Experiments with Multi-Domain Logic: Variable Merging and Split Strategies *

Tudor Jebelean[1] and Gábor Kusper[2]

[1] Johannes Kepler University Linz, RISC Institute, Austria
Tudor.Jebelean@risc.uni-linz.ac.at
[2] Eszterházy Károly College, Eger, Hungary
gkusper@aries.ektf.hu

Abstract. Multi-Domain Logic (MDL) is a generalization of signed logic, in which every variable has its own domain. This aspect increases the efficiency of direct solving of MDL satisfiability, because the solving process proceeds by reducing the size of the domains (contradiction appears as an empty domain). In contrast to the usual approach of translating signed logic satisfiability into boolean satisfiability, we implement the generalized DPLL directly for MDL, using a specific version of the techniques used for signed logic. Moreover, we use a novel technique – *variable merging*, which consists in replacing two or more variables by a new one, whose domain is the cartesian product of the old domains. This operation is used during the solving process in order to reduce the number of variables. Moreover, variable merging can be used at the beginning of the solving process in order to translate a boolean SAT problem into an MDL problem. This opens the possibility of using MDL solvers as an alternative to boolean solvers, which is promising because in MDL several boolean constraints can be propagated simultaneously. Our experiments with a prototype eager solver show the effects of the initial merging factor of boolean variables, as well as the effects of different design decisions on the efficiency of the method.

1 Introduction

Signed logic [2, 7] is a special type of multi-valued logic in which the set of satisfying values for the variables may differ in different clauses. Namely, a *signed formula* is a conjunction of *signed clauses*, and a signed clause is a disjunction of *signed literals* of the form $S : p$, where S is a set (the *sign*) and p is a variable. (S is called the *support* of the variable in the respective clause.) The union of all signs constitute the *domain* N of the formula. An interpretation I is a mapping from the set of variables P to the set of truth values N, and it satisfies a literal $S : p$ if $I(p) \in S$. We may assume that each variable occurs at most once in each clause (otherwise we merge the corresponding literals by union of their

* Partially supported by the RISC-Linz Transnational Aceess Programme supported by the European Commission FP6 for Integrated Infrastructures Initiatives under the project SCIEnce (contract No. 026133).

supports). When $S = \emptyset$ the literal may be omitted from the clause, and when $S = N$ then the whole clause is redundant.

The classical methods from boolean logic generalize in a natural way to signed logic – see e. g. [2], which also describes the generalization of the DPLL algorithm [3], including a specific aspect of it for signed logic, namely the elimination from of branches corresponding to certain redundant truth values. (We will describe and use this strategy in the sequel under the name of *elimination of weak assignments*.) However, we found only few implementations of a direct method for solving signed logic problems – e. g. [7], which is targeted at a restricted class of formulae. Rather, most approaches are based on translating signed logic into boolean logic and using some version of a SAT algorithm. For instance, [1] uses the information from the original signed logic problem in order to guide the SAT search.

We present here a direct approach for solving signed logic problems, based on the generalization of DPLL method, which exhibits two novel aspects:

- separation of the domains of the variable,
- *dynamic merging* of the domains of the variables.

In contrast to the current approaches, we demonstrate how to solve boolean SAT problems by transforming them into signed logic problems and then applying our direct solver. This may constitute an efficient alternative to the current SAT solvers based on unit propagation, because, in the context of signed logic, more boolean constraints can be propagated simultaneously. For the representation of the signs we use strings of bits in the current implementation, but this is not essential for the main algorithm.

We call *Multi-Domain Logic (MDL)* the generalization of signed logic in which the domains of the variables may differ. Although from the theoretical point of view the expressivity of MDL does not differ from signed logic, in practice this distinction leads to more efficient solving methods. This is because *the domains can be reduced* during the solving process, and finding a contradiction is expressed as the reduction of the domain of a variable to the empty set. Initially the domain of a variable which does not occur in unit clauses is the union of all its supports. Otherwise, the domain is the intersection of all the supports from the unit clauses containing the respective variable. Unit resolution consists in intersecting the support of a non-unit clause with the respective domain - when this is empty then the literal disappears. If a support includes the corresponding domain, then the whole clause can be deleted (unit subsumption). Whenever a new unit is obtained, this will reduce the respective domain. When no new unit can be obtained, then one must branch on one of the variables, by splitting its domain. (We present here experiments with few splitting strategies.)

