frontiers | Frontiers in Artificial Intelligence

TYPE Original Research
PUBLISHED 10 December 2025
DOI 10.3389/frai.2025.1614894

OPEN ACCESS

EDITED BY
Francesco Giannini,
Scuola Normale Superiore Classe di Scienze, Italy

REVIEWED BY
Omar A. Alzubi,
Al-Balqa Applied University, Jordan
Giuseppe Mazzotta,
University of Calabria, Italy
Yiming Tang,
Hefei University of Technology, China
Pan Hu,
Shanghai Jiao Tong University, China

*CORRESPONDENCE
Yisong Wang
yswang@gzu.edu.cn

RECEIVED 20 April 2025
ACCEPTED 13 November 2025
PUBLISHED 10 December 2025

CITATION
Xie Z, Wang Y, Yang L and Feng R (2025) Minimal reduct for propositional circumscription. *Front. Artif. Intell.* 8:1614894. doi: 10.3389/frai.2025.1614894

Minimal reduct for propositional circumscription

Zhongtao Xie[1], Yisong Wang[1]*, Lei Yang[2] and Renyan Feng[3]

[1]State Key Laboratory of Public Big Data, Key Laboratory of Advanced Medical Imaging and Intelligent Computing of Guizhou Province, College of Computer Science and Technology, Guizhou University, Guiyang, China, [2]College of Mathematical Sciences, Minzu Normal University of Xingyi, Guiyang, China, [3]School of Information, GuiZhou University of Finance and Economics, Guiyang, China

Circumscription is an important logic framework for representing and reasoning common-sense knowledge. With efficient implementations for circumscription, including `circ2dlp` and `aspino`, it has been widely used in model-based diagnosis and other domains. We propose a notion of minimal reduct for propositional circumscription and prove a characterization theorem, *i.e.*, that the models of a circumscription can be obtained from the minimal reduct of the circumscription. With the help of the minimal reduct, a new method circ-reduct for computing models of circumscription is presented. It iteratively computes smaller models under set inclusion (if possible), and the minimal reduct is used to simplify the circumscription in each iteration. The algorithm is proved to be correct. Extensive experiments are conducted on circuit diagnosis ISCAS85, random CNF instances, and some industrial SAT instances for the international SAT competition. These results demonstrate that the minimal reduct is effective in computing circumscription models. Compared to the widely used circumscription solver `circ2dlp` using the state-of-the-art answer set programming solver `clingo`, our algorithm `circ-reduct` achieves significantly shorter CPU time. Compared with `aspino` using glucose as the internal SAT solver and unsatisfiable core analysis technique, our algorithm achieves better CPU time for random and industrial CNF benchmarks, while it is comparable for circuit diagnosis benchmarks.

KEYWORDS

minimal reduct, propositional circumscription, satisfiability, answer set program, model-based diagnosis

1 Introduction

Recent breakthroughs in large language models (LLMs) and foundation models have spurred an intense interest in integrating symbolic reasoning with data-driven learning to improve interpretability and controllable inference. Within this trend, classical logic techniques are being revisited as essential tools to provide strong guarantees and structured reasoning capabilities that purely statistical methods often lack (Ryu et al., 2025; Ye et al., 2023; Pan et al., 2023; Olausson et al., 2023; Callewaert et al., 2025).

Beyond classical approaches, non-monotonic reasoning has attracted renewed attention because it naturally captures default assumptions and common-sense knowledge that must be revised when new information arrives (Reiter, 1988). Recent ASP-based reasoning optimizations illustrate this synergy (Yang et al., 2023; Zeng et al., 2024; Rajasekharan et al., 2023). In addition, recent data-driven clustering methods continue to inspire logical inference. For example, automated cluster elimination guided by high-density points (Hu et al., 2025) and constrained clustering with weak label prior (Zhang et al., 2023) illustrate how structural knowledge can guide efficient search.

A prominent formalism for non-monotonic reasoning is circumscription, originally introduced by McCarthy for common-sense reasoning. Circumscription minimizes the extension of specific predicates, embodying a closed-world assumption where statements not known to be true are considered false (McCarthy, 1980). To increase the knowledge representation capabilities of ordinary circumscription, Lifschitz proposed parallel circumscription incorporating atoms that are allowed to vary (Lifschitz, 1985). Circumscription has received considerable attention in areas such as knowledge formalization (McCarthy, 1986; Lifschitz, 1986), common-sense reasoning (Wang et al., 2015; Alviano, 2019), diagnosis (Wotawa and Kaufmann, 2022; Metodi et al., 2014; Stern et al., 2012; Friedrich et al., 1999), planning (Sierra-Santibáñez, 2000), and privacy protection applications (Ogunniye and Kökciyan, 2023).

Despite its theoretical and practical significance, circumscription presents substantial computational challenges. It was shown that determining whether a circumscription has a model is NP-complete, verifying whether an interpretation is a model of a given circumscription is coNP-complete, and determining whether a formula is a logical consequence of a circumscription is Σ_2^p-complete (Eiter and Gottlob, 1993; Cadoli and Lenzerini, 1994). Consequently, efficiently computing models for propositional circumscription remains a formidable task. To the best of our knowledge, the main approach for computing circumscription models involves translating circumscriptions into (general) disjunctive logic programs under answer set semantics (ASP in short) (Gelfond and Lifschitz, 1988) such that the answer sets of the logic program correspond to the models of circumscription, allowing the use of efficient ASP solvers.

Sakama and Inoue (1995) proposed a method for translating circumscription into general disjunctive logic programs; however, this method requires the calculation of characteristic clauses, which can lead to exponential explosion. Oikarinen and Janhunen (2005) introduced the `circ2dlp` method to translate propositional circumscription into disjunctive logic programs. This method is linear and faithful, but non-modular (Janhunen and Oikarinen, 2004). Zhang et al. (2011) proposed and implemented the T2LP to translate finite first-order circumscription without varying predicates into first-order ASP. Although varying predicates can be eliminated in circumscription (Cadoli et al., 1992), the worst case can still result in exponential explosion. Wan et al. (2014) proposed the `cfo2lp` to translate any first-order formula in negation normal form without implication symbols into first-order ASP. All the above translations for propositional circumscriptions either introduce fresh (predicate) symbols, or may result in exponential explosion. The ones for computing first-order circumscription primarily focus on translating first-order circumscription (or second-order theories) into first-order theories under certain constraints (Przymusinski, 1989; Doherty et al., 1997), for which there is no efficient implementation to the best of our knowledge.

Another alternative approach involves translating circumscription into the propositional satisfiability problem (SAT). Lee and Lin proposed a method that employs loop formulas and completions to translate circumscriptions into propositional theories. However, this method requires finding loop formulas, and the number of loop formulas may be exponential (Lee and Lin, 2006). Notably, enumerating models of circumscriptions is also interesting. For this purpose, Alviano proposed an unsatisfiable core analysis and implemented a solver `aspino` making use of the SAT solver `glucose` (Alviano, 2017), but this approach only employs solver which support cardinality constraints.

