DLV^{MC}: Enhanced Model Checking in DLV

Marco Maratea[2], Francesco Ricca[1], and Pierfrancesco Veltri[1]

[1] Department of Mathematics, University of Calabria, 87036 Rende (CS), Italy
{veltri,ricca}@mat.unical.it
[2] DIST, University of Genova, 16145 Genova, Italy
marco@dist.unige.it

Abstract. Stable Model Checking (MC) in Answer Set Programming systems is, in general, a co-NP task for disjunctive programs. Thus, implementing an efficient strategy is very important for the performance of ASP systems. In DLV, MC is carried out by exploiting the SAT solver SATZ, and the result of this operation also returns (in case the check fails) an "unfounded set", as by-product, which is also used for pruning the search space during answer set computation.
In this paper we report on the integration of a "modern" SAT solver, MINISAT, in DLV. The integration poses not only technological issues, but also challenges w.r.t. the "quality" of the returned unfounded set and w.r.t. the interplay with the existing DLV techniques.

1 Introduction

Disjunctive Logic Programming under the answer set semantics, a.k.a., Answer Set Programming (ASP for short, [1, 2]), is a powerful declarative formalism for knowledge representation and reasoning [3]. ASP is expressive in a precise sense: it allows to solve any problem belonging to the second level of the polynomial hierarchy. The idea of ASP is to represent a given computational problem by a logic program, the answer sets of which correspond to solutions; and, then, to use an answer set solver to find such solutions [4]. Answer set computation of propositional (i.e., without variables) ASP programs is carried out in DLV [5], as well as in most ASP systems (like e.g., GnT [6], Cmodels [7] and CLASPD [8]) by exploiting two main modules: model generator and a model checker. The first, which is similar to the DPLL procedure [9] for SAT, generates "candidate" answer sets; the second verifies if such candidates are indeed answer sets. On the class of non Head-Cycle-Free (HCF) [10] disjunctive programs, stable model checking is co-NP-complete. To handle this case, DLV exploits a reduction to a satisfiability (SAT) problem [11]: verifying if the candidate solution is "stable", i.e., if it is an answer set, is reduced to checking the unsatisfiability of a SAT formula ϕ, built from the ASP program and the candidate solution. If the formula is satisfiable, then the MC module of DLV also returns a set of "unfounded" atoms, i.e., a set of atoms which can be freely assigned to false. Such unfounded set corresponds to the set of atoms assigned to true in a satisfying interpretation of ϕ returned by the SAT solver (thus, it is available for free after a MC call) and is exploited in the model generator of DLV for enhancing the search together with other techniques like partial checking [11–14]. In particular, once a (total) model check fails during the search DLV goes back in the

search and performs partial model checks while backtracking, in order to unroll the choices in current partial interpretations causing the original stability failure. This technique is combined with another approach, in which DLV speculatively performs partial model checks while moving forwards (as opposed to backtracking) in the search tree. Those checks allows to detect in advance (i.e., before reaching a complete assignment) that the current branch of the search tree is actually inconsistent.

As a matter of fact, DLV performance on non-HCF programs depends on both the efficiency of the SAT checker employed and the "quality" of the returned "unfounded set" (in case the candidate solution is not stable).

In this paper, we report on the integration of a "modern" SAT solver, MINISAT, in DLV, which results in the enhanced system DLV^{MC}. The integration poses not only technological issues; indeed, modern SAT solvers are significantly different from previous propositional checkers such as SATZ (which is the SAT solver employed in the standard version of DLV). The difference is not only in the data structures and/or in the optimization techniques implemented but also in the termination conditions. A CNF formula is declared satisfiable if either all variables of the problem have been assigned (without any conflict) as in MINISAT; or when all clauses have been satisfied, as in SATZ. The choice of the termination condition clearly affects the nature of the returned assignment, and thus the "quality" of the unfounded set. The branching heuristic we used within MINISAT by default assigns all variables to false, i.e., it guarantees to return subset-minimal "minimal" unfounded sets (see, e.g., [15]). Theoretically, imposing an ordering to be followed while branching can have significant degradation in performance [16], but some preliminary experiments shows that this approach can pay off when exploited in DLV, sometimes by orders of magnitude.

