Master Thesis

Certified Circuit Reconstruction for QBF

March 23, 2024

Contents

1 Introduction

1.1 Aim of the thesis

1.2 Related work

1.3 Work structure

2 Preliminary

In this preliminary chapter, we are going to set the building blocks needed for later chapters without the need of prior knowledge. In all sections of this chapter, every definition will be accompanied by its respective example, and a corresponding illustration for each procedure.
A notion can have all sorts of definition, thus we shall include the citation for every introduced concept.
The structure of this chapter will be as follows: we start by introducing syntax and semantics for our formulas of interest, followed by a transformation of a generic formula in a conjunctive normal form. In plus, we present two proof systems for QBFs and show how an algorithm can solve a QBF. Lastly, we talk about types of format a QBF should be given in to be understood by a solver.

2.1 QBF

The quantified Boolean formula is an extension of the propositional Boolean formula with quantified variables. For example, $(x_1 \vee x_2 \wedge x_3) \rightarrow x_4$ is a propositional formula whilst $\forall x_1 x_3 \exists x_2 x_4 (x_1 \vee x_2 \wedge x_3) \rightarrow x_4$ is the prior formula quantified.
Worth mentioning is that, every propositional formula can be represented as a QBF, by existentially quantified the variables, from $x_1 \vee x_2$ to $\exists x_1 x_2 (x_1 \vee x_2)$.
In this section, the definition for QBF will be given in the prenex form, where all the quantification appear before a quantifier free formula named matrix. The matrix, as well, can be given in a special form, this will be conjunctive normal form. Any generic QBF can be transformed in the prenex conjunctive normal form. In this chapter, we will present only the transformation of the matrix in CNF in section 2.2 using Tseitin transformation.

Definition 2.1 (Literal [1])**.** A **literal** is a Boolean variable x or its negation $\overline{x}$. Let the function $\mathbf{var}(l) = x$.

Example 2.2. In formula $(x \vee \overline{y} \wedge z)$ the literals are $\{x, \overline{y}, z\}$. $\text{var}(\overline{y}) = y$. If we want the negation of the literal $l = \overline{y}$, we will have $\bar{l} = y$. Also, $x \vee \overline{y}$ is not a literal because it is the conjunction of two literals.

Definition 2.3 (Clause [1])**.** A **clause** is a disjunction of literals.

Example 2.4. $(x_1 \vee \overline{x_2} \vee x_3)$ is a clause. But $(y_1 \vee \overline{y_2} \wedge y_3)$ is not because it contains an and operator, neither $(z_1 \vee (\overline{z_2} \wedge z_3))$ because the second term is not a literal.

A clause can also be represented as a set, $(x_1 \vee \overline{x_2} \vee x_3)$ to $\{x_1, \overline{x_2}, x_3\}$. This representation is useful for computer programs due to its processing as a list.

Definition 2.5 (Cube [1])**.** A **cube** is a conjunction of literals.

Example 2.6. $(x_1 \wedge \overline{x_2} \wedge \overline{x_3})$ is a cube. Whilst, $(y_1 \wedge (\overline{y_2} \vee \overline{y_3})), (x_1 \rightarrow \overline{x_2} \wedge \overline{z_3})$ are not.

Definition 2.7 (CNF [1])**.** A propositional formula is in **conjunctive normal form** if it is a conjunction of clauses.

Example 2.8. $x_1 \wedge (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_2 \vee x_3)$ is in conjunctive normal form.

Definition 2.9 (QBF in PCNF [1])**.** A **quantified Boolean formula** in **prenex conjunctive normal form** is of form:

$$\Pi\psi$$

, which consists of a CNF ψ called **matrix**, and a **prefix** $\Pi = Q_1 X_1 \dots Q_k X_k$, with $Q_i \in \{\exists, \forall\}$, $Q_i \neq Q_{i+1}$, and X_i pairwise disjoint sets of variables.