Recently, Wang et al. proposed the concept of minimal reduct for answer set programs. It substantially improves the minimal model decomposition of propositional theories (Angiulli et al., 2022), in addition to a new characterization for answer sets of logic programs (Zhang et al., 2021; Wang et al., 2023). Informally, the minimal reduct of a logic program P *w.r.t.* an interpretation M is obtained by replacing all atoms that are false under M in P with *false*.

In this study, we extend the notion of minimal reduct from answer set programming to (parallel) propositional circumscriptions and show that the minimal reduct preserves models of circumscription. With the help of minimal reduct and efficient SAT solvers, two approaches for computing and enumerating models of circumscriptions are presented, namely `circ/circ-reduct` and `circ-enum`. Extensive experimental results show that `circ/circ-reduct` is comparable with the state-of-the-art circumscription solvers `circ2dlp` and `aspino`.

The main contributions of this study are as follows:

- We introduce the concept of minimal reduct for propositional circumscription and establish its theoretical soundness by proving that it preserves satisfiability. Intuitively, the minimal reduct allows a circumscription formula to be simplified with respect to a given interpretation while keeping its essential models unchanged. This provides a new foundation for developing efficient SAT-based algorithms for computing circumscription models;
- We propose two sound algorithms for computing and enumerating models of propositional circumscriptions: `circ`, `circ-reduct` and `circ-enum`, respectively;
- We implement the aforementioned algorithms based on an open-source SAT solver (Cai et al., 2022) and conduct extensive experiments on diagnosis, random and industrial CNF formulas SAT competitions. We show that the two methods are comparable with the state-of-the-art methods `circ2dlp` and `aspino`.

The structure of the study is as follows: In Section 2, we review the necessary background knowledge; In Section 3, we propose minimal reduct for circumscription; In Section 4, we discuss the enumeration algorithm `circ-enum`; Section 5 details our experiments, demonstrating that the `circ` and `circ-reduct` methods can effectively compute propositional circumscription models; finally, we conclude the study and outline future research directions in Section 7.

2 Circumscription

In the section, we briefly recall the basic notions and notations of circumscription (McCarthy, 1980; Lifschitz, 1985; McCarthy, 1986):

Assume that the propositional symbols of the propositional language $\mathcal{L}$ form a finite set of atoms $\mathcal{A}$. A literal l is either an atom p or the negation of an atom $\neg p$, while a clause α is a disjunction of literals:

$$l_1 \vee \cdots \vee l_n \quad (n \geq 0) \tag{1}$$

When $n = 0$, $\alpha \equiv \bot$. For convenience, the clause α can also be written as the set of literals $\{l_1, \cdots, l_n\}$. We denote $\alpha^+ = \mathcal{A} \cap \alpha$ and $\alpha^- = \{p \mid \neg p \in \alpha\}$. A clause theory A is a set of finite clauses.

If $S \subseteq \mathcal{A}$, we denote $\overline{S} = \mathcal{A} \setminus S$, $\neg S = \{\neg p \mid p \in S\}$, $\bigvee S = \bigvee_{p \in S} p$, and $\bigwedge S = \bigwedge_{p \in S} p$. The notions of formula, theory, interpretation, model, satisfaction ($\models$), non-satisfaction ($\not\models$), equivalence ($\equiv$), etc., in the propositional language $\mathcal{L}$ are the same as in classical propositional logic. We denote $uar(e)$ as the set of all atomic symbols appearing in the formula (or theory) e.

Abusing the notation, a tuple $(t_1, \ldots, t_k)$ is usually written as $\{t_1, \ldots, t_k\}$ when there is no confusion from its context. The expression $A(P, Q)$ denotes a formula/theory that contains atoms from $P \cup Q$, where P, Q are disjoint sets of atoms; we denote $A(P)$ as $A(P, Q)$ when $Q = \emptyset$. When X, Y are tuples of the same length as P, Q respectively, $A(X, Y)$ denotes the formula (or theory) obtained from A by simultaneously replacing the atoms from P, Q with the corresponding atoms from X, Y.

Example 1. Let $A(\{p, q\}, \{r\}) = (p \vee \neg q) \wedge (r \vee \neg p)$. Then $A(\{x, y\}, \{z\}) = (x \vee \neg y) \wedge (z \vee \neg x)$;

Let $P = \{p_1, \ldots, p_n\}$ and $Q = \{q_1, \ldots, q_n\}$ be two sets of atoms.

- $P \leq Q$ denotes $\bigwedge_{1 \leq i \leq n}(p_i \rightarrow q_i)$;
- $P = Q$ denotes $\bigwedge_{1 \leq i \leq n}(p_i \leftrightarrow q_i)$;
- $P < Q$ denotes $(P \leq Q) \wedge \neg(P = Q)$.

The basic idea of circumscription is to minimize the set of atoms assigned true as much as possible while keeping certain atoms fixed, thereby obtaining a "minimal" model.

Definition 1. Let A be a formula, and let P, Z be disjoint sets of atoms. The *circumscription* of P in formula A with Z allowed to vary, denoted as $CIRC[A; P; Z]$, is the following formula:

$$A(P, Z) \wedge \neg \exists XY(A(X, Y) \wedge (X < P)) \tag{2}$$

where X, Y are fresh disjoint tuples of atoms with the same length as P, Z respectively. When $Z = \emptyset$, $CIRC[A; P; Z]$ is shortened as $CIRC[A; P]$.

In Definition 1, the P is called the *minimizing set*, the Z is called the *varying set*, and the set of remaining atoms is called the *fixing set*. Intuitively, the circumscription $CIRC[A; P; Z]$ is to minimize (under set inclusion) the interpretation of P, while fixing the interpretation for $\overline{P \cup Z}$. The *varying set* Z consists of atoms whose truth values may change freely during this minimization, and as long as the original formula A remains satisfied, their assignment can be arbitrary.

Let P and Z be disjoint sets of atoms, M, M' be two interpretations (sets of atoms). We use $M \leq^{P;Z} M'$ to denote

- $M \cap P \subseteq M' \cap P$, and
- $M \setminus (P \cup Z) = M' \setminus (P \cup Z)$.

If $M \leq^{P;Z} M'$ holds but $M' \leq^{P;Z} M$ does not, we denote this as $M <^{P;Z} M'$. When $Z = \emptyset$, $M \leq^{P;Z} M'$ is shortened to $M \leq^{P} M'$, and $M <^{P;Z} M'$ shortened to $M <^{P} M'$.

Definition 2 (Circumscription model). Given a formula $A(P, Z)$, for any interpretation M, if M is a model of $A(P, Z)$ and there exists no model M' of $A(P, Z)$ satisfying $M' <^{P;Z} M$, then M is a model of $CIRC[A(P, Z); P; Z]$.

