $p \bullet (r_1, .., r_n)$ *then for all* $i = 1, .., n$, *push* r_i *to the stack.*

iv. If p and beyond is in the form $p = \!| r$ *or* pr, *then put r to the stack.*

v. If p and beyond is an atomic term, i.e. is in the form px then put the atomic term to VAR.

4. The subgraph scanned is an elementary product. Its statement correspondence is found by conjunction of the variables in VAR.

The e-product scan algorithm starting from an arc can be made recursive as follows.

Algorithm 7 *This algorithm chooses an e-product starting from p.* $e\{p\}$ *represents the path set of the e-product thus scanned.*

1. Do the followings starting from element p.

(a) If p and beyond is px then it becomes $e\{X\} = px$.

(b) If p and beyond is $p\bullet(r_1, .., r_n)$ *then for each* $r_1, .., r_n$ *apply this algorithm. As a result* $e\{X\} = p\bullet\{e\{r_1\}, .., e\{r_n\}\}$ *holds.*

(c) If p and beyond is $p\models (r_1, .., r_n)$, *then for only one* r_i, *apply this algorithm. As a result* $e\{p\} = p\models e\{r_i\}$ *holds.*

4.2 The value of arches

We can show that the value of an initial formula starting from an arc is the disjunction of the values of all the e-product scans of it.

Each formula can be written as a set of its e-products. Then we can write the equation $p() = \{\boldsymbol{\lambda} p_1, .., \boldsymbol{\lambda} p_n\}$ where $\boldsymbol{\lambda} p_i$ is the i'th e-product beginning from p.

Each formula is either a conjunction or disjunction of other formulas. So the e-product of a formula is composed of the e-product of other formulas.

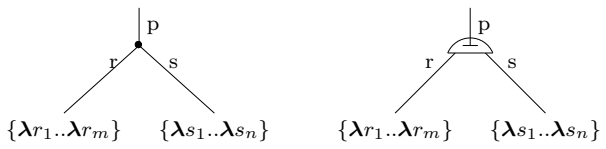


Figure 26: The types of formulas and e-products

In this case, some e-products chosen from p are shown below.
For each type of formula, the relationship among the e-products are given below.

- If p and the next element are in the form px then $\boldsymbol{\lambda} p = px$
- Let p and the next element be $p \bullet (r, s)$. In this case $p() = p \bullet \{r(), s()\}$ would be true. If $r() = \{\boldsymbol{\lambda} r_1, .., \boldsymbol{\lambda} r_m\}$ and $s() = \{\boldsymbol{\lambda} s_1, .., \boldsymbol{\lambda} s_n\}$ is true, then for any $i = 1, .., m$ and $j = 1, .., n$, $\boldsymbol{\lambda} p_{ij} = p \bullet \{\boldsymbol{\lambda} r_i, \boldsymbol{\lambda} s_j\}$ would be true.

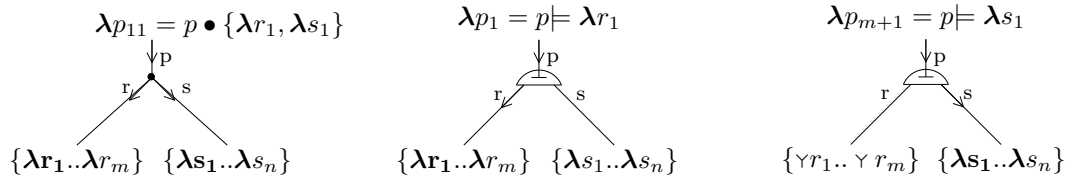


Figure 27: Some e-products found

- If p and the next element are in the form $p\models (r, s)$, then $p() = p\models \{r(), s()\}$ would be true. According to the e-product search algorithm, after p either r or s must be chosen. In that case, the number of all possible e-product would be $m+n$. If $r() = \{\boldsymbol{\lambda}r_1, .., \boldsymbol{\lambda}r_m\}$ and $s() = \{\boldsymbol{\lambda}s_1, .., \boldsymbol{\lambda}s_n\}$ is valid then $\boldsymbol{\lambda}p_i = p\models \boldsymbol{\lambda}r_i$ and $\boldsymbol{\lambda}p_{m+j} = p\models \boldsymbol{\lambda}s_j$ would be valid.

Now by using these relations, we show if $p() = \{\boldsymbol{\lambda}p_1, .., \boldsymbol{\lambda}p_k\}$ is true, then $D(p) = D(\boldsymbol{\lambda}p_1) \vee .. \vee D(\boldsymbol{\lambda}p_k)$ would be true by induction.

Theorem 1 *In an unreduced formula, let* $p() = \{\bigcup_{i=1}^{k} \boldsymbol{\lambda}p_i\}$ *be valid. Then* $D(p) = \bigvee_{i=1}^{k} D(\boldsymbol{\lambda}p_i)$ *would be true.*

Proof 2 *1. If p and the next element are in the form px then $\boldsymbol{\lambda}(p) = \{px\}$ would be true. In this case $|p| = x$ is valid.*

2. Let $p() = p \bullet \{r(), s()\}$ be true. Now assume $D(r) = (\bigvee_{i=1}^{m} D(\boldsymbol{\lambda}r_i))$ and $D(s) = (\bigvee_{j=1}^{n} D(\boldsymbol{\lambda}s_j))$ where $r() = \{\boldsymbol{\lambda}r_1, .., \boldsymbol{\lambda}r_m\}$ and $s() = \{\boldsymbol{\lambda}s_1, .., \boldsymbol{\lambda}s_n\}$. In this case we find

$$\begin{aligned} D(p) &= D(r)D(s) \\ &= (\bigvee_{i=1}^{m} D(\boldsymbol{\lambda}r_i))(\bigvee_{j=1}^{n} D(\boldsymbol{\lambda}s_j)) \\ &= \bigvee_{i=1}^{m}\bigvee_{j=1}^{n} D(\boldsymbol{\lambda}r_i)D(\boldsymbol{\lambda}s_j) \end{aligned}$$

On the other hand for any $i = 1, .., m$ and $j = 1, .., n$, we know $\boldsymbol{\lambda}p_{ij} = p \bullet \{\boldsymbol{\lambda}r_i, \boldsymbol{\lambda}s_j\}$ is true. From here we deduce $D(\boldsymbol{\lambda}p_{ij}) = D(p \bullet \boldsymbol{\lambda}r_i)D(p \bullet \boldsymbol{\lambda}s_j) = D(\boldsymbol{\lambda}r_i)D(\boldsymbol{\lambda}s_j)$. Then

$$\begin{aligned} D(p) &= \bigvee_{i=1}^{m}\bigvee_{j=1}^{n} D(\boldsymbol{\lambda}r_i)D(\boldsymbol{\lambda}s_j) \\ &= \bigvee_{i=1}^{m}\bigvee_{j=1}^{n} D(\boldsymbol{\lambda}p_{ij}) \end{aligned}$$

is found.