While the operations above are straightforward generalizations of boolean constraint propagation, MDL also benefits from specific strategies. Similarly to signed-logic, one may detect certain redundant elements in domains, which we call *weak assignments*: If an element a of a domain occurs in all supports together with another element b, then we say *a is weaker then b* and a can be eliminated from the domain. Novel in our approach is the use of a technique which we call

variable merging. This consists in replacing two or more variables by a new one, whose domain is the cartesian product of the old domains. In each clause, the disjunction of the literals containing the old variables is replaced by one literal whose support is constructed in a straightforward way. Variable merging is used at the beginning of the solving process in order to translate a boolean SAT problem into signed logic, but also during the solving process in order to reduce the number of variables, when some domains become relatively small.

Multi-domain logic (MDL) was introduced in [4][3], which also presents the first practical experiments with this method for solving signed logic problems directly, and in particular those which are constructed from boolean SAT problems. The idea of MDL solving improves on earlier boolean solvers based on simultaneous propagation of several boolean units [5, 6].

The purpose of this paper is to illustrate the effectiveness of the MDL version of the DPLL algorithm and to investigate the efficiency of different combinations of specific strategies. For rapid prototyping we use an eager algorithm implemented in Java, thus both the size of the SAT instances as well as the absolute timings are not impressive. However we obtain interesting facts when comparing different strategies details,

- Increasing the number of the boolean variables during the original translation leads to a significant speed-up until a certain threshold, which depends on the implementation environment but also on the structure of the original problem.
- The domain splitting strategies during branching have a significant effect on certain classes of problems.
- Both the deletion of weak assignments, as well as the dynamic merging have a notable impact on efficiency.

These findings constitute a good motivation for a more complex implementation using lazy techniques, and for further experiments and possible theoretical developments.

2 Proof Techniques

Variable merging (VM). Merging two variables x, x' (with domains D, D') consists in replacing x, x' by a new variable y, which (intuitively) represents the pair $\langle x, x' \rangle$ and ranges over $D \times D'$. A disjunction $A : x \vee A' : x'$ becomes $((A \times D') \cup (D \times A')) : y$. By induction, merging extends to an arbitrary number of variables.

Variable clustering. Classical boolean variables can be seen as ranging over the domain $\{0, 1\}$. As a first step of the MDL based SAT solving algorithm, the boolean problem is transformed by clustering the variables. The *clustering factor* (number of boolean variables per MDL variable) may be fixed (as it is used in

[3] Before we learned about signed logic.

the present experiments) or variable. Before clustering we perform in fact a preprocessing step for detecting which variables occur more often together in clauses, and we try to group them together. The experiments presented in our previous work [4] show that the preprocessing results in a moderate increase in the efficiency of the MDL solving process.

Dynamic Variable merging. We also experiment with *dynamic merging*, that is binary merging of variables during the execution of the DPLL algorithm. When no new units can be obtained, then merging of two variables which occur together in a binary clause creates a new unit, thus split can be avoided. We apply merging if the size of the new domain is not bigger than the a 2^{k+t}, where k is the original clustering factor and t is a *merging threshold*.

