QSOLE: Automatic QBF Equivalence Checking *

Peter Pfeiffer[1,2][0009−0008−8414−1463], Mark Peyrer[1][0009−0005−5751−9975], Daniel Große[2][0000−0002−1490−6175], and Martina Seidl[1][0000−0002−3267−4494]

[1] Institute for Symbolic Artificial Intelligence, JKU Linz, Austria
[2] Institute for Complex Systems, JKU Linz, Austria
{firstname.lastname}@jku.at

Abstract. Quantified Boolean Formulas (QBFs) extend propositional logic with existential and universal quantifiers, making their decision problem PSPACE-hard. Recent advances in QBF solvers have established QBFs as an attractive framework for encoding PSPACE-hard problems across domains such as formal verification, synthesis, and symbolic AI. Despite progress in solving techniques, less attention has been given to the infrastructure for constructing correct and efficient QBF encodings. For instance, it is often unclear whether two QBFs that encode the same problem in different ways yield the same solutions. Traditional QBF equivalence checking focuses only on free variables, yet in many cases, the quantified variables must also be considered.
In this paper, we present QSOLE, the first fully automatic checker for solution-based QBF equivalence. Based on a recently introduced approach, QSOLE decomposes equivalence checks into smaller entailment computations and is capable of generating witnesses for detected inequivalences, which can be used to debug encodings. Furthermore, it allows for explicit exclusion of variables from equivalence checks enabling comparison of formulas using different local auxiliary variables.

Keywords: QBF · QBF Solutions · Equivalence Checking.

1 Introduction

Quantified Boolean formulas (QBFs) [1,4] extend propositional logic with existential and universal quantifiers, thereby representing PSPACE-complete problems. QBFs play a key role in areas such as formal verification, reactive synthesis and planning [11]. Solutions to QBFs are functions that capture the dependencies between the different types of variables. These functions describe, for example, winning strategies in encodings of two-player games or automatically generated programs in encodings of synthesis problems.

Traditionally, QBF equivalence has been evaluated through satisfiability comparison over free variables [5,6]. Given two QBFs that are defined over the same

* Supported by the LIT AI Lab and the LIT Secure and Correct Systems Lab Lab funded by the state of Upper Austria and by the Austrian Science Fund (FWF) [10.55776/COE12]. A version of this paper without appendices appears in the proceedings of TACAS 2026.

set of free variables, it is checked if for all assignments of the free variables, the two formulas have the same truth value. This is a rather restricted notion of equivalence checking, because the quantified variables are not taken into account. In recent work [8] we introduced solution-based notions of equivalence grounded in Skolem and Herbrand functions, a well-explored way of expressing QBF solutions. With this, we obtain a more expressive notion of equivalence. If two QBFs are solution equivalent, on the one hand, they evaluate to the same truth value, on the other hand they also have the same set of solutions. If, for example, the second formula should be an optimized encoding of the first formula, it can be shown that the optimization did not change the set of solutions. This is valuable for debugging and validating practical encodings. Moreover, two formulas generated through completely different encodings can be compared to ensure correctness of encoding techniques.

The first practical approach to compare individual solutions of two QBFs was presented by Shaik et al. [10]. This approach is an interactive approach, i.e., manual intervention is necessary. To the best of our knowledge, so far there is no tool that offers automatic solution-based QBF equivalence checking.

In this work, we present QSOLE, an automatic checker for solution-based QBF equivalence notions as defined in [8]. Our tool QSOLE is implemented in C++, uses the QBF solver DepQBF [7] as a reasoning backend and allows for witness extraction of inequivalences using the SAT solver CaDiCaL [2]. We propose a flexible QBF encoding that supports the exclusion of auxiliary variables from equivalence checks and introduce the first equivalence-specific solving techniques, including substitution-based encoding optimization for formulas with shared clauses. To the best of our knowledge, QSOLE is the first practical framework for automated reasoning about semantic QBF equivalence. Our tool QSOLE is openly available at `https://doi.org/10.5281/zenodo.17356608`.