3. Let p and the next element are in the form $p\models (r, s)$. Now assume $D(r) = (\bigvee_{i=1}^{m} D(\boldsymbol{\lambda}r_i))$ and $D(s) = (\bigvee_{j=1}^{n} D(\boldsymbol{\lambda}s_j))$ is true where $r() = \{\boldsymbol{\lambda}r_1, .., \boldsymbol{\lambda}r_m\}$ and $s() = \{\boldsymbol{\lambda}s_1, .., \boldsymbol{\lambda}s_n\}$ is true. In this case, since $D(p) = D(r) \vee D(s)$ is true, we find $D(p) = D(r) \vee D(s) = D(\boldsymbol{\lambda}r_1) \vee .. \vee D(\boldsymbol{\lambda}r_m) \vee D(\boldsymbol{\lambda}s_1) \vee .. \vee D(\boldsymbol{\lambda}s_n)$

We know for any $i = 1, .., m$ and $j = 1, .., n$, $\boldsymbol{\lambda}p_i = p|= \boldsymbol{\lambda}r_i$ and $\boldsymbol{\lambda}p_{m+j} = p|= \boldsymbol{\lambda}s_j$ is true. This means that $D(\boldsymbol{\lambda}p_i) = D(p\models \boldsymbol{\lambda}r_i) = D(\boldsymbol{\lambda}r_i)$ and $D(\boldsymbol{\lambda}p_{m+j}) = D(p\models \boldsymbol{\lambda}s_j) = D(\boldsymbol{\lambda}s_j)$ is true. From here

$$\begin{aligned} D(p) &= D(\boldsymbol{\lambda}r_1) \vee .. \vee D(\boldsymbol{\lambda}r_m) \vee D(\boldsymbol{\lambda}s_1) \vee .. \vee D(\boldsymbol{\lambda}s_n) \\ &= D(\boldsymbol{\lambda}p_1) \vee .. \vee D(\boldsymbol{\lambda}p_m) \vee D(\boldsymbol{\lambda}p_{m+1}) \vee .. \vee D(\boldsymbol{\lambda}p_{m+n}) \end{aligned}$$

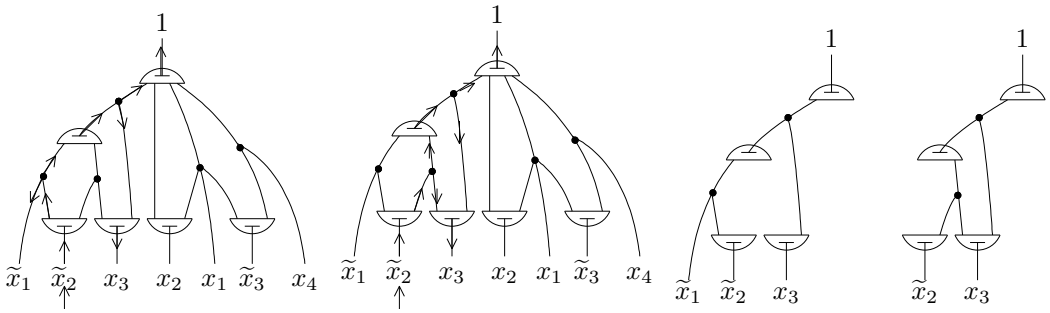
is found.

■

5 Symmetry in e-products and regular formulas

5.1 Symmetry in e-products

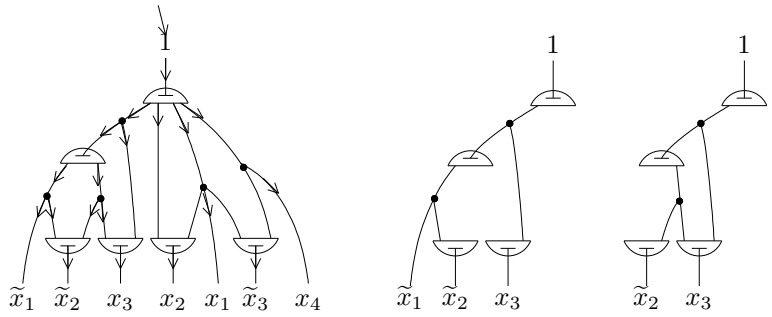
An e-product scan could also start from a variable. In the following figure, the e-products found by starting from $\widetilde{x}_2$ are shown.



(a) Scannings (b) e-products found

Figure 28: the e-products scanning from $\widetilde{x}_2$ and found

In a formula, all the e-products are belong to the formula. I.e. all the e-product found by starting from $\widetilde{x}_2$ must also be found by starting from the root term.

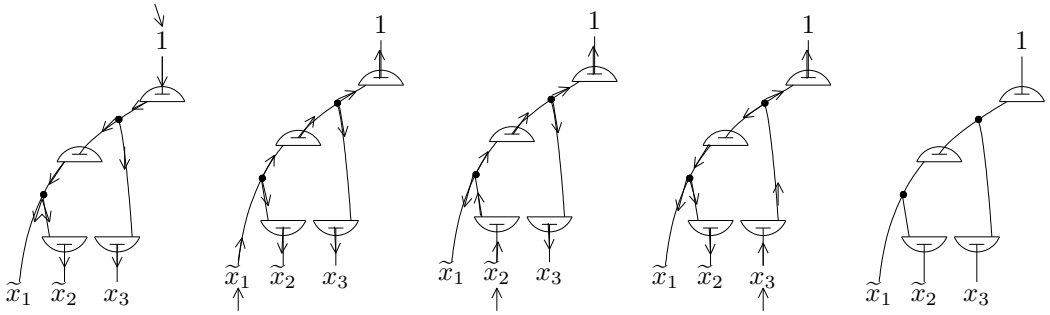


(a) The scannings starting from the root (b) The e-products found

Figure 29: The e-products found by scanning from the root and includes $\widetilde{x}_2$ as a terminal

As seen from the figure, the e-products containing $\widetilde{x_2}$ could be found both from the root and from $\widetilde{x_2}$.

In an e-product, there is a path from each terminal to the other and whatever terminal is used for scanning, the same e-product would be scanned. Each different scanning of the given e-product are illustrated in the following.



(a) Tha scanning starting from a terminal (b) The e-product found

Figure 30: All these scans find the same e-product

Let denote an e-product by $\curlyvee a[u_1, .., u_n]$ where $u_1, .., u_n$ are all the terminals of it. Let denote a scanning from a terminal by an arrow at that terminal. For example, we can write the scanning from the four terminals of this e-product as follows.

$$\boldsymbol{\lambda}a_1 = \curlyvee a[\Rightarrow 1, \widetilde{x_1}, \widetilde{x_2}, x_3]$$
$$\boldsymbol{\lambda}a_2 = \curlyvee a[1, \Rightarrow \widetilde{x_1}, \widetilde{x_2}, x_3]$$
$$\boldsymbol{\lambda}a_3 = \curlyvee a[1, \widetilde{x_1}, \Rightarrow \widetilde{x_2}, x_3]$$
$$\boldsymbol{\lambda}a_4 = \curlyvee a[1, \widetilde{x_1}, \widetilde{x_2}, \Rightarrow x_3]$$

Each e-product has an statement equivalence. The value of an e-product is the conjunction of the values of its terminals. For example

$$D(\curlyvee a[1, \widetilde{x_1}, \widetilde{x_2}, x_3]) = 1 \cdot \widetilde{x_1} \cdot \widetilde{x_2} \cdot x_3$$

. However as seen, an e-product can be found by scanning only from a single terminal because there is a path from every terminal to the others and a single scanning finds all the terminals. In this case, the value of each scanning would be the same and give the value of the e-product. In other words the following would be true.

$$D(\curlyvee a) = D(\boldsymbol{\lambda}a_1) = D(\boldsymbol{\lambda}a_2) = D(\boldsymbol{\lambda}a_3) = D(\boldsymbol{\lambda}a_4)$$

Scannings of an e-product In an e-product, we can determine all the other scans from one scanning. Also all these scans must give the same value. Let there are terminals x_i and x_j in an e-product. Thus, there is a path $x_i A x_j$ in the scanning starting from x_i. Let show such a scan having such a path as in the figure.

