A Framework for the Specification of Random SAT and QSAT Formulas*

Nadia Creignou[1], Uwe Egly[2], and Martina Seidl[3,4]

[1] Laboratoire d'Informatique Fondamentale CNRS UMR 7279,
Aix-Marseille Université, France
[2] Institut für Informationssysteme 184/3, Technische Universität Wien, Austria
[3] Institute for Formal Models and Verification, Johannes Kepler University, Austria
[4] Institut für Interaktive Systeme 188/3, Technische Universität Wien, Austria

Abstract. We present the framework [q]bfGen which allows the declarative specification of random models for generating SAT and QSAT formulas not necessarily in (prenex) conjunctive normal form. To this end, [q]bfGen realizes a generic formula generator which creates formula instances by interpreting the random model specification expressed in XML. Consequently, the implementation of specific random formula generators becomes obsolete, because our framework subsumes their functionality.

1 Motivation

Over the last years, tools for solving the satisfiability problem of propositional logic (SAT) showed to be powerful backend engines for various hardware and software verification problems [14]. The same hope is pined on extensions like QSAT, where quantifiers are introduced over the propositional variables allowing more succinct encodings of verification problems [2]. So far, QBF solvers have not reached the same maturity as SAT solvers in terms of efficiency and stability. Techniques which showed to be useful in SAT can often not directly be transferred to QBF. Whereas in SAT conjunctive normal form is the canonical input format, the pendant for QSAT, the prenex conjunctive normal form (PCNF), is not the commonly accepted representation format. In fact, the transformation to PCNF might negatively influence the behavior of a solver [6]. Consequently, QSAT solvers have been developed which process non-PCNF formulas, i.e., formulas of less restricted structure [7,10,11].

In SAT as well as in QSAT, random formulas find their *raison d'être* justified in two different use cases. From a theoretical point of view, random formulas provide the basis for investigations on properties like the phase transition phenomenon [9,8,4,5]. From a practical point of view, they are an important tool

* This work was partially funded by the Vienna Science and Technology Fund (WWTF) through project ICT10-018, by the Austrian Science Fund (FWF) under grant S11409-N23 and by the Agence Nationale de la Recherche under grant ANR-09-BLAN-0011-01.

A.D. Brucker and J. Julliand (Eds.): TAP 2012, LNCS 7305, pp. 163–168, 2012.

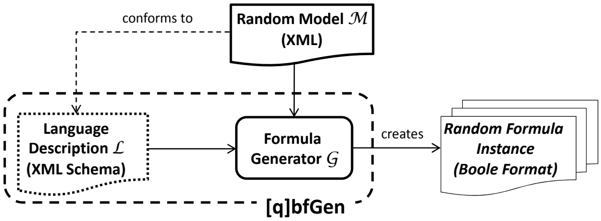
conforms to
Random Model $\mathcal{M}$ (XML)
Language Description $\mathcal{L}$ (XML Schema)
Formula Generator $\mathcal{G}$
creates
Random Formula Instance (Boole Format)
[q]bfGen

Fig. 1. The basic architecture of [q]bfGen

for testing and evaluating solvers. In particular, random formulas are used for fuzz testing which supports the automatic detection of various kinds of defects in solvers [3] by random inputs.

Random models provide control mechanisms for randomly generating formulas of a certain structure and size. The regularity of the formula structure allows for a characterization by statistical and combinatorial means, which in turn allows for a prediction of properties such as satisfiability and computational difficulty. Since most state-of-the-art solvers process formulas in (prenex) conjunctive normal form only, also most random models describe and generate formulas in PCNF. This restricted structure allows only a small set of parameters like number and distribution of variables, clause sizes, and the probability that a variable occurrence is negated to vary in the random model.

