

Doing Two-Level Logic Minimization 100 Times Faster

Olivier Coudert

Synopsys, Inc.
700 East Middlefield Road,
Mountain View, CA 94043-4033

Abstract

Two-level logic minimization consists in finding a minimal cost sum-of-products, i.e., disjunctive normal form, of a given Boolean function. This paper presents a new algorithm to solve this problem, and gives experimental evidences showing that it outperforms the leading minimizers of several orders of magnitude. We suspect that it may be possible to explain this improvement in performance theoretically, and hope that our empirical results will stimulate research along these lines.

1 Introduction.

Two-level logic minimization is a fundamentally important problem in computer science. It has applications in several fields, e.g., logic synthesis [2, 14], reliability analysis [12, 7], and automated reasoning [10, 15, 25, 20]. Since Quine states the problem of computing minimal sums-of-products in the 1950's [23], many efforts have been made to discover efficient minimization procedures [24, 16, 13, 2, 27, 28, 18, 14, 17].

A Boolean function f is a subset of $\{0,1\}^n$. A product p is a cartesian product $\times_{k=1}^n l_k$, where $l_k \subseteq \{0,1\}$. A set of products P covers a Boolean function f iff $(\bigcup_{p \in P} p) = f$. Let $Cost$ be a function mapping products onto real positive numbers.

DEFINITION 1. *The minimization of a Boolean function f consists in finding a set of products P that covers f and that minimizes $\sum_{p \in P} Cost(p)$ ¹.*

Clearly, 2-level logic minimization is a set covering problem (SCP in short). Let X and Y be two sets, R be a relation defined on $X \times Y$, and $Cost$ be a cost function defined on Y . One says that y covers x iff $x R y$.

DEFINITION 2. *The set covering problem $\langle X, Y, R \rangle$ consists of finding a subset S of Y that covers X and that minimizes $\sum_{y \in S} Cost(y)$.*

A product p is an *implicant* of the Boolean function f iff $p \subseteq f$. The set of *prime* implicants of f , which we will note $Prime(f)$, is the set of its maximal implicants w.r.t. \subseteq [23]. Assuming that $p \subseteq p' \Rightarrow Cost(p') \leq Cost(p)$ ², all minimal covers of f are made of prime implicants. Two-level minimization of f is thus the SCP $\langle f, Prime(f), \subseteq \rangle$.

It is convenient to represent a set covering problem $\langle X, Y, R \rangle$ with its *covering matrix*. Its rows are labeled with elements of X , and its columns are labeled with elements of Y . An element $[x, y]$ of the matrix is 1 if y covers x , otherwise it is 0. A solution of the covering matrix is a subset of columns that covers all the rows and that minimizes the cost function.

The Quine-McCluskey procedure, which is used world-wide to compute minimal sum-of-products, consists of the three following steps:

- (a) Compute all the prime implicants of f .
- (b) Build the covering matrix consisting of rows labeled by relevant minterms of f and of columns labeled by all the prime implicants of f .
- (c) Find a minimal cost subset of columns that cover all the rows using cyclic core computation and branch and bound.

The *cyclic core* of a SCP is the covering matrix obtained by repeatedly removing essential columns and dominated rows and columns. A y is essential if it is the only one that covers some x . An x is dominated if there is another $x' \neq x$ such that covering x' necessarily results in covering x . A y is dominated if it covers a subset of the elements covered by some $y' \neq y$ and $Cost(y') \leq Cost(y)$.

ESPRESSO-EXACT [2, 28] is an implementation of the Quine-McCluskey procedure, which uses several auxiliary techniques to reduce the covering matrix [16,

¹This definition can be extended to more complex structures as a paving problem. There are reductions of some paving problems to this particular problem [30, 9, 6].

²This constraint holds for real-life minimization problems because the more literals a product has, the more costly it is to implement.

11, 22, 26]. From the time of its introduction until 1993, it was the leading 2-level logic minimizer.

ESPRESSO-EXACT fails in minimizing a function either because the number of prime implicants is too large to be computed explicitly (indeed the number of prime implicants can be exponential [24, 4, 18]), or because the covering matrix it generates is too complex to be solved. In practice it can at best minimize Boolean functions that have no more than 20000 prime implicants.

In 1993, the minimizer ESPRESSO-SIGNATURE was introduced. It is able to generate a smaller covering matrix without necessarily computing all the prime implicants [17]. ESPRESSO-SIGNATURE can minimize any function that ESPRESSO-EXACT can, since the former generates a smaller covering matrix than the latter. Though ESPRESSO-SIGNATURE has been successfully used to minimize functions with huge numbers of prime implicants, it is still limited by the size of the covering matrix it must build.

SCHERZO, the minimizer this paper presents, differs from both ESPRESSO-EXACT and ESPRESSO-SIGNATURE in three fundamental ways.

First, SCHERZO uses two graph based data structures to represent boolean functions and sets of products, namely Binary Decision Diagrams (BDDs) and Combinational Sets (CSs). The reader is referred to Appendices A and B for a brief description of these data structures. Their main advantage is that there is no relation between the size of a BDD or a CS and the number of elements of the set they represent. Indeed, using such data structures, the complexity of the minimization problem is shifted, resulting in substantial performance gains in practice.

Second, in order to use these data structures, it was necessary to devise a new cyclic core computation algorithm that was capable of operating on them³. SCHERZO accomplishes this by lifting the set covering problem to a set covering problem on a lattice.

Third, in the branch-and-bound portion of the minimization, SCHERZO uses new lower bound based pruning techniques that typically reduce the number of recursions by a factor of 10 on examples that have a very large search space.

We have performed extensive empirical studies comparing the performance of SCHERZO against that of ESPRESSO-EXACT and ESPRESSO-SIGNATURE. We have found that SCHERZO is 10 to more than 100 times faster than the the others, and is able to solve much more complex problems that were open so far.

The rest of this paper is organized as follows. Sec-

tion 2 outlines the minimization algorithm of SCHERZO. Section 3 presents and discusses some experimental results comparing the three minimizers. As a conclusion we formulate some open theoretical questions that could help explain the effectiveness of SCHERZO.

2 Scherzo's Minimization Algorithm.

For the sake of simplicity, *Cost* is assumed to be a constant function. We note \mathcal{P} the set of all products, and $Sol(C)$ the set of solutions of a SCP C . SCHERZO's minimization procedure is as follows. More details and a proof of correctness can be found in Appendix C.

