A Grounder From Second-Order Logic To QBF

Matthias van der Hallen and Gerda Janssens

August 2, 2018

A Grounder From Second-Order Logic To QBF

Matthias van der Hallen and Gerda Janssens

KU Leuven firstname.lastname@kuleuven.be

Abstract. Recent solver research has developed powerful QBF solvers. Alas, we know of few tools that provide a modelling language on a higher level, translating this to QBF. This is surprising, as in the closely related field of SAT solvers, research has gone hand in hand with the development of such systems.
This extended abstract on work in progress reports on a system that allows the use of second-order logic as a high-level modelling language and that grounds (translates) models written in such a language to a QBF formula. We provide an example encoding, outline the grounding process and propose further research and experiments.

1 Introduction

In the research of solvers, many effort and progress has been made in the area of solving boolean formulas. One example is SAT, but the more expressive QBF is also the subject of more recent research. This has led to the development of powerful techniques, such as QDPLL with conflict-driven clause learning and solution-driven cube learning [6] or CEGAR [8].

While for years, the research in SAT has driven the development of systems that interface SAT solvers with a higher-level modelling language, e.g. IDP, this does not seem to be mirrored by the research into QBF. To our knowledge, the only system that does this for a QBF solver is SAT-TO-SAT [5] grounder SO2GROUNDER using second-order logic (SO logic) as modelling language [4]. As such, we identify the following contributions for our system:

- Extend the traditional rules used in grounding first-order (FO) logic with rules handling the SO quantifications.
- Improving on SAT-TO-SAT by:
  - Lifting syntactical restrictions imposed on the second-order logic model by SO2GROUNDER. SO2GROUNDER translates the alternation of SO quantifiers by nesting SAT solver calls, while FO quantifiers are handled by a regular FO grounder. As a result, SO2GROUNDER must bring all SO quantifiers to the front. However, it will not push an FO quantification past an SO quantification when their quantors differ. For example, with x an FO variable and Y an SO variable, it does not transform $\exists x : \forall Y : \phi$ as this would require changing the signature of Y to accept x as an argument. Likewise, it will not transform $\forall x : \exists Y : \phi$.
  - Interfacing with arbitrary QBF solvers by providing qdimacs [1] output, which is a widely accepted standard.

- Supporting arithmetic.

Section 2 introduces SO logic and shows an example encoding. Section 3, briefly sketches our approach to grounding. In Section 4 we indicate ongoing work, such as an experimental evaluation.

2 Second-Order Logic as a Modelling Language

First, we informally define second-order logic as the well known first-order logic, extended by allowing variables in quantifications to not only represent domain elements, but also let them represent predicates and functions over those domain elements. For convenience while modelling, we also extend SO logic with types.

The choice for second-order logic as a modelling language has two major advantages: It is expressive and intuitive.

To illustrate the intuitiveness of second-order logic as a modelling language, we refer to Bogaerts et al. [4], who discuss other research fields that use SO modellings to describe their problems, such as argumentation theory. Furthermore, we note that within QBF research Ansotegui et al. already use an SO-like notation in their description framework for adversarial games in QBF [3]. They discuss how adversarial games introduce the need for *cheat variables.* These variables complicate writing encodings and their presence lowers the efficiency of many QBF solvers. Our tool could reduce the need for such variables as many cases are covered by the use of implications and functions, and in time will be able to automatically use the techniques that Ansotegui et al. describe to minimize the computational cost of these kind of constructions.

Our final argument regarding the intuitiveness of second-order logic lies in the popularity and success of systems allowing first-order logic, such as the IDP system [7]. As such systems generally look for an interpretation of free symbols within the FO model, modellings correspond with existential second-order logic (SO∃). However, SO∃ is much more limited w.r.t. to expressive power than QBF. As such, second-order logic, which has a descriptive complexity in PH [9], is a prime candidate as the replacement for FO (or, computationally, SO∃).