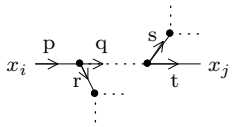


Figure 31: A scan having the path $x_i A x_j$

However, in this case, there exists a scanning which starts from x_j, has the path $x_j A' x_i$ and where all the arches of it, except the arches on A and A', are the same with the above scanning.

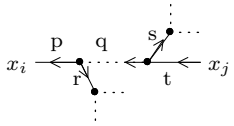


Figure 32: There is another e-product scan which has the path $x_j A' x_i$

Let call the second scanning the reverse scanning over the path $x_i A x_j$. This means that, each scan has reverse scans over each of its paths and all of these scans has the same terminals. Therefore they have the same value. So an e-product scan and it's all reverse scans determines the same e-product. This is true for every scan of an e-product. I.e. each scan of an e-product is a reverse scan of all the others.

5.2 Regular Formulas

The e-products scanning from the root has always a atomic. I.e. the same e-product can be forund from both the root and that terminal. However, this does not guarantee that all the e-products that can be found from x can also be found from the root. Two example that demonstrates this is shown in the figure.

However, such formulas does not derived from a statement formula by the method given. n a formula derived from a statement formula by the given method does not include any e-product which can be scanned from a leaf bu can not be scanned from the root.

Definition 4 *If in a formula, every e-product has the root terminal as a leaf then this is a regular formula.*

This definition in fact gives us a hint to define the root also.

Definition 5 *In a formula, if every e-product can be scanned from a atomic, then this atomic is also a root of the formula.*

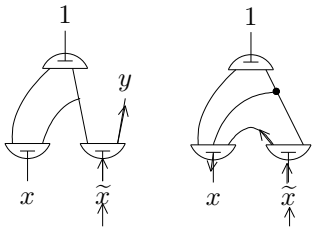
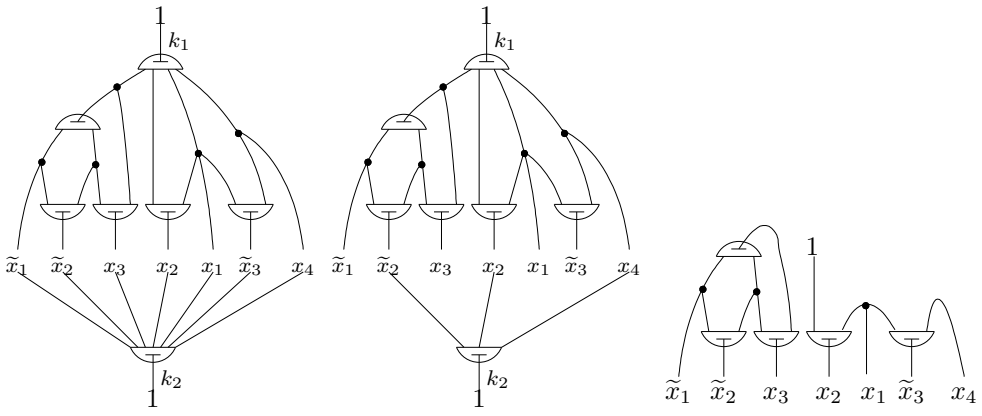


Figure 33: e-products that can be scanned from $\widetilde{x}$, but can not be scanned from the root



(a) Formulas with two root terms

(b) The formula without a root term

Figure 34: Formulas can have any number of root terms

Therefore, a root term can be created by joining arches to all the e-products, disjuncting all of them via and OR-node and terminating the main arc with 1.. For example, in the figure, by joining arches to all atomic terms ad discoining them via an OR-node and terminating the main arc of the node with a 1, a root term is obtained.

This lead us to express a formula in a more simple way, because we don't have to draw the roots any more. However, in this article, we will always draw the formulas with their roots. Also we will denote the root term with k to differentiate it from the other atomic terms, If there are more then one root term, they will be denoted with subindexes. For example the above formulas are shown as below with this notation.

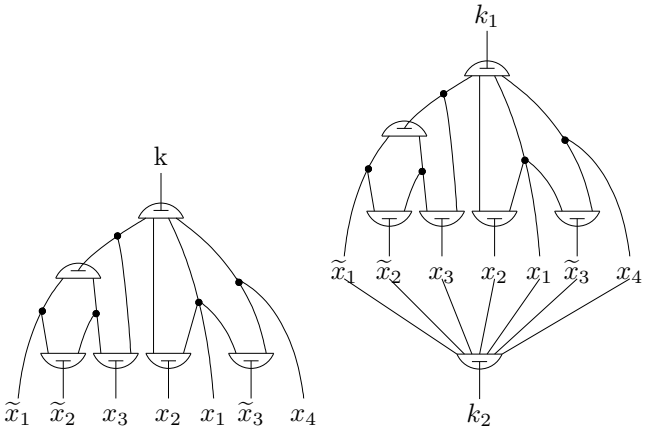


Figure 35: The root terms are denoted by k and with indices if there are more than one

6 Reduced formulas and e-product scanning

6.1 Validity of e-product scanning algorithm

As shown in the figure, in an elementary product including both x and x', there are paths from x to x' and from x' to x. However, after an x reduction, a closed path is produced.

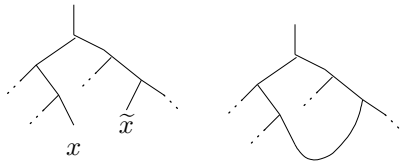


Figure 36: The invalid e-products produce closed paths when reduced

Then it seems that the closed paths indicates invalidity. If this is so, a way to find whether a formula is a tautology or not is to examine every e-product in a totally reduced formula and find whether all have a closed path. However, to be able to do this we have to know what is an elementary product and how to find them in a reduced formula

One way of doing this is to use the e-product scan algorithm developed for the initial formulas without any change on it. However, we need some evidences which shows that it can be used in the reduced formulas too.

