Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning
Main Track

ASP-QRAT: a Conditionally Optimal Dual Proof System for ASP

Leroy Chew , Alexis de Colnet , Stefan Szeider
Algorithms and Complexity Group, TU Wien, Vienna, Austria
[lchew,decolnet,sz]@ac.tuwien.ac.at

Abstract

Answer Set Programming (ASP) is a declarative programming approach that captures many problems in knowledge representation and reasoning. To certify an ASP solver's decision, whether the program is consistent or inconsistent, we need a certificate or proof that can be independently verified.

This paper proposes the dual proof system ASP-QRAT that certifies consistent and inconsistent ASPs. ASP-QRAT is based on a translation of ASP to QBF (Quantified Boolean Formulas) and the QBF proof system QRAT as a checking format.

We show that ASP-QRAT p-simulates ASP-DRUPE, an existing refutation system for inconsistent disjunctive ASPs. We show that ASP-QRAT is conditionally optimal for consistent and inconsistent ASPs, i.e., any super-polynomial lower bound on the shortest proof size of ASP-QRAT implies a major breakthrough in theoretical computer science. The case for consistent ASPs is remarkable because no analog exists in the QBF case.

1 Introduction

Answer Set Programming (ASP) is a declarative programming approach originating from non-monotonic reasoning and logic programming (Brewka, Eiter, and Truszczynski 2011). ASP finds extensive use in fields such as knowledge representation, artificial intelligence, and planning, facilitating succinct problem modeling (Baral, Provetti, and Son 2003; Pontelli et al. 2012). Within ASP, problems are formulated as sets of rules known as logic programs or answer set programs (ASPs), which are processed according to the stable model semantics (Gelfond and Lifschitz 1993) and result in specific sets of atoms called answer sets. Various solvers like clingo (Gebser et al. 2011; Gebser et al. 2014), WASP (Alviano, Dodaro, and Ricca 2014), and DLV (Alviano et al. 2017) have been developed for ASP.

As argued by Alviano et al. (2019), ASP supports the aims of explainable AI (XAI) because it relies on deductive reasoning, which is inherently transparent, but certifying the inconsistency of an ASP (i.e., it has no answer sets) needs additional efforts to avoid the ASP solver acting as a black box. If the ASP is normal (i.e., no disjunctions in the rule heads), then one can check in polynomial time whether a set of atoms is indeed an answer set, but inconsistency checking is co-NP complete. For general ASP (with disjunctions), the complexities of both tasks move up one step in the Polynomial Hierarchy and become co-NP and Π_2^P complete, respectively (Eiter and Gottlob 1995; Cadoli and Lenzerini 1994; Marek and Truszczynski 1991). For the certification of co-NP tasks, one can utilize the proof logging techniques developed in the context of SAT solving (Wetzler, Heule, and Hunt 2014). However, for the certification of Π_2^P tasks, one needs additional concepts. To this aim, Alviano et al. (2019) proposed the proof format ASP-DRUPE that builds upon the Reverse Unit Propagation (RUP) format designed for UNSAT proofs.

In this paper, we propose an alternative proof format for ASP, which utilizes proof systems developed for Quantified Boolean Formulas (QBFs). The new proof system ASP-QRAT employs the QBF proof system QRAT (Quantified Resolution Asymmetric Tautology; Heule, Seidl, and Biere, 2014a) via a translation from ASP to QBF. ASP-QRAT is *conditionally optimal* for proofs of both inconsistent and consistent ASPs, in the sense that a super-polynomial lower bound to the shortest proof size of ASP-QRAT would imply $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ or a super-polynomial lower bound for Extended Resolution, both implications would constitute a major breakthrough in theoretical computer science. Similar results have already been shown for proof systems in auto-epistemic logic (Beyersdorff 2013), QBF (Beyersdorff et al. 2020) and for propositional model counting (Chede, Chew, and Shukla 2024). The underlying translation from ASP to QBF does not follow the standard reduction proposed by Eiter and Gottlob (1995). However, it works via Clark's Completion (Clark 1978) and the generation of loop formulas (Lin and Zhao 2004) and fits into modern ASP solvers' inner workings. ASP-QRAT is dual because it allows the certification of consistent and inconsistent ASPs.

In this paper, we focus on a QBF translation based on the proof techniques of ASP-DRUPE and show a p-simulation of ASP-DRUPE. There are other ASP proof systems, such as the tableaux systems, that work with normal ASPs (Gebser and Schaub 2006; Järvisalo and Oikarinen 2007). Other translations to QBF appear in the literature (Amendola et al. 2022; Faber, Mazzotta, and Ricca 2023; Amendola, Ricca, and Truszczynski 2019), and since ASPs are complete for the second level of the Polynomial Hierarchy, the reverse translation from 2QBF to ASP has also been studied (Amen-

dola, Dodaro, and Ricca 2016).

Other work has expanded on the capabilities of QRAT, Blinkhorn (2020) created a version of QRAT that works for DQBF, and Lonsing and Egly (2018) generalized the rules of QRAT for QBF.

2 Preliminaries

2.1 Proof Complexity

A *proof system* (Cook and Reckhow 1979) is a polynomial-time function that maps proofs to theorems of a fixed logic (i.e., propositional logic). The *proof size* is determined by the number of symbols in the proof. We denote the proof size of proof π as $|\pi|$. A proof system is *polynomially bounded* if, for every theorem x, there is a proof π such that $f(\pi) = x$ and $|\pi| = |x|^{O(1)}$. We say a proof system f *p-simulates* a proof system g if there is a polynomial-time function γ that maps g-proofs to f-proofs of the same theorem; in other words $f(\gamma(\pi)) = g(\pi)$.

2.2 Propositional Logic

A *literal* is a propositional variable or its negation. We use $\bar{l}$ to denote the negative literal $\neg l$ when l is the positive literal of a variable. Otherwise, we use $\bar{l}$ to denote the positive literal x when l is the negative literal $\neg x$ for some variable x. A *clause* is a disjunction of literals, denoted by repeated use of the $\vee$ symbol, i.e., $\overline{x} \vee y \vee \overline{z}$ (or $\bigvee_{a \in A} a$). Whenever convenient, we use set notation to denote a clause, including disjuncts as members. A *unit clause* is a clause that contains only a single literal. A clause C subsumes clause D if C is a subset of D. *Subsumption* is the process of replacing a subsumed clause with the clause that subsumes it. We can consider the *negation of a clause* $\overline{C}$ (or $\neg C$) as a conjunction of unit clauses. A propositional formula is said to be in *conjunctive normal form* (CNF) if it is a conjunction of clauses. We can either denote it by repeated use of the $\wedge$ symbol or use set notation where members are clauses that are conjuncts.

Assume we are given two propositional formulas ϕ_1 and ϕ_2. We say they are *logically equivalent* if they both have exactly the same set of models. They are *satisfiability equivalent* or *equisatisfiable* if whenever there is a model α_1 for ϕ_1, there exists a model α_2 for ϕ_2 and vice versa.

Unit propagation ($\vdash_1$) is a technique for simplifying CNFs. We first detect any unit clauses, i.e., a clause with a single element l. We now consider all clauses that contain the literal $\bar{l}$, and we can safely remove $\bar{l}$ from the clause without altering the models of the formula. In simplifying the clauses, we may make more clauses unit, and we repeat the unit propagation steps until a fix-point or the empty clause is found. The number of unit propagation steps is bounded by the number of literals. As well as being a simplification rule, unit propagation can be considered a method of building up a partial assignment.

Reverse unit propagation (RUP) is a sound technique for finding clauses that can be added or removed while preserving logical equivalence. Consider a unit propagation from $\phi \wedge \overline{C}$ that results in the empty clause $\phi \wedge \overline{C} \vdash_1 \bot$. This means $\phi \wedge \overline{C}$ is unsatisfiable. Suppose ϕ has a model α, then α cannot also be a model for $\overline{C}$; therefore, it must satisfy C. Since all models of $\phi \wedge C$ satisfy ϕ by definition, $\phi \wedge C$ and ϕ are logically equivalent.

Every propositional formula can be feasibly transformed into a satisfiability equivalent CNF. For more complicated formulas with non-clausal definitions, we can use definition clauses to represent subformulas. The definition clauses for e when $e \Leftrightarrow \bigwedge_{l \in L} l$ for a set L of literals is given as the long clause $e \vee \bigvee_{l \in L} \neg l$ and a short clause for each $l \in L$,$\neg e \vee l$. This works similarly for disjunction by taking the definition clauses for the negation (a conjunction).