Generalized DPLL. The Davis-Putnam-Logemann-Loveland algorithm [3] generalizes to MDL by extending unit subsumption and unit resolution. Let x be a variable symbol, A, B constant sets, and $\mathcal{C}$ a disjunction of MDL literals. The unit clause $A : x$ *subsumes* the clause $(B : x) \vee \mathcal{C}$ if $A \subset B$. The *resolvent* of the unit clause $A : x$ and the clause $(B : x) \vee \mathcal{C}$ is the clause $(A \cap B : x) \vee \mathcal{C}$. In contrast to propositional logic, the literal containing x is not always *canceled*, but only when $A \cap B = \emptyset$. There may be more units containing the same variable. These *collapse* into one unit by resolution, and the resulting set is the new *domain* of the respective variable. During DPLL, each time a new unit is obtained, it is intersected with the current domain of the respective variable in order to obtain the new domain. When the domain becomes empty, we have a contradiction on the respective search branch. As in the DPLL method, we use unit subsumption and unit resolution for performing *unit propagation (UP)*, and *constraint propagation (BCP)*. When all units have been propagated, then either all clauses have been subsumed (we have a solution), or we can create a new unit out of a binary clause by variable merging, or we must *branch* the search tree – which is can done by splitting of one of the domains using in various strategies (see below).

Deletion of weak assignments (DoWA). An assignment (element of a domain) is *weak* if it appears in all supports together with another one, and weak assignments can be ignored. In [2] this is used in order to reduce the branching of the search tree, and we use it (see [4]) in order to reduce the domain.

Branching strategies. In MDL we branch on a partition of the domain of a selected variable. Our present experiments use various choices for the variable and for the partition.
Choice of variable:

- MinDom: a minimal domain;
- MinDomClause: a minimal domain from a minimal clause;
- MinLit: a minimal literal from a minimal clause.

Minimality of a clause, domain, and literal refer to the number of literals, cardinality of the set, and cardinality of the sign, respectively. *Choice of partition:*

– SplitLit: the sign of the literal and its complement (only with MinLit);
– SplitHalf: half of the domain on each branch;
– SplitAll: one branch for each assignment.

When SplitHalf and SplitAll are used together with MinLit, then we generate additionally a branch corresponding to the complement (w.r.t. the domain) of the sign of the respective literal. In signed logic the SplitLit and SplitAll techniques are known and they are described in [2], however without implementation.

3 Implementation and Experimental Results

Data representation. The representation of a MDL formula is described in [4]. Here we only illustrate the idea of our representation by exhibiting the representation of clusters of two propositional variables a, b. The table below lists the bit strings, the corresponding sign sets, and the formula which is encoded.

0000	$\{\}$	$\mathbb{F}$	1000	$\{00\}$	$\neg a \wedge \neg b$
0001	$\{11\}$	$a \wedge b$	1001	$\{00, 11\}$	$a \Leftrightarrow b$
0010	$\{10\}$	$a \wedge \neg b$	1010	$\{00, 10\}$	$\neg b$
0011	$\{10, 11\}$	a	1011	$\{00, 10, 11\}$	$a \vee \neg b$
0100	$\{01\}$	$\neg a \wedge b$	1100	$\{00, 01\}$	$\neg a$
0101	$\{01, 11\}$	b	1101	$\{00, 01, 11\}$	$\neg a \vee b$
0110	$\{01, 10\}$	$a \times b$	1110	$\{00, 01, 10\}$	$\neg a \vee \neg b$
0111	$\{01, 10, 11\}$	$a \vee b$	1111	$\{00, 01, 10, 11\}$	$\mathbb{T}$

These strings of bits can be represented as a list (or array) of computer words, and then union and intersection can be computed using hardware logical operations on words: union becomes *bitwise or* and intersection becomes *bitwise and*. For instance, if we merge 5 propositional variables into a multi-variable, then its domain has $2^5 = 32$ elements, so we can represent it on a 32-bit computer word. For higher clustering factors, however, the length of the representation in words grows exponentially, thus from a certain value the positive effect of clustering will be canceled by the higher cost of multi-word operations (as also confirmed by our experiments).

Experimental Results In order to demonstrate the effectiveness of multi-domain logic and to check the basic implementation principles, we implemented a variant of DPLL method described in Section 2.

The implementation is realized in Java using eager principles, with the main purpose is not to compete with the modern SAT solvers based on lazy data structures, but to allow us to determine what is the effect of using various techniques in various combinations. We used a HP Compaq nx6110 notebook (32 bits) with Pentium M 1.86 MHz processor and 512 MB memory.