We know that, in a formula containing x_i, every e-product contains either x_i or $\widetilde{x}_i$ or both x_i and $\widetilde{x}_i$ or neither of them.

$$\alpha_1 \vee .. \vee \alpha_k \quad = \quad (a_1 \vee .. \vee a_k) \cdot x \vee (b_1 \vee .. \vee b_l) \cdot \widetilde{x} \vee (c_1 \vee .. \vee c_m) \cdot x \cdot \widetilde{x} \vee (d_1 \vee .. \vee d_n)$$

If reduce this for x_i, we will show that after reduction, the newly created e-products are the compose of the parts of the e-product before reduction.

$$\begin{aligned}
&(a_1 \vee .. \vee a_k) \cdot x \vee (b_1 \vee .. \vee b_l) \cdot \widetilde{x} \vee (c_1 \vee .. \vee c_m) \cdot x \cdot \widetilde{x} \vee (d_1 \vee .. \vee d_n) \\
&\xrightarrow{x} (a_1 \vee .. \vee a_k) \cdot (b_1 \vee .. \vee b_l) \vee (d_1 \vee .. \vee d_n) \\
&= ..a_i \cdot b_j \vee ..d_1 \vee .. \vee d_n
\end{aligned}$$

It should be true for two dimensional formulas too. Therefore an e-product scan in a reduced formula can be seen as a unification of two elemantary scan in an unreduced formula.

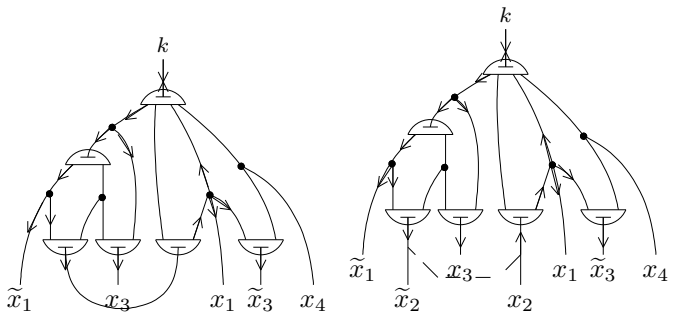


Figure 37: An e-product scan in the reduced formula is the unification of two e-product scans in the unreduced one

To scan an e-product, it is not necessary to begin from the beginning element and scan the rest in a sequential order. In the following, a scan beginning from p is shown. Then another scan beginning from the root are shown. However, this one stops at p. As a result, the whole e-product is scanned.

Therefore any e-product scan can be seen as a unification of two or more e-product scans. So, the e-product scan algorithm given can also be used to find e-products in the reduced formulas too.

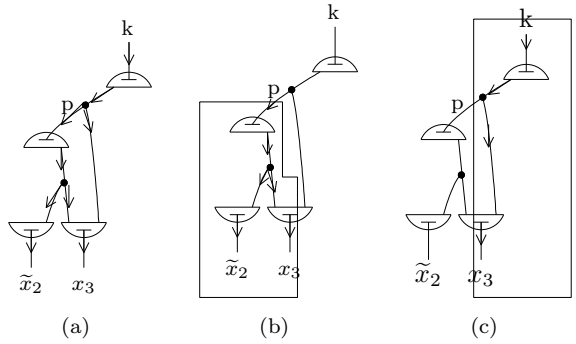
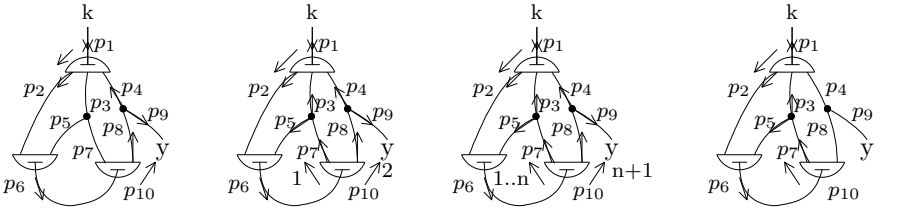


Figure 38: An e-product can be scanned seperately

6.2 An example

In the figure, an elementary scan is shown. Here the arrows near the or-nodes shows the selection and the numbers by shows the sequence of selections. In this example, there is no bound on the number of e-scans since the selection made in each pass from an or-node leads different e-scans. There is no bound on the number of passed of an or-node, therefore there is no bound on the number of different e-scans.



(a) An e-product scan; after p'_{10} is passed first time p'_8 is chosen

(b) Another e-product scan: After the first pass of p'_{10}, p'_7 is chosen and in the second pass, p'_8 is chosen

(c) The e-product scan where p'_7 is chosen after the first n pass of p'_{10} and p'_8 is chosen after the $n+1$'th pass

(d) The e-product scan where p'_7 is chosen after every pass of p'_{10}

Figure 39: Some e-product scans starting from the root with the given algorithm

Here in all the e-products, after p_1 , the arc p_2 is chosen. In the first figure, on the OR-node whose main arc is p'_{10}, the arc p'_8 is chosen. In the second figure, after p'_{10}, p'_7 is chosen and this lead to p_6 for the second time. After passing p'_{10} for the second time, the arc p'_8 is chosen.

6.3 Paths

In a reduced formula, the paths may be endless. For example in the figure, if for every time the arc p'_{10} is passed, the arc p'_7 is chosen, then the path would be $p_1.p_2.(p_6.p'_{10}.p'_7.p_5.)^\infty$ where $\{..\}^\infty$ shows that the subpath in the parenthesis is repeated infinity times. In general, $\{..\}^n$ shows that the part in parenthesis is repeated n times. So $(..)^0$ denote an empty path which means it does not indeed exist. For this reason $AB^0C = AC$ is accepted as a valid equation.

In a path, it can be encountered a node which is passed already. This is either an AND-node or an OR-node passed from main to base or an OR-node passed from base to main. These are cyclic paths. On the other hand, while following a path, one can encounter the reverse of an already passed arcs. Then these are returning paths.

We can classiffy the paths as follows.

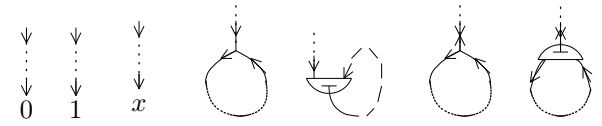
Definition 6 *1. If a path is in the form ..p..p.. it is a cyclic path.*

2. If a path is in the form ..p..p'.. then it is a returning or bidirectional path.

3. If a path is not returning then it is unidirectional.

4. If a path is neither cyclic nor returning then it is a simple path.

A path can both cyclic and returning. For example *..p..r..r'..p..* is both cyclic and returning.
Since the number of arches are finite, an infinite path is always cyclic.



(a) paths end with a atomic term (b) cyclic paths (c) returning paths

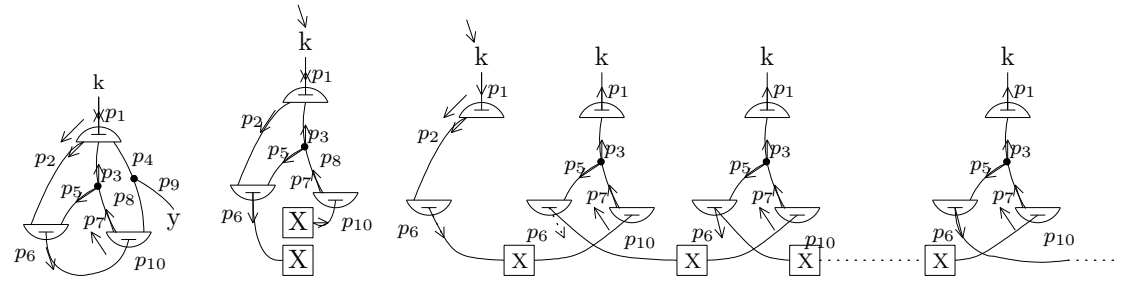
Figure 40: Types of paths

6.4 Paths and E-scans

An e-product scan beginning from an arc can be written as a set of paths beginning from that arc. For example the path set of the e-product scan shown in the figure is given below.

$$\boldsymbol{\lambda}a = \boldsymbol{\lambda}\{\Rightarrow kp_1.p_2.p_6.p'_{10}.p'_8 \bullet p_9 y, \Rightarrow kp_1.p_2.p_6.p'_{10}.p'_8 \bullet p'_4.p'_1 k\}$$

An e-product scan can have infinite number of paths. For example for the e-product scan where every time p'_{10} is passed p'_7 is chosen, the number of paths is infinite. To find these paths let write the e-product scan recursively as follows.