Observation 1. *Given a set of definition clauses θ for variable e under input variables X, given an assignment to X expressed as a set of unit clauses α. Either $\theta \wedge \alpha \vdash_1 e$, or $\theta \wedge \alpha \vdash_1 \bar{e}$.*

Resolution is a propositional proof system that refutes unsatisfiable CNFs (Robinson 1963). Resolution has as its single inference rule the *resolution rule*, which, utilizes existing clauses $C_1 \vee x$ and $C_2 \vee \neg x$ to add a new clause $C_1 \vee C_2$, the *resolvent*. A CNF is logically equivalent with or without the resolvent. Resolution aims to derive the empty clause and thus show a contradiction. A CNF has a resolution proof that reaches the empty clause if and only if it is a contradiction. Adding a clause through RUP is a stronger form of the resolution rule, and we will sometimes refer to a resolution step as a specific use of RUP.

Extended resolution (Ext Res) is a more powerful version of resolution. In addition to the resolution rule, extended resolution allows the addition of definition clauses for $\wedge$ and $\vee$ definitions with the *extension rule*, as long as the defined variable is new and all input variables are previously defined. Because the variable is new and well-defined, the extension rule preserves satisfiability.

2.3 Quantified Boolean Formulas

A Quantified Boolean Formula (QBF) is a propositional formula augmented with Boolean quantifiers $\forall$ and $\exists$ that bind propositional variables and range over the values $0, 1$. For a QBF Φ with free variable x, the QBF $\forall x \Phi$ is true if and only if $\Phi[0/x]$ and $\Phi[1/x]$ are both true ($\Phi[c/x]$ denotes that c replaces x in Φ), $\exists x \Phi$ is true if and only if at least one of $\Phi[0/x]$ or $\Phi[1/x]$ are true.

A prenex conjunctive normal form (PCNF) contains a CNF propositional matrix, and a quantifier prefix, that uses Boolean quantifiers $\forall$ and $\exists$ that bind propositional variables that appear in the matrix. The prefix is ordered linearly; each variable has a *quantifier level* starting at the leftmost variable which has level 1. As we move rightwards, we increase the level by one whenever the quantifier changes. Variables on the same level are said to be in the same *quantifier block*. A *closed* PCNF is a PCNF where every variable in the matrix appears in the prefix.

In this paper, we almost exclusively use PCNFs with quantifier alternation $\exists X \forall Y \exists Z$. We refer to the variables in block X as the *outer variables* and the variables in block Z as the *inner variables*. We refer to the variables in Y as the *universal variables*.

We can represent the semantics of a QBF (with a closed prefix) by a semantic two-player game. Following the order of the prefix players $\exists$ and $\forall$ each set a Boolean value to each variable that appears bound by their quantifier. $\exists$ is trying to get the propositional matrix to become true and $\forall$ is trying to get the matrix to become false. A QBF is true if and only if $\exists$ has a winning strategy. Likewise, a QBF is false if and only if $\forall$ has a winning strategy. *Skolem functions* return values for each $\exists$ variable so that the propositional matrix will be satisfied no matter the universal player's choices. Skolem functions for variable x need only to have universal variable inputs of the $\forall$ variables left of x in the prefix. Herbrand functions work the same for universal values and false QBFs. One can observe that replacing a propositional matrix with a logically equivalent matrix preserves Skolem and Herbrand functions, and therefore preserves the truth value of the QBF.

We can transform any QBF into an equisatisfiable propositional formula by $\forall$-*expansion*. Suppose we have a partial prefix Π, which we extend leftwards into a full prefix $\exists X \forall u \Pi$ and a propositional matrix ϕ. $\exists X \forall u \Pi \phi$ is true if and only if there exists a value of X such that $(\Pi\phi[0/u])$ and $(\Pi\phi[1/u])$ are both true. However, when writing this as a QBF, we must be careful about the use of variables being repeated in separate quantifiers, so we create two copies of each Π variable z: $z^{\bar{u}}$ and z^u. $\phi^{\bar{u}}[0/u]$ and $\phi^u[0/u]$ use renamings of ϕ when using these annotated variables. We get that $\exists X \forall u \Pi \phi$ is satisfiability equivalent to $\exists X \Pi^0 \Pi^1 \phi^{\bar{u}}[0/u] \wedge \phi^u[0/u]$. We can repeat this until all universal variables are removed; at this point we have a singular block of existential variables and a CNF, a satisfiability problem. The issue with $\forall$-expansion is that we can generate exponentially many clauses from a linear number of universal variables. Nonetheless, it forms the basis of many modern QBF solvers (Rabe and Tentrup 2015; Janota 2017).

2.4 Answer Set Programming

A *disjunctive answer set program* (or simply an *ASP*) P is a set of *rules* of the following form

$$x_1 \vee \cdots \vee x_l \quad \leftarrow \quad y_1, \ldots, y_m, \neg z_1, \ldots, \neg z_n$$

where $x_1, \ldots, x_l, y_1, \ldots, y_m, z_1, \ldots, z_n$ are atoms l, m, n are non-negative integers. Let r be a rule. We write $\{x_1, \ldots, x_l\} = H(r)$ (the *head* of r), $\{y_1, \ldots, y_m\} = B^+(r)$ (the positive body of r) and $\{z_1, \ldots, z_n\} = B^-(r)$ (the negative body of r). We denote the sets of atoms occurring in a rule r or in a program P by $\text{at}(r) = H(r) \cup B^+(r) \cup B^-(r)$ and $\text{at}(P) = \bigcup_{r \in P} \text{at}(r)$, respectively. A rule r is *normal* if $|H(r)| \leq 1$. We say that an ASP is normal if all its rules are normal.

A set M of atoms *satisfies* a rule r if $(H(r) \cup B^-(r)) \cap M \neq \emptyset$ or $B^+(r) \setminus M \neq \emptyset$. M is a *model* of P if it satisfies all rules of P. The *Gelfond-Lifschitz (GL) reduct* of an ASP P under a set M of atoms is the program P^M obtained from P by (1) removing all rules r with $B^-(r) \cap M \neq \emptyset$ and (2) removing all literals $\neg z$ where $z \in B^-(r)$ from the remaining rules r (Gelfond and Lifschitz 1993). M is an *answer set* of an ASP P if M is a minimal model of P^M.

Given an ASP P, the *positive dependency digraph* D_P uses atoms as nodes and has an edge (a, b) if there is a rule r such that $a \in B^+(r)$ and $b \in H(r)$. A loop in P is a loop in D_P. The ASP P is *tight* if it contains no loops.

Clark's Completion Clark's completion (Clark 1978) transforms an ASP P into a conjunctive normal form Δ_P. If P is consistent then Δ_P is satisfiable, and the consistency of P and the satisfiability of Δ_P are equivalent when P is tight. With disjunctive answer set programs, Δ_P remains polynomial in the size of P.

Clark's completion breaks disjunctive rule r into a series of implications with $a \in H(r)$ being implied by a new variable called an induced body b. b is an extension variable equivalent to a conjunction of literals, from the body of the rule, and the negation of the rest of the head.

$$b \Leftrightarrow \bigwedge_{c \in H(r) \setminus \{a\}} \bar{c} \wedge \bigwedge_{c \in B^+(r)} c \wedge \bigwedge_{c \in B^-(r)} \bar{c}$$

The definition clauses for each b are called *body definition clauses*. For each head atom in a rule there is a singleton set $\text{IB}(r, a)$ that contains the induced body. Every rule has as many induced bodies as atoms in its head. For every atom a, we collect all its induced bodies in the set $\text{IB}(P, a) = \bigcup_{r \in P}^{a \in H(r)} \text{IB}(r, a)$. We use the set $\text{Bod}(P)$ to collect all induced bodies for P. The set Δ_P^{Bdef} contains all body definition clauses for every member of $\text{Bod}(P)$.

Let b be the single member of $\text{IB}(r, a)$. As mentioned earlier, their use in rule r is seen as an implication, we will call a *completion support clause*. This is given as a binary clause $\neg b \vee a$. We say that a is true only if at least one induced body for some rule regarding a is true: $\neg a \vee \bigvee_{b \in \text{IB}(P,a)} b$, and call this the *completion rule clause*.

The Clark's completion Δ_P contains all definition clauses from Δ_P^{Bdef}, all completion support clauses and all completion rule clauses. Once again, it should be stated that completion rule clauses do not fully capture the minimization needed for P. If an induced body b in $\text{IB}(P, a)$ is found in a loop from a being true, it could still be satisfiable to assume both b and a to be true. This is why, for programs that contain loops, we require reasoning based on loop formulas.

Basic Loop Clauses In order to fully minimize, we need that atom a is false when it appears on a loop ω and no external criterion sets it true. We will detail how this is enforced with clauses.

Given a loop ω and an induced body b, we say b is *independent* of ω if no positive conjunct in the definition of b is in ω. We define $\text{EB}(P, \omega)$, the set of external bodies of a loop, these are induced bodies that imply some member of the loop without being part of it. $\text{EB}(P, \omega) := \{\, b \mid b \in \text{IB}(P, a), b \text{ independent of } \omega, a \in \omega \,\}$.