2 Preliminaries

A *Boolean formula* is built from a set of Boolean variables V and the logical connectives $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$. The set of variables occurring in a formula φ is denoted by $var(\varphi)$. An *assignment* $\sigma : V' \rightarrow \{\mathbf{1}, \mathbf{0}\}$ maps variables $V' \subseteq V$ to $\mathbf{1}$ and $\mathbf{0}$. By $[\varphi]_\sigma$ we denote the formula obtained from φ by setting all variables $v \in V'$ in to $\sigma(v)$ and simplifying the formula according to standard semantics. An assignment σ is a model of φ iff $[\varphi]_\sigma = \mathbf{1}$. If $[\varphi]_\sigma = \mathbf{0}$, it is a *counter-model*. A Boolean formula is in *Conjunctive Normal Form (CNF)* if it is a conjunction of disjunctions (clauses), where negation applies only to Boolean variables.

Quantified Boolean Formulas (QBFs) [1,4] extend Boolean formulas with *quantifiers* $\mathcal{Q} = \{\forall, \exists\}$ that bind variables in subformulas, which may themselves contain quantifiers. A QBF $\forall v : \Phi$ is true iff both $([\Phi]_{\{v=\mathbf{1}\}})$ and $([\Phi]_{\{v=\mathbf{0}\}})$ are true. Dually, a QBF $\exists v : \Phi$ is true iff $([\Phi]_{\{v=\mathbf{1}\}})$ or $([\Phi]_{\{v=\mathbf{0}\}})$ is true. Quantification over multiple variables $V = v_1, \ldots, v_n$ is abbreviated as $\mathcal{Q}V = \mathcal{Q}v_1 \ldots \mathcal{Q}v_n$. A sequence $P = \mathcal{Q}_1 v_1 \ldots \mathcal{Q}_n v_n$ of quantifiers is called a *prefix*, and $\overline{P}$ denotes its dual where all quantifiers are flipped. A QBF is in *Prenex Conjunctive Normal*

Form (PCNF) if it has the structure $P : \varphi$, where P is a prefix and φ (the *matrix*) is in CNF. Variables bound in P are denoted by $var(P)$. *Free variables* $free(\Phi) = var(\varphi) \setminus var(P)$ are not bound by a quantifier in a QBF $\Phi = P : \varphi$. A QBF without free variables is *closed*.

Given a QBF $\Phi = \mathcal{Q}_1 v_1 \dots \mathcal{Q}_n v_n : \varphi$, consider an assignment tree whose internal nodes correspond to variables v_i, where edges assign truth values $\{\mathbf{1}, \mathbf{0}\}$, and each path from the root to a leaf represents an assignment σ with leaf label $[\varphi]_\sigma$. A model of a true QBF is a subtree of the assignment tree such that existential nodes have one child, universal nodes have two, and all leaves are $\mathbf{1}$. Dually, a counter-model of a false QBF requires one child for universal nodes, two for existential nodes, and all leaves to be $\mathbf{0}$. We denote the sets of models and counter-models of Φ as $\mathbb{S}_\exists(\Phi)$ and $\mathbb{S}_\forall(\Phi)$ respectively. Classically, two QBFs Φ and Ψ are said to be (satisfiability-)equivalent if they evaluate to the same truth value under all (resp. some) assignment(s) to their common free variables [5,6]. Solution-based notions of entailment and equivalence have been introduced in [8] and are defined as follows: Given two true QBFs with the same prefix P. Then Φ Skolem-entails Ψ, denoted by $\Phi \models_{\mathsf{Sk}} \Psi$ iff $\mathbb{S}_\exists(\Phi) \subseteq \mathbb{S}_\exists(\Psi)$. Skolem equivalence $\Phi \Leftrightarrow_{\mathsf{Sk}} \Psi$ holds if $\mathbb{S}_\exists(\Phi) = \mathbb{S}_\exists(\Psi)$. In a dual manner, Herbrand entailment and Herbrand equivalence is defined for false formulas. Solution equivalence $\Phi \Leftrightarrow_{\mathsf{Sol}} \Psi$ holds iff Φ and Ψ are Skolem equivalent or Herbrand equivalent. Note that Herbrand equivalence checking can be reduced to Skolem equivalence checking by negating the formulas.