Example 2.10. $\forall x \exists y (x \vee y)$ is a QBF in PCNF. A QBF that is not a PCNF can be $\forall x(x) \rightarrow \exists y(y)$. Also, it is not allowed to have two consecutive quantifiers of the same time $\forall x \forall y$, instead $\forall xy$ should be used.

Definition 2.11 (Qunatifier Block [1])**.** A **quantifier block** is $Q_i X_i$ (from definition 2.9). In plus, $Q_1 X_1$ is the **outermost quantifier block** and $Q_k X_k$ is the **innermost quantifier block**.
A variable x is quantified at **level** i, if $x \in X_i$ and denoted by $\mathrm{lv}(x) = i$. We can extend it for literals with $\mathrm{lv}(l) = \mathrm{lv}(\mathrm{var}(x))$. Furthermore, we can define $\mathbf{quant}(\Pi, l) = Q_i$.

Example 2.12. Let's have the prefix $\Pi = \exists ab \forall uv \exists xyz$, then the outermost block is $\exists ab$, the innermost block is $\exists xyz$, $\mathrm{lv}(u) = 2$, and $\mathrm{quant}(\Pi, v) = \forall$.

Definition 2.13 (Substitution [1])**.** $\Pi\psi[t/x]$ denotes the replacement of x by t.

Example 2.14. $\forall xy \exists z (x \vee y) \wedge z[1/z]$ we get $\forall xy \exists z (x \vee y) \wedge 1$.

Definition 2.15 (QBF Semantics [1])**.** A QBF $\forall x \Pi\psi$ is true iff $\Pi\psi[0/x]$ and $\Pi\psi[1/x]$ are true. A QBF $\exists x \Pi\psi$ is true iff $\Pi\psi[0/x]$ or $\Pi\psi[1/x]$ is true.

Example 2.16. $\forall x (x \vee \overline{x})$ is a true QBF, we have $(0 \vee \overline{0})$ which evaluates to true and $(1 \vee \overline{1})$ which also evaluates to true. Another true QBF is $\exists x \forall y (x \vee y)$, because it will be true, when we assign x to 1. A false QBF can be $\forall x(x)$, because we can take $[0/x]$ producing a false, and the other case with $[1/x]$ doesn't matter because $\forall$ quantifier requires both cases to be true.

Definition 2.17 (Proof System [1]). CE E ASTA? rng(f) Didn't like thte definiton in citaiton need to find another and exemplify with a simple axiom and inference rule.

sat, qsat, equisat, free variable, level sign of prefix, assignment, unit propagation, RAT for propositional

2.2 Tseitin transformation

Tseitin transformation is a procedure that is taking a propositional formula and compute a new formula in conjunctive normal form that is equisatisfiable to the initial formula. As stated in [2], for each logical connective we introduce a new variable that is equivalent with its formula, and replace in the subformulas. These equivalences $T \leftrightarrow A \circ B$, where $\circ$ is a logical connective, can be transformed in a logical equivalent CNF, in table 1 we display a portion of those CNF.

should write its also linear in the size of the input.

Logical connective	Conjunctive normal form
$T \leftrightarrow A \wedge B$	$(T \vee \overline{A} \vee \overline{B}) \wedge (\overline{T} \vee A) \wedge (\overline{T} \vee B)$
$T \leftrightarrow A \vee B$	$(\overline{T} \vee A \vee B) \wedge (T \vee \overline{A}) \wedge (T \vee \overline{B})$

Table 1: Tseitin transformation of a logical connective into its conjunctive normal form.