2 System usage and options

DLV^{MC} has to be invoked as follows:

*./*DLV^{MC} *-solver <> [-heurm <>] [filename [filename [...]]]*

The command line of DLV^{MC} inherits all the options of DLV, and adds some new options. In more detail: (i) "-solver" indicates the SAT solver to be used for stable model checking. The two SAT solvers available are SATZ and MINISAT ("satz" and "minisat" are the specific strings to be specified in place of <>). DLV^{MC} relies on MINISAT, but the default setting of DLV (i.e., SATZ) can be also selected; and (ii) "-heurm" specifies the type of heuristic used inside MINISAT, by allowing to employ heuristics where variables are first assigned negatively, or positively, or randomly ("neg", "pos", "ran" are the specific strings to be specified in place of <>). Negative first heuristic is the default setting. The option has no effect in case of SATZ.

System Availability. The home page of the system, together with the Linux executables and the QBF benchmarks used are available at: *http://www.mat.unical.it/ ricca/DLVMC*.

3 Preliminary analysis

We have performed a preliminary experimental analysis involving both hard QBF benchmarks coming from the QBF evaluations, and StrategicCompanies benchmarks. The

instance	DLV -PC	DLV^{MC} -PC	#MC	DLV -noPC	DLV^{MC} -noPC	CLASPD
qbf1	0.85	0.88	65	0.92	0.96	2.31
qbf2	3.95	3.60	129	4.15	3.86	12.93
qbf3	14.53	14.14	257	15.43	15.17	63.47
qbf4	77.39	70.80	513	82.76	75.65	315.68
x100.0q	1.71	0.97	3	28.12	10.14	4.71
x110.0q	13.79	10.96	12/7	95.26	60.19	279.89
x135.02q	80.00	113.24	12/9	TIME	TIME	TIME
x145.0q	137.82	51.86	12/7	TIME	992.80	TIME
x150.02q	233.73	49.89	15/8	TIME	876.94	TIME
x150.04q	208.11	76.74	12/7	TIME	984.08	TIME
x165.03q	376.58	842.01	13/8	TIME	TIME	TIME
x165.04q	286.10	1084.68	12/8	TIME	TIME	TIME
x170.0q	TIME	503.42	−/8	TIME	TIME	TIME
x170.02q	919.03	475.11	15/7	TIME	TIME	TIME
x175.01q	TIME	763.78	−/8	TIME	TIME	TIME
x175.04q	816.79	689.10	15/9	TIME	TIME	TIME
x185.0q	TIME	527.99	−/9	TIME	TIME	TIME

Table 1. Experimental analysis: DLV and DLV^{MC} with (-PC) and without (-noPC), and CLASPD, on QBF and StrategicCompanies benchmarks.

second ones being the only "hard" benchmarks submitted to the last ASP competition. All experiments were run on a machine equipped with two Intel Xeon "Woodcrest" (quad core) processors clocked at 3.GHz with 4MB of Level 2 Cache and 4GB of RAM, running Debian GNU Linux 4.0. Time measurements have been done using the `time` command provided by the system, counting total CPU time for the respective process. We report the results in terms of execution time for finding one answer set, if any, within 20 minutes ("TIME" otherwise). The virtual memory available to the solvers has been limited to 512MB. qbf1-qbf4 instances are translation of the bigger MutexP QBF instances. Few StrategicCompanies benchmarks are non solved by any system.

