Multi-Domain Logic
as a Tool for Program Verification

Gábor Kusper$^1$ and Tudor Jebelean$^2$

$^1$ Eszterházy Károly College, Eger, Hungary
gkusper@aries.ektf.hu
$^2$ RISC–Linz, Johannes Kepler University, Austria
Tudor.Jebelean@jku.at

Many software and hardware verification problems are reduced to boolean satisfiability (SAT) of relatively large formulae. (In fact, most SAT challenges come from practical verification of information systems.) Therefore, an important issue in practical verification is the development of very efficient SAT algorithms, because this will enlarge significantly the class of verification problems which can be solved. Moreover, various software verification techniques (notable model checking and in particular bounded model checking) generate satisfiability problems which are not boolean, but they refer to a generalized model of propositional logic in which a variable may have more than two values. Currently these problems are reduced to boolean SAT because the availability of SAT tools, however this involves a certain loss of information (wrt. the original problem) and also a certain loss of efficiency. Therefore direct solving algorithms for multi-valued logic are also very important for software verification. The approach presented here addresses both of these problems: (1) it increases the efficiency of boolean SAT by transforming it into multi-domain SAT which propagates units more efficiently, and (2) it solves directly SAT problems in multi-valued logic.

Multi-Domain Logic (MDL)[3] is a generalization of signed logic[1], 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. In fact, 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.

An MDL problem is a set of clauses which are disjunctions of literals. Every literal is of the form $S : x$, where $x$ is a symbolic variable and $S$ is a finite set
which is called support. The domain of a variable is the set of possible values, and a solution is an assignment of variables which makes all clauses true.

Our experiments use the following techniques which are specific to MDL:

**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 $A : x \lor A' : x'$ becomes $((A \times D') \cup (D \times A')) : y$. Similarly, merging extends to an arbitrary number of variables.

**Variable clustering.** Boolean formulae are MDL formulae over the domain $\{0, 1\}$, by merging we can cluster several boolean variables into one. The clustering factor (number of boolean variables per MDL variable) may be fixed or variable. Before clustering we perform in fact a preprocessing step in order to reduce the length of the resulting clauses.

**Dynamic Variable merging.** The novel technique implemented here is dynamic merging, that is binary merging of variables during the execution of the DPLL algorithm [2]. This creates new units, thus branching is avoided. We apply merging if the size of the new domain is not bigger than $2^{k+t}$, where $k$ is the clustering factor and $t$ is the merging threshold, which is a non-negative constant.

**Generalized DPLL.** Let $x$ be a variable symbol, $A, B$ constant sets, and $C$ a clause. As in signed logic, a unit clause $A : x$ subsumes the a clause $(B : x) \lor C$ if $A \subseteq B$. The resolvent of the unit clause $A : x$ and the clause $(B : x) \lor C$ is the clause $(A \cap B : x) \lor C$. The literal containing $x$ canceled only when $A \cap B = \emptyset$.

Several units containing the same variable collapse into one unit by intersection (the new domain). During DPLL, new units reduce by intersection the respective domain (empty domain means contradiction). Unit propagation (UP), and constraint propagation (BCP) are performed as in DPLL. After propagation, 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 various strategies [4].

**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 [1] this is used in order to reduce the branching of the search tree, and we use it in order to reduce the domain.

**References**