2.1 Example

To illustrate the style and syntax of a model in SO logic, Listing 1.1 shows how to model *reachability* or *transitive closure* of a graph. This well-known problem is not expressible in FO, and while other constructs exist (e.g. inductive definitions), it is an easy yet educational problem to express in SO.

First, lines 1-3 specify a vocabulary, which contains a single type T containing the possible nodes. To reduce the number of quantifications, vocabularies can also contain symbols that will be *existentially* quantified within the entire model, such as r and g. Line 4 specifies that g actually has a fixed interpretation with 3 connections.

Now, we finally discuss the first expressions in SO logic. Line 6 specifies that for any combination of nodes a and b, r only holds iff a and b are either directly connected or when another node exists that is connected with both. Furthermore, an additional disjunct is added to make r symmetric. Line 7-8 express that no other predicate over T that is a subset of r (not equal and contained, line 7) satisfies the reachability rules (line 8).

For convenience, we have highlighted existential SO quantifications in yellow and universal SO quantifications in orange. This shows that our SO language can model problems with alternating SO quantifiers.

Listing 1.1. A second order modelling of reachability within a graph

```
1  type T = {a;b;c;d;e;f;g;h}          # type of nodes
2  g :: (T,T)                          # graph predicate
3  r :: (T,T)                          # reachability predicate
4  g = {a,b;b,c;a,d}                   # input of graph predicate
5
6  ∀a::T : ∀b::T :r(a,b) ⇔ (g(a,b) ∨ [∃z::T : r(a,z) ∧ r(z,b)] ∨ r(b,a)).
7  ¬(∃subs::(T,T) : subs ≠ r ∧(∀a::T : ∀b::T : subs(a,b) ⇒ r(a,b)) ∧
8    ∀a::T : ∀b::T : subs(a,b) ⇔ [g(a,b) ∨ (∃z::T : subs(a,z) ∧ subs(z,b)] ∨ subs(b,a))).
```

3 Grounding to QBF: A first approach

Our initial approach to grounding SO specifications to QBF proceeds as follows: After pushing negations inward until they only appear in front of atoms, we rewrite quantifications over functions, s.t. uniqueness (yellow) and existence (orange) are enforced: with $\rightarrow$ indicating a transformation, we have $\exists f : \phi \rightarrow \exists f : F(f) \wedge \phi$ and $\forall f : \phi \rightarrow \forall f : F(f) \Rightarrow \phi$ with

$$F(f) \equiv \forall \overline{x} : \exists y : f(\overline{x}) = y \wedge \forall y' : (f(\overline{x}) = y' \Rightarrow y = y')$$

We then hand off the resulting specification to a standard FO grounder [10], modified only to ignore second order quantifications. This eliminates all constructs except for SO $\exists/\forall$, $\wedge$ and $\vee$, producing a CNF.

When the FO grounder is finished, we introduce unique names for every remaining (second order) quantification, e.g.:

$$\forall f : \phi \wedge \exists f : \forall g : f(x) \vee \psi \qquad \rightarrow \qquad \forall f : \phi \wedge \exists f' : \forall g : f'(x) \vee \psi$$

After introducing unique names for every SO variable, we repeatedly use the rules from Fig. 1 to pull SO quantifiers to the front. By switching between applications of rule (1) or rule (2) only when we cannot further apply the active rule, we minimise the number of quantifier alternations. For example $\forall a : (\exists b : \exists c : (\phi \vee \forall d : \psi)) \wedge (\forall e : \phi' \vee \forall f : \exists g : \psi')$ becomes $\forall a, e, f : \exists b, c, g : \forall d : (\phi \vee \psi) \wedge (\phi' \vee \psi')$. Note how $\forall e$ and $\forall f$ are pulled to the level of $\forall a$ by rule (2), whereas $\forall d$ is *blocked* from being pulled to that level by the quantifications $\exists b$ and $\exists c$, as no rule allows switching the order of $\forall$ and $\exists$ quantifications.