Intuitively, given a propositional formula φ, its circumscription in a set of minimizing atoms is the formula having only the models of φ that do not assign minimizing atoms to **true** unless necessary. For example, consider the formula A in Example 1, it is not hard to verify that $\{p, q, r\} \models A$ but $\{p, q, r\} \not\models CIRC[A; \{p, q, r\}]$ because $\emptyset \models A$ and $\emptyset <^{\{p,q,r\}} \{p, q, r\}$. In fact, for any disjoint sets P, Q of atoms, $\emptyset$ is a model of $CIRC[A; P; Q]$.

Example 2 (Circumscription in common-sense reasoning).
Consider the use of circumscription in common-sense reasoning. Let

$$K = \{bird \wedge \neg ab \rightarrow fly\},$$

which states that birds normally fly unless an abnormality (ab) occurs. When computing Circ$[K; ab; fly]$, we obtain three models: $\{bird, fly\}$, $\{fly\}$, and $\emptyset$.

Here, ab is minimizing, meaning that we prefer situations where no abnormality occurs. Indeed, in all three models ab is false, which intuitively corresponds to "no abnormality happens."

- The first model can be read as: it is a bird, and it flies.
- The second and third models correspond to situations where it is not a bird, in which case flying may or may not hold.

In this setting: *Fixing atoms* can be understood as conditions that partition the cases under consideration. *Minimizing atoms* typically represents abnormalities, which are assumed to be false unless evidence suggests otherwise. *Varying atoms* corresponds to reasoning outcomes that may change depending on the situation.

The next corollary easily follows from Definition 2.

Corollary 1. Let A be a formula and $M \subseteq \mathcal{A}$. Then

1. $M \models CIRC[A; var(A)]$ if and only if M is a minimal model of A.
2. $M \models CIRC[A; \emptyset; var(A)]$ if and only if $M \models A$.
3. $CIRC[A; P; Z]$ has a model if and only if A has a model.

The next example shows that circumscription is a generalization of the theory of minimal models.

Example 3. Let $A = \{p \vee q\}$, $\mathcal{A} = \{p, q\}$. It's not hard to verify the following:

- $CIRC[A; \{p, q\}]$ has two models $\{p\}$ and $\{q\}$, that are exactly the minimal models of A.
- $CIRC[A; \{p\}; \{q\}]$ has only one model $\{q\}$ since $\{q\} <^{\{p\};\{q\}} \{p\}$.
- $CIRC[A; \emptyset; \{p, q\}]$ has three models $\{p\}$, $\{q\}$ and $\{p, q\}$.

3 Minimal reduct for circumscriptions

In this section, we introduce minimal reduct for circumscriptions. This reduct, based on the inherent structure of circumscription models, achieves efficient model search through iterative formula reduct and simplification.

Definition 3 (Minimal reduct). Given a clause theory $A(P,Z)$ and $M \subseteq \mathcal{A}$, the *minimal reduct* of A w.r.t. M, P and Z,[1] is the set of clauses consisting of, for each $\alpha \in A$,

$$(\alpha^+ \cap ((P \cap M) \cup Z)) \cup \neg(\alpha^- \cap ((P \cap M) \cup Z)) \tag{3}$$

such that

1. $\alpha^- \cap P \cap \overline{M} = \emptyset$,
2. $\alpha^+ \cap \overline{P \cup Z} \cap M = \emptyset$, and
3. $\alpha^- \cap \overline{P \cup Z} \cap \overline{M} = \emptyset$.

The *minimal reduct* of the circumscription $CIRC[A; P; Z]$ w.r.t. $M \subseteq \mathcal{A}$, denoted as $CRed[A; P; Z, M]$, is the circumscription $CIRC[Red(A; P; Z, M); P \cap M; Z]$.

For convenience, we denote Equation 3 by $Red[\alpha; P; Z, M]$. If one of the conditions (a), (b) or (c) in Definition (Equation 3) does not hold, then clause (Equation 3) does not belong to $Red[A; P; Z, M]$. In this case, we define $Red[\alpha; P; Z, M] = \top$.

Thus, if there is no clause in A that satisfies one of the conditions (a), (b) and (c) then $Red[A; P; Z, M] \equiv \top$.

Intuitively, the minimal reduct simplifies clauses with respect to a given interpretation. For a given clause and interpretation, if some atoms in the fixing set make the clause true, or if some atoms in the minimizing set that are **false** make the clause **true**, then the clause is reduced to $\top$. Otherwise, the clause is further simplified to retain only the varying atoms and those atoms in the minimizing set that are assigned **true**, so that minimization can continue on these relevant literals.

The next lemma follows from Definition 3 easily.

Lemma 1. Let $M \subseteq \mathcal{A}$, $\alpha(P,Z)$ be a clause, and $A(P,Z)$ be a clause theory.

1. If $M \not\models \alpha$, then $Red[\alpha; P; Z, M] = \alpha^+ \cap Z \cup \neg(\alpha^- \cap (P \cup Z))$.
2. $var(Red[A; P; Z, M]) \subseteq P \cap M \cup Z$.

The following example illustrates the notions of minimal reduct *CRed* and *Red*.

Example 4. Let $P = \{p, q\}, Z = \{z\}, M = \{q, a\}, M' = \{p\}, A_1(P,Z)$ consists of

$$\alpha_1 : \neg p \vee \neg z, \quad \alpha_2 : z \vee q, \quad \alpha_3 : \neg q \vee p \vee a.$$

It is evident that $M \models A_1$ and $M' \not\models A_1$. We have:

- $Red[A_1; P; Z, M] = \{z \vee q\}$, because α_1 does not satisfy the condition (a) of Definition 3, and α_3 does not satisfy condition (c) of Definition 3.
- $CRed[A_1; P; Z, M] = CIRC[z \vee q; \{q\}; \{z\}]$, which has only one model $\{z\}$, and it is also a model of $CIRC[A_1; P; Z]$.
- $Red[A_1; P; Z, M'] = \{\neg p \vee \neg z, z\}$, which has a unique minimal model $\{z\}$; $CRed[A_1; P; Z, M'] = CIRC[\{\neg p \vee \neg z, z\}; \{p\}; \{z\}]$, which has only one model $\{z\}$.

Intuitive explanation. The distinction between fixing and minimizing atoms plays a crucial role here. For any $M^\star$ such that $M^\star <^{P;Z} M$, the truth values of the *fixing* atom in $\{a\}$ remain unchanged between $M^\star$ and M. For the *minimizing* atoms in $\{p, q\}$, if an atom is false in M, it must also be false in $M^\star$. The varying atom z, on the other hand, may freely change across models. These assumptions drive the simplification of clauses in the reduct.

