Aligning CNF- and Equivalence-reasoning

Marijn Heule and Hans van Maaren

Department of Information Systems and Algorithms
Faculty of Electrical Engineering, Mathematics and Computer Sciences
Delft University of Technology
marijn@heule.nl, h.vanmaaren@its.tudelft.nl

Abstract. Structural logical formulas sometimes yield a substantial fraction of so called equivalence clauses after translating to CNF. The best known example of this feature is probably provided by the `parity`-family. The larger such CNF formulas cannot be solved in reasonable time if no extra reasoning with - and detection of - these clauses is incorporated. That is, in solving these formulas, there is a more or less separated algorithmic device in dealing with these equivalence clauses, called equivalence reasoning, and another dealing with the remaining clauses. In this paper we propose a way to align these two reasoning devices by introducing parameters for which we establish optimal settings over a variety of existing benchmarks. We obtain a truly convincing speed up in solving such formulas with respect to the best solving methods existing so far.

1 Introduction

The notorious `parity-32` benchmarks [3] remained unsolved by general purpose Sat solvers for a considerable time. In [11] a method was proposed which, for the first time, could solve these instances in a few minutes. The key to this method was to detect the clauses which represented so called *equivalences* $l_1 \leftrightarrow l_2 \leftrightarrow \cdots \leftrightarrow l_n$ (where the l_i are literals, or their negations, appearing in the formula at hand) and to pre-process the set of these equivalences in such a way that dependent and independent variables became visible. The remaining clauses then where tackled with a rather straightforward DPLL procedure but in such a way that kept track of the role of these dependent and independent variables. As it was developed, it was a two-phase method, where the equivalence part was established and transformed in a pre-processing phase.

The next important step was made by Li [5, 6] which incorporated a sort of equivalence reasoning in any node of an emerging search tree. His approach did not incorporate a pre-processing phase and thus he established the first one-phase Sat solver `eqsatz` which could tackle these instances in reasonable time.

A disadvantage of his method is the fact that he uses a full lookahead approach (very costly for larger size formulas) and evaluates the speed in reduction of the formula in a - to our opinion - not optimal way. Also, his equivalence reasoning is restricted to equivalences of length at most three.

Some years later Ostrowski *et al.* [9] extended the above pre-processing ideas from [11] to logical gates other than equivalences, resulting in the `lsat` solver. However, their DPLL approach to deal with the remaining CNF-part uses a Jeroslow-Wang branching rule and they do not perform a lookahead phase, which is - again to our opinion - not an optimal alignment.

In this paper we propose an alignment of equivalence reasoning and DPLL reasoning which does not assume a full lookahead approach. This will enforce us to introduce adequate pre-selection heuristics for selecting variables which are allowed to enter an Iterative Unit Propagation phase. Further, we will evaluate the progress in enrolling the formula at hand in a more detailed manner as was done in `eqsatz`. We are forced to introduce parameters to guide this search. This parameters are needed to aggregate the reduction of the equivalence part of the formula and that of the remaining CNF part. Further, our method is able to deal with equivalences of arbitrary size. This in turn leads us to an investigation of the relative importance of equivalences of different size. Surprisingly, this relative importance turns out to be rather differently measured as could be expected when taking similar relative importance of ordinary clause-lengths as a guiding principle.

We optimise the various parameters to ensure a convincing speed up in solving formulas with a substantial equivalence part, both with respect to the various alternative solvers available and with respect to a variety of benchmarks known of this type.

2 Equivalence reasoning in pre-processor

The first goal of the pre-processor after initialisation is to simplify the formula as much as possible. This is achieved by iterative propagation of all unary clauses and binary equivalences. After this procedure, the equivalence clauses are detected using some simple syntactical search procedure, extracted from the formula and placed in a separate data-structure. We refer to this data-structure as the Conjunction of Equivalences (CoE). The aim of our equivalence reasoning enhanced pre-processor is to solve the extracted CoE sub-formula.