(a) An e-product scan (b) Separate it into two parts (c) Expansion of the scan

Figure 41:

$$\begin{aligned} Y &= \Rightarrow\{kp_1.p_2.p_6X\} \\ X &= \Rightarrow\{p'_{10}.p'_7.p'_3.p'_1 1, p'_{10}.p'_7.p_5.p_6X\} \end{aligned}$$

We separate the scan into two parts where the second one is expressed recursively. If we expand this recursive definition, then we get

$$\begin{aligned} X &= \Rightarrow\{p'_{10}.p'_7.p'_3.p'_1 1, p'_{10}.p'_7.p_5.p_6X\} \\ &= \{A, BX\} \\ &= \{A, B\{A, BX\}\} \\ &= \{A, \{BA, BBX\}\} \\ &= \{A, BA, BBX\} \\ &= . \\ &= . \\ &= \{A, \ddot{\bigcup_{i=1}} B^i A, B^\infty\} \\ &= \Rightarrow\{p'_{10}.p'_7.p'_3.p'_1 1, \{\ddot{\bigcup_{i=1}} (p'_{10}.p'_7.p_5.p_6.)^i p'_{10}.p'_7.p'_3.p'_1 1\}, (p'_{10}.p'_7.p_5.p_6)^\infty\} \end{aligned}$$

Here $\ddot{\bigcup_{i=1}}$ means from $i = 1$ to infinity but not includes infinity. In general, for $l >= k$, $\{\overset{l}{\underset{i=k}{\cup}} AB^i C\} = \{AB^k C, AB^{k+1}, .., AB^l C\}$. For the obvious reasons, $B^\infty A = B^\infty$ must be a valid equivalence. Then the following equivalence would be valid.

$$X = \{A, \ddot{\bigcup_{i=1}} B^i A, B^\infty\} = \{\overset{\infty}{\ddot{\bigcup_{i=0}}} B^i A\}$$

where $B^0 A = A$. Therefore, the above set can be written as

$$\boldsymbol{\lambda} b = \boldsymbol{\lambda}\{\overset{\infty}{\ddot{\bigcup_{i=0}}} \Rightarrow (p'_{10}.p'_7.p_5.p_6)^i p'_{10}.p'_7.p'_3.p'_1 k\}$$

As in the initial formulas, we accept that the value of an e-product scan is the conjunction of the values of its paths.

6.5 Values of the arches

The value of a formula is the value of it's root term or arc. Therefore we only need to find the value of arches in general to know the value of the formulas.

We have shown that the value of an arch in an initial formula is equal to the disjunction of the values of the e-product scans starting from that arc. However, we can not prove a similar thing for reduced formulas by induction, because in reduced formulas there can be no end of the paths and therefore the necessity of induction can not be satisfied. On the other hand, we can see this is true for reduced formula too by the following reasoning.

Theorem 2 *Suppose the value of an arc is equal to the disjunction of the values of its e-product scans. This property does not change with reduction.*

Proof 3 *Consider the arches of the terms px and $r\widetilde{x}$ in a formula. Assume all of the arches p, r, p', r' have this property. Let reduce the formula for x. After reduction, the connection would be in the form $..pr'...$ If r' still has this property, p would have had this property by induction. However, there is no reason for r' to loss this property by reduction. If there is no path from r' to p, this property of r' does not affected by p. On the other hand, suppose there is a path $pr'..p$. In this case, this property of p depends on itself. I.e. if we assume that p does not loss this property, by induction, p does not loss this property and it seems that there is no reason for p to loss this property. Therefore, for any arc in the formula, a reduction does not change this property*

On the other hand, whether we express this as a theorem or assumption is not important. We are totally free to make any assumption and the only restricting factor here is whether the reduced formula keeps the tautological value of the original one under these assumptions. If this is so, they can be used to identify tautology in two dimensional formulas. I.e, we can write this property as a postulate instead of a theorem.

Postulate 1 *the value of an arc in any formula is equal to the disjunction of the values of all the e-product scans starting from that arc.*

7 Elementary sums

7.1 E-sum scans

it can be said of elementary sums of the formulas as well as elementary products. The elementary sums can be found with e-sum scan. The e-sums of a formula can be found with the following algorithm.

Algorithm 8 *1. start with the root term.*

2. Put the current element into AC.

3. Do the followings until there is no more elements.

(a) get an element from AC.

(b) f this is an arc follow it and put the connected element into AC.

(c) If this is a top2base or-node then put all base arches into AC.

(d) If this is an and-node then choose one of the outgoing arches an put to AC.

(e) f this is a atomic term put it to VAR.

4. The subformula scanned is an e-sum scan and its statement equivalence is found by disjuncting the elements in VAR.

We call this an e-sum scan.
In the figure, an e-sum scan is shown.
On the other hand, we can again mention the e-sum scans starting from an arc.

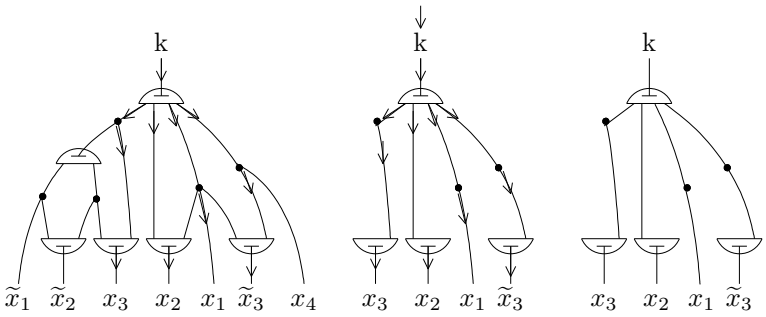


Figure 42: An e-sum scan and the e-sum found

Algorithm 9 *1. This algorithm find an e-sum scan starting from p*

2. Put p into stack.

3. Repeat the following until the stack is empty.

(a) Pop an arc from the stack. Let this be p.

(b) If p and beyond is in the form $p \bullet (r_1, .., r_n)$ then for one i, push r_i to the stack.

(c) If p and beyond is in the form $p \models (r_1, .., r_n)$, then for $i = 1, .., n$, push r_i to the stack.

(d) If p and beyond is in the form $p \dashv r$ or pr, then push r to the stack.

(e) If p is connected to a atomic term, i.e. if p and beyond is in the form px, then put x into VAR.