- (a) Compute the CS P of $Prime(f)$.
- (b) Compute the CS Q of $\{q \in \mathcal{P} \mid x \in f, \{x\} = q\}$.
- (c) Compute the fixpoint, say C , produced by the following rewriting rules on the *implicit* set covering problem $\langle Q, P, \subseteq \rangle$,

$$\langle Q, P, \subseteq \rangle \rightarrow \langle \max_{\subseteq} \tau_P(Q), \max_{\subseteq} \tau_Q(P), \subseteq \rangle$$

$$\langle Q, P, \subseteq \rangle \rightarrow \langle Q - E, P - E, \subseteq \rangle, \quad \text{with } E = Q \cap P$$

where τ_P and τ_Q are defined from \mathcal{P} into \mathcal{P} by:

$$\tau_Q(r) = \sup_{\subseteq} \{q \in Q \mid q \subseteq r\}$$

$$\tau_P(r) = \inf_{\subseteq} \{p \in P \mid r \subseteq p\}$$

- (d) Solve the resulting fixpoint C using branch and bound and cyclic core computation.
- (e) Let F be the union of the sets E found during the computation of the fixpoint C . Then the set of all solutions of the 2-level logic minimization of f is:

$$\bigcup_{S \in Sol(C)} \times_{r \in S \cup F} \{p \in Prime(f) \mid r \subseteq p\}$$

This minimization algorithm is very different from all the previous ones. First, it uses two endomorphisms τ_P and τ_Q to capture dominance relations and compute the fixpoint C , which is indeed isomorphic to the cyclic core (see Appendix C.2.1). Second, CSs are used to represent sets of products, which makes the cost of the cyclic core computation independent of the number of products it needs to manipulate, in particular the number of prime implicants.

The CS based recursive algorithms needed for steps (a) to (c) can be found in [8]. We rather describe how the branch-and-bound is done, and present a new pruning technique that dramatically reduces in practice the search space one has to explore at step (d). We assume that $R = \subseteq$ to lighten the notation.

³The straightforward approach that mimics the Quine-McCluskey procedure does not work in practice [32].

2.0.1 Branching. If the cyclic core is empty, the minimization is done, and the minimal solutions can be recovered using step (e).

If the cyclic core $C = \langle X, Y, \in \rangle$ is not empty, an element y is chosen to generate the two set covering problems $\langle X - y, Y - \{y\}, \in \rangle$ and $\langle X, Y - \{y\}, \in \rangle$. The former assumes that y belongs to the minimal solution, and the later assumes that y does not belong to the minimal solution. These two problems are recursively solved and the best solution is kept. Heuristics to properly choose a branching element y are discussed in [16, 2, 28].

2.0.2 Lower Bound. Let $C = \langle X, Y, \in \rangle$ be a set covering problem produced at some point of the binary search tree. Let $C.min$ be the cost of a minimal solution of C , and $C.path$ be the cost of the path that yields C , i.e., the sum of the costs of all y 's that have been considered as belonging to the minimal solution until producing C . Let $C.upper$ be the best global solution found so far in the search tree, and $C.lower$ be a lower bound of $C.min$. Then C can be pruned as soon as $C.path + C.lower \geq C.upper$. Consequently it is critical to compute an accurate lower bound to terminate useless searches as early as possible.

Let X' be a subset of X such that for any two different elements x_1 and x_2 of X' , all elements y that cover x_1 do not cover x_2 and conversely. The set X' , which is called an *independent set* of C , provides us with the lower bound

$$C.lower = \sum_{x \in X'} \min_{y \ni x} Cost(y),$$

since this is the minimum cost necessary to cover the elements of X' . Though finding an independent subset that maximizes this lower bound is an NP-complete problem, heuristics used to build an independent set X' yield in practice fairly good lower bounds [28].

2.0.3 The Limit Lower Bound Theorem. This result is the most important in practice. It reduces the number of recursions as well as the CPU time by more than 100 on complex examples that have a very large search space.

THEOREM 2.1. *Let $C = \langle X, Y, \in \rangle$ be a set covering problem, and X' be an independent set of C . Let $C.lower = \sum_{x \in X'} \min_{y \ni x} Cost(y)$ the lower bound of C obtained thanks to X' . Let Y' be the set of y 's that do not cover any element of X' , and such that $C.path + C.lower + Cost(y) \geq C.upper$. Then C can be reduced to $\langle X, Y - Y', \in \rangle$.*

Reducing $\langle X, Y, \in \rangle$ to $\langle X, Y - Y', \in \rangle$ prevent us from looking at $|Y'|$ unsuccessful branches, which can yield

a $O(2^{|Y'|})$ gain. In practice the use of the limit lower bound makes the recursion terminate immediately, i.e., the lower bound of $\langle X, Y - Y', \in \rangle$ nearly always exceeds the upper bound, or a better solution is found.

3 Experimental Results.

We compare 3 two-level logic minimization methods: the Quine-McCluskey based procedure ESPRESSO-EXACT [28], ESPRESSO-SIGNATURE that is similar to ESPRESSO-EXACT excepts that ESPRESSO-SIGNATURE generates a smaller covering matrix [17], and SCHERZO, outlined in Section 2.

The 134 examples of the MCNC benchmark [33] are used for years as a wide range of representative Boolean functions in VLSI. ESPRESSO-EXACT can minimize 114 of these examples, that we will call the “easy” examples. It failed to minimize the 20 remaining examples, 17 of them because it cannot compute the prime implicants, and the other 3 ones because it cannot solve the covering matrix though it succeeded in building it. We will call these 20 examples the “hard” examples.

To solve the 114 easy examples on a DEC 5000, ESPRESSO-EXACT needs 9264 seconds [29], ESPRESSO-SIGNATURE 7777 seconds [29], and SCHERZO only 471 seconds. SCHERZO is 16 times faster than ESPRESSO-SIGNATURE and 19 times faster than ESPRESSO-EXACT on this set of benchmarks. Table 1 shows the results obtained on 33 examples representing 90% of the cumulative time for solving the 114 easy examples.

Table 2 compares the results of ESPRESSO-SIGNATURE reported in [17] and those of SCHERZO on the 20 hard examples. The prime implicants are computed thanks to the CS based implicit method introduced in [5]. We clearly see it is not limited by the huge number of prime implicants of some functions, thanks to the implicit representation of sets of products it uses.

While both ESPRESSO-EXACT and ESPRESSO-SIGNATURE can fail to generate a covering matrix because of its too large size, SCHERZO is not limited by the size of the covering matrix since it computes it (indeed the cyclic core) in an *implicit* way⁴. For instance, ESPRESSO-SIGNATURE is unable to build a covering matrix for *ti*, *mainpla*, and *soar*, because the number of rows ESPRESSO-SIGNATURE needs to produce is 17716, 10364, and 6707717 respectively⁵. SCHERZO is 48 times faster than ESPRESSO-SIGNATURE on the 14 hard exam-

⁴As far as we looked from both the practical and theoretical point of view, we did not find a function that could be treated by ESPRESSO-SIGNATURE but not by SCHERZO. This is one of the open problems stated in the conclusion.

⁵We can prove that the number of rows of the matrix generated by ESPRESSO-SIGNATURE is equal to $|\max_{\subseteq} \tau_P(Q)|$, which can be computed linearly on CSs.