This transformation works as follows: given a propositional formula ϕ, for each subformula we introduce a Tseitin variable and add this cnf to ϕ_T, after we introduce all the subformulas, ϕ is equisatisfiable with $\phi_T \wedge T_0$, where T_0 corresponds to the first logical connective of ϕ.
We illustrate this transformation in the next example:

Example 2.18. Let's have the formula $(A \wedge B) \vee C$, first we should point that, this is not in CNF. We introduce a Tseitin variable for each subformula $T_1 \leftrightarrow (A \wedge B)$ and $T_0 \leftrightarrow (T_1 \vee C)$. We write the conjunctive normal form for each equivalence $\varphi_T = (T_1 \vee \overline{A} \vee \overline{B}) \wedge (\overline{T_1} \vee A) \wedge (\overline{T_1} \vee B)(\overline{T_0} \vee T_1 \vee C) \wedge (T_0 \vee \overline{T_1}) \wedge (T_0 \vee \overline{C})$. Finally, $\varphi_T \wedge T_0$ is the transformation of the initial formula in CNF.

A sketch of why it is working, and why we need to add the last variable to the formula is that each of those introduce Tseitin variable just takes the value of their equivalence, and the need for the conjunction with T_0 is that we want our end formula to be true, without it, we do not take into account the output of the formula, and just make a notation for all the subformulas evaluations. And a formal proof of this is that, given φ and $\varphi_T \wedge T_0$, if φ is satisfiable, then using the assignment we can set our T_i accordingly, and T_0 is true because φ is true to be satisfiable, in case φ is unsatisfiable, suppose $\varphi_T \wedge T_0$ is satisfiable, but this cannot be the case, because we can get an assignment that should be true for φ thus $\varphi_T \wedge T_0$ is also unsatisfiable.

With the transformation for propositional formula in a CNF in place, in a QBF PCNF setting we are raising the question of where we put the Tseitin variables in the prefix. If we do not assign them in and quantified block, they will be treated as free variables, thus in the outermost existential block (if the first block is universal, we just add another existential block outside). But this has the following counter-example $\forall x(x \wedge \overline{x})$ that is true to $\exists t \forall x(t \vee \overline{x} \vee x) \wedge (\overline{t} \vee x) \wedge (\overline{t} \vee \overline{x}) \wedge t$ which is false. Thus, we can form the following claim:

Claim 2.19. *Given a QBF of form $\Pi\varphi$, where φ is quantifier free but is not in conjunctive normal form. $\Pi\varphi$ is equisatisfiable with $\Pi\exists T(\varphi_T \wedge T_0)$, where T is the set of Tseitin variables, φ_T is the Tseitin transformation of subformulas and T_0 is the first logical connective that is applied to the formula. (If last quantified block of Π is $\exists$, then T is appended to it.)*

Proof. Similar with the propositional case, by applying the semantic definition 2.15 of QBF, we end up with a matrix that is formed prefixed only by existential block of Tseitin variables. Due to the prefix being only existential we can use the reasoning at propositional level, and we can use the prior sketch proof. □

2.3 Q-Resolution Proof System

In SAT solving, if a proposition is satisfiable, we can give a satisfying assignment, if it is unsatisfiable we will need to check every possible solution to be sure that the formula is not satisfiable, this is where a proof system comes in our help. For an unsatisfiable formula, we can give the steps in a proof system do derive a contradiction. One proof system for proposition formulas is the resolution proof system.
In QBF solving, we can elaborate a quantified version for the prior resolution proof system. This is presented in figure 1. Q-resolution proof system is refutational-complete for QBFs in PCNF, [1], this means if a formula is unsatisfiable, we can derive the empty clause, $\bot$, by applying rules from our proof system given the formula.

what about satisfiable formulas

$$\frac{C \cup \{l\}}{C} \quad \text{for all } x \in \mathsf{vars}(\Pi)\colon \{x, \bar{x}\} \not\subseteq (C \cup \{l\}),\ \mathsf{quant}(\Pi, l) = \forall,\ \text{and } l' \leq_\Pi l \text{ for all } l' \in C \text{ with } \mathsf{quant}(\Pi, l') = \exists \qquad (red)$$

$$\frac{C_1 \cup \{p\} \qquad C_2 \cup \{\bar{p}\}}{C_1 \cup C_2} \quad \text{for all } x \in \mathsf{vars}(\Pi)\colon \{x, \bar{x}\} \not\subseteq (C_1 \cup C_2),\ \bar{p} \notin C_1,\ p \notin C_2, \text{ and } \mathsf{quant}(\Pi, p) = \exists \qquad (res)$$

$$\frac{}{C} \quad \text{for all } x \in \mathsf{vars}(\Pi)\colon \{x, \bar{x}\} \not\subseteq C \text{ and } C \in \psi \qquad (cl\text{-}init)$$

Figure 1: Q-resolution proof system from [1].