A solution is obtained by performing the first phase of the algorithm by Warners and Van Maaren [11]: We initialise set $\mathscr{I} = \{x_1, \cdots, x_m\}$, the set of *independent* variables, with m referring to the initial number of variables. We loop through the equivalency clauses once, choosing variable x_i in each one to eliminate from all other equivalence clauses. Subsequently we remove x_i from $\mathscr{I}$, and call it a *dependent* variable. Thus we end up with a set of equivalence clauses for which all satisfiable assignments can be constructed by assigning all possible combinations of truth values to the independent variables. The values of the dependent variables are uniquely determined by an assignment of the independent variables. Note that during the elimination process a contradiction might be derived, which implies unsatisfiability of the original problem.

Numerous of such independent sets could be obtained by this algorithm. The performance of a solver might vary significantly under different choices of the independent set, as we have observed using the march solver(developed by Joris van Zwieten, Mark Dufour and Marijn Heule and van Maaren, and which participated in the Sat 2002 [7] and Sat 2003 [8] competitions). Therefore, two enhancements are added to the original algorithm to find an independent set that would result in a relative fast performance: The first addition is an explicit prescription for the selection of the variables to eliminate: For every equivalence clause the variable is selected that occurs (in a weighted fashion) least frequently in the CNF. Occurrences in binary clauses are counted twice as important as occurrences in n-ary clauses.

The motivation for selecting the least occurring variable is twofold: First, if the selected variable x_i does not occur in the CNF at all, the equivalence clause in which x_i occurs, becomes a *tautological clause* after elimination, because x_i could always be taken such to satisfy it. Neglecting tautological clauses during the solving phase could result in a considerable speed-up. Secondly, faster reduction of the formula is expected when the independent variables occur frequently in the CNF-part: independent variables will then be forced earlier to a certain truth value by constraints from both the CoE- and CNF-part.

The second addition is a procedure that reduces the sum of the lengths of all non-tautological equivalences in the CoE. This procedure consists of two steps: The first searches for pairs of equivalence clauses that could be combined to created a binary equivalence. Notice that binary equivalence clauses are always tautological, since one of its literals could be removed from the CNF by replacing it by the other. The second step loops through all equivalence clauses and checks whether a different choice for the dependent variable in that clause would result in a smaller sum of lenghts of non-tautological equivalences. Both methods are iteratively repeated until both yield no further reduction.

Several benchmark families in the Sat 2003, Sat 2002 and Dimacs benchmark suites[1] can be solved by merely applying the pre-processing presented above. One of these families is `xor-chain` which contains the smallest unsolved unsatisfiable instances from the Sat 2002 competition. Table 1 shows the required time to solve these families for various solvers. Notice that march uses the proposed pre-processing. In the table, the numbers after the family names refer to the number of instances in a family. The last five columns show the total time required to solve a family. In these columns, numbers between braces express the number of benchmarks that could not be solved within a 120 seconds time limit. Judging from the data in the table, only lsat is able to solve all these families with comparable speed as the march pre-processor.

[1] All three suites are available at www.satlib.org

family		contributer	suite	march	eqsatz		satzoo		lsat	zchaff	
bevhcube	(4)	Bevan	SAT '03	0.02	2.64	(2)	2.74	(2)	0.02	0.01	(3)
dodecahedron	(1)	Bevan	SAT '03	0.01	0.01		0.08		0.01	0.01	
hcb	(4)	Bevan	SAT '03	0.16	0.01	(3)	0.01	(3)	2.53	0.01	(3)
hypercube	(4)	Bevan	SAT '03	0.09	0.40	(3)	0.08	(3)	0.33	2.56	(3)
icos	(2)	Bevan	SAT '03	0.01	2.29	(1)	4.12	(1)	0.02	——	(2)
marg	(17)	Bevan	SAT '03	0.12	195.83	(5)	52.93	(5)	0.13	0.08	(11)
urqh	(26)	Bevan	SAT '03	0.19	102.58	(20)	16.76	(20)	0.47	1.19	(22)
hardmn	(18)	Moore	SAT '03	0.87	0.75		——	(18)	0.30	——	(18)
genurq	(10)	Ostrowski	SAT '03	0.95	0.07	(7)	0.68	(1)	0.57	0.39	(6)
Urquhart	(30)	Simon	SAT '02	0.27	——	(30)	58.09	(25)	0.01	0.13	(29)
urquhart	(6)	Chu Min Li	SAT '02	0.03	0.29	(4)	95.78	(3)	0.01	0.04	(5)
xor-chain	(27)	Zhang-Lintao	SAT '02	0.16	0.17	(25)	0.77	(25)	0.02	2.25	(24)
dubois	(13)	Dubois	DIMACS	0.02	0.1		0.75		0.01	0.06	
pret	(8)	Pretolani	DIMACS	0.03	0.08		20.81		0.01	——	(8)