$$\begin{aligned}(\exists \alpha : \phi) \wedge \psi &\rightarrow \exists \alpha : (\phi \wedge \psi)\\ (\exists \alpha : \phi) \vee \psi &\rightarrow \exists \alpha : (\phi \vee \psi)\end{aligned} \quad (1) \qquad \begin{aligned}(\forall \alpha : \phi) \wedge \psi &\rightarrow \forall \alpha : (\phi \wedge \psi)\\ (\forall \alpha : \phi) \vee \psi &\rightarrow \forall \alpha : (\phi \vee \psi)\end{aligned} \quad (2)$$

Fig. 1. Rules for pulling quantifications to the front.

Now that all quantifiers are pulled to the front, we introduce a proposition for each possible predicate atom $p(\overline{x})$ and possible function atom $f(\overline{x}) = y$. By replacing atoms with their corresponding propositions in the entire SO model, we effectively produce a qdimacs encoding.

4 Ongoing work

Currently, our grounder does not implement advanced grounding techniques from FO such as *Ground With Bounds* or *Lifted Unit Propagation* [10] as for these techniques, ignoring SO quantifications does not suffice. In addition, we see great value in supporting binary quantifications, which allow the user to restrict quantifications to instantiations for which a certain formula (the generator) would be satisfied. We are currently working towards an implementation of these advanced techniques, which will lead to a reduction in grounding sizes.

We also work towards an evaluation of our grounders current performance, and propose the following experiment: We model the *Strategic Companies* problem, as submitted to QBFLib [2] by Faber et al. (a working draft of which can be found on https://bit.ly/2KnN2Zg) and use their probabilistic method described in [11] to generate problem instances for strategic companies similar to the ones in QBFLib. We will use these problem instances to generate QBF encodings using both our tool, as well as the instantiation scheme used for QBFLib. We will then report running times for our tool, combined with #literals, #clauses and solving time as comparative methods for both resulting QBF encodings.

References

1. Qdimacs standard, http://www.qbflib.org/qdimacs.html, last accessed 27 Apr 2018
2. Stategic companies - qbflib, http://www.qbflib.org/suite_detail.php?suiteId=19, last accessed 30 Apr 2018
3. Ansótegui, C., Gomes, C.P., Selman, B.: The achilles' heel of QBF. In: AAAI. pp. 275–281. AAAI Press / The MIT Press (2005)
4. Bogaerts, B., Janhunen, T., Tasharrofi, S.: Declarative solver development: Case studies. In: KR. pp. 74–83. AAAI Press (2016)
5. Bogaerts, B., Janhunen, T., Tasharrofi, S.: Solving QBF instances with nested SAT solvers. In: AAAI Workshop: Beyond NP. vol. WS-16-05. AAAI Press (2016)
6. Cadoli, M., Schaerf, M., Giovanardi, A., Giovanardi, M.: An algorithm to evaluate qbf and its experimental evaluation. J. Autom. Reasoning **28**(2), 101–142 (2002)
7. de Cat, B., Bogaerts, B., Bruynooghe, M., Janssens, G., Denecker, M.: Predicate logic as a modelling language: The IDP system. CoRR **abs/1401.6312** (2014)
8. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: CAV. Lecture Notes in Computer Science, vol. 1855, pp. 154–169. Springer (2000)
9. Immerman, N.: Descriptive complexity. Graduate texts in computer science, Springer (1999)
10. Jansen, J.: Advanced Techniques for Grounding and Solving in the IDP Knowledge Base System. Ph.D. thesis, Katholieke Universiteit Leuven, Belgium (2016)
11. Maratea, M., Ricca, F., Faber, W., Leone, N.: Look-back techniques and heuristics in DLV: implementation, evaluation, and comparison to QBF solvers. J. Algorithms **63**(1-3), 70–89 (2008)