As part of minimization insists that if a, a member of loop w is true, at least one external body of the loop is true. We call this a *basic loop clause* $\lambda(a, \omega) := \neg a \vee \bigvee_{b \in \text{EB}(P,\omega)} b$.

The set of all basic loop clauses of disjunctive ASP P is defined as Λ_P, but since the number of loops is potentially exponential, Λ_P can explode exponentially.

$\Delta_P \wedge \Lambda_P$ is satisfiable if and only if P is a consistent ASP.

ASP-DRUPE ASP-DRUPE (Alviano et al. 2019) is a format for checking the inconsistency of ASPs implicitly using the translation to $\Delta_P \wedge \Lambda_P$. In the original paper, ASP-DRUPE is defined using *no-goods*, these are exactly the negations of clauses. Here we define ASP-DRUPE using clauses directly, but this does not affect the proof complexity of the system. One more significant alteration here is that ASP-DRUPE is a more flexible format that allows for disjunctive ASPs with *weight rules*, we have not included weight rules in our definition of disjunctive ASPs. Here, for simplicity, we focus on the part of ASP-DRUPE that deals with disjunctive rules only.

ASP-DRUPE starts with an empty set ∇, and ends with ∇ containing the empty clause.

Completion-Axiom : Add any clause from Δ_P.
Loop-Axiom : Add any clause from Λ_P.
Deletion: Remove any clause from ∇.
RUP: Add any clause C to ∇ if $\nabla \wedge \bar{C} \vdash_1 \bot$.
Ext: For a new variable e add all the definition clauses to ∇.

Despite ASP-DRUPE using the powerful technology of extended resolution, it may be possible that an exponential number of loop clauses are needed to even get a semantic contradiction. In the next section we explain how using QBFs can shrink the number of necessary loop clauses.

On the other hand ASP-DRUPE is not automatically lower bounded whenever Λ_P is exponential. It can opt to only lazily introduce basic loop clauses and it might be that a propositional contradiction is possible with a small subset of the loop clauses.

3 ASP to QBF

In this section, we succinctly represent the ASP consistency problem as a QBF.

Observation 2. *Falsifying a candidate answer set for program P is in NP, where the polynomial-time checkable witness is a loop.*

Given an answer set, we want to know whether it (along with all computed induced body variables) falsifies a clause from $\Delta_P \wedge \Lambda_P$. We can check in polynomial time whether Δ_P is falsified. Otherwise, we know where to look in Λ_P if we are given the right loop. Then we only need to check $|\text{at}(P)|$ many basic loop clauses.

Another way of looking at this is that checking an answer set is in coNP i.e., we can express checking an answer if we universally quantify over all possible loops. Using this idea we can compress the CNF used in ASP-DRUPE, by succinctly hiding the loops with quantification

$$\Pi := \exists\langle\text{atom vars}\rangle\ \exists\langle\text{body vars}\rangle\ \forall\langle\text{loop slots}\rangle\ \exists\langle\text{tse defs}\rangle.$$

Here, $\langle$atom vars$\rangle$ are the variables that appear as atoms in the ASP (P). The Clark's completion and basic loop formulas contain variables from $\langle$atom vars$\rangle$ and $\langle$body vars$\rangle$. We will also use the two new levels $\forall\langle$loop slots$\rangle$ (see Section 3.1) and $\exists\langle$tse defs$\rangle$ (see Section 3.2), that will succinctly represent the necessary information the loop clauses provided. Note that the third level, that solely contains definition variables, is only necessary because we use a CNF.

In ASP-DRUPE, we have a set of clauses for every atom a and all its loops ω in a set Λ_P. Each basic loop clause $\neg a \vee \bigvee_{b \in \text{EB}(P,\omega)} b$ states that a is minimized to false if none of the external bodies of the loop ω that contains it are true.

Here instead we replace this with one clause for each variable a, which we will call the *compact loop clause*:

$$\neg a \vee \neg u_a \vee \neg\mathsf{valid\_loop} \vee \bigvee_{b \in \text{Bod}(P)} \mathsf{extern}(b)$$

This says that if a is truly a member of *the* loop (specified by the $\forall$ variables) and if none of the external bodies of that loop are true then a is minimized to false.

u_a is the universal variable selecting atom a to be included in the loop. $\mathsf{extern}(b)$ and $\mathsf{valid\_loop}$ have to occur in the innermost $\exists$ level as Tseitin variables, because they are defined using the choice of loop slots. Before we define them fully, let us examine their use in this clause.

The expression $\mathsf{valid\_loop}$ must be false if the loop is not valid. The validity of the loop depends on whether all the loop slots follow from the previous loop slot and whether the end wraps around again to the beginning.

Notice that the clause now contains an $\mathsf{extern}(b)$ literal for every induced body. This means that we want to be able to ignore the literal $\mathsf{extern}(b)$ when the loop is chosen so that b is not an external body. In other words, $\mathsf{extern}(b)$ should be false if b is not an external body; it should also be false if b as an induced body variable is false.

We contain all compact loop clauses from program P in the set Ω_P. Along with the definition clauses used for the Tseitin variables contained in set Θ_P we get our QBF translation of the ASP program P:

$$\Psi_P = \Pi(\Delta_P \wedge \Omega_P \wedge \Theta_P).$$

3.1 Loop Slot Variables

For ASP P we have two kinds of universal variables in the QBF Ψ_P.

u_a: this specifies that variable a is contained in the loop.
u_a^b: this specifies that the next variable in the loop after a is b.

These universal variables appear in the clauses of Θ_P to define the Tseitin variables that appear in Ω_P.

3.2 Tseitin Variables and their Definitions

The universal variables are important to the QBF Ψ_P but have to be used as definitions when we make the compact loop clauses. Since we use definition clauses, quantification of new variables has to be existential and it has to happen after the loop slot variables. This is standard in QBF (Jussila et al. 2007; Beyersdorff, Chew, and Janota 2016), but to see this, think of the existential player being forced to satisfy the definition clauses, he can only do this after the other variables are chosen, else the universal player can easily falsify the definition clauses.

- $\mathsf{v\_next}(a, b)$: if atom b is next after atom a it must also be specified in the loop. The definition is: $\mathsf{v\_next}(a, b) \Leftrightarrow \bar{u}_a^b \vee u_b$.
- $\mathsf{v\_prev}(a, b)$: if atom a is previous from atom b it must also be specified in the loop. The definition is: $\mathsf{v\_prev}(a, b) \Leftrightarrow \bar{u}_a^b \vee u_a$.
- $\mathsf{slot}(i, a)$: atom a comes next, after the $i-1$th slot. For $i = 1$ we fix some ordering $<$ on the atom variables, the definition is: $\mathsf{slot}(1, a) \Leftrightarrow u_a \wedge \bigwedge_{b\in \mathrm{at}(P)}^{b<a} \bar{u}_b$. For $i \geq 2$, $\mathsf{slot}(i, a) \Leftrightarrow \bigvee_{b\in \mathrm{at}(P)} u_b^a \wedge \mathsf{slot}(i-1, b)$. We create auxillary variables $\mathsf{aux\_slot}(i, a, b) \Leftrightarrow u_b^a \wedge \mathsf{slot}(i-1, b)$, to succinctly represent slot variables as disjunctions.
- $\mathsf{conn}(a)$: atom a comes in as loop variable in or after the first slot. The definition is $\mathsf{conn}(a) \Leftrightarrow \bigvee_{i\geq 1}^{i\leq n} \mathsf{slot}(i, a)$.
- $\mathsf{in\_at\_most}(a, b, c)$: the loop has at most one edge arriving to atom a. When $(b, a), (c, a) \in D_P$ and $b \neq c$ the definition is $\mathsf{in\_at\_most}(a, b, c) \Leftrightarrow \bar{u}_b^a \vee \bar{u}_c^a$.
- $\mathsf{out\_at\_most}(a, b, c)$: the loop has at most one edge leaving from atom a. When $(a, b), (a, c) \in D_P$ and $b \neq c$ the definition is $\mathsf{out\_at\_most}(a, b, c) \Leftrightarrow \bar{u}_a^b \vee \bar{u}_a^c$.
- $\mathsf{in\_at\_least}(a)$: the loop has at least one edge arriving to a, if a is specified as in the set. The definition is $\mathsf{in\_at\_least}(a) \Leftrightarrow \bar{u}_a \vee \bigvee_{(b,a)\in D_P} u_b^a$.
- $\mathsf{out\_at\_least}(a)$: the loop has at least one edge leaving from a, if a is specified as in the set. The definition is $\mathsf{out\_at\_least}(a) \Leftrightarrow \bar{u}_a \vee \bigvee_{(a,b)\in D_P} u_a^b$.
- $\mathsf{no\_extra}(a)$: no extra disconnected loop contains an a. The definition is $\mathsf{no\_extra}(a) \Leftrightarrow \bar{u}_a \vee \mathsf{conn(a)}$.
- $\mathsf{valid\_loop}$: all checks are fine. We give this definition as $\mathsf{valid\_loop} \Leftrightarrow \bigwedge_{a\in \mathrm{at(P)}} \mathsf{no\_extra}(a) \wedge \mathsf{in\_at\_least}(a) \wedge \mathsf{out\_at\_least}(a) \wedge \bigwedge_{b\in \mathrm{at}(P)}^{(a,b)\in D_P} \mathsf{v\_next}(a, b) \wedge \mathsf{v\_prev}(a, b) \wedge \bigwedge_{c\in \mathrm{at}(P)\setminus\{a\}}^{(c,b)\in D_P} \mathsf{in\_at\_most}(b, a, c) \wedge \bigwedge_{d\in \mathrm{at}(P)\setminus\{d\}}^{(a,d)\in D_P} \mathsf{out\_at\_most}(a, b, d)$.
- $\mathsf{loop\_adj}(b)$: as a necessary condition of being an external body, that states that induced body b is the body for some loop member. $\mathsf{loop\_adj}(b) \Leftrightarrow \bigvee_{a\in \mathrm{at}(P)}^{b\in \mathrm{IB}(a)} u_a$.
- $\mathsf{is\_extern}(b)$: as a consequence of the loop choice, b is an external body. This means no positive atom in its definition can be in the loop (independence). $\mathsf{is\_extern}(b) \Leftrightarrow \mathsf{loop\_adj}(b) \wedge \bigwedge_{c\in \mathrm{B}^+(r), r\in P,}^{a\in \mathrm{H}(r), \mathrm{IB}(r,a)\ni b} \bar{u}_c$.
- $\mathsf{extern}(b)$: b is both external and true, this is the only inner Tseitin variable to depend on an outer variable and not just the loop choice universal variables. $\mathsf{extern}(b) \Leftrightarrow \mathsf{is\_extern}(b) \wedge b$.