Example 2.20. $\exists y \forall x z \exists p (y \vee x \vee z) \wedge p$ by applying only (red) for $(y \vee x \vee z)$ we get $(y \vee x)$ because $l \leq_\Pi x$ and p is not in the clause, also we can get $(y \vee z)$ because z needs to be higher than existential variables and ignore the universal quantified variables, in plus, if we apply it repeatedly we can also get (y).

why tautology so probebatic

how to check sound of a red, with semantics?

Example 2.21. $\forall x y \exists z (x \vee z) \wedge (y \vee \overline{z})$ with (res) on those 2 clauses, we get $(x \vee y)$.

Example 2.22. In this example we will apply the rules to get a refutation for $\forall x_1 x_2 \exists y (x_1 \vee x_2 \vee \overline{y}) \wedge (y \vee \overline{x_1}) \wedge (y \vee \overline{x_y}) \wedge y \wedge (x_1 \vee \overline{x_1})$. Firstly we apply (cl-init) where we get $(x_1 \vee x_2 \vee \overline{y}) \wedge (y \vee \overline{x_1}) \wedge (y \vee \overline{x_y}) \wedge y$ without last clause because that is a tautology. On the first and last clause we can apply (res) and get $(x_1 \vee x_2)$. On $(x_1 \vee x_2)$ we apply twice (red) and get the empty clause, $\bot$. Thus, our formula is false and have a proof in Q-resolution proof system.

2.4 QRAT Proof System

The main objective of this work, is to provide a QRAT proof for an input PCNF, where QRAT is derived from the quantified circuit of the PCNF. Thus, we dedicate this section to provide the prerequisite for a QRAT proof.
The QRAT proof system

...

Definition 2.23 (Outer Resolvent [1])**.** The **outer resolvent** of clauses $C \vee l$ and $D \vee \bar{l}$ on literal l w.r.t. qunatifiers Π is:

$$\mathrm{OR}(\Pi, C \vee l, D \vee \bar{l}, l) = C \cup \{k \mid k \in D, k \leq_\Pi l, k \neq \bar{l}\}, \text{for } \mathrm{quant}(\Pi, l) = \exists,$$

and:

$$\mathrm{OR}(\Pi, C \vee l, D \vee \bar{l}, l) = (C \backslash \{l\}) \cup \{k \mid k \in D, k \leq_\Pi l, k \neq \bar{l}\}, \text{for } \mathrm{quant}(\Pi, l) = \forall.$$

Example 2.24. Given $\Pi = \forall x_1 x_2 \exists y_1 y_2 \forall x_3 x_4$.
For $C = (x_2 \vee x_4 \vee y_1), D = (x_1 \vee x_3 \vee \overline{y_1})$, with the existential quantifier y_1 as the pivot, we get $\mathrm{OR} = (x_2 \vee x_4 \vee y_1 \vee x_1)$.
For $C = (y_1 \vee y_2 \vee x_1), D = (x_2 \vee \overline{x_1})$, with the universal quantifier x_1 as the pivot, we get $\mathrm{OR} = (y_1 \vee y_2 \vee x_2)$.

Definition 2.25 (Implies via Unit Propagation [1])**.** A propositional formula ψ **implies via unit propagation** a clause C, denoted by $\psi \vdash_1 C$,
iff applying unit propagation on $\psi \wedge \overline{C}$ we can derive empty clause $\bot$.