We tested our implementation on some unsatisfiable problems from the SATLIB – Benchmark Problems homepage (`www.satlib.org`). Namely, we used the `uuf-50-01.cnf` problem and the `hole6.cnf` problem, because they belong to a scalable class of problems, and are of a complementary nature. The former

is randomly generated and has no structure or symmetry, while the latter is symmetric and well structured.

We experiment with unsatisfiable problems, because this gives more reliable information on the general behavior of the program. On satisfiable problems the running time may be influenced by the randomness of early found solutions.

We investigate the main aspects of the novel method: cluster preprocessing, branching strategies, deletion of weak assignments (DoWA), variable merging (VM), and the variable merging threshold.

All experiments are performed with increasing clustering sizes (number of original propositional variables per MDL variable).

Cluster preprocessing. The effect of cluster preprocessing confirms what we reported in the previous paper: a speed-up between $40 - 60\%$ for the randomly generated problems, and almost no speed-up for the pigeon–hole problems (because they are already well clustered). Therefore we will show in detail only the effect on the randomly generated problem `uuf-50-01.cnf`.

Branching strategies. In the previous section we presented 3 split strategies and 3 choice strategies, which we present in 4 combinations – which appear to be more efficient.

First we show the running time of this methods on the `uuf50-01.cnf` problem from the SATLIB page. Figure 1 shows the case where we do not use any simplification techniques.

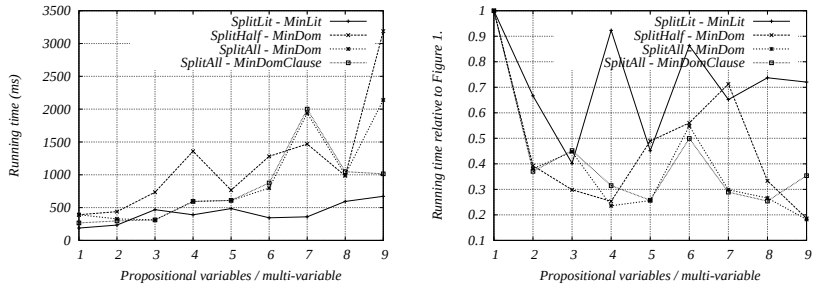

Fig. 1. `uuf50-01.cnf`: Absolute timings without-, and relative timings with clustering.

Dynamic variable merging. To create a new unit out of a binary clause is quite expensive, because we have to know the exact domains of the two variables. The current implementation is lazy in the sense that it recalculates a domain only if a new unit is found for the corresponding variable. Therefore, we use the following heuristic for deciding wether to compute the domains or not. Assume the binary clause is $A : x \vee B : y$. If $|A| * |B| \leq 2^{k+t-1}$, where k is the clustering factor

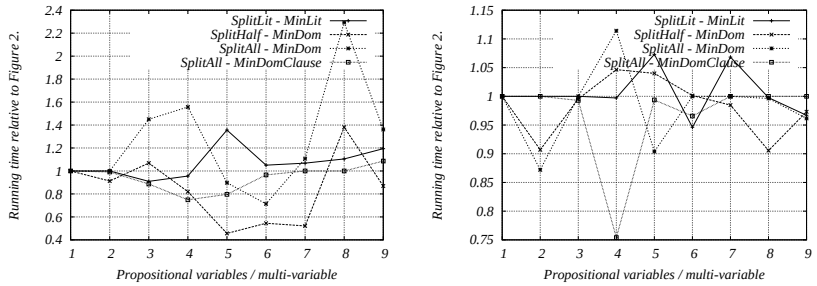

Fig. 2. `uuf50-01.cnf`: DoWA and VM (timings relative to Fig. 1. with clustering).