Lemma 1. *The associated QBF Ψ_P is polynomial in the size of P.*

Proof. There are polynomially many new body variables in $\mathrm{Bod}(P)$, each of which has a linear size definition.

For the completion support clauses, each standard variable a shares a clause with its induced body variables which is bounded above by $|\mathrm{Bod}(P)|$. Likewise, for each atom a, there is a completion rule clause for every induced body in $\mathrm{IB}(a)$. There are polynomially many definition clauses in the inner Tseitin variables. The sizes of loop clauses are bounded above by $|\mathrm{Bod}(P)|$ and there is one for each standard variables. □

Lemma 2. *An ASP P is consistent if and only if the associated QBF Ψ_P is true.*

Proof. In the forward direction, we show Ψ_P is true by exhibiting its Skolem functions.

We know P is consistent if and only if the propositional formula $\Delta_P \wedge \Lambda_P$ is satisfiable (Gebser et al. 2012). If P is consistent, we take the Skolem function for the outer variables as the constant values that satisfy $\Delta_P \wedge \Lambda_P$. For the inner variables, these are Tseitin variables, we use the Skolem function that follows the definition of the Tseitin variables. This Skolem function will satisfy all definition clauses. We have to show that all compact loop clauses will be satisfied by this Skolem function. Suppose there is a loop slot assignment γ such that under these Skolem functions a loop clause of variable a will be falsified. a must be true in our answer set for this to possibly cause a contradiction, γ must relate to a valid loop w that contains a and the external bodies of w must all be false. But then the assignment must falsify the clause $\lambda(a, w) \in \Lambda_P$, contradicting our assumption.

In the other direction, we show the QBF is false by exhibiting its Herbrand functions. We know that if P is inconsistent then the propositional formula $\Delta_P \wedge \Lambda_P$ is unsatisfiable. So we take an assignment α to $\Delta_P \wedge \Lambda_P$ (which will be an outer assignment in QBF) and show how to choose the universal value $\beta(\alpha)$ that guarantees that at least one clause in our QBF's matrix will be falsified. If α already falsifies Δ_P then $\beta(\alpha)$ can be set arbitrarily. Otherwise α falsifies a loop clause $\lambda(a, w) \in \Lambda_P$, in this case we pick the assignment $\beta(\alpha)$ to construct loop w. α must set a to true as well as all external bodies of w to false. The existential player has to play all Tseitin variables in the inner block according to their definitions; otherwise, a clause in the definition is falsified. This will force the loop clause of a to simplify to $\lambda(a, w)$, which is falsified by α. □

4 The ASP-QRAT Proof System

For clause C we use $O(C)$ to define the outer clause, the subclause of C containing only and all existential outer literals.

Definition 1. *Consider a PCNF $\Pi\phi$, where Π is of the form $\exists\forall\exists$. A* resolution path *from clause C is a list of inner literals $p_1 \dots p_k$ and a list of clauses $C_1 \dots C_k$ in ϕ such that $p_1 \in C$ and for every $1 \leq i < k$, $\bar{p}_i \in C_i$, $p_{i+1} \in C_i \setminus \{\bar{p}_i\}$.*

The resolution paths tell us which clauses can be connected at the inner levels via a resolution proof. Basically, once all outer and universal variables are assigned we have a propositional formula whose unsatisfiability is connected to the existence of a resolution proof (via the refutational completeness of propositional resolution). The lack of resolution

path guarantees that two clauses are never in the same unsatisfiable core for the inner levels, no matter the assignment to the outer and universal variables.

We will now describe the ASP-QRAT format. We maintain a propositional matrix Φ and a $\exists\forall\exists$ prefix, which we can add to.

4.1 Initialization and Processing Rules

To prove consistency or inconsistency for ASP P we initialize ASP-QRAT with the QBF Ψ_P. Then we have the following rules.

ATA: We can add clause C if $\Phi \wedge \neg C \vdash_1 \bot$.
ATE: We can remove clause C if $\Phi \setminus \{C\} \wedge \neg C \vdash_1 \bot$.
ORATA: We can add clause C if there is an outer literal $l \in C$ such that for every clause D that contains $\bar{l}$: $\Phi \wedge \overline{O(D) \setminus \{\neg l\}} \wedge \bar{C} \vdash_1 \bot$.
ORATE: We can remove clause C if there is a outer literal $l \in C$ such that for every clause D that contains $\bar{l}$: $\Phi \setminus \{C\} \wedge \overline{O(D) \setminus \{\neg l\}} \wedge \bar{C} \vdash_1 \bot$.
IRATA: We can add clause C if there is an inner literal $l \in C$ such that for every clause D that contains $\bar{l}$: $\Phi \wedge \overline{D \setminus \{\neg l\}} \wedge \bar{C} \vdash_1 \bot$.
IRATE: We can remove clause C if there is an inner literal $l \in C$ such that for every clause D that contains $\bar{l}$: $\Phi \setminus \{C\} \wedge \overline{D \setminus \{\neg l\}} \wedge \bar{C} \vdash_1 \bot$.
Local-Purity: This rule allows us to remove a loop-slot $\forall$ literal u from a clause if it is "locally" pure. For our purposes here for the ASP use, the local purity rule is as follows: if no resolution path from C reaches a clause containing $\bar{u}$ we can safely remove u from C. Reachability via resolution paths is polynomial-time checkable (Slivovsky and Szeider 2014; Slivovsky and Szeider 2016).

We provide an example of the Local Purity rule in Example 1. For readers unfamiliar which RAT rules, in Section 5, Example 3 doubles as an example of ORATA.

Example 1. *Consider the CNF $(x \vee u \vee y) \wedge (\bar{x} \vee \bar{u} \vee y) \wedge (\bar{y})$ where x is an outer variable, u a universal variable and y an inner variable. The only resolution paths from each of u and $\bar{u}$ clauses in the inner variables y lead to $(\bar{y})$ and cannot proceed. Hence there are no resolution paths in the inner variables that connect clauses $(x \vee u \vee y)$ and $(\bar{x} \vee \bar{u} \vee y)$, which makes u locally pure in $(x \vee u \vee y)$ and $\bar{u}$ locally pure in $(\bar{x} \vee \bar{u} \vee y)$. Safely removing these literals gives us $(x \vee y) \wedge (\bar{x} \vee y) \wedge (\bar{y})$. The empty clause can now be derived with ATA.*

A proof that uses only these rules is what we call a *search mode* ASP-QRAT proof, because it can used when there is a lack of prior knowledge about the consistency of the ASP. However when simply checking a proof we can know in advance whether the proof returns consistent or inconsistent, when it ends with a empty CNF, or an empty clause, respectively. Additional rules can be used. For *refutation mode*, we can arbitrarily **delete** clauses, (in other words we can ignore ATE, ORATE, IRATE). Refutation mode ASP-QRAT proofs end with a CNF containing the empty clause. For *satisfaction mode* we can arbitrarily **add** clauses, (in other words we can ignore ATA, ORATA, IRATA, we can ignore Local-Purity as well because of subsumption). Satisfaction mode proofs end in an empty CNF.