Take $Red[A_1; P; Z, M] = \{z \vee q\}$ as an example: For $\alpha_1 = \neg p \vee \neg z$, since p is false in M, $\neg p$ is always true; hence α_1 is satisfied and can be reduced to $\top$; For $\alpha_2 = z \vee q$, the values of z and q cannot be determined from M, so the whole clause is retained; For $\alpha_3 = \neg q \vee p \vee a$, since a is true in M, the clause is always satisfied and reduces to $\top$.

Thus, only α_2 survives in the reduct, yielding $\{z \vee q\}$.

Next, we discuss the properties of minimal reduct. This lemma establishes the fundamental correspondence between a clause and its minimal reduct under a given interpretation M. It explains how the satisfaction of a reduced clause by an interpretation M' reflects the satisfaction of the original clause by a combined interpretation that agrees with M on the fixing atoms while possibly minimizing atoms in P. In particular, Lemma 2 guarantees that the reduct construction preserves the semantic relationship between M and M' with respect to the partial order $\leq^{P;Z}$. This property is crucial for proving that the minimal reduct maintains the models of the original theory (Theorem 1) and, consequently, that the overall reduction process is sound with respect to the circumscription semantics.

Lemma 2. Let P, Z, M be the ones in Definition 3, the clause α satisfies conditions (a)-(c) in Definition 3, and $M' \subseteq \mathcal{A}$. Then

1. $M' \cap (M \cap P \cup Z) \cup M \cap \overline{P \cup Z} \leq^{P;Z} M$;
2. $\alpha^- \cap (M \cap P \cup Z) = \alpha^- \cap (P \cup Z)$;
3. $M' \models Red[\alpha; P; Z, M]$ if and only if $M' \cap (M \cap P \cup Z) \cup (M \cap \overline{P \cup Z}) \models \alpha$.

The next theorem shows that the minimal reduct for clause theories preserves the models of A that have the same assignment for fixing atoms, while the minimizing atoms may be minimized further.

Theorem 1. Let $A(P,Z)$ and M be the ones in Definition 3, and $M' \subseteq \mathcal{A}$. Then $M' \models Red[A; P; Z, M]$ if and only if $M' \cap (M \cap P \cup Z) \cup (M \cap \overline{P \cup Z}) \models A$.

Informally, if a set of atoms M' is a model of $Red[A; P; Z, M]$, then one can construct a model of A that agrees with M on the fixing atoms and agrees with M' on varying atoms, while the minimizing atoms in M are further minimizing in terms of M'.

Example 5. Let $M = \{a, p, q\}, A(P; Z) = \{\neg a \vee p \vee \neg q, \quad q \vee \neg p\}$, where $P = \{p, q\}$ and $Z = \emptyset$. It can be readily verified that M is a model of $A(P,Z)$, and $Red[A; P; Z, M] = \{p \vee \neg q, \quad q \vee \neg p\}$.

1 P and Z acts as the minimizing set of atoms and varying set of atoms, respectively.

The intersection $M \cap \overline{P \cup Z}$ is $\{a\}$. We consider the following two cases:

1. When $M' = \emptyset$, we have $M' \models \mathit{Red}[A; P; Z, M]$. Furthermore, $M' \cap (M \cap P \cup Z) \cup (M \cap \overline{P \cup Z}) = \{a\}$, which is a model of A.
2. When $M' = \{p\}$, we have $M' \not\models \mathit{Red}[A; P; Z, M]$. The resulting set is $M' \cap (M \cap P \cup Z) \cup (M \cap \overline{P \cup Z}) = \{a, p\}$, which is not a model of A.

The following theorem demonstrates how to construct a model of $\mathit{CIRC}[A; P; Z]$ from a model $M'$of $\mathit{CRed}[A; P; Z, M]$.

Theorem 2. Let $A(P, Z)$ be a clause theory, and $M', M \subseteq \mathcal{A}$. Then $M' \models \mathit{CRed}[A; P; Z, M]$ if and only if $M' \cap (M \cap P \cup Z) \cup M \cap \overline{P \cup Z} \models \mathit{CIRC}[A; P; Z]$.

Example 6. Let $A(P; Z) = \{p \vee \neg q, \quad q \vee \neg p, \quad p\}$, where $P = \{p\}$ and $Z = \emptyset$. It is not hard to verify that $\mathit{CIRC}[A; P; Z]$ has a unique model $\{p, q\}$. We consider the following cases:

1. Let $M_1 = \{p\}$. Then $\mathit{Red}[A; P; Z, M_1] = \{p, \neg p\}$, which is obviously unsatisfiable, so $\mathit{CRed}[A; P; Z, M_1]$ has no model. In fact, $\mathit{CIRC}[A; P; Z]$ also has no such model M which assigns all fixing atoms **false**, *i.e.*, $M \cap \overline{P \cup Z} = M \cap \{q\} = \emptyset$.
2. Let $M_2 = \{p, q\}$. Then $\mathit{Red}[A; P; Z, M_2] = \{p\}$, so $\mathit{CRed}[A; P; Z, M_2]$ has a unique model $\{p\}$. It's easy to see that $\{p, q\} \models \mathit{CRed}[A; P; Z, M_2]$, and in fact, there is only one M' such that $M' \cap (M_2 \cap P \cup Z) \cup M_2 \cap \overline{P \cup Z} \models \mathit{CIRC}[A; P; Z]$.

The next follows easily from Theorem 2.

Corollary 2. Let $A(P, Z)$ be a clause theory, $Z = \emptyset$, $M, M' \subseteq \mathcal{A}$, M be a model of $A(P, Z)$. The following statements are equivalent to each other:

i. $M' \models \mathit{CRed}[A; P; Z, M]$;
ii. M' is a minimal model of $\mathit{Red}[A; P; Z, M]$;
iii. $M' \cap M \cap P \cup (M \cap \overline{P}) \models \mathit{CIRC}[A; P; Z]$.

Note that a simple approach to compute circumscription models is to iteratively compute subset minimal models by calling an SAT solver in terms of Definition 1. With the help of Theorem 2, we obtain a new approach to compute circumscription models. It is based on the simple one and makes use of the minimal reduct to simplify the circumscription in each iteration. The typical one and the new one are named as `circ` and `circ-reduct`, respectively. They are presented as Algorithms 1, 2, while *Model*(.) extracts a model by calling an SAT solver which returns *unsat* if its input is unsatisfiable. This algorithm can be viewed as an extension of the works by Koshimura et al. (2009) and Ben-Eliyahu and Dechter (1993). Specifically, when the fixing set is empty, this algorithm aligns with the methods described in Koshimura et al. (2009) and Ben-Eliyahu and Dechter (1993).