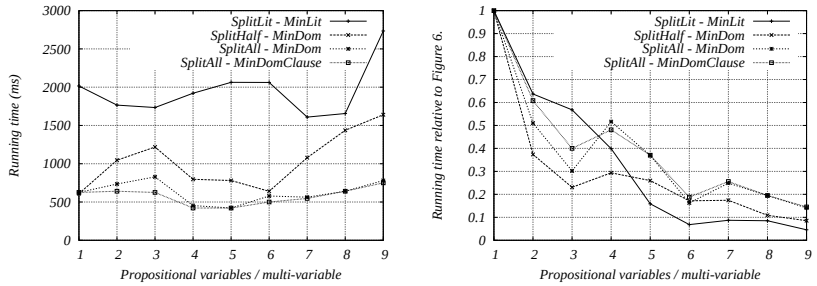

Fig. 3. `hole6.cnf`: Absolute-, and relative timings when using DoWA and VM.

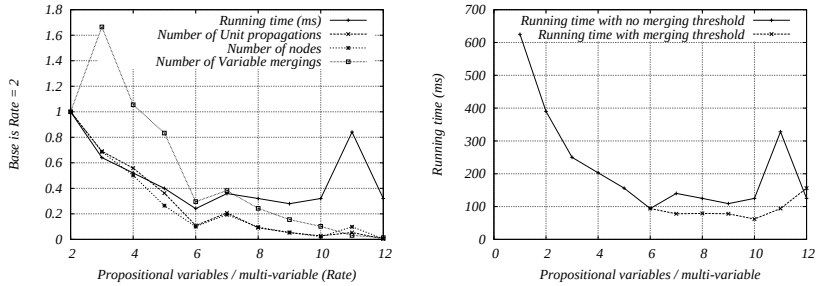

Fig. 4. `hole6.cnf`: Statistics of SplitAll - MinDomClause ; Effect of merging threshold.

and t is a merging threshold, then we calculate the domains of x and y. This heuristic performed very well. It covered the 88% of the cases where the the new domain really fits in 2^{k+t} bits and its guess was never wrong.

Deletion of weak assignments and Variable merging. The problems are already cluster preprocessed.

In Figure 2 we can see the effect of deletion of weak assignments, for short DoWA. Note that in some cases it even slows down the search, and in some cases it gives $5-15\%$ speed-up. The average of these values is 101%. For a clustering factor of 5 the average is 86%.

In the same Figure 2 we can see the effect of variable merging (VM). The effect ranges from slow down to $2-8\%$ speed-up. It appears that DoWA gives slightly better speed-up than VM, but in fact the average of these values are 99%, which is a bit better than in case of DoWA, but DoWA gives bigger speed-up.

In conclusion, on this uniform random 3-SAT problem, the effect of cluster preprocessing is large but the one of DoWA and VM is small.

We have also tested this simplification techniques on the "pigeonhole problem" `hole6.cnf` (7 pigeons and 6 holes). Figure 3 shows the absolute running time of 4 branching strategies, as well as the improvement of efficiency (as relative time) when using DoWA and VM together. It is difficult to judge which branching method is the best. The effect of DoWA and VM together on `hole6.cnf` is very significant, like $40-90\%$ speed-up. The average of these values is 39%. So in this case we obtain very good results with the same simplification techniques which gave virtually no speed-up in case `uuf50-01.cnf`. This is probably due to the regular structure of the pigeonhole problem.

Dynamic merging of variables. As mentioned in section 2, merging of variables can be further stimulated by allowing new MDL variables of greater size than the original ones (as controlled by the parameter *merging threshold*). In Figure 4 (left) we see that bigger merging threshold leads in general to a higher number of variable merging and, therefore, less nodes in the search tree. We would expect here an exponential curve since the bigger the merging threshold is the bigger the probability that two domains can be merged (note, that in this implementation the size of a domain is limited to 2^k, where k is rate of propositional variables / multi-variable, k is fixed). If the merging threshold is $2k$, then virtually all binary clauses can be merged (only those not, which contains an already merged multi-variable). But we see that the number of variable merging starts to increase and then remains almost the same, then drops.