Example 2. *Consider the program P with the following rules:*

$$q \leftarrow p \quad p \leftarrow q \quad q \leftarrow \neg p \quad p \leftarrow q$$

The QBF $\Psi_P =$

$$\exists p, q, b_1, b_2, b_3, b_4 \forall u_p, u_q, u_p^q, u_q^p \exists \langle \textit{tse defs} \rangle \Delta_P \wedge \Omega_P \wedge \Theta_P$$

$$\Delta_P = (b_1 = p) \wedge (b_2 = q) \wedge (b_3 = \bar{p}) \wedge (b_4 = \bar{q}) \wedge (\bar{p} \vee b_2 \vee b_4) \wedge (\bar{q} \vee b_1 \vee b_3) \wedge (\bar{b}_1 \vee q) \wedge (\bar{b}_2 \vee p) \wedge (\bar{b}_3 \vee q) \wedge (\bar{b}_4 \vee p)$$

$$\Omega_P = (\neg p \vee \neg u_p \vee \textit{valid\_loop} \vee \bigvee_{i=1}^{4} \textit{extern}(b_i)) \wedge (\neg q \vee \neg u_q \vee \textit{valid\_loop} \vee \bigvee_{i=1}^{4} \textit{extern}(b_i)).$$

We will not go into the full detail of Θ_P, instead we know through ASP-QRAT we can derive the following through propositional (ATA) reasoning.

- $(\bar{u}_p \vee \bar{u}_q \vee \bar{u}_p^q \vee \bar{u}_q^p \vee \textit{valid\_loop})$. *That the p, q loop is valid*
- $(\textit{extern}(b_1) \vee \bar{u}_p) \wedge (\textit{extern}(b_2) \vee \bar{u}_q)$. *That b_1, b_2 cannot be external unless their constituent atoms are not part of the loop.*
- $(\textit{extern}(b_3) \vee \neg p) \wedge (\textit{extern}(b_4) \vee \neg q)$. *That b_3, b_4 are external only when their constituent literals are true.*

With resolution (using ATA) we can derive $(\neg p \vee \neg q \vee \bar{u}_p \vee \bar{u}_q \vee \bar{u}_p^q \vee \bar{u}_q^p)$ by resolving with one of the compact loop clauses. Using Local Purity this reduces to $\neg p \vee \neg q$ which resolves to a contradiction with clauses from Δ_P.

4.2 Soundness and Completeness of ASP-QRAT

Theorem 1 (Soundness)**.** *Let P be an ASP, consider π an ASP-QRAT proof that initializes from P. If π is in refutation or search mode and returns a line containing the empty clause, then P is an unsatisfiable ASP. Likewise, if π is in satisfaction or search mode and returns a line containing no clauses, then P is a satisfiable ASP.*

Each rule is covered by a QRAT rule, so it suffices to use the original soundness proof of QRAT. We show how each rule of ASP-QRAT is sound for the benefit of the reader.

Proof. Consider the prefix Π fixed, in the sense that all lines should have the largest prefix, which will always be the prefix of the final line. This will give meaning to the idea of preserving or modifying Skolem functions. We first use Lemma 2 to match the ASP P's satisfiability to Ψ_P. The rules of ASP-QRAT preserve satisfiability.

Using ATA or ATE adds or removes a clause based on logical equivalence of the propositional matrix. Therefore, the same Skolem functions work.

Suppose ORATA adds a clause $C \vee l$ to a satisfiable QBF $\Pi\phi$. There is a satisfying outer assignment α to the outer variables of ϕ. We can modify α to get α' which we will show to satisfy $\Pi\phi \wedge (C \vee l)$. Consider all clauses D such that $\bar{l} \in D$. If all $O(D) \setminus \{\bar{l}\}$ are satisfied by α and $\alpha(l) = 0$ we take $\alpha'(l) = 1$, and $\alpha'(a) = \alpha(a)$ on all other values $a \notin \{l, \bar{l}\}$. The remaining Skolem functions for the inner variables remain unchanged, all clauses will be satisfied by α' and the Skolem functions because only clauses that contain $\bar{l}$ can be affected by a positive change to l. On the other

hand, if some subclause $O(D) \setminus \{\bar{l}\}$ is falsified by α, $C \vee l$ is automatically satisfied by the ORATA side condition, so $\alpha' = \alpha$ suffices, in this case.

Suppose IRATA adds clause $C \vee l$ to a true QBF $\Pi\phi$. Let α be the outer assignment chosen by the constant Skolem functions, let β be the assignment to the universal variables, and let γ be the inner assignment chosen by the Skolem functions in response so that (α, β, γ) satisfies ϕ. We can modify γ to satisfy $\phi \wedge (C \vee l)$. If γ satisfies $C \vee l$ then no modification is needed. Otherwise, let γ' be identical to γ on all inner variables except l (so $\gamma(l) = 0$ and $\gamma'(l) = 1$). Consider all clauses D such that $\neg l \in D$, since $\phi \wedge \overline{D \setminus \{\neg l\}} \wedge \neg(C \vee l)$ is unsatisfiable but (α, β, γ) satisfies both ϕ and $\neg(C \vee l)$, we have that (α, β, γ) falsifies every conjunction of the form $\overline{D \setminus \{\neg l\}}$. In other words, (α, β, γ) satisfies ϕ even if we remove all occurrences of $\neg l$, therefore (α, β, γ') also satisfies ϕ and, since $\gamma'(l) = 1$, it satisfies $\phi \wedge (C \vee l)$. So if there are Skolem functions for $\Pi\phi$, there are Skolem functions for $\Pi\phi \wedge (C \vee l)$.

Now we look at the local purity rule. Suppose $\Pi\phi \wedge (C \vee u)$ is a satisfiable QBF, where u is a universal variable. It has a satisfying outer assignment α and inner Skolem functions. Now suppose given that outer assignment, and a universal assignment β that $(\phi \wedge C)|_{\alpha,\beta}$ is an unsatisfiable CNF. Then there is a resolution refutation π of $(\phi \wedge C)|_{\alpha,\beta}$. However since $(\phi \wedge (C \vee u))|_{\alpha,\beta}$ is satisfiable, π must contain $C|_{\alpha,\beta}$. We can remove any clause not connected by a common descendant to $C|_{\alpha,\beta}$, which will remove any clause that is not on a resolution path from C using the inner variables. If every clause D containing $\bar{u}$ is not on a resolution path from C then $(\phi \setminus \{ D \mid \bar{u} \in D \} \wedge C)|_{\alpha,\beta}$ is false. But consider β' as a modified β with u set to 0. $(\phi \setminus \{ D \mid \bar{u} \in D \} \wedge (C \vee u))|_{\alpha,\beta'}$ must be false. This contradicts $(\phi \wedge C)|_{\alpha}$ having valid Skolem functions. If $\Pi\phi \wedge C$ has valid Skolem functions, then we can use the same Skolem functions for $\Pi\phi \wedge (C \vee u)$.

In refutation mode, if we remove a clause, the Skolem function does not need to be be modified. Likewise in satisfaction mode, if we add a clause there must have been a valid Skolem function, if the new QBF has one.

Once we reach the empty clause, if we use the search or refutation rules then there cannot have been valid Skolem functions for the original QBF, otherwise we would have a set of Skolem functions that satisfies the empty clause. Likewise, once we reach the empty CNF there will be a trivial Skolem function for each variable and if we used the search or satisfaction rules we can work backwards to find the original Skolem functions. □

Theorem 2. *Given a satisfaction proof of an ASP P, we can extract in polynomial time an answer set of P.*

Proof. QRAT is known to have Skolem function extraction for satisfaction proofs of QBF (Heule, Seidl, and Biere 2014b). Since every ASP-QRAT rule is covered by a QRAT rule, we will get Skolem functions for the QBF Ψ_P. The Skolem functions for the atom variables are constant functions which tell us an answer set. □

Theorem 3 (Completeness)**.** *Let P be an ASP, if P is consistent then there is an ASP-QRAT proof in search mode that initializes from P and ends in a line containing no clauses. If P is inconsistent then there is an ASP-QRAT proof in search mode that initializes from P and ends in a line containing the empty clause.*

Proof. First note that the consistency of P is equivalent to the truth of the initialized QBF in ASP-QRAT. We proceed with the following steps to find a proof: Pick an inner variable x. Perform DP-Resolution, i.e., resolve all clauses with x with all clauses with $\bar{x}$ and add the resolvents with ATA unless they are tautological. Every clause containing $\bar{x}$ can now be removed with IRATE, then x only appears positively so every clause containing x can be removed with IRATE. This completely eliminates x. Repeat this for every inner variable.

Since no clause now contains an inner variable every $\forall$ literal is locally pure, we remove all universal literals.

We proceed with DP on the outer variables. If at any point we resolve to the empty clause we stop and the QBF is false. Otherwise we will remove all clauses containing the outer variables and this will end with the empty CNF. □

The completeness for satisfaction mode and refutational mode ASP-QRAT follows from the completeness of search mode ASP-QRAT.