Table 1. Performances of the solvers march, eqsatz, satzoo, lsat and zchaff in seconds on several families that could be solved by merely pre-processing.

3 Combined lookahead evaluation

Lookahead appears to be a powerful technique to solve a wide range of problems. The pseudo-code of an elementary lookahead procedure is presented in Algorithm 1. The lookahead procedure in march closely approximates this elementary procedure. Notice that it does not perform any equivalence reasoning during this phase.

Algorithm 1 LOOKAHEAD( )

Let $\mathcal{F}'$ and $\mathcal{F}''$ be two copies of $\mathcal{F}$
for each variable x_i in $\mathcal{P}$ **do**
  $\mathcal{F}'$:= ITERATIVEUNITPROPAGATION($\mathcal{F} \cup \{x_i\}$)
  $\mathcal{F}''$:= ITERATIVEUNITPROPAGATION($\mathcal{F} \cup \{\neg x_i\}$)
  if empty clause $\in \mathcal{F}'$ **and** empty clause $\in \mathcal{F}''$ **then**
    return "unsatisfiable"
  else if empty clause $\in \mathcal{F}'$ **then**
    $\mathcal{F} := \mathcal{F}''$
  else if empty clause $\in \mathcal{F}''$ **then**
    $\mathcal{F} := \mathcal{F}'$
  else
    $\mathrm{H}(x_i) = 1024 \times \mathrm{DIFF}(\mathcal{F}, \mathcal{F}') \times \mathrm{DIFF}(\mathcal{F}, \mathcal{F}'') + \mathrm{DIFF}(\mathcal{F}, \mathcal{F}') + \mathrm{DIFF}(\mathcal{F}, \mathcal{F}'')$
  end if
end for
return x_i with greatest $\mathrm{H}(x_i)$ to branch on

An effective lookahead evaluation function (DIFF in short) is critical for the effectiveness of the branching variable the lookahead returns. Experiments on random 3-SAT instances showed that using a DIFF that counts newly created binary clauses, results in fast performances on these benchmarks and many other families. Addition of new clauses of length > 2 to the DIFF requires weights that express the relative importance of clauses of various length. Weights that result in optimal performance on random k-SAT formulas could be described by linear regression: e.g. Kullmann [4] uses weights in his OKsolver that could be approximated by 0.22^{n-2}. In this equation n refers to the length of a clause, with $n \geq 2$.

Little is known about effective evaluation functions to measure the importance of a new equivalence clause. In eqsatz by Li [5, 6] only new binary equivalences are counted. These are weighted twice as important as a new binary clause. The importance of the new equivalence clauses of various length could be obtained by measuring the reduction of its translation into CNF. Applying the approximation of the weights by Kullmann [4] results in a weight function of $2^{n-1} \times 0.22^{n-2} \approx 10.33 \times 0.44^n$ for a new equivalence of length n. However, this reference should be labelled as vague since the weights are optimised with respect to random formulas.

Although we have indications that other models might be more appropriate when equivalence clauses are involved, we take this regression model as a first start. Performances were measured for various parameter settings of equation (1). In this equation, n refers to the reduced length of an equivalence clause. Parameter q_{base} denotes the factor that describes the decreasing importance of equivalence clauses of various length and parameter q_{const} expresses the relation between the reduction of the CNF-clauses and the equivalence clauses. Since march uses a 3-Sat translator, only new binary clauses are created. The evaluation of the lookahead is calibrated by defining the importance of a new binary clause to value 1. The result of eq_n then defines the relative importance of a new equivalence clause of length n in relation to a new binary clause.