The explanation of this observation is that the number of unit propagations steps starts to decrease quite fast. It is so because each variable merging prevents a split and allow a unit propagation instead (note that we merge multi-variables only if they occur in a binary clause, which, of course, becomes a unit after variable merging). This cuts the search spaces (an early variable merge during the search can dramatically cut it) and, therefore, the number of unit propagation steps decreases. But if the search space is smaller then the probability of variable merges is also smaller. Furthermore, a merged multi-variable is less likely takes

place again in a merge, because its domain has $A * B$ elements, where A, B are the size of the domains before merging. Hence, the number of variable merges cannot grow exponentially as the merging threshold grows.

In Figure 4 (right) we see that to solve the `hole6.cnf` problem we need only few milliseconds if we use variable merging, rate 6 (which is fixed in this figure) of propositional variables / multi-variable, and merging threshold 7 . This is even better if we have rate 13 but no dynamic merging. The explanation of this observation is that with rate 6 all clauses of the problem are already binary and they can be immediately merged because of the high merging threshold, $6 + 7 \geq 2 * 6$. Note that merging threshold 6 also was this property; indeed lot of the input clauses can be merged, but not those which contain already merged multi-variables. So with this high merging threshold after variable merging we have all units, so we spend only time on initialization and variable merging to solve hole6.cnf. One would except the same behavior in case rate 13 if we have no dynamic merging. But this does not guarantee that all clauses are units, we have lot of units but also some binaries, this is why it needs more time. To summarize, it is better to use rate k and merging threshold t than rate $k + t$.

This would suggest to choose very big numbers for the clustering factor k, and and for the threshold t, but then we need 2^{k+t} bits to store a literal (at least in the current implementation), which results in the unfeasibility of the operations on literals after certain limits.

4 Conclusions

The multi-domain approach to satisfiability solving represents an effective alternative to boolean SAT algorithms, although its possible superiority in efficiency remains doubtful in absence of more sophisticated theoretical and practical investigations.

However, the results obtained with this preliminary eager implementation constitute a good motivation for further refinements of the techniques and of the solving strategies.

Further work includes more comprehensive experiments with more classes of problems and with larger instances, as well as more sophisticated lazy implementations generalizing the current efficient methods used in SAT solvers.

In contrast to boolean logic, MDL presents more rich opportunities for theoretical insights and algorithmic developments regarding various aspects of the solving process: the preprocessing for static and for variable clustering, the choice of domain for splitting, the domain splitting strategies for branching, the strategies for the merging of variables, the representation of domains and of signs, as well as the data structures representing variables, clauses, etc.

References

[1] C. Ansotegui, J. Larrubia, Chu Min Li, and F. Manyà. Mv-Satz: A SAT solver for many-valued clausal forms. Proceedings of the 4th International Conference on Knowledge discovery and discrete mathematics, Metz, France, 2003, pp. 143–150.

[2] B. Beckert, R. Hähnle, and F. Manyà. The SAT problem of signed CNF formulas. Labelled Deduction, volume 17 of Applied Logic Series, pp. 61–82. Kluwer, Dordrecht, May 2000.

[3] M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem Proving. Communications of the ACM, 5:394–397, 1962.

[4] T. Jebelean, G. Kusper. Multi-Domain Logic and its Applications to SAT. Proceedings of SYNASC'08, IEEE Computer Society Press, ISBN 978-0-7695-3523-4, pp. 3–8, 2008.

[5] G. Kusper. Solving the SAT Problem by Hyper-Unit Propagation. RISC Technical Report 02-02, 1-18, University Linz, Austria, 2002.

[6] G. Kusper, L. Csőke. Better test results for the graph coloring and the Pigeonhole Problems using DPLL with k-literal representation. Proceedings of ICAI-2007, Volume II. 127-135, Eger, Hungary, January 2007.

[7] F. Manyà, R. Béjar, and G. Escalada-Imaz. The satisfiability problem in regular CNF-formulas. Soft Computing: A Fusion of Foundations, Methodologies and Applications, 2(3):116–123, 1998.