4.3 An Important Difference to QRAT

The main difference with QRAT is that here we fix the quantifiers levels to 3. In general QRAT is able to fit new variables between the universal variables. It is unclear if this makes our new system weaker, but it will not affect our results. Another noticeable difference is that we do not provide a QRATU rule. This rule is not necessary as we show completeness without it. These differences mean that QRAT checkers will be more lenient than the proof system we have presented here, however since the QBF conversion is correct (Lemma 2), QRAT checkers will still be sound.

5 P-simulation of ASP-DRUPE

In this section, we show that ASP-QRAT p-simulates ASP-DRUPE for disjunctive ASPs. This means that any solver that outputs an ASP-DRUPE proof can convert that proof further to an ASP-QRAT proof and only receive a polynomial increase in size.

The following lemma is a basic use of ASP-QRAT's rules.

Lemma 3. *The following sets of definition clauses D for variable e can be added or removed in ASP-QRAT, as long as e does not appear in formula ϕ.*

- *If e is an outer variable and defined only on outer variables, we can use ORATA to add all clauses of D to ϕ, or ORATE to remove all clauses of D from $\phi \wedge D$.*
- *If e is an inner variable, we can use IRATA to add all clauses of D to ϕ, or IRATE to remove all clauses of D from $\phi \wedge D$.*

Proof. The resolvent of any two definition clauses for e is always a tautology. If e is defined only in the outer variables the tautology happens over two outer literals. □

Example 3. *Consider CNF ϕ, where we want to add definition clauses for a new outer variable n that does not appear in ϕ. We can add clause $C = (\bar{n} \vee \bar{x} \vee \bar{y})$ via ORATA, where x and y are outer variables, the side condition is vacuously satisfied (no clause of ϕ contains n, so no condition to check). Adding clauses $C_1 = (n \vee x)$ and $C_2 = (n \vee y)$ after that is also possible via ORATA. We only have to check C_1 against C: $\phi \wedge \neg(C \backslash \{\bar{n}\}) \wedge \neg C_1 = \phi \wedge x \wedge y \wedge \bar{x} \wedge \bar{n} \vdash_1 \bot$. And similarly for C_2.*

The next lemma proves the main part of the simulation argument.

Lemma 4. *Given a subset of basic loop clauses used in an ASP-DRUPE proof, we can derive these basic loop clauses in a polynomial-size ASP-QRAT derivation (polynomial in the size of the ASP-DRUPE proof).*

Proof. We follow the idea from (Kiesl and Seidl 2019). Given a basic loop clause $\lambda(a, \omega)$, we represent the loop ω with a corresponding assignment α to the loop slot variables.

We will take the clause $\neg a \vee \neg u_a \vee \neg\mathsf{valid\_loop} \vee \bigvee_{b \in \mathrm{Bod}(P)} \mathsf{extern}(b)$ along with the definition clauses that help define the inner literals, $\mathsf{valid\_loop}$, $\mathsf{extern}(b)$ given here and their definition clauses and so on. We will create a modified copy of each of these clauses specific to α.

For every inner variable v, we want to replace it with the new variable v^α. To do this we introduce IRATA clauses $(\bar{\alpha} \vee \bar{v} \vee v^\alpha)$ and $(\bar{\alpha} \vee v \vee \bar{v}^\alpha)$. $\bar{\alpha}$ is a clause because α can be seen as a conjunction of literals.

To do the replacement, we take $\neg a \vee \neg u_a \vee \neg\mathsf{valid\_loop} \vee \bigvee_{b \in \mathrm{Bod}(P)} \mathsf{extern}(b)$ and, for each inner literal l that it contains, we resolve it with $(\bar{\alpha} \vee \bar{l} \vee l^\alpha)$. l itself is a definition variable so in its definition clauses we do the same to all inner literals. Now, l^α has a conditional definition under α.

We repeat this for every loop clause. It is essential that we do this first for every loop clause for w now, while $(\bar{\alpha} \vee \bar{v} \vee v^\alpha)$ and $(\bar{\alpha} \vee v \vee \bar{v}^\alpha)$ are available before we start deleting clauses. After all inner variables are replaced, we remove all clauses with universal tautologies. We now delete all clauses that mix inner literals annotated with α with other inner literals. This severs the resolution paths from the clauses that contain subclause $\bar{\alpha}$ and full replacements of the inner variables, to any clauses with the opposite universal literals, because all inner variables annotated with α are *only* connected resolution path-wise with *other literals annotated with* α, and all these clauses contain $\bar{\alpha}$, with no tautologies. In other words among these paths the literals of $\bar{\alpha}$ are pure.

We can then remove all literals of $\bar{\alpha}$ from each of these "annotated" clauses by local purity. And this process can be repeated for each loop w. This is essentially how QRAT p-simulates ∀Exp (Kiesl and Seidl 2019).

In our case we are not finished. We need to simplify clause $\neg a \vee \neg\mathsf{valid\_loop}^\alpha \vee \bigvee_{b \in \mathrm{Bod}(P)} \mathsf{extern}(b)^\alpha$ to $\neg a \vee \bigvee_{b \in \mathrm{EB}(P,w)} b$. So we need a proof of $\mathsf{valid\_loop}^\alpha$, $\neg\mathsf{extern}(b)^\alpha$ for every induced body b that is not an external body of w and a proof of $\neg\mathsf{extern}(b)^\alpha \vee b$ for every body b that is an external body. The Tseitin variables of $\mathsf{extern}(b)$ depend on both inner and universal variables, in fact $\mathsf{extern}(b) = \mathsf{is\_extern}(b) \wedge b$, where $\mathsf{is\_extern}(b)$ depends only on universal variables.

If we look at the α-annotated Tseitin variables that are defined only on universal variables, since $\bar{\alpha}$ is reduced via local purity we get them as unit clauses, or via unit propagation in the case of nested definitions (Observation 1). $\mathsf{valid\_loop}^\alpha$ is derived via unit propagation in this way. Likewise $\mathsf{is\_extern}(b)^\alpha$ or its negation are derived from unit propagation depending on whether b is external to w. □

Theorem 4. *ASP-QRAT in refutation mode p-simulates ASP-DRUPE.*

Proof. We go through the five rules of ASP-DRUPE. For *Completion-Axiom*, we already have all Δ clauses at the start of the ASP-QRAT proof. For *Loop-Axiom*, we can generate all introduced basic loop clauses using Lemma 4. For *RUP*, we can simulate it using ATA. For *Ext*, we use IRATA to add extension clauses using Lemma 3. Finally, for *Deletion*, in refutation mode we can remove clauses for free. □

We conjecture that ASP-QRAT (in refutation mode) is exponentially stronger than ASP-DRUPE. For 3 level QBFs QRAT is strictly stronger than ∀Exp+Ext Res. The QBF examples are the Clique-CoClique formulas, which make use of the law of non-contradiction over a formula describing a graph. Much like our translation Ψ_P, Clique-CoClique's inner variables are all definitions. ∀Exp+Ext Res requires an exponential number of clauses from the ∀-expansion of Clique-CoClique to prove a contradiction, hence the lower bound (Chew 2018). There are short proofs of these formulas in eFrege + ∀red (Beyersdorff et al. 2018) which is p-simulated by QRAT(Heule, Seidl, and Biere 2014a).

It is unclear how to directly use a semantic QBF lower bound like this in our ASP setting. Instead of exponentially many expanded clauses we would want exponentially many basic loop clauses. One could perhaps find a way to reverse engineer a similar lower bound into an ASP.

6 Proof Complexity of ASP-QRAT

In this section, we show the conditional optimality of ASP-QRAT. By *conditional optimality*, we mean that showing a super-polynomial lower bound to the shortest proof size of ASP-QRAT gives an answer to an open problem.

The most important open problem is whether the propositional proof system extended resolution has a super polynomial lower bound. The way we use this is to link short extended resolution proofs to short ASP-QRAT proofs.

Note that IRATA works as a rule on propositional formulas, which is called RATA in the DRAT propositional refutational system (Wetzler, Heule, and Hunt 2014). It is known that extended resolution and DRAT are equivalent (Kiesl, Rebola-Pardo, and Heule 2018), but we use a precise Lemma to link refutation to derivation.

Lemma 5. *Given an propositional formula $\phi \wedge \bar{C}$ with an extended resolution refutation π, there is a series of ATA and RATA and deletion steps that derive $\phi \wedge C$ from ϕ in a proof of size polynomial in π.*

Proof. Replacing resolution steps with ATA steps and extension steps with RATA steps, we follow the derivation of the empty clause from $\phi \wedge \neg C$. However we weaken all intermediate clauses (using ATA) to additionally include all literals of C. This will create tautologous clauses including clauses from $\neg C$, we can remove these without affecting the derivation. Suppose $D_1 \vee \bar{x}$ resolves with $D_2 \vee x$. If $C \vee D_2 \vee x$ is tautologous then either $C \vee D_1 \vee D_2$ is tautologous or $\bar{x} \in C \vee D_2$. In both cases $C \vee D_1 \vee D_2$ can be added via ATA from $C \vee D_1 \vee \bar{x}$ alone. □

When using ASP-QRAT we can use this lemma, but we again use IRATA instead of RATA.