3 Solution-Based QBF Equivalence Checking

We introduce QSOLE, an automatic checker for solution-based notions of QBF equivalence. It works for true and for false formulas in a similar manner. Our tool takes two input formulas in QDIMACS format and can determine whether they are Skolem entailed, Skolem equivalent, or solution equivalent. Additionally, it requires a mandatory numeric parameter `PCOUNT` to denote the length of the quantifier prefix used for comparison. To compute equivalence wrt. the full prefix this value has to be set to $|var(P)|$. In the next section, we explain what happens if not the full quantifier prefix is considered.

Figure 1 illustrates how QSOLE checks whether two QBFs Φ and Ψ are solution equivalent. For this, it first evaluates the truth value of Φ (purple). If Φ evaluates to true, i.e., $\mathbb{S}_\forall(\Phi) = \emptyset$, Skolem entailment needs to checked in both directions (blue part of Figure 1). If $\Phi \models_{\mathsf{Sk}} \Psi$ holds, it follows that Ψ is true as well. Analogously, if Φ is false (yellow), then QSOLE instead checks for Skolem equivalence of the negated formulas $\neg\Phi \models_{\mathsf{Sk}} \neg\Psi$ and $\neg\Psi \models_{\mathsf{Sk}} \neg\Phi$ (red). If any Skolem entailment check fails during this process, $\Phi \Leftrightarrow_{\mathsf{Sol}} \Psi$ does not hold. Then a witness can be generated that is part of the solution of one formula but not of the other.

QSOLE computes Skolem entailment $\Phi \models_{\mathsf{Sk}} \Psi$ checks based on the QBF encoding presented in [8]. The Plaisted-Greenbaum transformation [9] is applied to obtain a PCNF representation of this formula, which is subsequently solved

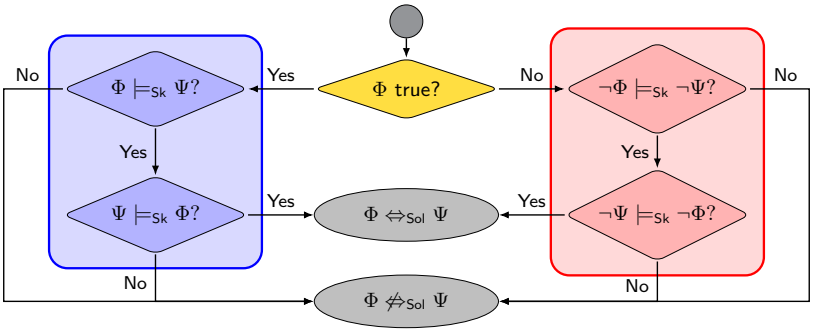

Fig. 1: Solution equivalence $\Phi \Leftrightarrow_{\mathsf{Sol}} \Psi$ check in QSOLE.

using the QBF solver DepQBF. A counter-model to $\Phi \models_{\mathsf{Sk}} \Psi$ represents a model of Φ, which does not satisfy Ψ, indicated by a propositional countermodel for ψ, which we refer to as a *witness*. If the `--models` flag is set, QSOLE generates witnesses for failed Skolem entailment checks. They are constructed from the terminating assignment of DepQBF using the SAT solver CaDiCaL.

We demonstrate this solving process on an example: Consider the two QBFs

$$\Phi = P : \varphi = \forall a \exists b \exists c : (a \vee c) \wedge (\neg a \vee \neg c) \wedge (b \vee c) \wedge (\neg b \vee \neg c)$$
$$\Psi = P : \psi = \forall a \exists b \exists c : (a \vee \neg b) \wedge (\neg a \vee b) \wedge (a \vee c)$$

Suppose it is claimed that Ψ encodes the same problem as Φ, i.e., that $\Phi \Leftrightarrow_{\mathsf{Sol}} \Psi$ holds. Figure 2 shows the input formulas Φ (left) and Ψ (center) in QDIMACS format. The quantified variables a, b, and c correspond to QDIMACS variables `1`, `2`, and `3`, respectively. The assignment tree (right) represents the propositional models of Φ and Ψ simultaneously. The leaves are labeled by φ/ψ where φ (resp., ψ) represents the truth value of Φ (resp., Ψ) under the assignment on the path to the root of the tree. The highlighted (yellow) subgraph marks a model of Φ, which is not a model for Ψ as indicated by a propositional counter-model for ψ (red). This refutes the claim that $\Phi \Leftrightarrow_{\mathsf{Sol}} \Psi$.