Intuitively, in Algorithm 1, the set Y determines the truth values of the fixing atoms of the model. Specifically, as long as $M' \models Y$, the assignment of these atoms in M' remains the same as in M. The set T ensures that fewer atoms in the minimizing set are assumed to be true. That is, $M' \cap P \subset M \cap P$ if $M' \models T$. Therefore, the main idea of this algorithm is to find an assignment with fewer true atoms in the minimizing set, while keeping the assignment of the fixing atoms, and satisfying the original theory A. Note that if

```
Require: A circumscription CIRC[A; P; Z].
Ensure: A model of CIRC[A; P; Z], and unsat otherwise.
1: M ← Model(A)  ▷ Computing a model of A by calling
      some SAT solver
2: if M = unsat then
3:    return unsat
4: end if
5: Y ← (M \ (P ∪ Z)) ∪ ¬(P̄ ∪ Z̄ \ M)    ▷ Fix assigments of
      fixing atoms
6: while M ∩ P ≠ ∅ do
7:    T ← ¬(P \ M) ∪ {⋁ ¬(P ∩ M)}    ▷ Constraint to be
      added for finding a smaller model under P
8:    M′ ← Model(T ∪ A ∪ Y)
9:    if M′ = unsat then
10:      break
11:   end if
12:   M ← M′
13: end while
14: return M
```

Algorithm 1. circ(*A*, *P*, *Z*).

the theory A is unsatisfiable, we assume that `Model(A)` returns `unsat`; otherwise, it returns a model of A. This algorithm calls the SAT solving procedure at most $|P| + 1$ times in the worst case.

Theorem 3. Algorithm `circ`$(A; P; Z)$ is correct. That is, if `circ`(A, P, Z) returns `unsat`, then there exists no model of $\mathit{CIRC}[A; P; Z]$; otherwise, the set of atoms returned by `circ`(A, P, Z) is a model of $\mathit{CIRC}[A; P; Z]$.

The idea of Algorithm 1 is to compute circumscription models by adding block clauses, whereas Algorithm 2 computes them by reducing the size of the clause set. Clearly, Algorithm 2 also calls the SAT solver $|P| + 1$ times in the worst case.

Theorem 4. Algorithm `circ-reduct`(A, P, Z) is correct. That is, `circ-reduct`(A, P, Z) returns a model of $\mathit{CIRC}[A; P; Z]$ if A is satisfiable; otherwise, `circ-reduct`(A, P, Z) returns `unsat`.

The following example intuitively demonstrates the computation process of Algorithms 1, 2.

Example 7 (Continuation of Example 4). Here $\mathcal{A} = \{p, q, z, a\}$. Note that $P = \{p, q\}$, $Z = \{z\}$, and the clause theory $A_1(P, Z)$ consists of

$$\alpha_1 : \neg p \vee \neg z, \quad \alpha_2 : z \vee q, \quad \alpha_3 : \neg q \vee p \vee a.$$

(a) The execution process of Algorithm 1 on A_1, P, Z is as follows:

1. Assume that $\mathit{Model}(A_1)$ line 1 returns $M = \{q, a\}$, then at line 5, $Y = \{a\}$;
2. In the first iteration of the WHILE loop, $T = \{\neg p\} \cup \{\neg q\}$ (line 7);
   $M' = \{a, z\}$ (line 8, at this time the only model of $T \cup A \cup Y$ is M');
   $M' \neq$ unsat (line 9);

Require: A circumscription $CIRC[A; P; Z]$.
Ensure: A model of $CIRC[A; P; Z]$, and unsat otherwise.
1: $M \leftarrow Model(A)$ ▷ Computing a model of A by calling some SAT solver
2: **if** M = unsat **then**
3: **return** unsat
4: **end if**
5: $Y \leftarrow M \setminus (P \cup Z)$ ▷ Retain the fixing atoms part of the model
6: $A' \leftarrow A$
7: **while** $M \cap P \neq \emptyset$ **do**
8: $T \leftarrow \{\bigvee \neg(P \cap M)\}$ ▷ Constraint to be added for finding a smaller model on P
9: $A' \leftarrow Red[A'; P; Z, M]$ ▷ reduct the clause set by P, Z, M
10: $M' \leftarrow Model(T \cup A')$
11: **if** M' = unsat **then**
12: **break**
13: **end if**
14: $M \leftarrow M'$
15: **end while**
16: **return** $M \cup Y$ ▷ Reconstruct the full model including fixing atoms

Algorithm 2. circ-reduct (A, P, Z).

$M = M'$ (line 12);
the loop ends (line 6), since $M \cap P = \emptyset$;
3. At line 14, return $M = \{z, a\}$. It's easy to see that $M \models CIRC[A_1; P; Z]$.

(b) The execution process of Algorithm 2 on A_1, P, Z is as follows:

1. Assume that $Model(A_1)$ line 1 returns the model $M = \{q, a\}$, then at line 5, $Y = \{a\}$;
2. In the first iteration of the WHILE loop, $T = \{\neg q\}$ (line 8);
$A' = \{z \vee q\}$ (line 9);
$M' = \{z\}$ (line 10, at this time the only model of $T \cup A'$ is M');
$M' \neq$ unsat (line 11);
$M = M'$ (line 14);
the loop ends (line 7), since $M \cap P = \emptyset$;
3. At line 16, return $M = \{z, a\}$. It is easy to see that $M \models CIRC[A_1; P; Z]$.

3.1 Complexity analysis

In this section, we analyze the time complexity of the two algorithms presented: Algorithm 1 (*circ*) and Algorithm 2 (*circ-reduct*). The complexity mainly depends on the number of SAT solver calls and the auxiliary set operations. Let the input clause theory A contain n clauses over m atoms. While propositional satisfiability has a worst-case complexity of $O(2^m)$, modern SAT solvers typically perform much better in practice due to sophisticated heuristics.

3.1.1 Time complexity of algorithm (*circ*)

The complexity can be analyzed as follows:

- Initial SAT call: computing an initial model M of A requires one SAT solver call, in $O(2^m)$ time.
- Set operations: constructing Y by combining and negating subsets of M involves $O(m)$ operations.
- Main loop: the loop iterates at most $|P|$ times. In each iteration:
  - Constructing T takes $O(|P|)$ time.
  - A SAT solver call on $T \cup A \cup Y$ costs $O(2^m)$ time.

Thus, Algorithm 1 makes at most $|P| + 1$ SAT calls, with polynomial overhead in m. The overall complexity is $O(m \cdot 2^m)$ since $|P| \leq m$.

3.1.2 Time complexity of algorithm *circ-reduct*

Algorithm 2 refines models by applying reducts until minimality is achieved:

- Initial SAT call: one SAT call on A, in $O(2^m)$ time.
- Set operations: constructing Y requires $O(m)$ operations.
- Main loop: the loop executes at most $|P|$ times. In each iteration:
  - Constructing T requires $O(|P|)$ time.
  - Applying the reduct $Red[A'; P; Z, M]$ requires at most $O(n \cdot m)$ operations.
  - A SAT solver call on $T \cup A'$ runs in $O(2^{|Z|+|M\cap P|})$ time in the worst case.

Therefore, the loop contributes at most $O(2^{|Z|+|P|})$ complexity. Including the initial SAT call, the total complexity is $O(2^m)$ since $|Z| + |P| \leq m$.