Theorem 5. *If there exists a family of consistent ASPs which form a super-polynomial-size lower bound for ASP-QRAT in satisfaction mode, then extended resolution has a super-polynomial-size lower bound.*

Proof. We describe how to construct an ASP-QRAT satisfaction proof from an answer set α and a series of Ext Res proofs. First we will detail which extended resolution proofs are used. Let α be an answer set in the form of a conjunction of unit clauses (including negative literals for non-inclusion). Δ_P^{Bdef} is the part of the completion that contains only definition variables for the induced bodies. Consider the set of definition clauses Θ_P of the inner variables and let $C_a := \neg u_a \vee \neg \mathsf{valid\_loop} \vee \bigvee_{b \in \text{Bod}(P)} \mathsf{extern}(b)$ for every atom a. Each C_a subsumes a compact loop clause.

$\Delta_P^{\text{Bdef}} \wedge \alpha \wedge \Theta_P \wedge \neg C_a$ is a contradiction when $\alpha(a) = 1$, otherwise a would be minimized by some loop found by the universal variables, contradicting the fact that α is an answer set. So by completeness there will be an extended resolution refutation of $\Delta_P^{\text{Bdef}} \wedge \alpha \wedge \Theta_P \wedge \neg C_a$. If we further assume that extended resolution does not have a super-polynomial lower bound, these proofs will be polynomially bounded.

Now we explain how to generate satisfaction ASP-QRAT proofs from α and the short extended resolution proofs.

1. First we add singleton clauses, using the arbitrary clause addition that specify the answer set α as we did above.
2. Next we follow each extended resolution proof as a series of IRATA and ATA additions to derive each C_a for a positive in α using Lemma 5. Remember that these additions only need Δ_P^{Bdef}, α and Θ_P.
3. Once we subsume the loop clause for a we delete it with ATE. For a negative in α, subsumption is immediate.
4. We reverse the clause additions for 2. using IRATE and ATE. The deleted loop clause was never part of the forward derivation so in reverse it is not needed.
5. After these reversals we remove all clauses relating to the inner variables, since they are only definitions (and thus removable by IRATE). Only singleton clauses specifying the answer set and the Clark's completion clauses remain.
6. By following unit propagation we get singleton clauses for each of the induced bodies and we add them. Every clause in the Clark's completion is satisfied by the assignment from singleton clauses by assumption, this means every one is subsumed and can be removed with ATE.
7. Finally we are left with a consistent assignment of singleton clauses. We can remove each by ORATE. □

Theorem 5 is not implied by previous results on QRAT's proof complexity. Here we take advantage of the fact that the third level contains purely definition variables and so we only have two active quantifier levels. The first level is $\exists$ and is given by the answer set and the second is $\forall$ and where we use the propositional proofs. For refutation mode we also get conditional optimality.

Theorem 6. *If there exists a family of inconsistent ASPs which form a super-polynomial-size lower bound for ASP-QRAT in refutation mode, then extended resolution has a super-polynomial-size lower bound or* $\mathsf{NP} \not\subseteq \mathsf{P/poly}$.

We re-use the argument used to show the proof system $\mathsf{eFrege} + \forall\mathsf{red}$ has conditional optimality (Beyersdorff et al. 2020). One minor difference here is our self-imposed restriction to creating extension variables in-between universal variables. However we have at our disposal a local purity rule stronger than the reduction rule (Beyersdorff et al. 2020).

Proof. Assume that $\mathsf{NP} \subseteq \mathsf{P/poly}$ and that all families of propositional contradictions have polynomially bounded extended resolution proofs. Let P be an inconsistent ASP and consider the QBF Ψ_P. Let $U = \{u_1, u_2, \dots\}$ be the set of universal variables of Ψ_P. Since Ψ_P is false, then each $u_i \in U$ has Herbrand function h_i. As previously stated in regard to Observation 2, finding a falsifying loop is an NP problem, and since $\mathsf{NP} \subseteq \mathsf{P/poly}$ there are small circuits that calculate h_i. Each circuit is encoded in CNF using definition clauses for each of its gates and, as a result, we obtain an extension variable σ_i for the output gate of the circuit that calculates h_i. We create extension variables d_i, for $1 \leq i \leq |U|$. $d_1 \Leftrightarrow u_1 \oplus \sigma_1$. $d_{i+1} \Leftrightarrow d_i \vee (u_{i+1} \oplus \sigma_{i+1})$. Firstly, note that when considering these in ASP-QRAT, these will have to be inner variables. Secondly, note that these are extension variables that are not just repeated conjunctions or repeated disjunctions. We can define d_1 using the four clauses $(d_1 \vee u_1 \vee \bar{\sigma}_1)$, $(d_1 \vee \bar{u}_1 \vee \sigma_1)$, $(\bar{d}_1 \vee \bar{u}_1 \vee \bar{\sigma}_1)$, $(\bar{d}_1 \vee u_1 \vee \sigma_1)$. We can define d_{i+1} using the five clauses $(\bar{d}_i \vee d_{i+1})$, $(d_{i+1} \vee u_{i+1} \vee \bar{\sigma}_{i+1})$, $(d_{i+1} \vee \bar{u}_{i+1} \vee \sigma_{i+1})$, $(\bar{d}_{i+1} \vee d_i \vee \bar{u}_{i+1} \vee \bar{\sigma}_{i+1})$ and $(\bar{d}_{i+1} \vee d_i \vee u_{i+1} \vee \sigma_{i+1})$. These clauses, can be added with IRATA, nonetheless.

Let Σ be the set of definition clauses for the extension variables σ_u, let D be the set of definition clause for all d variables and let ϕ be the propositional matrix of Ψ_P. $\phi \wedge \Sigma \wedge D \wedge (\bar{d}_{|U|})$ is a propositional contradiction and thus has a short extended resolution proof.

In our ASP-QRAT proof we can use ORATA to add all Σ clauses to ϕ, next we use IRATA to add all D clauses. Next, using Lemma 5 we can add clause $(d_{|U|})$ to $\phi \wedge \Sigma \wedge D$ in a short process of ATA and IRATA. Now we can delete all clauses of ϕ. We also delete the "positive" definition clauses

for each d_i. For $i = 1$ these are $(d_1 \vee \bar{u}_1 \vee \sigma_1)$ and $(d_1 \vee u_1 \vee \bar{\sigma}_1)$ and for $i > 1$ these are $(\bar{d}_i \vee d_{i+1})$, $(d_{i+1} \vee u_{i+1} \vee \bar{\sigma}_{i+1})$, $(d_{i+1} \vee \bar{u}_{i+1} \vee \sigma_{i+1})$. These should be deleted with only the "negative" definition clauses remaining. This means $\bar{d}_i$ only appears in clauses with d_j for $j < i$.

We will use the derived clause (d_{i+1}) to derive (d_i) and eventually reach the empty clause after $i = 1$. We start with $i + 1 = |U|$. We resolve (d_{i+1}) with the negative definition clauses $(\bar{d}_{i+1} \vee d_i \vee \bar{u}_{i+1} \vee \bar{\sigma}_{i+1})$, $(\bar{d}_{i+1} \vee d_i \vee u_{i+1} \vee \sigma_{i+1})$ to get $(d_i \vee \bar{u}_{i+1} \vee \bar{\sigma}_{i+1})$ and $(d_i \vee u_{i+1} \vee \sigma_{i+1})$. We now delete the three antecedents used here. Both $(d_i \vee \bar{u}_{i+1} \vee \bar{\sigma}_{i+1})$ and $(d_i \vee u_{i+1} \vee \sigma_{i+1})$ have local purity for the literal $\bar{u}_{i+1}$ and u_{i+1}, respectively. Indeed, the only resolution paths are through the positive d_i literals, but these can only continue through more positive d_j literals for $j < i$ and never can reverse direction. After we reduce the universal literals we get $(d_i \vee \bar{\sigma}_{i+1})$ and $(d_i \vee \sigma_{i+1})$, which we resolve together to get (d_i). Again we can delete the antecedents.

Once we reach (d_1) we resolve with the negative definitions to get $(u_1 \vee \bar{\sigma}_1)$ and $(\bar{u}_1 \vee \sigma_1)$. These are again locally pure, so we reduce the universal literals then resolve (σ_1) and $(\bar{\sigma}_1)$ to get the empty clause. Thus we have a polynomially bounded ASP-QRAT refutation of P. □

7 Conclusion

With some help from QBF proof complexity, we proposed a dual, sound, complete, and conditionally optimal proof system for ASP. It will be interesting to see how our proof system performs when implemented in an ASP solver. Another interesting avenue for future research is to extend our system to certify ASPs with quantifiers (Amendola et al. 2022).