Listing 1 illustrates how QSOLE can be used to automatically show that the two formulas do not have the same solutions. In a first call, it checks solution equivalence (parameter `--check psole`). The prefix length is set to 3 to compare over the full quantifier prefix $P = \forall a \exists b \exists c$. QSOLE first calculates the value of Φ, which evaluates to true. Given this information, it proceeds with a Skolem equivalence check for $\Phi \models_{\mathsf{Sk}} \Psi$. Since this check also succeeds, QSOLE has to check $\Psi \models_{\mathsf{Sk}} \Phi$. However, this check fails, ultimately refuting the solution equivalence. To generate a witness, we invoke QSOLE again with the option `--models`. We only rerun the check $\Psi \models_{\mathsf{Sk}} \Phi$ (parameter `--check skent`) which previously returned false. QSOLE reports the witness `"v 1 2 3 0"` assigning all variables to **1**. This corresponds directly to the counter-model (red) for ψ in the binary tree of Figure 2.

```
p cnf 3 4
a 1 0
e 2 3 0
1 3 0
-1 -3 0
2 3 0
-2 -3 0
```

```
p cnf 3 3
a 1 0
e 2 3 0
1 -2 0
-1 2 0
1 3 0
```

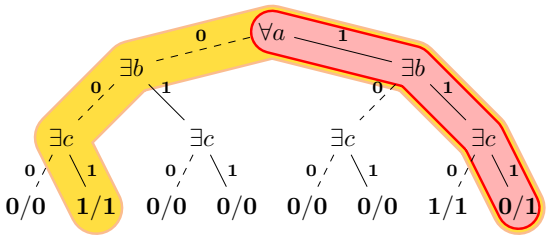

Fig. 2: Φ (left) and Ψ (center) in QDIMACS format next to assignment tree (right) on variables $V = \{a, b, c\}$ with leaf labels φ/ψ and model of Ψ (yellow) containing counter-model for φ (red).

4 Auxiliary Variables

Some PCNF encodings, such as efficient normal form transformation [12,9] of formulas, require that auxiliary variables are added to the formula. For example, consider the QBF $\Phi = \forall a \exists b \exists c : (a \vee c) \wedge (\neg a \vee \neg c) \wedge (b \vee c) \wedge (\neg b \vee \neg c)$ from before. The variable c indirectly enforces $(a \leftrightarrow b)$ through $(a \not\leftrightarrow c) \wedge (b \not\leftrightarrow c)$. Suppose we wanted to disregard the semantics of c and only focus on the relation between a and b enforced through it. Figure 3 illustrates this idea: The full assignment tree for Ψ (left) is pruned after b (right), whereas leaves correspond to the evaluation of the subformula that was eliminated. The outer leaves of the pruned tree represent $(a \leftrightarrow b)$ and are, therefore, labeled with **1**, whereas the inner leaves denote $a \not\leftrightarrow b$ and are labeled with **0**. The same relation between a and b is also present in $\Psi = \forall a \exists b \exists c : (a \vee \neg b) \wedge (\neg a \vee b) \wedge (a \vee c)$. To verify this, we can separate the outer prefix $P = \forall a \exists b$ from the auxiliary variable c, which is treated as local to each formula (with $P_1 = \exists c$ and $P_2 = \exists c'$), to obtain $\Phi = P : P_1 : \varphi$ and $\Psi = P : P_2 : \psi$ and compare them based on P. QSOLE allows restriction to a common outer prefix with the parameter `PCOUNT`. In this case, we can set

```
> qsole phi.qdimacs psi.qdimacs 3 --check psole --info
[QSOLE] [INF] Polarity check of f1 resulted in 10
[QSOLE] [INF] Skolem Entailment f1 |=SK f2 resulted in 10
[QSOLE] [INF] Skolem Entailment f2 |=SK f1 resulted in 20
[QSOLE] [INF] Polarity Solution Equivalence (f1 <->PSOL f2)
    resulted in 20
> qsole psi.qdimacs phi.qdimacs 3 --check skent --models
c f1 |=SK f2
v 1 2 3 0
```