4. The subformula scanned is an e-sum scan and its statement equivalence is found by disjuncting the elements in VAR..

As for the e-product scans, the value of the e-sum scan is defined as follows:
"The value of an e-sum scan is the disjunction of the values of it's paths.

7.2 Simple and elementary sums

Each e-sum scan can be written as a set of its paths. For example the paths of the e-sum scan in the figure can be written as

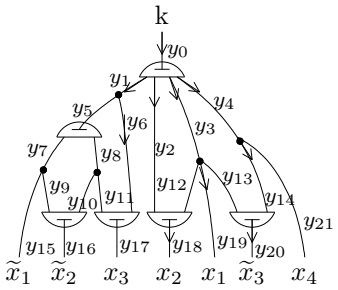


Figure 43:

$$\{\Rightarrow y_0.y_1.y_5.y_8.y_{10}.y_{16}\widetilde{x_2},\ \Rightarrow y_0.y_1.y_5.y_8.y_{10}.y_{11}.y_{17}x_3,\ \Rightarrow y_0.y_1.y_6.y_{17}x_3\}$$

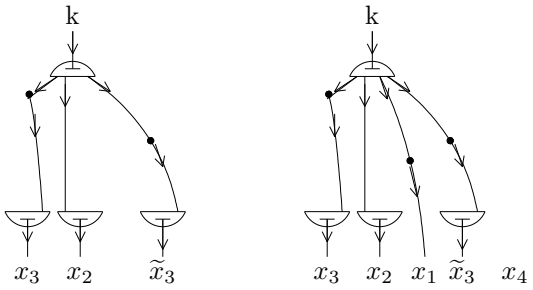
We call such a set as the path set of an e-sum scan.

Every set of paths does not correspond to an e-sum scanning. In the path set of an e-sum scan, a path forked from other paths at a common OR node. On the other hand, all the paths forked from an OR-node found on a member path belong to this set.

Let the structure where a path forked from the other ones by an OR-node be simple sum scans. So, naturally, all the e-sum scans are simple sum scans. However, not every simple sum scans are e-sum scans.

Thus an e-sum scan would have the following properties.

1. All the paths starts from the same element. The element can be an arc or atomic term.



(a) A simple sum scan (b) An e-sum scan

Figure 44:

2. All the paths are seperated via an OR-node.

3. All the arches out from an OR-node which is on the scan belongs to one path of the scan.

Let τ determine a simple product scan and $\boldsymbol{\pi}$ determine e-product scans. In that case, τa and $\boldsymbol{\pi} b$, indicate a simple sum scan and an e-sum scan respectively.

The value of an e-sum scan is defined as the disjunction of the values of it's paths.

7.3 e-product scans vs e-sum scans

In a 2d formula, choosing one path from all the e-sum scans starting from an arc, is the same with choosing all the paths of an e-product scan starting from that arc. Similarly, choosing one path from all the e-product scans starting from an arc, is the same with choosing all the paths of an e-sum scan starting from that arc.

Theorem 3 *A path starting from an element belongs to at least one e-sum scan and at least one e-product scan starting from that element.*

Proof 4 *Trivial.* ■

Theorem 4 *There is one and only one common path between any e-sum scan and any e-product scan starting from one element.*

Proof 5 *First we will prove that there is at least one common path between an e-product scan and e-sum scan. Let $\boldsymbol{\lambda} a$ be an e-product scan and $\boldsymbol{\pi} b$ be an e-sum scan. Let consider two uncommon paths A and B where $A \in \boldsymbol{\lambda} a$ and $B \in \boldsymbol{\pi} b$. These two paths must have a common beginning part and then fork either in an and-node or or-node. Suppose they fork in an and-node and let this be $q \bullet (r, s)$. Now suppose $A = pA_1(q \bullet r)A_2$ and $B = pA_1(q \bullet s)B$. However this means that there is another path $pA_1(q \bullet s)A_3$ which belongs to $\boldsymbol{\lambda} a$. Therefore there are two paths $pA \bullet sC$ and $pA \bullet sD$ which belongs to $\boldsymbol{\lambda} a$ and $\boldsymbol{\pi} b$ respectively and have a longer common initial part. Similarly, for the case of OR-node, we again obtain two paths which have longer common initial path and belongs to $\boldsymbol{\lambda} a$ and $\boldsymbol{\pi} b$. These two paths are either the same peth or different. However, we can repeat the same reasoning as long as the paths are different. So this means that there is at least one common path between $\boldsymbol{\lambda} a$ and $\boldsymbol{\pi} b$.*

On the other hand, suppose there is more than one common path between $\boldsymbol{\lambda} a$ and $\boldsymbol{\pi} b$. Consider two of them. These are both belong to an e-product scan and an e-sum scan. If two paths belongs to the same e-product scan they must fork in an and-node. Similarly, if two paths belongs to the same e-sum scan they must fork in an or-node. Since two paths can fork in one single node, and this node can not be both an and-node and an or-node, assumption is wrong.

■

Theorem 5 *1. Let construct a set by taking one and only one path from each e-product scan of an element. This set is the path set of an e-sum scan of that element.*

2. Let construct a set by taking one and only one path from each e-sum scan of an element. This set is the path set of an e-product scan of that element.

Proof 6 *1. Let the set constructed be A. Any arbitrary pair of the paths in A must fork in an OR-node. Otherwise both of them would belong to an e-product scan. Therefore they set up a simple sum scan. I.e. A must be a subset of a path set of an e-sum scan such as $\boldsymbol{\pi} a$. Suppose one of the paths of $\boldsymbol{\pi} a$ is not an element of A. Each path must be an element of at least one e-product sum. Therefore this path is an element of an e-product scan, let say $\boldsymbol{\lambda} b$. However,*

there is also a path in A which is also an element of $\boldsymbol{\lambda}b$ since A includes a path from every e-product scan. This is not possible because then there would be two common paths between $\boldsymbol{\lambda}b$ and $\boldsymbol{\pi}a$ which contradics with the previous theorem. . Therefore, there can not be no path of $\boldsymbol{\pi}a$ which is not in A. This means that A is the path set of an e-sum scan.

2. Similar to the previous item.

■

We have postulated that the value of an arc is the disjunction of the values of it's e-product scan. We can do the same for e-sum scans also.

Postulate 2 *The value of an arc in any formula is equal to the conjunction of the values of all the e-sum scans starting from that arc.*

Part III
Tautological properties

8 Tautological properties in last reduced forms

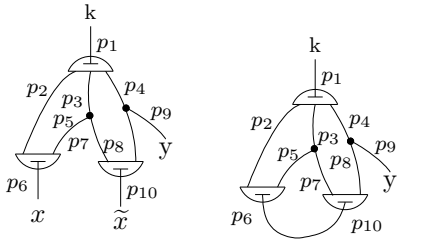
8.1 The values of the infinite paths

We have said that the value of an e-product scan is equal the conjunction of the values of its paths and the values of a path is equal to the value of its last element.

$D(pX) \quad = \quad D(X).$

However, the infinite or closed paths does not have a last element. The infinite paths can exist in any formula and the value of any infinite path with the same form must be the same without considering its place in a formula. This means that the value of any infinite path must be either 0 or 1. Let call them as valid and inconsistent paths respectively. Now our aim is to investigate which properties might differentiate a valid path from an inconsistent one.