Results are presented in Table 1, where the first column is the instance, the second and third columns contain results for DLV with partial check (-PC) employing SATZ and MINISAT, respectively, the fourth column contains the number of stability checks performed (if "DLV -PC" and "DLV^{MC} -PC" perform the same number of checks there is one number), the fifth and sixth columns are the same as the second and third, but with partial check disabled (-noPC), and the last column contain results for CLASPD [8], as reference. "DLV -PC" and "DLV^{MC} -PC" always perform the same number of stability checks on these benchmarks.

On QBF benchmarks, the advantages of DLV^{MC} over DLV is of around a 10% on both configurations, i.e., with and without PC: the number of stability checks is here always the same, and the advantages of DLV^{MC} seem to be due to the efficiency of MINISAT. All DLV-based versions are much faster than CLASPD. On the StrategicCompanies benchmarks, the impact of MINISAT is instead much higher: on all but two instances, "DLV^{MC} -PC" performs much better than "DLV -PC", solving three

instances more among the biggest showed. Considering the related number of stability checks, here is very important both the efficiency of MINISAT *and* the quality of the returned unfounded sets. As a matter of fact, PC is of fundamental importance for the efficiency of DLV; nonetheless, the results without MC show an even more significant and consistent gain for the version with MINISAT, even if the number of stability checks is the same. CLASPD solves only the two smallest instances of this set.

Acknowledgements. We thank Nicola Leone for fruitful discussion about the topic of the paper. This work has been partially supported by the Calabrian Region under PIA (Pacchetti Integrati di Agevolazione industria, artigianato e servizi) project DLVSYSTEM approved in BURC n. 20 parte III del 15/05/2009 - DR n. 7373 del 06/05/2009. Part of this work has been done while the first author has been with the Università degli Studi e-Campus.

References

1. Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming. In: ICLP/SLP 1988, Cambridge, Mass., MIT Press (1988) 1070–1080
2. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. NGC **9** (1991) 365–385
3. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. CUP (2003)
4. Lifschitz, V.: Action Languages, Answer Sets and Planning. In: The Logic Programming Paradigm – A 25-Year Perspective. (1999) 357–373
5. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM TOCL **7**(3) (2006) 499–562
6. Janhunen, T., Niemelä, I., Seipel, D., Simons, P., You, J.H.: Unfolding Partiality and Disjunctions in Stable Model Semantics. ACM TOCL **7**(1) (2006) 1–37
7. Lierler, Y.: Disjunctive Answer Set Programming via Satisfiability. In: LPNMR'05. LNCS 3662, (2005) 447–451
8. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-driven answer set solving. In: IJCAI 2007,(2007) 386–392
9. Davis, M., Logemann, G., Loveland, D.: A Machine Program for Theorem Proving. Communications of the ACM **5** (1962) 394–397
10. Ben-Eliyahu, R., Dechter, R.: Propositional Semantics for Disjunctive Logic Programs. AMAI **12** (1994) 53–87
11. Koch, C., Leone, N., Pfeifer, G.: Enhancing Disjunctive Logic Programming Systems by SAT Checkers. AI **15**(1–2) (2003) 177–212
12. Leone, N., Rullo, P., Scarcello, F.: Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics and Computation. Inf.Comp. **135**(2) (1997) 69–112
13. Pfeifer, G.: Improving the Model Generation/Checking Interplay to Enhance the Evaluation of Disjunctive Programs. In: LPNMR-7. LNCS 2923, (2004) 220–233
14. Janhunen, T., Niemelä, I., Simons, P., You, J.H.: Partiality and Disjunctions in Stable Model Semantics. In: KR 2000, 12-15,(2000) 411–419
15. Giunchiglia, E., Maratea, M.: Solving optimization problems with DLL. In: Proc. of the 17th European Conference on Artificial Intelligence (ECAI 2006). Volume 141 of Frontiers in Artificial Intelligence and Applications., IOS Press (2006) 377–381
16. Järvisalo, M., Junttila, T.A., Niemelä, I.: Unrestricted vs restricted cut in a tableau method for boolean circuits. Annals of Mathemathics and Artificial Intelligence **44**(4) (2005) 373–399