$$eq_n = q_{\text{const}} \times {q_{\text{base}}}^n \tag{1}$$

Wide scale experiments were troubled by the lack of useful benchmarks: Many benchmark families that contain a significant part of equivalence clauses are easily solved with solely the pre-processing procedures: Either the solving procedure for the CoE results in a contradiction, or the propagation of the unary clauses and the binary equivalences found during pre-processing are sufficient to solve the formula. Many benchmarks families with a significant CoE-part that require a sophisticated solving procedure after pre-processing are also not useful for these experiments, because most or all of their equivalence clauses have length 3. For comparison: The Sat 2003 [8] competition suite consisted of 11 families which are solved in pre-processing while only five needed further search. Of those five only two had a large number of long equivalences after the pre-processing.

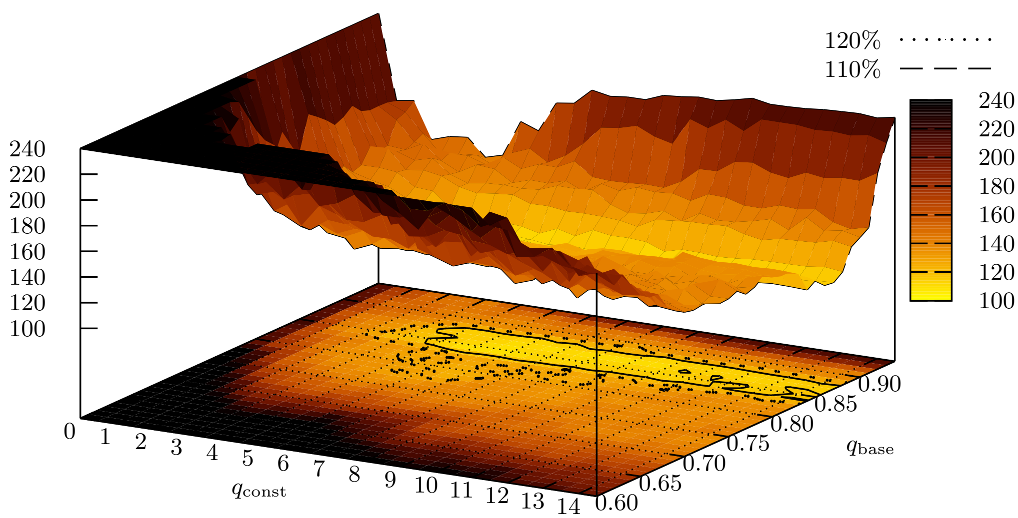

Fig. 1. Performances achieved by march on various settings of q_{base} and q_{const}. The values on the z-axis are the cumulated performances on the whole `parity-32` and `hwb-n20` families in seconds. Contour lines are drawn at 110% and 120% of optimal performance.

These two families are the `parity32` and the `hwb`. The first family consists of the Sat-encoding of minimal disagreement parity problems contributed by Crawford *et al.* [3]. The second consists of equivalence checking problems that arise by combining two circuits computed by the hidden weighted bit function. These latter are contributed by Stanion [8]. Both families have been used to determine the parameter setting for equation (1) that results in optimal performance. The results of these experiments are shown in Fig. 1. The values $q_{\text{const}} = 5.5$ and $q_{\text{base}} = 0.85$ appeared optimal during our experiments. Two conclusions can be drawn regarding the results: (1) Parameter

q_{base} has a much larger influence on the performance than q_{const}. (2) Using optimal settings, the reduction of equivalences is considered far more important than the reduction of the equivalent CNF-translations would suggest: Tab. 2 shows the used weights for both settings.

Reduced length (n):	2	3	4	5	6	7	8	9	10
CNF-reference:	2.00	0.88	0.39	0.17	0.07	0.03	0.01	0.01	0.00
Found optimum:	3.97	3.38	2.87	2.44	2.07	1.76	1.50	1.27	1.08

Table 2. Weights to measure the reduction of equivalence clauses of various length.