Name	#var	#P	B	ESP – EXCT	ESP – SIGN	SCHZ
<i>add6</i>	19	8568	355	234	78.44	0.5
<i>al2</i>	63	9179	66	349	324.89	4.3
<i>alcom</i>	53	4657	40	97	160.09	2.6
<i>b2</i>	33	928	104	14	47.76	3.9
<i>b9</i>	21	3002	119	27	7.93	1.0
<i>bc0</i>	37	6596	177	643	302.99	18.2
<i>bca</i>	72	305	180	267	308.62	8.3
<i>becb</i>	65	255	155	79	69.07	5.3
<i>bcd</i>	64	172	117	56	49.16	2.0
<i>ex7</i>	21	3002	119	31	7.88	1.0
<i>exep</i>	93	558	108	30	160.28	12.3
<i>in1</i>	33	928	104	15	47.54	4.0
<i>in3</i>	64	1114	74	25	11.65	3.3
<i>in6</i>	56	6174	54	191	1.66	2.8
<i>in7</i>	36	2112	54	44	2.53	1.1
<i>prom1</i>	49	9326	472	1698	790.12	86.7
<i>Z9sym</i>	10	1680	84	63	83.15	4.5
<i>addm4</i>	17	1122	189	9	27.42	5.6
<i>b3</i>	52	3056	210	168	38.63	4.5
<i>bcc</i>	71	237	137	123	76.37	6.1
<i>cps</i>	133	2487	157	111	816.77	38.6
<i>exps</i>	46	852	132	11	40.12	9.1
<i>in4</i>	52	3076	211	134	35.14	5.3
<i>intb</i>	23	6522	629	270	567.94	28.2
<i>lin.rom</i>	43	1087	128	753	755.85	55.2
<i>m181</i>	24	1636	41	39	10.92	2.1
<i>mlp4</i>	16	606	121	21	29.91	5.0
<i>mp2d</i>	28	469	30	16	27.00	3.4
<i>pope.rom</i>	54	593	59	15	13.97	11.5
<i>spla</i>	72	4972	248	209	714.10	25.1
<i>sym10</i>	11	3150	210	386	463.29	18.1
<i>t1</i>	43	15135	100	2302	801.83	6.0
<i>tial</i>	22	7145	575	319	685.70	36.3
Total	1496	110701	5559	8749	7558.7	421.9
Mean	45	3354	168	265.1	229.0	11.5
Ratio				20.7	17.9	1.0

#var / #P: #variables / #prime implicants of the Boolean function.

B : size of the minimal sum-of-products of the Boolean function.

Table 1. 33 examples with ESPRESSO-EXACT, ESPRESSO-SIGNATURE, and SCHERZO (CPU time on a DEC 5000).

ples the later can solve. Moreover, SCHERZO is able to minimize the 6 remaining hard examples that were open problems so far.

Table 3 gives the results of SCHERZO on some other hard examples. None of them can be handled by ESPRESSO-EXACT because of the too large number of prime implicants. ESPRESSO-SIGNATURE cannot handle examples with a “*” because of the too large size of the covering matrix it would generate. For instance, *rip08*, *s382*, *seq*, and *addsub* would have respectively 11131, 29455, 44374, and 33270071 rows.

Figure 1 shows the \log_{10} of the CPU time required by the three minimizers on 45 examples of increasing complexity. We clearly see the double expo-

ponential time complexity of both ESPRESSO-EXACT and ESPRESSO-SIGNATURE, due to the fact that when treating a function of n variables, the covering matrix can be in $\Omega(3^n/n)$, and solving it is NP-complete. SCHERZO has a completely different behavior. Its exponential growing is expected to occur on much more complex examples.

4 Conclusion.

Two-level logic minimization is a fundamental problem that applies in logic synthesis, reliability analysis, and automated reasoning. The widely used Quine-McCluskey algorithm (e.g., ESPRESSO-EXACT) is limited by the number of prime implicants of the Boolean function to be minimized. The more recent procedure

Name	Features			ESPRESSO – SIGNATURE			SCHERZO		
	#var	#P	B	row × col	Tmat	Time	row × col	Tcc	Time
<i>accpla</i>	119	1758057	175	385 × 768	<i>n/a</i>	7670	8 × 8	430	431
<i>ex4</i>	156	1.83 e+14	279	509 × 838	<i>n/a</i>	163.2	20 × 20	6.6	6.7
<i>ibm</i>	65	1.04 e+09	173	173 × 174	<i>n/a</i>	1.6	0 × 0	8.4	8.4
<i>jbp</i>	93	2496809	122	5192 × 37644	<i>n/a</i>	5891	324 × 209	52.3	72.2
<i>misg</i>	79	6.49 e+09	69	134 × 200	<i>n/a</i>	13.9	0 × 0	7.3	7.3
<i>mish</i>	137	1.12 e+15	82	160 × 239	<i>n/a</i>	49.2	0 × 0	9.6	9.6
<i>misj</i>	49	139103	35	79 × 101	<i>n/a</i>	2.1	0 × 0	2.7	2.7
<i>pdc</i>	56	23231	96	6550 × 18923	<i>n/a</i>	9759	1498 × 1141	91.1	106.8
<i>shift</i>	35	165133	100	100 × 100	<i>n/a</i>	0.2	0 × 0	4.1	4.1
<i>signet</i>	47	78735	119	132 × 153	<i>n/a</i>	55.3	0 × 0	56.1	56.1
<i>ts10</i>	38	524280	128	128 × 128	<i>n/a</i>	0.3	0 × 0	4.5	4.5
<i>x2dn</i>	138	1.14 e+16	104	846 × 2006	<i>n/a</i>	223.8	0 × 0	8.8	8.8
<i>x7dn</i>	81	566698631	538	2602 × 5966	<i>n/a</i>	2478	1968 × 664	51.3	67.2
<i>xparc</i>	114	15039	254	1843 × 2974	<i>n/a</i>	12726	224 × 149	23.8	26.3
Total						39033.6			811.7
Ratio						48.0			1.0
<i>ti</i>	119	836287	213*	–	> 40h	–	874 × 430	28.2	34.1
<i>mainpla</i>	81	87692	172*	–	> 40h	–	0 × 0	1480	1480
<i>soar</i>	177	3.30 e+14	352*	–	> 40h	–	22540 × 8927	421.1	1638
<i>prom2</i>	30	2635	287*	1763 × 2604	282.5	> 40h	1524 × 1813	60.2	5.5h
<i>max1024</i>	16	1278	259*	1054 × 1278	47.4	> 40h	916 × 904	5.5	20.3h
<i>ex5</i>	71	2532	65*	795 × 2451	123.2	> 40h	686 × 974	64.0	34.4h

#var : #variables of the Boolean function.

#P : #prime implicants of the Boolean function.

B : size of the minimal sum-of-products of the Boolean function.

***** : indicates a previously open problem.

row × col: size of the yielded covering matrix.

Tmatrix : time needed by ESPRESSO-SIGNATURE to yield its covering matrix.