With the advent of non-PCNF solvers, also random models are required which generate formula instances of less restricted structure and which, consequently, introduce additional degrees of variability like the nesting depth of the formula tree. To support the specification of such random models, we introduce the framework [q]bfGen which provides a dedicated language for the description of random models as well as a generic formula generator which creates random formula instances according to such descriptions. By using [q]bfGen, the functionality of specific random formula generators like [13,9,4] can be realized by giving simple declarative descriptions of the according random model. For demonstration purposes, we extend the shape model of Navarro and Voronkov [13] for quantified Boolean formulas (QBF).

2 The Architecture of [q]bfGen

Our framework [q]bfGen allows the description of SAT and QSAT random models in XML from which formula instances are directly created. The current prototype uses the Boole format[1] for the the representation, in future implementations we consider to support also other output formats. As illustrated in Fig. 1, [q]bfGen consists of two main components: (i) the *language specification* $\mathcal{L}$ and (ii) the *formula generator* $\mathcal{G}$. Within [q]bfGen a random model must be expressed in conformance to $\mathcal{L}$. The resulting random model description $\mathcal{M}$ is then passed

[1] http://www.qbflib.org/boole.html

to $\mathcal{G}$, which generates random formula instances according to $\mathcal{M}$. Finally, these formula instances are provided to a SAT/QBF solver and evaluated. In the following, both $\mathcal{L}$ and $\mathcal{G}$ are presented in detail.

The Language Specification $\mathcal{L}$. In our implementation, $\mathcal{L}$ is realized as XML Schema. For ease of presentation, we use the notation of the UML Class Diagram to visualize a selection of concepts provided by $\mathcal{L}$. In Fig. 2, we show a simplified Class Diagram of the language specification $\mathcal{L}$. Each random model has one single element Root, which contains an arbitrary number of parameters and the actual formula. A Parameter element has a unique name within a random model and is characterized by a minimum value attribute, a maximum value attribute, and a step width attribute. With parameters it is possible to specify iterations for generating multiple formula instances with different settings. A Formula is either a Quantified Formula or a Connective Formula. A Quantified Formula has a unique name and introduces a new quantifier scope of a specified size which is either of existential, of universal, or of random type. In the case of random type, the quantifier is randomly selected for each instance. The size is either a fixed number or it may be assigned by a parameter. For example, a QBF of the form $\forall x_1 x_2 x_3\ \phi$ may be described by a Quantified Formula where the name of the scope is x, the size is 3, the type is universal, and ϕ is a Formula. A Connective Formula is translated to a conjunction or a disjunction. These connectives are of arbitrary arity. For example $(\neg x \wedge y \wedge \phi)$ could be an instantiation of a conjunction where x and y are variables and ϕ may be a complex formula. The variables occurring in a Connective Formula are specified by a VarSet element which states the probability for a variable being negated as well as from which quantifier block how many variables shall be selected. When a random formula instance is created, it can be assumed that within all instantiations of a VarSet, each variable occurs at most once. To specify that the variables shall occur in all branches of the subformula where the VarSet is defined, the position attribute must be set to a positive integer (the relative distance from the current position in the formula tree). In this way, it is possible to ensure that a variable is only instantiated once within the subformula. An example for this feature follows in the next section. The Formula element contains an attribute duplicates. For example, if a conjunction shall contain three clauses of a certain size, then the clause is specified only once and the duplication attribute is set to 3.

The Formula Generator $\mathcal{G}$. With the language specification $\mathcal{L}$ formulated in XML Schema, specifications of random models are expressed in XML. For the creation of formula instances out of random model specifications, we provide the formula generator $\mathcal{G}$, which is implemented as a command line tool in Java using Apache XMLBeans. When $\mathcal{G}$ is started it requires arguments like the random model, a set name for the formulas, and the number of formula instances to be generated. First $\mathcal{G}$ parses the provided random model and checks if it is conformant to $\mathcal{L}$ and if no constraints are violated. Such constraints assert that no parameter name is used which has not been specified, that no minimum value is greater than a maximum value, etc. When the random model passed