Example 2.26. Given $\psi = (\overline{a} \vee b) \wedge (\overline{c} \vee d) \wedge (\overline{c} \vee \overline{d})$ and the clause $C = (a \vee b)$.

continue

Definition 2.27 (QRAT [1]). A clause C has **QRAT on literal** $l \in C$ w.r.t. QBF $\Pi\psi$, iff for all $D \in \psi$ with $\bar{l} \in D$:

$$\psi \vdash_1 \mathrm{OR}(\Pi, C, D, l).$$

Example 2.28.

continue

With the QRAT definition in place, in order to make use of it we use the following theorems, from [3], that help us to transform a QBF in a satisfiable equivalent QBF:

Theorem 2.29 (QRAT for existential [3]). *Given a QBF $\phi = \Pi.\psi$ and a clause $C \in \psi$ with QRAT on existential literal $l \in C$ w.r.t. QBF $\phi' = \Pi'.(\psi \backslash \{C\})$, where Π' is adapted for $(\psi \backslash \{C\})$. Then ϕ and ϕ' are equisatisfiable.*

Theorem 2.30 (QRAT for universal [3]). *Given a QBF $\phi = \Pi.\psi$ and a clause $C \in \psi$ with QRAT on universal literal $l \in C$ w.r.t. QBF $\Pi.(\psi \backslash \{C\})$. Then ϕ and $\Pi.(\psi \backslash \{C\} \cup \{C \backslash \{l\}\})$ are equisatisfiable.*

speak whats with them

EUR

the proof system cheking

2.5 QCDCL

In this section, we will present an algorithm that can solve a QBF instance, with the underlying proof of the solution given in the Q-Resolution framework. The quantified conflict driven clause learning is the quantified version for the CDCL used for SAT solving. For the propositional problem in the satisfiable case it's enough to have an assignment, thus the CDCL algorithm tries to guess an assignment that evaluates the input to true, but in case the decisions result in a false formula, the procedure tries to derive a clause that minimize the search space. Similarly, in QCDCL we learn clauses to prune the search space of falsifying assignments, and learn cubes for running the satisfying assignment, [1]. Propozitie sa introduc ce urmeaza.

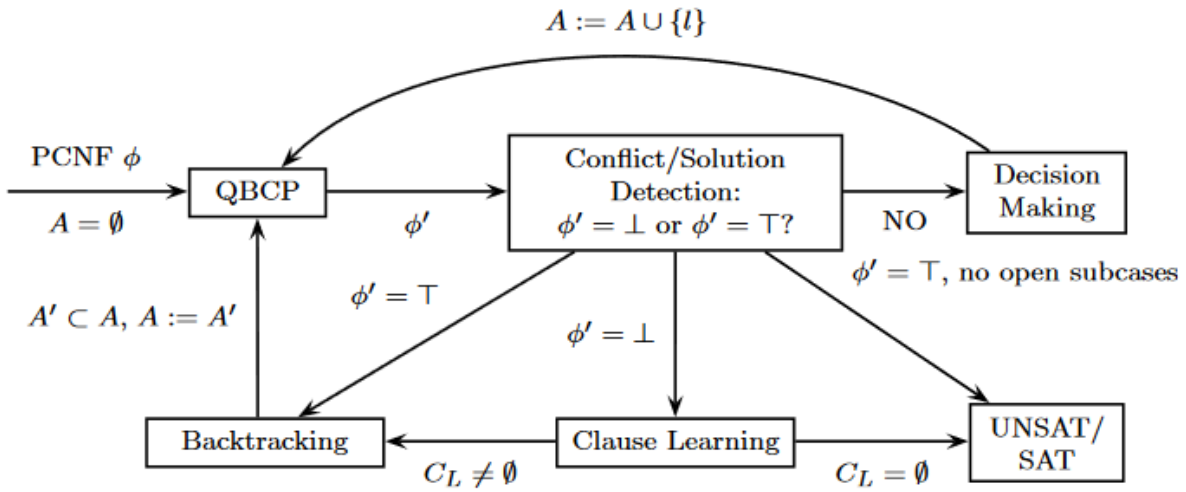

Figure 2: Flowchart of QCDCL from [1].

cum merge figura

Definition 2.31 (Unit literal detection [1])**.** A clause $C \in \phi$ is **unit** iff C contains one literal, and the literal is existential quantified. That literal is also called a **unit literal**. **Unit literal detection** applied to a QBF collects all unit clauses in the QBF.