Tcc : time needed by SCHERZO to yield the cyclic core.

Time : total minimization time.

Table 2. Hard examples with ESPRESSO-SIGNATURE and SCHERZO (CPU time on DEC 5000).

ESPRESSO-SIGNATURE is still limited by the size of the covering matrix it produces. We have outlined a new BDD/CS based algorithm that *implicitly* compute the cyclic core of the set covering problem through endomorphisms. This makes the minimizer SCHERZO no longer limited by the size of the covering matrix of the function being treated. We have also presented a very efficient recursion pruning technique that dramatically reduces the search space. These different techniques enables SCHERZO to be from 10 to more than 100 times faster than the best methods known so far, and allows the 2-level minimization of functions that are out of reach of any other algorithm.

It is significant to notice that the more complex the problem is (because of the number of prime implicants of the function, the size of its covering matrix, or the inherent difficulty of solving the covering matrix), the greater the efficiency of SCHERZO is compared to the one of ESPRESSO-EXACT or ESPRESSO-SIGNATURE. SCHERZO

breaks the smooth evolution of the performances of minimizers for the last 20 years. However it is very difficult to explain theoretically this efficiency. We believe that this efficiency is primarily due to a shifting of the complexity thanks to the BDD and CS data structures. This raises the following open questions:

- (1) The worst case complexity of combining two BDDs by a binary operator is quadratic. What is the average complexity? We suspect it is quasi-linear.
- (2) What is the relation between the size of a sum-of-products and the size of the BDD of the function it represents? We suspect that there exist functions with an $O(n)$ size sum-of-products, but with non-polynomial size BDDs for all variable orderings.
- (3) What is the relation between the size of a BDD and the size of the CS of the set of its prime implicants?
- (4) Does it exist a function that can be polynomially minimized by ESPRESSO-SIGNATURE but not by

Name	#var	#P	B	Time
cont	58	17060286	229	1.9
s298	37	106307	58	1.4
s420	53	125924	58	2.1
s526	51	21523813	130	9.2
pitch	64	27560	95	20.5
add8	26	338732	2519	0.1
alupla	30	25677	2144	37.5
cbp16	50	$6.87 e+10$	655287*	0.4
cbp32	98	$2.84 e+21$	42949672823*	1.0
s382	51	11142527	150*	27.7
seq	128	$9.85 e+09$	472*	62.4
addsub	46	$3.60 e+09$	163649*	171
rip08	25	182893	1499*	1.9

Table 3. Results of SCHERZO on other hard examples.

SCHERZO? More precisely, let f be a Boolean function, and $\sigma(x) = \bigcap_{p \in \text{Prime}(f), p \ni x} p$ the *signature cube* of x [17]. The set of rows that ESPRESSO-SIGNATURE generates is $\max_{\subseteq} \sigma(f)$. Does it exist a function f such that $|\max_{\subseteq} \sigma(f)|$ is polynomial and such that f has a non polynomial size BDD for any variable ordering⁶?

- (5) What is the average gain produced by the limit lower bound when exploring the binary search tree?

Acknowledgment.

The author would like to thank Anna R. Karlin for her helpful comments on this paper, and Jean-Christophe Madre and Hervé Touati that helped develop SCHERZO.

References

- [1] S. B. Akers, “Binary Decision Diagrams”, *IEEE Trans. on Comp.*, 27:509–516, 1978.
- [2] R. K. Brayton, G. D. Hachtel, C. T. McMullen, A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Ac. Pub., 1984.
- [3] R. E. Bryant, “Graph-Based Algorithms for Boolean Functions Manipulation”, *IEEE Trans. on Comp.*, 35:8:677–692, August 1986.
- [4] A. K. Chandra, G. Markowsky, “On the Number of Prime Implicants”, in *Dis. Math.*, 24:7–11, 1978.
- [5] O. Coudert, J. C. Madre, “Implicit and Incremental Computation of Primes and Essential Primes of Boolean Functions”, in Proc. of *29th Design Aut. Conf.*, Anaheim CA, June 1992.
- [6] O. Coudert, J. C. Madre, “A New Graph Based Prime Computation Technique”, in *Logic Synthesis and Optimization*, T. Sasao Ed., Kluwer Ac. Pub., pp. 33–57, 1993.
- [7] O. Coudert, J. C. Madre, “Fault Tree Analysis: 10^{20} Prime Implicants and Beyond”, in Proc. of *Annual Reliability and Maintainability Symp.*, pp. 240–245, Atlanta GA, January 1993.
- [8] O. Coudert, J. C. Madre, H. Fraise, “A New Viewpoint on Two-Level Logic Minimization”, in Proc. of *30th Design Aut. Conf.*, Dallas TX, June 1993.
- [9] R. B. Cutler, S. Muroga, “Useless Prime Implicants of Incompletely Specified Multiple-Output Switching Functions”, in *Int’l Jour. of Comp. and Inf. Sci.*, V. 9-4, 1980.
- [10] J. Doyle, “A Truth Maintenance System”, in *Art. Int.*, 12:231–271, 1979.
- [11] J. F. Gimpel, “A Reduction Technique for Prime Implicant Tables”, in *IEEE Trans. on Elec. Comp.*, 14:535–541, 1965.
- [12] D. F. Hasl, “Advanced Concepts in Fault Tree Analysis”, in Proc. of *System Safety Symp.*, Seattle WA, June 1965.
- [13] S. J. Hong, R. G. Cain, D. L. Ostapko, “MINI: A Heuristic Approach for Logic Minimization”, in *IBM Jour. R&D*, pp. 443–458, 1974.
- [14] S. J. Hong, S. Muroga, “Absolute Minimization of Completely Specified Switching Functions”, in *IEEE Trans. on Comp.*, 40:53–65, 1991.
- [15] J. De Kleer, B. C. Williams, “Diagnosing Multiple Faults”, in *Art. Int.*, 32:97–130, 1987.

⁶Here the size of the BDD is the bottleneck, because we can easily show that for any set of products P made of m products built on n variables, the size of its CS is bound by $O(n \times m)$, whatever is its variable ordering.

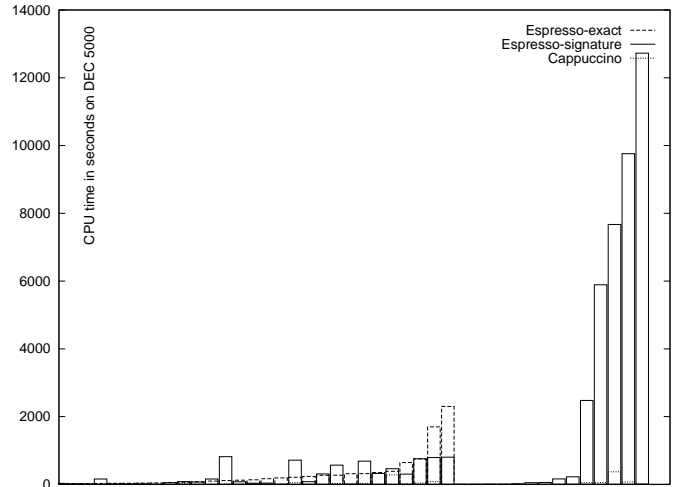


Figure 1. ESPRESSO-EXACT, ESPRESSO-SIGNATURE, and SCHERZO.