Listing 1: QSOLE check for $\Phi \Leftrightarrow_{\mathsf{Sol}} \Psi$ with `PCOUNT` $= 3$ and witness generation

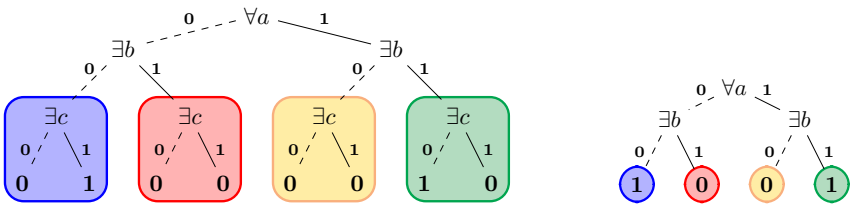

Fig. 3: Left: Assignment tree of Ψ. Right: Pruned assignment tree with $P = \forall a \exists b$. Subformulas with prefix $\exists c$ are replaced with leaves indicating their truth value.

`PCOUNT` $= 2$ resulting in Φ and Ψ being solution equivalent wrt. $P = \forall a \exists b$. We confirm this with QSOLE in Listing 2.

```
> qsole phi.qdimacs psi.qdimacs 2 --check psole --info
[QSOLE] [INF] Polarity check of f1 resulted in 10
[QSOLE] [INF] Skolem Entailment f1 |=SK f2 resulted in 10
[QSOLE] [INF] Skolem Entailment f2 |=SK f1 resulted in 10
[QSOLE] [INF] Polarity Solution Equivalence (f1 <->PSOL f2)
    resulted in 10
```

Listing 2: QSOLE check for $\Phi \Leftrightarrow_{\mathsf{Sol}} \Psi$ with `PCOUNT` $= 2$

The encoding used by QSOLE to compute whether $\Phi \models_{\mathsf{Sk}} \Psi$ holds for some QBFs $\Phi = P : \varphi$ and $\Psi = P : \psi$ requires the negation of ψ. This introduces at least one additional auxiliary variable per clause, leading to a large overhead. To mitigate this, QSOLE can perform *subsumption* of clauses in ψ with clauses of φ before applying negation in the encoding. Since the Skolem entailment check looks for a model of Φ that includes a counter-model of ψ, all clauses in φ must be satisfied. Therefore, any clause in ψ that is subsumed by a clause in φ cannot refute ψ. Thus, any such clause can be removed from ψ for the check. This can considerably reduce the number of auxiliary variables, particularly when φ and ψ share many clauses. This optimization is available in QSOLE via the flag `--subsumption`.

5 Evaluation

We tested QSOLE on two sets of synthetic benchmarks derived from pseudo-Boolean constraints: For "equals-k" constraints exactly k out of n variables must be true. We used PySAT [3] to generate two different standard encodings, namely `seqcounter` and `kmtotalizer`, and added a randomly generated quantifier prefix containing u universal quantifiers over the common n variables. Auxiliary variables are existentially quantified. This construction yields pairs of formulas that are guaranteed to be solution equivalent wrt. their n common variables. Analogously, we used "at-least-k" constraints (at least k of n variables must be

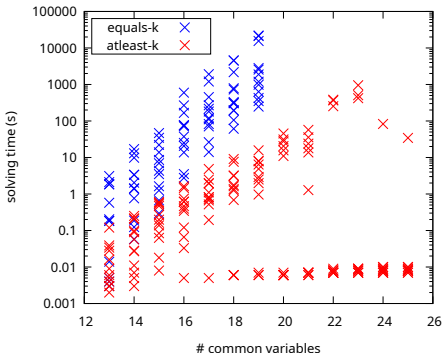

Fig. 4: QSOLE Solution Equivalence checking on synthetic benchmarks equals-k (blue) and atleast-k (red); x-axis: number of common variables; y-axis: time in seconds.

true) to generate pairs of formulas that are not solution equivalent. Any model for a larger bound k is also a model for a smaller bound, but not vice versa. This holds inversely for counter-models in false formulas. This setup can be scaled along several dimensions: total number of variables, the bounds k, the encoding type, and both the ratio and ordering of universal vs. existential variables in the prefix. However, the lift to QBF is conducted with random prefixes and these benchmarks may, therefore, not be reliable performance indicators for concrete QBF applications.