Example 2.32. Let our formula be $\forall x_1 x_2 \exists y_1 \forall x_3 \exists y_2 (x_1 \vee x_2) \wedge x_3 \wedge y_1 \wedge \overline{y_2}$. The unit literal detection gives us $\{y_1, \overline{y_2}\}$.

Definition 2.33 (QBCP [1])**.** Given a PCNF ϕ and the empty assignment $A = \{\}$. We apply the following:

1. Apply universal reduction (UR) on $\phi[A]$ to get $\phi[A]'$.
2. Apply unit literal detection (UL) on $\phi[A]$ and append the result to A.
3. Repeat from 1. Stop if A haven't changed, or the formula is true or false.

Example 2.34. Given $\phi = \forall x_1 x_2 \exists y (x_1 \vee x_2 \vee \overline{y}) \wedge (y \vee \overline{x_1}) \wedge (y \vee \overline{x_y}) \wedge y$ we apply QBCP, we start with A empty:

- We cannot apply UR.
- From $\phi[]$ with UL, we get $A = \{y\}$.
- $\phi[y] = \forall x_1 x_2 (x_1 \vee x_2)$, we apply UR on x_2, and we get $\phi[y] = x_1$
- We cannot apply UL, because no existential literal is present.
- By applying UR, we get $\phi[y] = \bot$. Thus, we stop.

Using the previous example 2.34,
how a proof would look like for $\exists xy(\neg x \vee y)$ after assign of x

the scope of the work is not the solver but to derive a prove briefly bla bla

2.6 QDIMACS Format

2.7 QCIR Format

3 Quantified circuit Reconstruction Certification

In this chapter, we present the main aim of the work: quantified circuit reconstruction certification. Given a QBF ϕ in PCNF and a QCIR converter, our goal is to verify that the output of the converter is equisatisfiable with the given input. Restricting our approach only for the false instances of QBF, in order to check the satisfiable equivalence we propose the following solution: after we apply the converter, and get the formula ϕ_{QCIR}, we can reconstruct a refutation proof for ϕ from ϕ_{QCIR} refutation proof. This way, with the initial proof reconstruction, we can reassure the sound of the QCIR converter.

In the following, we assume that the input formula is false, and a QCIR converter gives a circuit that uses variables from the input without addition of other new variables.
In the first section we present the detailed steps for the main procedure, followed by sections that prove soundness of our procedure.

3.1 Certified QCIR reconstruction by initial proof

Before presenting the proof reconstruction we need to have a concise definition of what a QDIMACS to QCIR reconstruction program does.

Definition 3.1 (QCIR reconstruction)**.** A ϕ_{QCIR}, in QCIR format, is a **QCIR reconstruction** of ϕ, in PCNF format, iff there exists an assignment of the remaining variables, that are found in the ϕ but not in ϕ_{QCIR}, to a gate, such that $\models \psi \leftrightarrow \psi_{\text{QCIR}}$, where ψ_{QCIR}, ψ are the matrices of ϕ_{QCIR}, ϕ, respectively. In other words, the reconstruction comes from the fact that some variables can be deterministically derived from evaluation of the remaining ones.

Example 3.2. In the PCNF $\forall xy \exists t(x \vee y \vee \bar{t}) \wedge (\overline{x} \vee t) \wedge (\overline{y} \vee t)$, we can see t is a Tseitin variable for an OR-gate, thus a reconstruction can remove this added variable and keep it as $\forall xy \underbrace{(x \vee y)}_{g}$. And we can see that we can assign $t = g$, and the logical equivalence of the matrices is preserved.