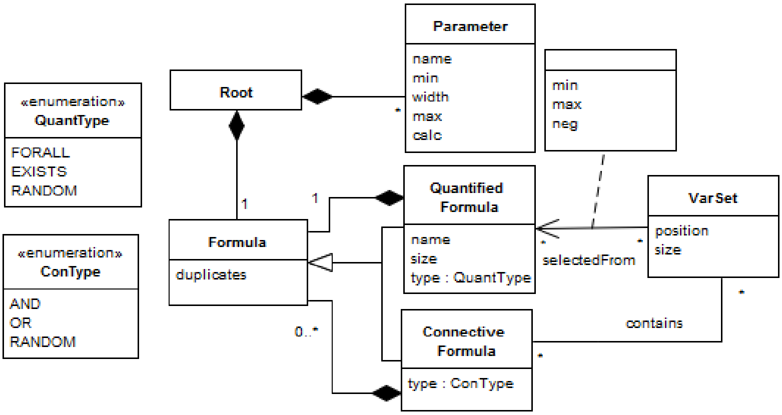

Fig. 2. Language specification

these tests, the formula instances are generated. Each formula instance is stored in an individual file. In the current version of $\mathcal{G}$, the output format is Boole, a standard format for QBF. If the `-p` flag is set, then no quantifiers are printed, i.e., the generated formulas are propositional formulas.

3 Tool Demo: The Fixed-Shape Model for QBF

In this section, we first present an extension of the fixed-shape model by Navarro and Voronkov [13] to QBF and then show how it is specified within our framework. For pedagogical reasons we focus on the specific $\langle 2,2,3\rangle$-*shape*. The method can then easily be extended to any *balanced shape* as defined in [13].

The Fixed-Shape Model for SAT and QBF. A $\langle 2,2,3\rangle$-*shape* is an alternating-$\{\vee,\wedge\}$-formula tree where the root node is a disjunction having two conjunctions as subformulas. Each of these conjunctions contains two clauses of size three. A $\langle 2,2,3\rangle$-*constraint* over the set of variables X is any instantiation of the above tree obtained by replacing the variables in the tree by possibly negated distinct variables from X. A $\langle 2,2,3\rangle$-*formula* is a conjunction of $\langle 2,2,3\rangle$-constraints. Observe that such formulas are in negation normal form

We are interested in creating random QBF instances of the form $\forall X \exists Y \phi$, where X and Y are sets of variables and ϕ is a $\langle 2,2,3\rangle$-formula. This extension of the fixed-shape random model introduced in [13] to quantified formulas requires the following additional parameters:

- The first parameter is the pair (m, n) specifying the number of variables in each quantifier block (size of X, size of Y).
- The second parameter is a pair (u, e), which fixes the number u of universal variables and the number e of existential variables that occur in each deepest, non-leaf subtree of every $\langle 2,2,3\rangle$-constraint of ϕ. Thus, $u + e = 3$. Here we fix $u = 1$ and $e = 2$.
- The third parameter L is the number of $\langle 2,2,3\rangle$-constraints in ϕ.

Thus, we obtain a random (m, n)-$(1, 2)$-L-$\langle 2, 2, 3\rangle$-formula in choosing uniformly, independently and with replacement L constraints among all the possible ones that fulfill the above requirements. Note that the random model we propose is inspired by [8] and [4] for QBF in PCNF.

Realization in [q]bfGen. For the specification of the random model described above, the following steps are necessary:

1. To vary the ranges of X and Y, we specify two parameters m and n.
2. As we are interested in the probability that a formula instance is satisfiable when the ratio number of existential variables to number of constraints increases, we range our experiments over $L = rn$ where r is a real value. Therefore we introduce another parameter called width.
3. In the next step, we introduce a Quantified Formula element of universal type for X of size m which itself contains a Quantified Formula element of existential type for Y of size n.
4. Then we specify the outmost conjunction by a Connective Element which contains a description of the $\langle 2, 2, 3\rangle$-constraints. Note that it suffices to specify such a constraint only once, as the duplication may be achieved by the duplicates attribute which is set to the value of the parameter width.
5. The $\langle 2, 2, 3\rangle$-formula starts with a disjunction. Here we also specify a VarSet consisting of one variable selected from X and two variables selected from Y. The position attribute of this VarSet is set to two, stating that the variables are not inserted immediately, but in the clauses occurring two levels below in the formula tree. So we can insure, that these clauses do not share any variables.
6. Finally, we specify a conjunction containing a disjunction which are both duplicated twice realizing the two "2" in $\langle 2, 2, x\rangle$. The last disjunction is of arity three due to the literals obtained from the VarSet specified above.