To find what could be the value of an infinite path, let consider the formula given in the figure.



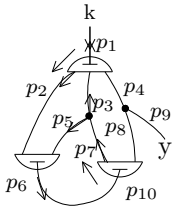
(a) 2d formula (b) reduced form for x

Figure 45:

The statement equivalence of this sample formula is $x \vee x \cdot \widetilde{x} \vee \widetilde{x} \cdot y$. If we do an x reduction, we get

$$\begin{aligned} x \vee x \cdot \widetilde{x} \vee \widetilde{x} \cdot y \quad &\xrightarrow{x} \quad (0 \vee 0 \cdot 1 \vee 1 \cdot y) \cdot (1 \vee 1 \cdot 0 \vee 0 \cdot y) \\ &= y \end{aligned}$$

Then this must be the statement equivalence of the two dimensional formula obtained in the figure. However, there is an e-product scan with an infinite path as seen in the figure.



(a) An e-product scan

Figure 46: What is the value of the path $\Rightarrow kp_1.p_2.p_6.(p'_{10}.p'_7 \bullet p_5.p_6)^\infty$?

Let write the paths of this e-product scan.

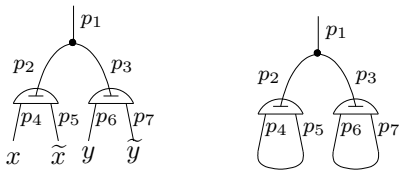
$$\boldsymbol{\lambda} c = \boldsymbol{\lambda} \Rightarrow kp_1.p_2.p_6.\{p'_{10}.p'_7 \bullet p'_3.p'_1 k, \quad \{\overset{\cdot\cdot}{\bigcup_{i=1}}(p'_{10}.p'_7 \bullet p_5.p_6.)^i p'_{10}.p'_7 \bullet p'_3.p'_1 k\}, \quad (p'_{10}.p'_7 \bullet p_5.p_6)^\infty\}$$

All the paths except the last one finishes with root term. Since value of the root term is 1, the value of this e-product scan must be equal to the value of it's last path.

$$D(\boldsymbol{\lambda} c) = D(\Rightarrow kp_1.p_2.p_6.(p'_{10}.p'_7 \bullet p_5.p_6)^\infty)$$

It is obvious that, the value of this path must be 0. Otherwise, the value of the e-product scan and therefore the value of the formula would be 1 and the reduction would be wrong.

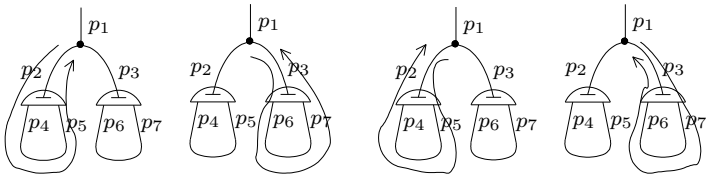
On the other hand, consider the following example.



(a) An example (b) The last reduced form

Figure 47: What is the value of $p_1.(p_2 \models p_4 p_5 = \nmid p_2'.p_3 \models p_6 p_7 = \nmid p_3')^\infty$ in the second figure?

The statement equivalence of the original formula is $(x \vee \widetilde{x}) \cdot (y \vee \widetilde{y})$ and we get 1 when totally reduced. However, as seen, in the last reduced two form, there is at least one infinite path in every e-product scan. I.e. there is no one e-product scan whose every path is finite. So, for the result be true, all the infinite paths of at least one e-product scan must be taken as 1. On the other hand, there is a difference between the infinite paths in this and in the previous example.



(a) The path A (b) The path B (c) The path A' (d) The path B'

Figure 48: All the closed paths in this example are formed from the subpaths A, B, A', B'

All the closed paths in the second example are formed by connecting the subpaths

$$AB, AB', A'B, A'B'$$

and all of them are bidirectional or returning paths. I.e. all the closed paths here are returning paths. Therefore we can argue that the value of any infinitely returning path is 1.

Therefore, it seems that the unidirectional infinite paths must be treated as inconsistent while the returning infinite ones are treated as valid in general. However, there is a problem here. For example let the subpath A is an unidirectional infinite path in the returning path $..p..p'A$. In this case. the value of this path can not be 1 although it is returning because the value of a path is equal to its last part and the value of last part A is 0. On the other hand it can be shown that all infinite paths fall into the following three categories.

Definition 7 *1. An infinite path which is unidirectional.*

2. An infinite path whose one last part is unidirectional.

3. An infinite path which has no unidirectional last part, It is called an infinitely returning path.

So we can assume that the infinite paths belong to the first two categories are inconsistent while the ones belong the last category are valid.

8.2 A demonstration

Thus we have made many assumptions up to know. The followings are among them.

1. The value of an arc is equal to the disjunction of the values of the e-product scans starting from that arc.
2. The value of an e-product scan is the conjunction of the values of it's paths.
3. The value of a path is equal to the value of its last part.

In addition the these, we made the following assumptions for infinite paths.

1. The value of a unidirectional infinite path is 0.
2. The value of an infinitely returning path is 1.

Obviously an e-product is valid if it's all paths are valid and inconsistent if at least one path of it is inconsistent. A valid formula is a formula whose at least one e-product is valid and an inconsistent formula is a formula whose all e-products are inconsistent. Therefore if a formula is a tautology, we must find an a valid e-product scan in its last reduced form.

In a last reduced form, every path is either infinite or finite and terminates with either a 0 or 1. So if the a formula is a tautology, then its last reduced formula is a valid formula and if it is not a tautology, its last reduced form is inconsistent. From the above assumptions we can write the followings.

1. If each path of an e-product scan is either ends with 1 or infinitely returning then this is a valid e-product scan.
2. If one path of an e-product scan is either ends with 0 or the last part of it is unidirectional infinite then this is an inconsistent e-product scan.
3. If a formula is a tautology, then there is a valid e-product scan in its last reduced form.
4. If a formula is not a tautology, then every e-product scan of it is inconsistent in its last reduced form.

A tautology example For example, the formula given in the figure is a tautology and in the totally reduced form there must be a valid e-product.