- [16] E. L. Jr. McCluskey, "Minimization of Boolean Functions", in *Bell Sys. Tech. Jour.*, 35:1417-1444, April 1959.
- [17] P. C. McGeer, J. Sanghavi, R. K. Brayton, A. L. Sangiovanni-Vincentelli, "ESPRESSO-SIGNATURE: A New Exact Minimizer fo Logic Functions", in *IEEE Trans. on VLSI*, 1-4:432-440, Dec. 1993.
- [18] C. McMullen, J. Shearer, "Prime Implicants, Minimum Covers, and the Complexity of Logic Simplification", in *IEEE Trans. on Comp.*, 35:761-762, Aug. 1986.
- [19] J. C. Madre, J. P. Billon, "Proving Circuit Correctness using Formal Comparison Between Expected and Extracted Behaviour", in Proc. of the *25th Design Aut. Conf.*, Anaheim CA, July 1988.
- [20] J. C. Madre, O. Coudert, "A Logically Complete Reasoning Maintenance System Based on Logical Constrain Solver", in Proc. of *Int'l Join Conf. on Art. Int.*, Sydney, Australia, August 1991.
- [21] S. Minato, "Zero-Supressed BDDs for Set Manipulation in Combinatorial Problems", in Proc. of *30th Design Aut. Conf.*, Dallas TX, June 1993.
- [22] I. B. Pyne, E. L. McCluskey, "An Essay on prime Implicant tables", *SIAM*, 9:604-632, 1961.
- [23] W. V. O. Quine, "The problem of Simplifying Truth Functions", in *Am. Math. Month.*, 59:521-531, 1952.
- [24] W. V. O. Quine, "On Cores and Prime Implicants of Truth Functions", in *Am. Math. Month.*, 66:755-760, 1959.
- [25] R. Reiter, J. de Kleer, "Foundations for Assumption-Based Truth Maintenance Systems" in Proc. of *AAAI National Conf. '87*, pp. 183-188, Seattle WA, July 1987.
- [26] S. Robinson, R. House "Gimpel's Reduction Technique Extended to the Covering Problem With Costs", in *IEEE Trans. on Elec. Comp.*, 16:509-514, Aug. 1967.
- [27] R. L. Rudell, A. L. Sangiovanni-Vincentelli, "Multiple Valued Minimization for PLA Optimization", in *IEEE Trans. on CAD*, 6-5:727-750, Sept. 1987.
- [28] R. L. Rudell, *Logic Synthesis for VLSI Design*, PhD thesis, UCB/ERL M89/49, 1989.
- [29] J. Sanghavi, personal communication, June 1994.
- [30] T. Sasao, "An Application of Multiple-Valued Logic to a Design of Programmable Logic Arrays", Proc. of *Int'l Symp. on Multiple-Valued Logic*, 1978.
- [31] C. E. Shannon, "The Synthesis of Two-Terminal Switching Function", in *Bell Sys. Tech. Jour.*, 28-1:59-98, 1949.
- [32] G. M. Swamy, P. McGeer, R. K. Brayton, "A Fully Quine-McCluskey Procedure using BDD's", in Proc. of *Int'l Workshop on Logic Synthesis*, Lake Tahoe CA, May 1993.
- [33] S. Yang, *Logic Synthesis and Optimization Benchmarks User Guide*, Microelectronics Center of North Carolina, January 1991.

A Binary Decision Diagrams.

Let $f(x_1, \dots, x_n)$ be a Boolean function. We note f_{x_k} (respectively $f_{\overline{x_k}}$) the function obtained by setting variable x_k to 1 (respectively to 0). Shannon tree of f is the binary tree obtained by recursively decomposing f as $f = \overline{x_k}f_{\overline{x_k}} + x_kf_{x_k}$ for all variables x_k in a given order [31]. Figure 2 shows the Shannon tree of the function $x_1(x_3 \oplus x_4) + x_2(x_3 \Leftrightarrow x_4)$. The value of the function for any assignment of its variables can be found by following the path that this assignment defines in the graphs, for instance the path defined by the assignment $x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0$ is marked with a dotted line (left branch corresponds to an assignment to 0, right branch to 1).

The Binary Decision Diagram (BDD) of a Boolean function $f(x_1, \dots, x_n)$ is a binary directed acyclic graph that is a compacted representation of f 's Shannon tree [1, 3]. It can be obtained from the Shannon tree by sharing isomorphic graphs, and eliminating vertices that have the same left and right branches (for example vertices marked with a "*" in Figure 2).

The size of the BDD of a formula f , noted $|f|$, depends heavily on the variable ordering that has been chosen to build this graph [3]. For instance the BDD of the formula $(x_1 \oplus x_{2n}) + (x_2 \oplus x_{2n-1}) + \dots + (x_n \oplus x_{n+1})$ has $\Theta(n)$ vertices with the variable ordering $x_1 < x_{2n} < x_2 < x_{2n-1} < \dots < x_{n-1} < x_n$, but has $\Theta(2^n)$ vertices with the variable ordering $x_1 < x_2 < \dots < x_{2n}$. Moreover, there exist formulas that do not have polynomial size BDDs for any variable ordering.

The binary Boolean operators can be evaluated with a quadratic time and memory worst case complexity on BDDs built with the same variable ordering [3]. Negation can be evaluated in linear time on BDDs, and in constant time on BDDs using typed edges to denote the negation [19]. BDDs are a very interesting representation of Boolean functions and sets because there is *no relation at all* between the number of elements in a set

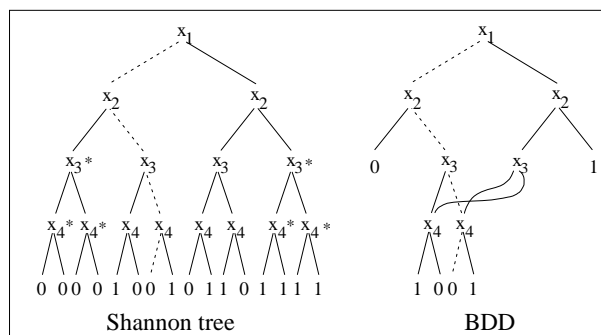


Figure 2. Shannon tree and BDD of the formula $x_1(x_3 \oplus x_4) + x_2(x_3 \Leftrightarrow x_4)$.