We kindly refer to our project site [1] where we discuss the random model in detail and where we also show first experiments. Further (P)CNF random models from literature like [4,5,8,12] can also easily be handled.

4 Conclusion and Future Work

We introduced the framework [q]bfGen which provides a language for specifying SAT and QSAT random models and a formula generator which interprets such specifications and which creates formula instances accordingly. [q]bfGen is not only valuable in the context of empirical investigations of the properties of randomly generated formulas, but it is also a valuable tool within the solver development process. The randomly generated formulas may then serve as input data for fuzz testing [3], which is a powerful testing technique for such complex tools as solvers. So conceptual as well as programming bugs can be tracked down automatically and the robustness of the solvers increases. When used as backend engine for verification tasks, buggy solvers are worthless. Thus, [q]bfGen might become very helpful for the development of stable software.

We demonstrated our approach by describing a shape model for prenex QBF. The XML Schema, the prototypical implementation of the formula generator, and the detailed specification of the random model presented above are available at our project site [1].

In future work and driven by practical needs, we will include additional language elements like XOR and realize more advanced duplication and iteration mechanisms as provided at the moment.

References

1. qbfGen Project Site, `http://fmv.jku.at/qbfgen/`
2. Benedetti, M., Mangassarian, H.: QBF-Based Formal Verification: Experience and Perspectives. JSAT 5(1-4), 133–191 (2008)
3. Brummayer, R., Lonsing, F., Biere, A.: Automated Testing and Debugging of SAT and QBF Solvers. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 44–57. Springer, Heidelberg (2010)
4. Chen, H., Interian, Y.: A Model for Generating Random Quantified Boolean Formulas. In: Proc. of IJCAI 2005, pp. 66–71. Professional Book Center (2005)
5. Creignou, N., Daudé, H., Egly, U., Rossignol, R.: (1,2)-QSAT: A Good Candidate for Understanding Phase Transitions Mechanisms. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 363–376. Springer, Heidelberg (2009)
6. Egly, U., Seidl, M., Tompits, H., Woltran, S., Zolda, M.: Comparing Different Prenexing Strategies for Quantified Boolean Formulas. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 214–228. Springer, Heidelberg (2004)
7. Egly, U., Seidl, M., Woltran, S.: A solver for QBFs in negation normal form. Constraints 14(1), 38–79 (2009)
8. Gent, I., Walsh, T.: Beyond NP: The QSAT Phase Transition. In: Proc. of AAAI/IAAI 1999, pp. 648–653 (1999)
9. Gent, I.P., Walsh, T.: The SAT Phase Transition. In: Proc. of ECAI 1994, pp. 105–109 (1994)
10. Goultiaeva, A., Iverson, V., Bacchus, F.: Beyond CNF: A Circuit-Based QBF Solver. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 412–426. Springer, Heidelberg (2009)
11. Klieber, W., Sapra, S., Gao, S., Clarke, E.: A Non-Prenex, Non-Clausal QBF Solver with Game-State Learning. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 128–142. Springer, Heidelberg (2010)
12. Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., Troyansky, L.: 2+p-sat: Relation of typical-case complexity to the nature of the phase transition. Random Struct. Algorithms 15(3-4), 414–435 (1999)
13. Navarro, J., Voronkov, A.: Generation of Hard Non-Clausal Random Satisfiability Problems. In: Proc. AAAI/IAAI 2005, pp. 436–442 (2005)
14. Prasad, M.R., Biere, A., Gupta, A.: A survey of recent advances in SAT-based formal verification. STTT 7(2), 156–173 (2005)