4 Pre-selection heuristics

Although lookahead is a powerful technique, it pays off to restrict the number of variables which enter this procedure. In Algorithm 1 this partial behaviour is achieved by performing only lookahead on variables in set $\mathcal{P}$. At the beginning of each node, this set is filled by pre-selection heuristics that are based on an approximation function of the combined evaluation of the lookahead (Ace). The ranking of variable x is calculated by multiplying $\textsc{Ace}(x)$ and $\textsc{Ace}(\neg x)$. $\mathcal{E}(x)$, used to obtain $\textsc{Ace}(x)$, refers to the set of all equivalence clauses in which x occurs and $occ_3(x)$ refers to the number of occurrences of x in ternary clauses.

$$\textsc{Ace}(x) = occ_3(\neg x) + \sum_{Q_i \epsilon \mathcal{E}(x)} eq_{|Q_i|-1} + \sum_{\neg x \vee y \epsilon \mathcal{F}} \left(occ_3(\neg y) + \sum_{Q_i \epsilon \mathcal{E}(y)} eq_{|Q_i|-1} \right) \quad (2)$$

In the versions of march without equivalence reasoning fast performance is achieved on average by performing only lookahead on the “best” 10 % of the variables. This constant percentage is not optimal in general. It is not even optimal for the benchmarks used in this paper, but because of space limitations we restricted ourselves to this 10 %. To illustrate the diversity of partial lookahead optima, march requires 1120 secondes to solve a benchmark provided by Philips using the 10 % setting (see table 3), while it requires only 761 seconds at the optimal setting of 8 ‰.

5 Additional equivalence reasoning

Various addition forms of equivalence reasoning are tested. These include:

- Removal of equivalence clauses that have became tautological during the solving phase. This results in a speed-up because of faster propagation.
- Propagation of binary equivalences in the CoE: Replacing one of its literals by the other. This increases the change that a variable occurs twice in an equivalence clause, so both could be removed.
- Prevention of equivalent variables to enter the lookahead procedure, since equivalent variables will yield an equivalent Diff.

Only the last adjustment realised a noticeable speed-up of about 10 %. The gain that other procedures accomplished were comparable to their cost, resulting in a status quo in terms of time.

6 Results

Four solvers are used to compare the results of march: eqsatz[2], lsat[3], satzoo[4] and zchaff[5]. The choice for eqsatz and lsat is obvious since they are the only other Sat solvers performing equivalence

[2] version 2.0, available at http://www.laria.u-picardie.fr/∼cli/EnglishPage.html
[3] version 1.1, provided by authors
[4] version 1.02, available at http://www.math.chalmers.se/∼een/Satzoo/
[5] version 2003.07.01, available at http://www.ee.princeton.edu/∼chaff/zchaff.php

reasoning. Since equivalence clauses merely occur in handmade and industrial problems, we added some solvers that are considered state-of-the-art in these categories: satzoo and zchaff, respectively. In an extended version of this paper, performances of other solvers will be added.

All solvers were tested on a AMD 2000+ with 128Mb memory running on Mandrake 9.1. Besides the `parity32` and the `hwb` benchmarks, we experimented on the `barrel` and `longmult` families that arise from bounded model checking [2], five unsolved benchmarks (`pyhala-braun-x` and `lisa21-99-a`) from the SAT 2002 competition contributed by Pyhala and Aloul, respectively [7] and three factoring problems (`2000009987x`) contributed by Purdom [10]. Except from both bounded model checking families and the benchmark provided by Philips, all benchmarks were used in the SAT 2003 competition. To enable a comparison with the SAT 2003 results[6], we used the shuffled benchmarks generated for this competition during our experiments. However, these shuffled benchmarks caused a slowdown in performance of eqsatz: e.g. eqsatz solves most original `parity32` benchmarks within the 2000 seconds time limit.