Both algorithms are dominated by the number of SAT solver calls. Algorithm *circ* requires at most $|P| + 1$ SAT solver calls, while Algorithm *circ-reduct* may involve additional clause reductions. In practice, modern SAT solvers significantly mitigate the theoretical exponential bound, and our empirical results confirm that the overhead of reduct operations is manageable.

4 Circumscription models enumeration

In this section, we further investigate the model enumeration problem for circumscriptions with the help of the algorithm `circ` and `circ-reduct`.

Let P, Z be disjoint finite subsets of $\mathcal{A}$ and $M \subseteq \mathcal{A}$. By $\mathcal{S}(M, P, Z)$ we denote the formula

$$\left(\bigvee \neg (M \setminus Z)\right) \vee \left(\bigvee (\overline{P \cup Z} \setminus M)\right). \tag{4}$$

If P and Z are sets of "minimizing" and "varying" atoms, respectively, then the counter-models of constraint (Equation 4) correspond to the interpretations that are P, Z-greater than M, as

Require: A clause theory A and $M, Z \subseteq \mathcal{A}$.
Ensure: The set MS of models of A agreeing with M on $\overline{Z}$.
1: $MS \leftarrow \emptyset$
2: $Block \leftarrow (M \cap \overline{Z} \cup \neg(\overline{Z} \cap \overline{M}))$ ▷ Fix the assignment of minimizing and fixing atoms
3: $C \leftarrow (\bigvee \neg(Z \cap M)) \vee (\bigvee (Z \setminus M))$ ▷ Clause excluding the current Z-assignment
4: $A' \leftarrow A \cup Block \cup \{C\}$
5: **while** A' is satisfiable **do**
6: $M' \leftarrow Model(A')$ ▷ Computing a model of A' by calling some SAT solver
7: $MS \leftarrow MS \cup \{M'\}$
8: $C \leftarrow (\bigvee \neg(Z \cap M')) \vee (\bigvee (Z \setminus M'))$ ▷ Exclude the new Z-assignment
9: $A' \leftarrow A' \cup \{C\}$
10: **end while**
11: **return** MS

Algorithm 3. circWithZ(*A*,*M*,*Z*).

shown by the next lemma. Thus, it can be used to exclude such M in the search for circumscription models.

Lemma 3. Let P, Z be disjoint finite subsets of $\mathcal{A}$ and $M \subseteq \mathcal{A}$. Then, for each $M' \subseteq \mathcal{A}$, $M' \not\models \mathcal{S}(M, P, Z)$ if and only if $M \leq^{P;Z} M'$.

Example 8. Consider the clause theory $A(P,Z) = \{\alpha_1 : p \vee q, \quad \alpha_2 : p \vee r\}$ where $P = \{p, q, r\}$, $Z = \emptyset$, and $M = \{p\}$ We observe the following:

- $\mathcal{S}(M, P, Z) = \neg p$.
- For any M' such that $p \in M'$, we have $M' \not\models \mathcal{S}(M, P, Z)$ and $M \leq^{P;Z} M'$.

The next corollary easily follows from the above Lemma 3. It shows that the constraint (Equation 4) can indeed be used to compute some new circumscription models different on varying atoms.

Corollary 3. Let $A(P,Z)$ be a clause theory, M be a model of $CIRC[A; P; Z]$ and $M' \subseteq \mathcal{A}$. Then, $M' \models CIRC[A \cup \{\mathcal{S}(M, P, Z)\}; P; Z]$ if and only if $M \setminus Z \neq M' \setminus Z$ and $M' \models CIRC[A; P; Z]$.

According to Corollary 3, we propose the algorithm `circWithZ` to iteratively compute such models that are merely different on given varying atoms of Z. The following example shows how this algorithm is involved in computing such models.

Example 9. Let $A = \{\alpha_1 : p \vee q, \quad \alpha_2 : a \vee \neg p, \quad \alpha_3 : q \vee z\}$, $P = \{p, q\}$, $Z = \{z\}$, and $M = \{a, q, z\}$. It is evident $M \models A$. The computation process using `circWithZ`(A, M, Z) is as follows:

1. At line 1, $MS = \emptyset$;
2. At line 2, $Block = \{a, q, \neg p\}$;
3. At line 3, $C = \{\neg z\}$;
4. At line 4 $A' = \{p \vee q, a \vee \neg p, q \vee z, a, q, \neg p, \neg z\}$;

Require: A circumscription $CIRC[A; P; Z]$.
Ensure: The set M of all models of $CIRC[A; P; Z]$.
1: $MS \leftarrow \emptyset$
2: **while** A is satisfiable **do**
3: $M \leftarrow$ circ(A, P, Z) ▷ Or equivalently call circ-reduct(A, P, Z)
4: $M_Z \leftarrow$ circWithZ(A, M, Z) ▷ Enumerate all variants under Z
5: $MS \leftarrow MS \cup \{M\} \cup M_Z$
6: $A \leftarrow A \cup \{\mathcal{S}(M, P, Z)\}$ ▷ Add blocking clause to exclude M and its Z-variants
7: **end while**
8: **return** MS

Algorithm 4. circ-enum(*A*,*P*,*Z*).

5. In the first iteration of the WHILE loop, $M' = \{a, q\}$ (line 6)
   $MS = \{\{a, q\}\}$ (line 7);
   $C = \{z\}$ (line 8);
   $A' = \{p \vee q, a \vee \neg p, q \vee z, a, q, \neg p, \neg z, z\}$ (line 9);
   the loop ends (line 5), since A' is satisfiable;
6. At line 11 returns $MS = \{\{a, q\}\}$.

The correctness of the algorithm `circWithZ` is guaranteed by the next theorem.

Theorem 5. Algorithm 3 is sound and complete.

We are now in the position to present the algorithm of enumerating circumscription models, as shown by Algorithm 4. Informally, this algorithm `circ-enum` iteratively computes a circumscription model and then immediately enumerates all models that merely differ from the just-computed model. The core concept underlying this approach is akin to the strategies outlined in prior works, such as those by Dvořák et al. (2015) and Niskanen and Järvisalo (2020).

The correctness of `circ-enum` is guaranteed by the next theorem.

Theorem 6. Algorithm 4 is sound and complete.

The next example illustrates how `circ-enum` can be used to compute all circumscription models.