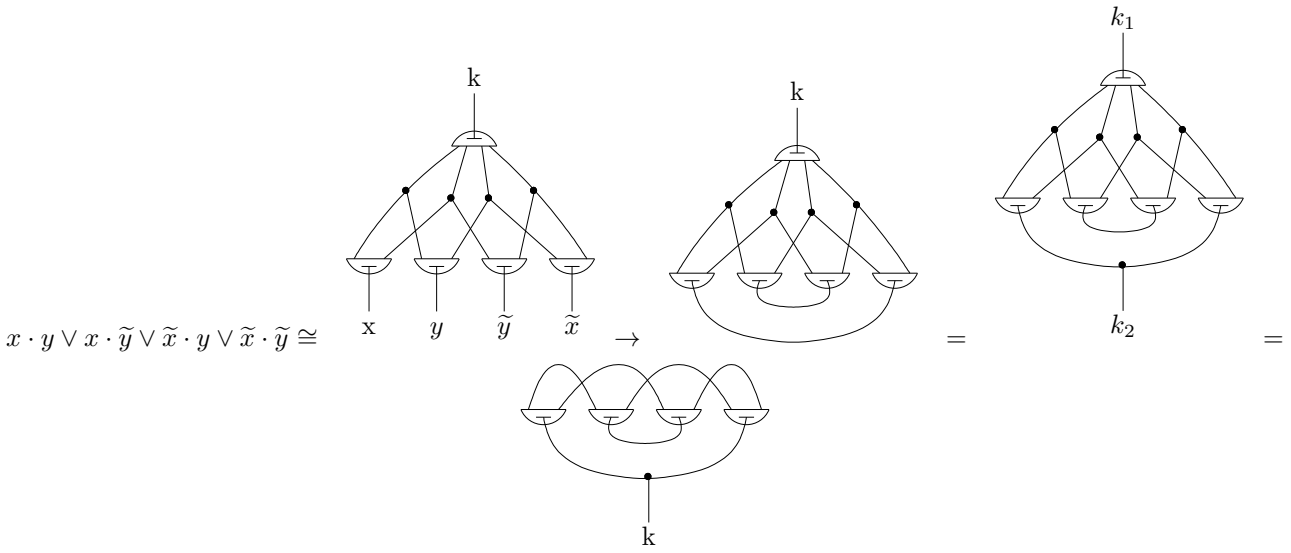
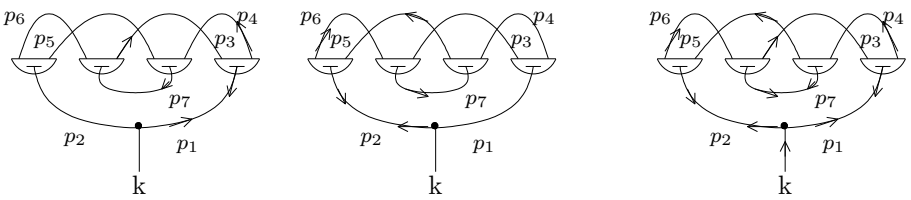


Figure 49: A valid formula and its reduced forms

As seen in the figure, there is an e-product scan whose infinite paths are in the form $\Rightarrow p(AB)^{\infty}$ or $\Rightarrow p(BA)^{\infty}$.



(a) $A = \Rightarrow p_1.p_4.p_7.p'_3.p'_1$ (b) $B = \Rightarrow p_2.p_6.p'_7.p'_5.p'_2$ (c) An e-product scan whose infinite paths are in the form $\Rightarrow p(AB)^{\infty}$ or $\Rightarrow p(BA)^{\infty}$

Figure 50: There is a valid e-product scan whose all paths are either infinitely returning or terminates with the root term k

These paths are infinitely returning paths. On the other hand, all the finite paths of this scan terminates with root term whose value is 1, so this is a valid e-product.

A non tautology example On the other hand, the formula in the figure is not a tautology. So, at least one path of every e-product in its last reduced form must be either unidirectional infinite or terminates with 0. However, the number of e-products here is infinite and checking this is not an easy task.

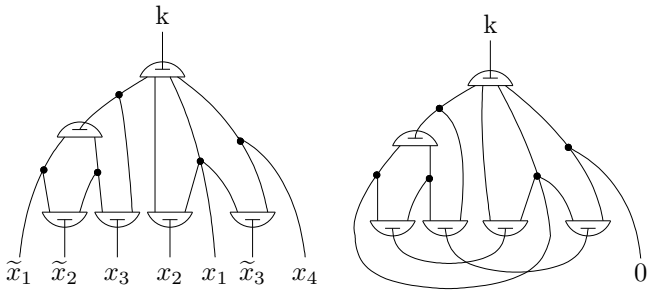
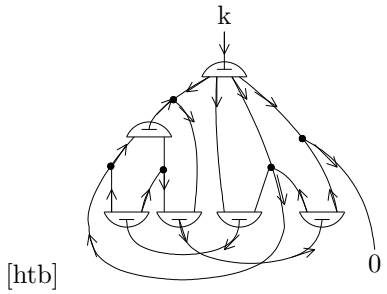


Figure 51: A nontautology formula and its last reduced form

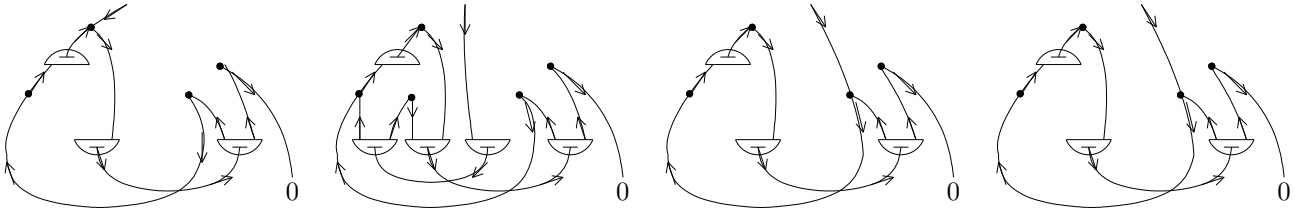
However, we can check not only e-product but e-sums also. we can write the followings abouth the valid and inconsistent e-sum scans..

1. If one path of an e-sum scan is either ends with 1 or infinitely returning then this is a valid e-sum scan.
2. If each path of an e-product scan is either ends with 0 or the last part of it is unidirectional infinite then this is an inconsistent e-sum scan.
3. If a formula is a tautology, then every e-sum scan of it is valid in its last reduced form.
4. If a formula is not a tautology, then at least one e-sum scan of it is inconsistent in its last reduced form.

So there must be at least one inconsistent e-sum scan in the last reduced form of the example formula. As seen, there is ane-sum scan in this formula whose every pat is either unidirectional infinite or ends with 0.



(a) An inconsistent e-sum scan



(b) All the paths of this e-sum scan either terminates with 0 or unidirectional infinite

Figure 52: There is an inconsistent e-sum scan in this formula

9 Further studies

Thus we have demonstrated some assumptions in 2d formulas which differentiate a tautology from a non tautology. However, this is not enough. To use them to find tautology in two dimensional formulas, we need to prove they are true for every case. I.e. we need to prove that they determine the true properties. This can be achieved by proving the following assertion under these assumptions.

"Let $2D(x_i)$ and $S(x_i)$ represent a 2d formula and its statement equivalence respectively. I.e. $2D(x_i) \cong S(x_i)$. If $2D(x_i) \stackrel{x_i}{\rightarrow} 2D()$ and $S(x_i) \stackrel{x_i}{\rightarrow} S()$ then $2D() \cong S()$"

Also, we must develop polynomial algorithms which decides whether a last reduced two dimensional formula is a tautology or not by using these properties. These two jobs would be the subject of the next articles.

References

[1] P. Lammens, L. Claesen, H. De Man, 1989, "Correctness Verification of VLSI Modules Supported by a Very Efficient Boolean Prover", Proceedings: IEEE International Conference on Computer Design, October 1989, pp266-269

[2] John E. Hopcroft, Jeffrey D. Ullman, "Introduction to Automata Theory, Languages and Computation",p.322,pp328-329, Addison-Wesley, 1979