Acknowledgements

Thanks to Johannes Fichte for the helpful discussions. The authors acknowledge the support from FWF the Austrian Science Funds, projects 10.55776/P36420, 10.55776/P36688, 10.55776/COE12, 10.55776/ESP197 , and 10.55776/ESP235.

References

Alviano, M.; Calimeri, F.; Dodaro, C.; Fuscà, D.; Leone, N.; Perri, S.; Ricca, F.; Veltri, P.; and Zangari, J. 2017. The ASP system DLV2. In Balduccini, M., and Janhunen, T., eds., *Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings*, volume 10377 of *Lecture Notes in Computer Science*, 215–221. Springer.

Alviano, M.; Dodaro, C.; Fichte, J. K.; Hecher, M.; Philipp, T.; and Rath, J. 2019. Inconsistency proofs for ASP: the ASP - DRUPE format. *Theory Pract. Log. Program.* 19(5-6):891–907.

Alviano, M.; Dodaro, C.; and Ricca, F. 2014. Preliminary report on WASP 2.0. *CoRR* abs/1404.6999.

Amendola, G.; Cuteri, B.; Ricca, F.; and Truszczynski, M. 2022. Solving problems in the polynomial hierarchy with ASP(Q). In Gottlob, G.; Inclezan, D.; and Maratea, M., eds., *Logic Programming and Nonmonotonic Reasoning - 16th International Conference, LPNMR 2022, Genova, Italy, September 5-9, 2022, Proceedings*, volume 13416 of *Lecture Notes in Computer Science*, 373–386. Springer.

Amendola, G.; Dodaro, C.; and Ricca, F. 2016. ASPQ: an asp-based 2qbf solver. In Lonsing, F., and Seidl, M., eds., *Proceedings of the 4th International Workshop on Quantified Boolean Formulas (QBF 2016) co-located with 19th International Conference on Theory and Applications of Satisfiability Testing (SAT 2016), Bordeaux, France, July 4, 2016*, volume 1719 of *CEUR Workshop Proceedings*, 49–54. CEUR-WS.org.

Amendola, G.; Ricca, F.; and Truszczynski, M. 2019. Beyond NP: quantifying over answer sets. *CoRR* abs/1907.09559.

Baral, C.; Provetti, A.; and Son, T. C. 2003. Introduction to the special issue on programming with answer sets. *Theory Pract. Log. Program.* 3(4-5):387–391.

Beyersdorff, O.; Chew, L.; Mahajan, M.; and Shukla, A. 2018. Understanding cutting planes for QBFs. *Information and Computation* 262:141 – 161.

Beyersdorff, O.; Bonacina, I.; Chew, L.; and Pich, J. 2020. Frege systems for quantified Boolean logic. *J. ACM* 67(2).

Beyersdorff, O.; Chew, L.; and Janota, M. 2016. Extension variables in QBF resolution. In *Beyond NP, Papers from the 2016 AAAI Workshop*.

Beyersdorff, O. 2013. The complexity of theorem proving in autoepistemic logic. In *Proc. 16th International Conference on Theory and Applications of Satisfiability Testing*, volume 7962 of *Lecture Notes in Computer Science*, 365–376.

Blinkhorn, J. 2020. Simulating DQBF preprocessing techniques with resolution asymmetric tautologies. *Electron. Colloquium Comput. Complex.* TR20-112.

Brewka, G.; Eiter, T.; and Truszczynski, M. 2011. Answer set programming at a glance. *Commun. ACM* 54(12):92–103.

Cadoli, M., and Lenzerini, M. 1994. The complexity of propositional closed world reasoning and circumscription. *J. Comput. Syst. Sci.* 48(2):255–310.

Chede, S.; Chew, L.; and Shukla, A. 2024. Circuits, proofs and propositional model counting. *Electron. Colloquium Comput. Complex.* TR24-081.

Chew, L. 2018. Hardness and optimality in QBF proof systems modulo NP. *Electron. Colloquium Comput. Complex.* TR18-178.

Clark, K. L. 1978. Negation as failure. In Gallaire, H., and Minker, J., eds., *Logic and Data Bases, Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, France, 1977*, Advances in Data Base Theory, 293–322. New York: Plemum Press.

Cook, S. A., and Reckhow, R. A. 1979. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic* 44(1):36–50.

Eiter, T., and Gottlob, G. 1995. On the computational cost of disjunctive logic programming: Propositional case. *Ann. Math. Artif. Intell.* 15(3-4):289–323.

Faber, W.; Mazzotta, G.; and Ricca, F. 2023. An efficient solver for ASP(Q). *Theory Pract. Log. Program.* 23(4):948–964.

Gebser, M., and Schaub, T. 2006. Tableau calculi for answer set programming. In Etalle, S., and Truszczyński, M., eds., *Logic Programming*, 11–25. Berlin, Heidelberg: Springer Berlin Heidelberg.

Gebser, M.; Kaminski, R.; König, A.; and Schaub, T. 2011. Advances in *gringo* series 3. In Delgrande, J. P., and Faber, W., eds., *Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings*, volume 6645 of *Lecture Notes in Computer Science*, 345–351. Springer.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2012. Answer set solving in practice. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6:1–238.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2014. Clingo = ASP + control: Preliminary report. *CoRR* abs/1405.3694.

Gelfond, M., and Lifschitz, V. 1993. Representing action and change by logic programs. *J. Log. Program.* 17(2/3&4):301–321.

Heule, M.; Seidl, M.; and Biere, A. 2014a. A unified proof system for QBF preprocessing. In *7th International Joint Conference on Automated Reasoning (IJCAR)*, 91–106.

Heule, M. J. H.; Seidl, M.; and Biere, A. 2014b. Efficient extraction of Skolem functions from QRAT proofs. In *2014 Formal Methods in Computer-Aided Design (FMCAD)*, 107–114.

Janota, M. 2017. Qfun: Towards machine learning in qbf.

Järvisalo, M., and Oikarinen, E. 2007. Extended ASP tableaux and rule redundancy in normal logic programs. In Dahl, V., and Niemelä, I., eds., *Logic Programming*, 134–148. Berlin, Heidelberg: Springer Berlin Heidelberg.

Jussila, T.; Biere, A.; Sinz, C.; Kröning, D.; and Wintersteiger, C. M. 2007. A first step towards a unified proof checker for QBF. In *SAT 2007*, 201–214.

Kiesl, B., and Seidl, M. 2019. QRAT polynomially simulates ∀Exp+Res. In *SAT 2019*, volume 11628 of *Lecture Notes in Computer Science*, 193–202. Springer.

Kiesl, B.; Rebola-Pardo, A.; and Heule, M. J. 2018. Extended resolution simulates DRAT. In *Automated Reasoning: 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, 516–531. Springer.

Lin, F., and Zhao, Y. 2004. Assat: computing answer sets of a logic program by sat solvers. *Artificial Intelligence* 157(1):115–137. Nonmonotonic Reasoning.

Lonsing, F., and Egly, U. 2018. QRAT+: Generalizing QRAT by a more powerful QBF redundancy property. In Galmiche, D.; Schulz, S.; and Sebastiani, R., eds., *Automated Reasoning*, 161–177. Cham: Springer International Publishing.

Marek, V. W., and Truszczynski, M. 1991. Computing intersection of autoepistemic expansions. In Nerode, A.; Marek, V. W.; and Subrahmanian, V. S., eds., *Logic Programming and Non-monotonic Reasoning, Proceedings of the First International Workshop, Washington, D.C., USA, July 1991*, 37–50. The MIT Press.

Pontelli, E.; Son, T. C.; Baral, C.; and Gelfond, G. 2012. Answer set programming and planning with knowledge and world-altering actions in multiple agent domains. In Erdem, E.; Lee, J.; Lierler, Y.; and Pearce, D., eds., *Correct Reasoning - Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, volume 7265 of *Lecture Notes in Computer Science*, 509–526. Springer.

Rabe, M. N., and Tentrup, L. 2015. CAQE: A certifying QBF solver. In *FMCAD 2015*, 136–143. FMCAD Inc.

Robinson, J. A. 1963. Theorem-proving on the computer. *Journal of the ACM* 10(2):163–174.

Slivovsky, F., and Szeider, S. 2014. Variable dependencies and q-resolution. In Sinz, C., and Egly, U., eds., *Theory and Applications of Satisfiability Testing – SAT 2014*, 269–284. Cham: Springer International Publishing.

Slivovsky, F., and Szeider, S. 2016. Quantifier reordering for QBF. *J. Autom. Reason.* 56(4):459–477.

Wetzler, N.; Heule, M.; and Hunt, W. A. 2014. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *SAT 2014*, volume 8561 of *Lecture Notes in Computer Science*, 422–429. Springer.