Example 10. Let $A = \{\alpha_1 : p \vee q, \quad \alpha_2 : a \vee \neg p, \quad \alpha_3 : q \vee z\}$, $P = \{p, q\}$, and $Z = \{z\}$. The computation steps for `circ-enum`(A, P, Z) are detailed below:

1. In the first iteration of the WHILE loop, we assume that `circ` (or `circ-reduct`) returns $M = \{a, q, z\}$ at line 3, then
   $M_z = \{\{a, q\}\}$ (line 4);
   $MS = \{\{a, q, z\}, \{a, q\}\}$ (line 5);
   $A = \{p \vee q, a \vee \neg p, q \vee z, \neg a \vee \neg q\}$ (line 6);
   enter next iteration of the WHILE loop, since A is satisfiable;
2. In the second iteration of the WHILE loop, we assume that `circ` (or `circ-reduct`) returns $M = \{q, z\}$ at line 3, then
   $M_z = \{\{q\}\}$ (line 4);
   $MS = \{\{a, q, z\}, \{a, q\}, \{q, z\}, \{q\}\}$ (line 5);
   $A = \{p \vee q, a \vee \neg p, q \vee z, \neg a \vee \neg q, \neg q \vee a\}$ (line 6);
   the loop ends (line 2), since A is unsatisfiable;
3. At line 8 returns $MS = \{\{a, q, z\}, \{a, q\}, \{q, z\}, \{q\}\}$.

5 Experimental results

Based on one of the state-of-the-art SAT solvers LSTech-Maple,[2] all three algorithms `circ`, `circ-reduct`, `circ-enum` have been implemented. This SAT solver integrates stochastic local search techniques into conflict-driven clause learning (CDCL) SAT solvers and keeps its completeness (Cai and Zhang, 2022; Cai et al., 2022).

To evaluate their performance, we selected three categories of benchmarks: model-based circuit diagnosis, random propositional circumscription, and various industrial test cases from the SAT competition.

We compare the three algorithms implemented `circ` and `circ-reduct` with `aspino`[3] and `circ2dlp`[4] The experimental environment includes a Linux server running Ubuntu 20.04 with an AMD EPYC 7742 CPU and 1007 GB of memory. The implementation and experimental data have been uploaded to GitHub.[5]

5.1 Note on unsatisfiable instances

For formulas that are unsatisfiable, the SAT solver invoked in the first step immediately reports UNSAT, so the core circumscription procedure is not executed. Consequently, the runtime and other performance metrics on such inputs are determined almost entirely by the underlying SAT solver, with the overhead of our circ/circ-reduct algorithms being negligible. UNSAT search itself can be costly, typically involving extensive branching and conflict analysis; approaches that extract unsatisfiable cores may introduce additional overhead, which likewise stems from the solver rather than our method.

In this section, we only foucs on CPU time, the memory will be reported in Supplementary material 2.

It should be emphasized that the issues observed with `circ2dlp` stem from minor implementation inconsistencies rather than from any flaw in the underlying theory of circumscription. In particular, `circ2dlp` may occasionally produce incorrect results when handling fixing atoms. For example, for $A = \{x_1 \vee x_2 \vee x_3\}$, $P = \emptyset$, and $Z = \emptyset$, the rules generated from A by `circ2dlp` are not recognizable by `clasp`. Likewise, for $A = \{\alpha_1 : x_1 \vee x_3, \quad \alpha_2 : \neg x_1 \vee x_2\}$ and $P = \{x_2, x_3\}$, `circ2dlp` returns only $\{x_3\}$ as the unique model of $CIRC[A; P]$, whereas $\{x_1, x_2\}$ is also a valid model. To handle such cases in practice, we developed the tool `cnf2any`,[6] which eliminates fixing atoms following the method proposed by de Kleer and Konolige (1989).

5.2 Integration with existing SAT frameworks

Our algorithms invoke an external SAT solver through its standard CNF interface. Modern SAT solvers such as Maple, Glucose, and MiniSat all provide C/C++ source code and well-documented APIs, so `circ` and `circ-reduct` can be integrated into other systems or embedded in larger software projects with minimal effort. This integration requires only routine engineering work–such as linking against the solver library or calling its incremental interface–and does not affect the theoretical correctness or performance guarantees of our approach. We emphasize that our implementation intentionally targets SAT solvers via CNF interfaces. Attempting to integrate the method with an ASP toolchain (e.g., clingo) is, in our assessment, not practically feasible without changing the problem or incurring prohibitive costs.

5.3 Model-based circuit diagnosis

Formally, a model-based circuit diagnosis is a triple $D = (SD, COMP, OBS)$, where SD is a propositional theory that describes the circuit system, $COMP$ is a finite set of components (atoms), and OBS is a set of literals representing system observations. A component set M is a *diagnosis* of D if and only if $SD \cup OBS \cup \{\neg Ab(a) \mid a \in COMP \setminus M\} \cup \{Ab(c) \mid c \in M\}$ is satisfiable, where $Ab(a)$ indicates that component a is abnormal. If M is a diagnosis of D and no proper subset of M is a diagnosis of D, then M is referred to as a *minimal diagnosis* of D (Reiter, 1987).

We used 11 standard ISCAS85 circuits [7] as our benchmarks. These circuit descriptions are translated into CNF formulas that serve as the system description SD, the circuit gates as the components $COMP$, and random assignments of circuit inputs and outputs as the observations OBS. The minimal diagnoses of $D = (SD, COMP, OBS)$ correspond to the circumscription models of $CIRC[SD \cup OBS; COMP]$. It is important to note that all circuit components are the minimizing atoms of $CIRC[SD \cup OBS; COMP]$, while all other propositional symbols are treated as varying atoms of $CIRC[SD \cup OBS; COMP]$.

For each circuit, we randomly generated 20 observation instances, ran 5 times for each instance, with a CPU time limit of 1,800 s, and calculated the average CPU time for solved instances. The experimental results are presented in Table 1. The best CPU time for each circuit is in bold face. It can be seen that both `circ` and `circ-reduct` significantly outperform `circ2dlp`, and are comparable with `aspino`. In particular, while `circ` and `circ-reduct` solved all the 20 instances of the c6828 circuit, `aspino` ran out of 30 min for 19 instances. Additionally, the minimal reduct seems not very helpful for the benchmark since `circ` and `circ-reduct` have very similar average CPU time.

2 https://github.com/shaowei-cai-group/LSTech-Maple

3 https://github.com/alviano/aspino

4 http://www.tcs.hut.fi/Software/circ2dlp/

5 https://github.com/gzu-ai/circ_solver, https://github.com/gzu-ai/circumscription_experiment

6 https://github.com/gzu-ai/cnf2any

7 https://sportlab.usc.edu/~msabrishami/benchmarks.html

TABLE 1 The average CPU time in seconds for minimal circuit diagnosis.

Instances	Circ	Circ-reduct	Aspino	Circ2dlp
c17	**0.00018**	0.000211	0.000958	0.0123
c432	0.023595	0.022939	**0.002233**	–
c499	0.030602	0.032227	**0.008674**	–
c880	0.090575	0.087977	**0.004910**	–
c1355	0.160254	0.157518	**0.027911**	–
c1908	0.472890	0.502259	**0.280355**	–
c2670	0.532106	0.549403	**0.025147**	–
c3540	**1.530130**	1.613932	2.445828	–
c5315	1.746824	**1.672224**	2.904894	–
c6288	8.322000	**7.39054**	64.491870[a]	–
c7552	5.782249	5.669364	**2.869035**	–

[a] There are 19 instances timeout.
–: Timeout.
Bold values denote the smallest CPU time among the compared solvers for each benchmark (i.e., the fastest solver).