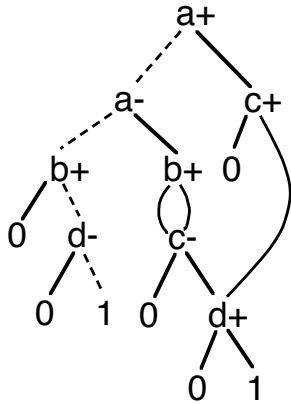


Figure 3. CS of the set of products $\{\bar{b}\bar{d}, \bar{a}\bar{b}\bar{c}d, acd\}$.

and the size of the BDD that denotes this set through its characteristic function, so that huge sets can potentially be represented by small BDDs [5].

B Combinational Sets

A set of products is nothing but a set of combinations of the literals $\{x_1, \bar{x}_1, \dots, x_n, \bar{x}_n\}$. Moreover, when dealing with prime implicants, most of the literals (indeed at least half of them) do not occur in the combinations. So we are interested in representing *sparse* sets of combinations.

CSs is another compacted representation of f 's Shannon tree, where f is the characteristic function of the set we one represents. CSs differs from BDDs by the compaction rules that are especially effective when dealing with sparse sets. There are: sharing all isomorphic graphs, and eliminating vertices whose left hand side branch is 0 [21].

Figure 3 shows the CS of the set of products $P = \{\bar{b}\bar{d}, \bar{a}\bar{b}\bar{c}d, \bar{a}\bar{b}\bar{c}d, acd\}$. We associate the variable x^+ (respectively x^-) with each literal x (respectively \bar{x}). Every path from the root of this graph to the leaf "1" can be interpreted to produce an assignment of the variables x^+ and x^- that defines a product belonging to P . For instance, the dotted path in this CS yields an assignment that correspond to the product $\bar{b}\bar{d}$.

Set operations can be performed on CSs with a cost related to the size of the CSs, but not to the number of combinations of the combinational sets denoted by the CSs. Union, intersection, and set difference can be performed on CSs with a quadratic worst case complexity, as disjunction and conjunction for BDDs. Complementation is also quadratic on CS, while the negation on BDDs can be done in constant time. Attributed edges can also be used to indicate whether or not the empty set belongs to a combinational set. This allows to add or remove the empty set from a combinational

set in constant time, which is very useful when combinational sets represent sets of products. It is also very useful for implementing low cost set operations. The main advantage of CSs compared to BDDs is their efficiency to represent sets of sparse combinations. This explain why CSs are particularly effective when addressing prime cover computation problems.

C Correctness of the Minimization Algorithm.

SCHERZO's minimization algorithm is an instance of a more general minimization algorithm dealing with SCP over a lattice. This appendix first introduces a few notations and definitions. It then details and proves an algorithm that compute the cyclic core of a SCP over a lattice. Applying this result to the lattice (\mathcal{P}, \subseteq) , where \mathcal{P} is the set of all products, proves the correctness of the SCHERZO's minimization algorithm outlined in Section 2.

C.1 Definition of the Cyclic Core. The concepts of essential columns and of dominance relations have been introduced to reduce the size of a covering matrix by removing some rows and columns [16, 22, 26]. The *cyclic core* is the smallest SCP that can be achieved when iterating these reductions. However the literature [16, 2, 27] exposes cyclic core computation is such a way that the set covering problem it yields is only *weakly* equivalent (see below). In this Section we formalize the cyclic core that guarantees a *strong* equivalence. The results presented here are valid for any function *Cost*.

A SCP C_2 is *weakly equivalent* to a SCP C_1 if there exists a recursive function μ such that $\mu(\text{Sol}(C_2)) \subseteq \text{Sol}(C_1)$. We say that C_1 and C_2 are *strongly equivalent* if there are two recursive functions that map $\text{Sol}(C_2)$ onto $\text{Sol}(C_1)$ and vice versa.

Let $\langle X, Y, R \rangle$ be a SCP. An element y of Y is *essential* iff it is the only one that covers some x . Obviously any minimal solution must contain the set of all essential elements.

To formalize dominance relations, we introduce the following notations. A *quasi order* is a reflexive and transitive relation, and is noted \prec . A *partial order* is a reflexive, transitive, and antisymmetric relation, and is noted \preceq . Given a quasi order \prec on a set Z , we note \equiv the equivalence relation defined by $(z \equiv z') \Leftrightarrow ((z \prec z') \wedge (z' \prec z))$. The projection of the quasi order \prec on the quotient set Z/\equiv is a partial order, which we denote with \preceq .

Let $\langle X, Y, R \rangle$ be a SCP, and *Cost* any cost function. The dominance relations that we will note \prec_Y and \prec_X in the following, are defined on X and Y respectively as follows:

$$x \prec_Y x' \Leftrightarrow \{y \in Y \mid x' R y\} \subseteq \{y \in Y \mid x R y\}$$

$$y \prec_X y' \Leftrightarrow \{x \in X \mid x R y\} \subseteq \{x \in X \mid x R y'\} \wedge \text{Cost}(y') \leq \text{Cost}(y)$$

The dominance relations \prec_Y and \prec_X are obviously quasi orders on X and Y respectively. With the concepts of essentiality and dominance, we are now ready to define the cyclic core.

DEFINITION 3. (CYCLIC CORE) *The cyclic core of a SCP is the fixpoint yielded by the following rewriting rules⁷:*

$$\begin{aligned} \langle X, Y, R \rangle &\rightarrow \langle \max_{\preceq_Y}(X/\equiv_Y), \max_{\preceq_X}(Y/\equiv_X), R \rangle \\ \langle X, Y, R \rangle &\rightarrow \langle X - \{x \in X \mid \exists y \in E, x R y\}, Y - E, R \rangle \end{aligned}$$

where E is the set of essential elements of $\langle X, Y, R \rangle$.

We can show that a SCP is strongly equivalent to its cyclic core. However recovering the original solutions from the cyclic core's ones is fairly complicated. This is why the "usual" cyclic core definition, e.g., the one used by ESPRESSO-EXACT and ESPRESSO-SIGNATURE, does not guarantee a strong equivalence. Basically, it consists in representing an equivalence class with one of its elements, which makes the minimization procedures lose some minimal solutions.

C.2 Set Covering Problem Over Lattices. We now present an original result that applies on set covering problem over a lattice (Z, \sqsubseteq) . Applying this result to the lattice (\mathcal{P}, \subseteq) proves the correctness of SCHERZO's minimization algorithm presented Section 2.

A SCP $\langle X, Y, \sqsubseteq \rangle$, where both X and Y are subsets of a lattice (Z, \sqsubseteq) ⁸, is the most general set covering problem in the sense that any SCP $\langle U, V, R \rangle$ can be rewritten into an equivalent SCP over a lattice. For let $X = \{\{u\} \mid u \in U\}$ and $Y = \{\{u \in U \mid u R v\} \mid v \in V\}$. Both X and Y are subsets of the lattice $(2^U, \subseteq)$, and clearly $x \subseteq y \Leftrightarrow u R v$, which proves that $\langle X, Y, \sqsubseteq \rangle$ is isomorphic to $\langle U, V, R \rangle$.