instance	#Cls	#Var	#Ind	#Eq	#Nt	march°	march*	eqsatz	satzoo	lsat	zchaff
par32-1	10227	3176	157	1158	218	**0.55**	>2000	568.31	>2000	90.85	>2000
par32-2	10253	3176	157	1146	218	**0.3**	>2000	>2000	>2000	88.43	>2000
par32-3	10297	3176	157	1168	218	**1.08**	>2000	>2000	>2000	7.54	>2000
par32-4	10313	3176	157	1176	218	**7.93**	>2000	>2000	>2000	79.87	>2000
par32-5	10325	3176	157	1182	218	**8.82**	>2000	>2000	>2000	34.41	>2000
par32-1-c	5254	1315	157	1158	218	**0.47**	>2000	>2000	>2000	3.91	>2000
par32-2-c	5206	1303	157	1146	218	7.82	>2000	>2000	>2000	**4.45**	>2000
par32-3-c	5294	1325	157	1168	218	**5.06**	>2000	>2000	>2000	33.59	>2000
par32-4-c	5326	1333	157	1176	218	**0.39**	>2000	>2000	>2000	52.39	>2000
par32-5-c	5350	1339	157	1182	218	**6.77**	>2000	>2000	>2000	71.98	>2000
hwb-n20-1	630	134	96	36	35	**18.05**	39.43	78.51	47.04	771.48	300.72
hwb-n20-2	630	134	96	36	35	**23.24**	50.82	83.25	73.38	738.16	461.49
hwb-n20-3	630	134	96	36	35	**16.16**	38.56	75.09	24.37	564.81	257.07
hwb-n22-1	688	144	104	38	37	**68.27**	164.65	299.45	108.	>2000	785.89
hwb-n22-2	688	144	104	38	37	**53.67**	145.51	297.3	85.79	>2000	1097.33
hwb-n22-3	688	144	104	38	37	**58.6**	148.29	306.71	60.6	>2000	1710.10
hwb-n24-1	774	162	116	44	43	**556.25**	796.56	>2000	624.17	>2000	>2000
hwb-n24-2	774	162	116	44	43	**463.46**	832.48	>2000	862.86	>2000	>2000
hwb-n24-3	774	162	116	44	43	**332.**	670.21	>2000	471.73	>2000	>2000
hwb-n26-1	832	172	124	46	45	**1203.99**	>2000	>2000	>2000	>2000	>2000
hwb-n26-2	832	172	124	46	45	**1777.78**	>2000	>2000	>2000	>2000	>2000
hwb-n26-3	832	172	124	46	45	**1703.12**	>2000	>2000	>2000	>2000	>2000
barrel-5	5383	1407	-	720	-	**0.07**	**0.07**	0.15	0.85	1.9	0.52
barrel-6	8931	2306	-	1260	-	**0.15**	**0.15**	0.33	4.94	644.49	2.73
barrel-7	13765	3523	-	2016	-	**0.29**	**0.29**	0.38	23.63	>2000	11.38
barrel-8	20083	5106	-	3024	-	**0.5**	**0.5**	0.61	69.94	>2000	31.22
barrel-9	36606	8903	-	5760	-	**1.19**	**1.19**	1.33	258.37	>2000	240.29
longmult-6	8853	2848	1037	174	90	7.13	3.98	11.96	5.1	66.32	**1.59**
longmult-7	10335	3319	1276	203	105	35.79	21.37	55.15	21.74	109.19	**13.32**
longmult-8	11877	3810	1534	232	120	100.92	70.8	185.85	**66.22**	192.56	68.17
longmult-9	13479	4321	1762	261	135	202.03	**130.46**	347.75	138.8	289.53	131.46
longmult-10	15141	4852	2014	290	150	260.06	**168.21**	520.1	232.96	404.27	252.95
longmult-11	16863	5403	2310	319	165	226.35	**151.36**	662.19	307.62	542.81	344.64
longmult-12	18645	5974	2620	348	180	128.57	**89.37**	741.19	338.48	709.15	305.07
longmult-13	20487	6565	2598	377	195	67.92	**52.69**	855.14	272.67	901.13	255.96
longmult-14	22389	7176	2761	406	210	44.33	**31.78**	985.37	383.08	1112.49	266.54
longmult-15	24351	7807	2784	435	225	25.26	**24.54**	1108.80	215.12	1236.6	207.72
pb-sat-40-4-03	31795	9638	2860	3002	3001	>2000	510.51	>2000	**184.21**	>2000	357.98
pb-sat-40-4-04	31795	9638	2860	2936	2935	>2000	**600.41**	>2000	>2000	>2000	>2000
pb-unsat-35-4-03	24320	7383	2132	2220	2219	771.24	**698.22**	>2000	>2000	>2000	>2000
pb-unsat-35-4-04	24320	7383	2131	2277	2276	821.43	**736.31**	>2000	>2000	>2000	>2000
lisa21-99-a	7967	1453	1310	460	459	**21.26**	1170.12	>2000	>2000	>2000	>2000
2000009987fw	12719	3214	1615	1358	1319	175.68	**115.89**	521.47	267.81	181.86	116.4
2000009987nc	10516	2710	1303	1286	1262	137.41	**84.16**	197.27	167.4	159.55	94.61
2000009987nw	11191	2827	1342	1322	1299	135.25	**87.9**	157.03	218.09	166.55	104.84
philips	4456	3642	1005	342	224	1120.61	**1032.82**	3390.59	2114.64	1277.68	>3600