With the definition 3.1 we want to capture the notion of the reconstruction of the formula where some existential quantified variables can be computed to the exact value such that the formulas are logical equivalent, and have a formula that uses less quantified variables.

Algorithm 1 Procedure for initial proof reconstruction from QCIR conversion.

Require: False PCNF: ϕ, QCIR converter procedure: QcirConv
Ensure: QRAT refutation proof P for ϕ

```
1: procedure GetInitialProof(φ, QcirConv)
2:     φ_QCIR ← QcirConv(φ)
3:     φ_Tseitin ← TseitinOfQcir(φ_QCIR)
4:     P_Q-Res ← QBFSolver(φ_Tseitin)
5:     P_QRAT ← QresToQrat(P_Q-Res)
6:     P_Initial-QRAT ← InitialQratReconstruction(φ, φ_Tseitin, P_QRAT)
7:     return P_Initial-QRAT
8: end procedure
```

In algorithm 1, we present the main procedure for getting a refutation proof for a given QBF using its circuit reconstruction form. First step is the application of the QCIR reconstruction on the initial QBF and get a circuit QBF ϕ_{QCIR}, respecting definition 3.1.

In the second step, we generate the PCNF of the QBF using Tseitin transformation, this way we introduce a variable for each of the gate, but more details will be presented in a dedicated section 3.2. In the third step, we apply a QBF solver based on QCDCL on the new formula, and this will produce a Q-Resolution proof for the ϕ_{QCIR} in PCNF. In the following step, with the Q-Resolution proof we can transform it in a QRAT proof, detailed in section 3.3. In the last step, with the initial QBF, the QCIR PCNF formula and its QRAT proof, we can derive a QRAT proof for the initial formula.
With the initial formula deducted from the QCIR we can use it as a certification for QCIR conversion, but more will be explained in the section 3.4.

3.2 Tseitin transformation of QCIR

Algorithm 2 PCNF from QCIR using Tseitin transformation procedure.

Require: QCIR formula: ϕ_{QCIR}

Ensure: Satisfiable equivalent PCNF form of input: ϕ_{Tseitin}

1: **procedure** TseitinOfQcir(ϕ_{QCIR})
2: $\quad\phi_{\text{Tseitin}} \leftarrow$ empty formula
3: $\quad$**for** gate in gates **do**
4: $\quad\quad\phi_{\text{Tseitin}} \leftarrow \phi_{\text{Tseitin}} \cup$ Tseitin(gate)
5: $\quad$**end for**
6: $\quad\phi_{\text{Tseitin}} \leftarrow \phi_{\text{Tseitin}} \cup t_{\text{output}}$ ▷ Tseitin encoding variable of output gate
7: $\quad$**return** ϕ_{Tseitin}
8: **end procedure**

In algorithm 2, we present the procedure that takes as an input a formula in QCIR and output the PCNF satisfiable equivalence of it. The procedure starts by initializing an empty formula where we will append clauses. Then, for each gate we introduce a variable t, and write the CNF formula of the ($t \leftrightarrow$ gate), that is the role of Tseitin function. For simplicity, if we have a gate that takes multiple inputs, we break down the formula for each 2 variables with auxiliary Tseitin variables, for example having the gate $a \wedge b \wedge c \wedge d$ we will use t_1 for $a \wedge b$, then use $t_1 \wedge c$ for t_2, etc. After we translate all the gates, to keep a CNF that is equisatisfiable with a given formula we also need to add the last Tseitin variable as a clause. As for the prefix, the prefix of ϕ_{Tseitin} will be the same as ϕ_{QCIR}, and each addition of a Tseitin variable will be added in the innermost existential block.
For the later use for proving the main result of the transformation, let's formulate the claim that states that the procedure 2 is sound, and produce what we want.