The cost function is assumed here to be constant. One can generalize to any cost function but this requires technical details that are irrelevant for this presentation.

⁷The relation R is modified in the usual way on the quotient sets, and the cost of an equivalence class is the cost of any of its element.

⁸A complete lattice (Z, \sqsubseteq) is a partially ordered set such that any non empty subset of Z has a greatest lower bound and a least upper bound with respect to \sqsubseteq , respectively noted \inf_{\sqsubseteq} and \sup_{\sqsubseteq} . Note that since (Z, \supseteq) is also a lattice, the principle of duality implies that, if a statement is true on (Z, \sqsubseteq) , then the dual statement obtained by interchanging \sqsubseteq and \supseteq can also be deduced. For instance $\inf_{\sqsubseteq} = \sup_{\supseteq}$.

THEOREM C.1. *Let $\langle X, Y, \sqsubseteq \rangle$ be a SCP over the complete lattice (Z, \sqsubseteq) . The fixpoint C obtained by applying the following rewriting rules is isomorphic to the cyclic core of the original SCP.*

$$\begin{aligned} \langle X, Y, \sqsubseteq \rangle &\rightarrow \langle \max_{\sqsubseteq} \tau_{Y, \supseteq}(X), \max_{\sqsubseteq} \tau_{X, \sqsubseteq}(Y), \sqsubseteq \rangle \\ \langle X, Y, \sqsubseteq \rangle &\rightarrow \langle X - E, Y - E, \sqsubseteq \rangle, \quad \text{with } E = X \cap Y \end{aligned}$$

where $\tau_{W, \sqsubseteq}$ is the function defined from Z into Z by:

$$\tau_{W, \sqsubseteq}(z) = \sup_{\sqsubseteq} \{w \in W \mid w \sqsubseteq z\}$$

Moreover we have:

$$\text{Sol}(\langle X, Y, \sqsubseteq \rangle) = \bigcup_{S \in \text{Sol}(C)} \bigtimes_{z \in \text{SUF}} \{y \in Y \mid z \sqsubseteq y\}$$

where F is the union of the sets E found when using the second rewriting rule during the fixpoint computation.

We prove Theorem C.1 in three steps. We first show that $\tau_{W, \sqsubseteq}$ is a morphism that captures dominance relations on both X and Y , which justifies the first rewriting rule. We then prove that the second rewriting rule consists in computing the set E of essential elements of Y and applying the corresponding reduction. Finally we prove that the minimal solutions of the original problem can be recovered as it is described.

C.2.1 The Transposing Function $\tau_{W, \sqsubseteq}$

DEFINITION 4. (TRANSPOSING FUNCTION) *Let (T, \prec) be a quasi ordered set. A transposing function τ is a morphism⁹ that maps (T, \prec) onto a partially ordered set $(\tau(T), \sqsubseteq)$, such that $\tau(t) \sqsubseteq \tau(t')$ iff $t \prec t'$.*

If τ is a transposing function, then $(T/\equiv, \preceq)$ is isomorphic to $(\tau(T), \sqsubseteq)$. We obtain the following commutative diagram:

$$\begin{array}{ccc} (T/\equiv, \preceq) & \longrightarrow & \max_{\preceq}(T/\equiv) \\ \nearrow & & \uparrow \\ (T, \prec) & & (\tau(T), \sqsubseteq) \\ \searrow \tau & & \downarrow \\ & & \max_{\sqsubseteq} \tau(T) \end{array}$$

The following lemma comes directly from the definition of $\tau_{W, \sqsubseteq}$.

LEMMA C.1. *Let (Z, \sqsubseteq) be a complete lattice, W be a subset of Z , and $\tau_{W, \sqsubseteq}$ be the function defined in Theorem C.1. We have:*

$$\begin{aligned} \forall z \in Z, \quad \tau_{W, \sqsubseteq}(z) \sqsubseteq z & \quad (3.1) \\ \forall z \in Z, \forall w \in W, \quad (w \sqsubseteq z) \Leftrightarrow (w \sqsubseteq \tau_{W, \sqsubseteq}(z)) & \quad (3.2) \end{aligned}$$

We are now ready to prove the following theorem.

⁹Most of the time it is not injective.

THEOREM C.2. *Let (Z, \sqsubseteq) be a complete lattice, and W be a subset of Z . Let $\prec_{W, \sqsubseteq}$ be the quasi-order defined on Z by:*

$$z \prec_{W, \sqsubseteq} z' \Leftrightarrow \{w \in W \mid w \sqsubseteq z\} \subseteq \{w \in W \mid w \sqsubseteq z'\}$$

The function $\tau_{W, \sqsubseteq}$ is a transposing function from $(Z, \prec_{W, \sqsubseteq})$ into (Z, \sqsubseteq) .

Proof. Since for any subsets S and S' of Z , $S \subseteq S'$ implies that $\sup S \sqsubseteq \sup S'$, we have $z \prec_{W, \sqsubseteq} z'$ implies $\tau_{W, \sqsubseteq}(z) \sqsubseteq \tau_{W, \sqsubseteq}(z')$. Conversely, assume that $\tau_{W, \sqsubseteq}(z) \sqsubseteq \tau_{W, \sqsubseteq}(z')$. Then by (3.1) and (3.2) we have $w \sqsubseteq \tau_{W, \sqsubseteq}(z) \sqsubseteq \tau_{W, \sqsubseteq}(z') \sqsubseteq z'$ for all element w of W such that $w \sqsubseteq z$. This means that we have $w \sqsubseteq z'$ for all element w of W such that $w \sqsubseteq z$, i.e., $z \prec_{W, \sqsubseteq} z'$. So $z \prec_{W, \sqsubseteq} z'$ is equivalent to $\tau_{W, \sqsubseteq}(z) \sqsubseteq \tau_{W, \sqsubseteq}(z')$. \square

COROLLARY C.1. *Let (Z, \sqsubseteq) be a complete lattice. Let $\langle X, Y, \sqsubseteq \rangle$ be a set covering problem, where X and Y are subsets of Z . It is strongly equivalent to $\langle \max_{\sqsubseteq} \tau_{Y, \sqsupseteq}(X), \max_{\sqsubseteq} \tau_{X, \sqsubseteq}(Y), \sqsubseteq \rangle$.*