We computed solution equivalence for one test case per combination of $k \in \{8, 10, 12\}$ and $u \in \{8, 10, 12, 14\}$ with $n \in \{13, \ldots, 19\}$ for equals-k and $n \in \{13, \ldots, 25\}$ for atleast-k. Quantifier prefixes were generated independently for each test case. Our experiments were run on an AMD Ryzen Threadripper PRO 5955WX with 16-Cores and 248Gi RAM. Figure 4 shows the result grouped by the number of common variables. The performance gap within atleast-k benchmarks can be attributed to early returns in cases where the first Skolem entailment check yields a refutation.

6 Conclusion and Future Work

In this paper, we introduced QSOLE, which automatically checks if two QBFs have the same set of solutions. The approach is implemented for true and for false formulas based on the encoding presented in [8]. We extended the encoding to handle local auxiliary variables that are at the end of the quantifier prefix. In future work, we plan to lift this restriction and allow for local variables at arbitrary positions in the quantifier prefix. Furthermore, we plan to improve on the user interface to facilitate the development and the debugging of QBF encodings.

Data Availability Statement The QSOLE artifact containing the tool implementation, benchmarks, and scripts required to reproduce the experiments presented in this paper is openly available at `https://doi.org/10.5281/zenodo.18590551`.

References

1. Beyersdorff, O., Janota, M., Lonsing, F., Seidl, M.: Quantified boolean formulas. In: Handbook of Satisfiability - Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336, pp. 1177–1221. IOS Press (2021)
2. Biere, A., Faller, T., Fazekas, K., Fleury, M., Froleyks, N., Pollitt, F.: Cadical 2.0. In: Gurfinkel, A., Ganesh, V. (eds.) 36th International Conference on Computer Aided Verification (CAV). pp. 133–152. Springer Nature Switzerland, Cham (2024)
3. Ignatiev, A., Tan, Z.L., Karamanos, C.: Towards universally accessible SAT technology. In: 27th International Conference on Theory and Applications of Satisfiability Testing (SAT). pp. 4:1–4:11 (2024). `https://doi.org/10.4230/LIPICS.SAT.2024.16`, `https://doi.org/10.4230/LIPIcs.SAT.2024.16`
4. Kleine Büning, H., Bubeck, U.: Theory of quantified boolean formulas. In: Handbook of Satisfiability - Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336, pp. 1131–1156. IOS Press (2021)
5. Kleine Büning, H., Lettmann, T.: Aussagenlogik – Deduktion und Algorithmen. Teubner (1994)
6. Kleine Büning, H., Zhao, X.: Equivalence models for quantified boolean formulas. In: 7th International Conference on Theory and Applications of Satisfiability Testing (SAT). Lecture Notes in Computer Science, vol. 3542, pp. 224–234. Springer (2004)
7. Lonsing, F., Egly, U.: Depqbf 6.0: A search-based qbf solver beyond traditional qcdcl. In: de Moura, L. (ed.) 26th International Conference on Automated Deduction (CADE). pp. 371–384. Springer International Publishing, Cham (2017)
8. Pfeiffer, P., Große, D., Seidl, M.: Refined notions of qbf equivalences. In: 19th European Conference on Logics in Artificial Intelligence (JELIA). p. 159–165. Springer-Verlag, Berlin, Heidelberg (2025). `https://doi.org/10.1007/978-3-032-04590-4_11`
9. Plaisted, D.A., Greenbaum, S.: A structure-preserving clause form translation. Journal of Symbolic Computation **2**(3), 293–304 (1986). `https://doi.org/https://doi.org/10.1016/S0747-7171(86)80028-1`
10. Shaik, I., Heisinger, M., Seidl, M., van de Pol, J.: Validation of QBF Encodings with Winning Strategies. In: 26th International Conference on Theory and Applications of Satisfiability Testing (SAT). Leibniz International Proceedings in Informatics (LIPIcs), vol. 271, pp. 24:1–24:10. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2023)
11. Shukla, A., Biere, A., Pulina, L., Seidl, M.: A Survey on Applications of Quantified Boolean Formulas. In: 31st International Conference on Tools with Artificial Intelligence (ICTAI). pp. 78–84. IEEE (2019)
12. Tseitin, G.S.: On the Complexity of Derivation in Propositional Calculus, pp. 466–483. Springer Berlin Heidelberg, Berlin, Heidelberg (1983)