Claim 3.3. *Given a QCIR ϕ_{QCIR} the algorithm 2 produce a satisfiable equivalent PCNF $\phi_{Tseitin}$.*

Proof. □ need to think

3.3 QRAT proof from Q-Resolution proof

Algorithm 3 Q-Resolution to QRAT proof format.

Require: Q-Resolution proof of the ϕ_{Tseitin}: $P_{\text{Q-Res}}$

Ensure: QRAT proof format from Q-resolution: P_{QRAT}

```
1: procedure QResToQRat(P_Q-Res)
2:     P_QRAT ← empty list
3:     for line in P_Q-Res do          ▷ line is of form (resolvent, premise1, premise2)
4:         P_QRAT.append(resolvent)
5:         while resolvent highest level of a variable is universal do
6:             P_QRAT.append(universal reduction)
7:         end while
8:     end for
9:     return P_QRAT
10: end procedure
```

In algorithm 3, from a Q-resolution proof we produce a QRAT proof. We start by initializing an empty list where we will store each step of the QRAT proof. Then we iterate through all the lines in the Q-resolution proof. Each of this line contains the resolvent and the premises it came from. From this line, we are only interested in the resolvent, because the premises should already be present at this step of the proof (either a resolvent that was added before or as a clause in the initial formula). In a QRAT proof the QRAT literal is on the first position, thus we need to see what literal we put in the resolvent, but of the claim 3.5 it doesn't matter. After that, we need to check if we can apply the universal reduction, thus we check the variable that have the highest level and see if it is universal and apply the respective rule.
In addition, it will be important for our implementation to know what variable is the pivot in the Q-resolution proof. But this can be easily found using the claim 3.4, that states only one pivot it's available between two premises.

Claim 3.4. *Given a proof line from Q-resolution proof that is formed of two premises and their resolvent, only one pivot is available between the premises.*

Proof. Suppose two literals or more are available as pivot for the resolution rule, having the following premises $C \vee a \vee b$ and $D \vee \overline{a} \vee \overline{b}$ this cannot be the case, because if we apply the resolution rule on one literal the other one will produce a tautology and is not allow by the rule, false. Thus, our supposition was false, and between 2 premises there are only one pivot. □

Claim 3.5. *Every literal from a resolvent has the QRAT property.*

Proof. Our premises are of form $C \vee p$ and $D \vee \overline{p}$. Our resolvent is $C \vee D$. We want to check that $C \vee D$ has QRAT on an arbitrary literal l. Thus, we want to check that we can use implicit unit propagation to derive the outer resolvent of $C \vee D$ and another clause that has $\overline{l}$. But, the outer resolver already includes $C \vee D$ and if we use the negated literals from this clause, we can apply unit propagation on the premises and derive p and $\overline{p}$, thus producing the $\bot$ showing that we can apply QRAT on any literal of the resolvent. □

3.4 Initial QRAT reconstruction

Algorithm 4 QRAT proof for a QBF from its QCIR.

Require: Q-Resolution proof of the ϕ_{Tseitin}: $P_{\text{Q-Res}}$
Ensure: QRAT proof format from Q-resolution: P_{QRAT}
1: **procedure** QRESTOQRAT($P_{\text{Q-Res}}$)
2: **return** P_{QRAT}
3: **end procedure**

In algorithm 4,

Claim 3.6. *s*

4 Implementation

5 Experiment

References

[1] O. Beyersdorff, M. Janota, F. Lonsing, and M. Seidl, "Quantified boolean formulas," in *Handbook of Satisfiability*, pp. 1177–1221, IOS Press, 2021.

[2] W. Meert, J. Vlasselaer, and G. Van den Broeck, "A relaxed tseitin transformation for weighted model counting," in *Proceedings of the Sixth International Workshop on Statistical Relational AI (StarAI)*, pp. 1–7, 2016.

[3] M. J. Heule, M. Seidl, and A. Biere, "A unified proof system for qbf preprocessing," in *International Joint Conference on Automated Reasoning*, pp. 91–106, Springer, 2014.