Proof. The dominance relation on Y is $\prec_{X, \sqsubseteq}$, and the dominance relation on X is $\succ_{Y, \sqsupseteq}$. Thus $\langle X, Y, \sqsubseteq \rangle$ is strongly equivalent to $\langle \max_{\succeq_{Y, \sqsupseteq}} (X / \equiv_{Y, \sqsupseteq}), \max_{\preceq_{X, \sqsubseteq}} (Y / \equiv_{X, \sqsubseteq}), \sqsubseteq \rangle$. Since $\tau_{X, \sqsubseteq}$ is a transposing function from $(Z, \prec_{X, \sqsubseteq})$ into (Z, \sqsubseteq) , and by duality, $\tau_{Y, \sqsupseteq}$ is a transposing function from $(Z, \prec_{Y, \sqsupseteq})$ into (Z, \sqsupseteq) , i.e., a transposing function from $(Z, \succ_{Y, \sqsupseteq})$ into (Z, \sqsubseteq) . Thus $\langle \max_{\succeq_{Y, \sqsupseteq}} (X / \equiv_{Y, \sqsupseteq}), \max_{\preceq_{X, \sqsubseteq}} (Y / \equiv_{X, \sqsubseteq}), \sqsubseteq \rangle$ is isomorphic to $\langle \max_{\sqsubseteq} \tau_{Y, \sqsupseteq}(X), \max_{\sqsubseteq} \tau_{X, \sqsubseteq}(Y), \sqsubseteq \rangle$. \square

We have justified the first rewriting rule, namely the set covering problem $\langle X, Y, \sqsubseteq \rangle$ is strongly equivalent to $\langle \max_{\sqsubseteq} \tau_{Y, \sqsupseteq}(X), \max_{\sqsubseteq} \tau_{X, \sqsubseteq}(Y), \sqsubseteq \rangle$.

C.2.2 Essential Elements. We now prove the second result. This nice property is a consequence of the definition of $\tau_{Y, \sqsupseteq}$, which is an endomorphism that acts as a filter w.r.t dominance relations on the lattice.

THEOREM C.3. *Let (Z, \sqsubseteq) be a complete lattice. Let $\langle X, Y, \sqsubseteq \rangle$ be a set covering problem, where X is a subset of Z , and Y is a maximal subset of Z w.r.t \sqsubseteq . Then the set of essential elements of Y is $Y \cap \tau_{Y, \sqsupseteq}(X)$, which is also equal to $Y \cap \max_{\sqsubseteq} \tau_{Y, \sqsupseteq}(X)$.*

Proof. If y is essential, then it is the only element of Y such that $x \sqsubseteq y$ for some x . We have $\tau_{Y, \sqsupseteq}(x) = \sup_{\sqsupseteq} \{y \in Y \mid y \sqsupseteq x\} = \sup_{\sqsupseteq} \{y\} = y$, and so y belongs to $\tau_{Y, \sqsupseteq}(X)$.

Conversely, let $y \in Y \cap \tau_{Y, \sqsupseteq}(X)$. Then there exists some x such that $\tau_{Y, \sqsupseteq}(x) = y$, and thus we have $x \sqsubseteq y$. Let y' be an element of Y such that $x \sqsubseteq y'$. We have $y = \tau_{Y, \sqsupseteq}(x) \sqsubseteq y'$, i.e., $y \sqsubseteq y'$. Since Y is maximal w.r.t \sqsubseteq , this implies that $y = y'$. So y is the only element of Y such that $x \sqsubseteq y$, i.e., y is essential.

We proved that the set of essential elements is $Y \cap \tau_{Y, \sqsupseteq}(X)$. Clearly it contains $Y \cap \max_{\sqsubseteq} \tau_{Y, \sqsupseteq}(X)$. Conversely, assume that there exists y such that $y \in Y \cap \tau_{Y, \sqsupseteq}(X)$ and $y \notin Y \cap \max_{\sqsubseteq} \tau_{Y, \sqsupseteq}(X)$. This means that y is not a maximal element of $\tau_{Y, \sqsupseteq}(X)$ w.r.t. \sqsubseteq , so there exists an element x of X such that $y \sqsubseteq \tau_{Y, \sqsupseteq}(x)$ and $y \neq \tau_{Y, \sqsupseteq}(x)$. Then let y' be an element of Y such that $x \sqsubseteq y'$. We have $y \sqsubseteq \tau_{Y, \sqsupseteq}(x) \sqsubseteq y'$, i.e., $y \sqsubseteq y'$ with $y \neq y'$, which is impossible because Y is maximal w.r.t. \sqsubseteq . \square

Provided that Y is *maximal* with respect to \sqsubseteq , the set of essential elements is thus $Y \cap \max_{\sqsubseteq} \tau_{Y, \sqsupseteq}(X)$. The first rewriting rule produces a set Y that has the form $\max_{\sqsubseteq} \tau_{X, \sqsubseteq}(Y)$, which is obviously maximal, and a set X that has the form $\max_{\sqsubseteq} \tau_{Y, \sqsupseteq}(X)$. This establishes the correctness of the second rewriting rule.

C.2.3 Recovering the Original Solutions. Recovering the original solutions from the minimal solutions of the fixpoint can be done very easily thanks to the uniformization of the workspace to the lattice (Z, \sqsubseteq) .

Consider the set covering problem $\langle X, Y, \sqsubseteq \rangle$ over the lattice (Z, \sqsubseteq) . Let $C = \langle X, \max_{\sqsubseteq} \tau_{X, \sqsubseteq}(Y), \sqsubseteq \rangle$. Thanks to the isomorphism $\tau_{X, \sqsubseteq}$ establishes, the set of minimal solution of the original problem $\langle X, Y, \sqsubseteq \rangle$ can be written as:

$$\bigcup_{S \in \text{Sol}(C)} \times_{z \in S} \tau_{X, \sqsubseteq}^{-1}(z)$$

Proving the following result is then sufficient to show the correctness of the minimal solution recovering process.

THEOREM C.4. *The function $\tau_{X, \sqsubseteq}^{-1}$ whose domain is $\text{Sol}(C)$ and whose range is restricted to Y satisfies the following identity:*

$$\tau_{X, \sqsubseteq}^{-1}(z) = \{y \in Y \mid z \sqsubseteq y\}.$$

Proof. Let y be an element of $\tau_{X, \sqsubseteq}^{-1}(z)$. This means that $\tau_{X, \sqsubseteq}(y) = z$, and since we have $\tau_{X, \sqsubseteq}(y) \sqsubseteq y$, we have $z \sqsubseteq y$. Conversely, let y be an element of Y such that $z \sqsubseteq y$. Then $\tau_{X, \sqsubseteq}(z) \sqsubseteq \tau_{X, \sqsubseteq}(y)$. But since $\tau_{X, \sqsubseteq}(z)$ belongs to $\max_{\sqsubseteq} \tau_{X, \sqsubseteq}(Y)$, it is maximal w.r.t \sqsubseteq within $\tau_{X, \sqsubseteq}(Y)$, and so we must have $\tau_{X, \sqsubseteq}(z) = \tau_{X, \sqsubseteq}(y)$. Moreover since z belongs to $\tau_{X, \sqsubseteq}(Y)$, and since $\tau_{X, \sqsubseteq}$ is idempotent, we have $\tau_{X, \sqsubseteq}(z) = z$. Thus $z = \tau_{X, \sqsubseteq}(y)$, and so $y \in \tau_{X, \sqsubseteq}^{-1}(z)$. \square