Table 3. Performances of the solvers march, eqsatz, satzoo, lsat and zchaff in seconds on various benchmarks with equivalence clauses.

Two versions of our solver are used to evaluate performance: The first, march° uses the equation $eq_n = 5.5 \times 0.85^n$ to measure the reduction of the CoE during the lookahead, and applies it on the calculation of ACE from the pre-selection heuristics. The second variant, march* does not use the

[6] Results of the SAT 2003 competition are available at www.lri.fr/∼simon/contest03/results/

CoE-part during the lookahead but operates using the original CNF instead. Both march variants use a 10% partial lookahead.

In table 3 the performances are presented for these solvers together with five properties of each benchmark:

#Cls refers to the initial number of clauses
#Var refers to the initial number of variables
#Ind refers to the number of variables in the independent set
#Eq refers to the number of detected equivalence clauses.
#Nt refers to the number of non-tautological equivalences after pre-processing.

We conclude that aligning Equivalence- and CNF- reasoning as carried out pays off convincingly, but that, although some instances are *not* solved without incorporating the CoE reductions during the lookahead phase (march$^{\circ}$), others suffer from this additional overhead and are better solved by updating and investigating this CoE part at the chosen path only (march*).

References

1. D. Le Berre and L. Simon, *The essentials of the SAT'03 Competition*. Springer-Verlag, LNCS 2919, pages 452-467, 2004.
2. A. Biere, A. Cimatti, E. M. Clarke, Y. Zhu, *Symbolic model checking without BDDs.* in Proc. Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems, Vol. 1579 of Lecture Notes in Comput. Sci., pp. 193-207, 1999.
3. J.M. Crawford, M.J. Kearns, R.E.Schapire, *The Minimal Disagreement parity problem as a hard satisfiability problem*. Draft version 1995.
4. O. Kullmann, *Investigating the behaviour of a SAT solver on random formulas*. Submitted to Annals of Mathematics and Artificial Intelligence, September, 2002.
5. C.M. Li, *Integrating Equivalency Reasoning into a Davis-Putman Procedure.* In Proceedings of the AIII-2000, pages 291-296, Austin, USA, August 2000.
6. Chu Min LI, *Equivalent literal propagation in Davis-Putnam procedure.* In Discrete Applied Mathematics, Vol 130/2 pp 251-276, 2003.
7. L. Simon, D. Le Berre, and E. Hirsch, *The sat2002 competition*. Submitted to AMAI, August 2002.
8. L. Simon, *Sat'03 competition homepage*. http://www.lri.fr/ simon/contest03/results/
9. R. Ostrowski, E. Gregoire, B. Mazure, L. Sais, *Recovering and exploiting structural knowledge from CNF formulas*, in Proc. of the Eighth International Conference on Principles and Practice of Constraint Programming, Vol. 2470 of Lecture Notes in Comput. Sci., Springer Verlag, pp 185-199, Ithaca (N.Y.), Septembre 2002.
10. P. Purdom and A. Sabry, *CNF Generator for Factoring Problems.* http://www.cs.indiana.edu/cgi-pub/sabry/cnf.htm
11. J.P. Warners, H.van Maaren, *A two phase algorithm for solving a class of hard satisfiability problems.* Operations research letters 23, 81-88, 1998.