A Skolem Entailment Encoding

To compute whether Skolem entailment $\Phi \models_{\mathsf{Sk}} \Psi$ holds for two QBFs $\Phi = P : \varphi$ and $\Psi = P : \psi$, QSOLE uses the formula from [8], which evaluates to true iff $\Phi \not\models_{\mathsf{Sk}} \Psi$, as a base:

$$\Delta(\Phi, \Psi) = \exists X' : P : \left(\varphi \wedge \left(\left(\bigwedge_{x'_i \in X'} x_i \leftrightarrow x'_i \right) \rightarrow \neg\psi \right) \right)$$

where X' represents a set of fresh variables corresponding to all universally quantified variables X in the common prefix P. QSOLE applies the Plaisted-Greenbaum [9] transformation to introduce fresh sets of variables Y and Z. The negation of each Y variable is implied by $x \leftrightarrow x'$, whereas the Z are used to negate ψ: Each $z \in Z$ implies the negation of a clause of ψ. Finally, a variable $z_{\overline{\psi}}$ implies that one $z \in Z$ is true resulting in the negation of ψ. We denote this with the propositional subformulas

$$\zeta = \bigwedge_{x'_i \in X'} (x_i \leftrightarrow x'_i) \rightarrow \neg y_i \quad \text{and} \quad \xi = \left(z_{\overline{\psi}} \rightarrow \bigvee_{z \in Z} z \right) \wedge \bigwedge_{c \in \text{clauses}(\psi)} z \rightarrow \neg c.$$

These formulas are conjoined with an additional clause over the Y variables and $z_{\overline{\psi}}$. This clause enforces that either at least one $y_i \in Y$ is assigned $\mathbf{1}$, which requires $x_i \not\leftrightarrow x'_i$, or that $\neg\psi$ holds. QSOLE constructs the PCNF formula

$$\Delta(\Phi, \Psi) = \exists X' : P : \exists Y : \exists Z : \left(\varphi \wedge \zeta \wedge \xi \wedge \left(z_{\overline{\psi}} \vee \bigvee_{y \in Y} y \right) \right)$$

and invokes DepQBF to solve it. If the formula evaluates to false, $\Phi \models_{\mathsf{Sk}} \Psi$ holds. Otherwise, QSOLE can produce a propositional model for $\neg\psi$, which is a part of a model of Φ. Specifically, it substitutes the X variables with the corresponding assignments to X' returned by DepQBF (denoted by $\sigma_{X \leftarrow X'}$), and then uses CaDiCaL to compute a propositional model for

$$\exists \{ var(P) \setminus X \} : [\phi]_{\sigma_{X \leftarrow X'}} \wedge \neg [\psi]_{\sigma_{X \leftarrow X'}}.$$

The result is a witness to refute Skolem entailment.

To check Skolem entailment wrt. an outer prefix P for formulas $\Phi = P : P_1 : \varphi$ and $\Psi = P : P_2 : \psi$, QSOLE uses the amended formula

$$\Delta(\Phi, \Psi) = \exists X' : P : \left((P_1 : \varphi) \wedge \left(\left(\bigwedge_{x'_i \in X} x_i \leftrightarrow x'_i \right) \rightarrow \neg P_2 : \psi \right) \right)$$

resulting in the PCNF encoding

$$\Delta(\Phi, \Psi) = \exists X' : P : P_1 : \overline{P_2} : \exists Y : \exists Z : \left(\varphi \wedge \zeta \wedge \xi \wedge \left(z_{\overline{\psi}} \vee \bigvee_{y \in Y} y \right) \right).$$