5.4 Random circumscriptions

We examined our algorithms on random CNF instances whose number of atoms in $\mathcal{A}$ ranges from 50 to 1000 with an interval of 50, and clause-to-atom ratios are from 3 to 5 with an interval of 0.5. The CNF instances were generated using the method proposed by Amendola et al. (2020). We generated 10 instances for each combination, resulting in $20 \times 5 \times 10 = 1000$ random circumscriptions in total.

For each CNF φ over $\mathcal{A}$, we created random circumscriptions $CIRC[\varphi; P; Z]$ by randomly selecting disjoint subsets P and Z from $\mathcal{A}$, with $|P|$ and $|Z|$ set to 10%, 30%, or 50% of $|\mathcal{A}|$.

Tables 2, 3 present the average CPU time in seconds and the number of satisfiable and unsatisfiable instances solved by `circ`, `circ-reduct`, `aspino`, and `circ2dlp` within a 2-hour CPU time limit, respectively. It can be observed that both `circ` and `circ-reduct` significantly outperform `aspino` and `circ2dlp` by four orders of magnitude on satisfiable instances. Our methods solved a few hundred instances more than both `aspino` and `circ2dlp`.

For unsatisfiable instances, `circ` and `circ-reduct` also demonstrate an advantage by solving a greater number of instances, though there is no big difference in the average CPU time.

Beyond average CPU time, we also analyze how the clause set shrinks across reduct rounds. For each circumscription instance we record the number of clauses C_r after round r and report the remaining-clauses ratio $\rho_r = C_r/C_0$ (lower is better). Since different instances may terminate after different numbers of rounds, the statistic at round r aggregates only those instances that reached at least r rounds (no imputation). Figure 1 shows per-round pruning trajectories split by $|P|/|\mathcal{A}| \in \{0.1, 0.3, 0.5\}$ (three panels), with curves stratified by $|Z|/|\mathcal{A}| \in \{0.1, 0.3, 0.5\}$; bands indicate IQR across instances. Figure 2 overlays all nine $(|P|/|\mathcal{A}|, |Z|/|\mathcal{A}|)$ settings in one panel for a global view.

Guided by the trajectories in Figures 1, 2, we observe:

1. Holding $|Z|/|\mathcal{A}|$ (resp. $|P|/|\mathcal{A}|$) fixed, the total number of reduct rounds increases as $|P|/|\mathcal{A}|$ (resp. $|Z|/|\mathcal{A}|$) grows, i.e., larger fixed or varying sets require more rounds before convergence. (ii) Across all configurations the **first** reduct yields the smallest remaining ratio (largest pruning step).
2. Quantitatively, after the first round we observe $\rho_1 \approx 0.10$ at $(|Z|/|\mathcal{A}|, |P|/|\mathcal{A}|) = (0.1, 0.1)$ (about 90% pruned), whereas even in the least favorable setting $(0.5, 0.5)$ the first round still leaves $\rho_1 \approx 0.60$ (about 40% pruned). These trends are consistent across all clause densities considered.

5.5 Circumscription of industrial CNF benchmarks

We choose moderate size industrial CNF benchmarks from SAT competitions.[8] They are *Collatz* and *Johnson* from SAT-2019, *Giraldez* from SAT-2016, and *crypto* and *grieu* from SAT-2007. While both *Collatz* and *Johnson* contain 19 instances, *Giraldez* has 29 instances, *crypto* and *grieu* have 10 instances. To generate circumscription instances, we applied the same strategy to generate P and Z as described in Section 5.4. We report the metrics only for satisfiable instances, since there is no big difference for unsatisfiable instances, as shown in Table 3 for random CNFs.

Please note that while `aspino`, `circ` and `circ-reduct` specify varying and fixing predicates at the end of input files, `circ2dlp` specifies varying predicates and fixing predicates in the command line as arguments. Due to the very large number of atoms or clauses in *Collatz* and *crypto*, `circ2dlp` either outputs a "argument list too long" or runs out of time on the two benchmarks, while `aspino` always runs out of time. For clarity, the experimental results on *Collatz* and *crypto* are not reported in detail for `aspino` and `circ2dlp`. Tables 4–8 present the test results for the solved satisfiable instances by `circ`, `circ-reduct`, `aspino`, and `circ2dlp` in *Collatz, crypto, Johnson, Giraldez*, and *grieu* benchmarks, respectively. The experimental results further confirm the effectiveness of `circ` and `circ-reduct` compared with `aspino` and `circ2dlp`,

8 http://satcompetition.org/

TABLE 2 The number of solved satisfiable instances and the average CPU time in seconds for random circumscriptions.

$\|P\|/\|\mathcal{A}\|$	$\|Z\|/\|\mathcal{A}\|$	# Solved satisfiable instances				Avg. CPU time (s)			
		Circ	Circ-reduct	Aspino	Circ2dlp	Circ	Circ-reduct	Aspino	Circ2dlp
0.1	0.1	605	605	447	485	0.12	0.12	105.02	551.57
	0.3	605	605	455	485	0.13	0.13	161.86	349.15
	0.5	605	605	458	472	0.13	0.12	150.57	297.11
0.3	0.1	605	605	232	487	0.13	0.12	328.23	508.93
	0.3	605	605	233	424	0.12	0.13	278.90	3,445.2
	0.5	605	605	223	363	0.14	0.14	235.98	345.23
0.5	0.1	605	605	152	462	0.12	0.13	342.85	466.58
	0.3	605	605	152	364	0.12	0.13	267.34	381.35
	0.5	605	605	153	344	0.15	0.14	344.73	306.30

TABLE 3 The number of solved unsatisfiable instances and the average CPU time in seconds for random circumscriptions.

$\|P\|/\|\mathcal{A}\|$	$\|Z\|/\|\mathcal{A}\|$	# Solved unsatisfiable instances				Avg. CPU time (s)			
		Circ	Circ-reduct	Aspino	Circ2dlp	Circ	Circ-reduct	Aspino	Circ2dlp
0.1	0.1	172	170	160	139	366.60	308.47	433.19	376.22
	0.3	170	170	159	142	284.59	322.54	361.46	495.63
	0.5	171	170	156	140	336.75	307.47	315.77	321.30
0.3	0.1	170	170	157	142	300.84	311.12	324.04	497.34
	0.3	170	170	158	140	298.65	320.17	310.49	417.73
	0.5	172	170	157	142	362.98	310.19	339.42	431.81
0.5	0.1	171	171	157	141	336.08	361.29	316.01	466.71
	0.3	170	169	157	141	310.72	267.67	356.45	426.14
	0.5	171	170	158	141	340.94	296.64	372.04	398.69

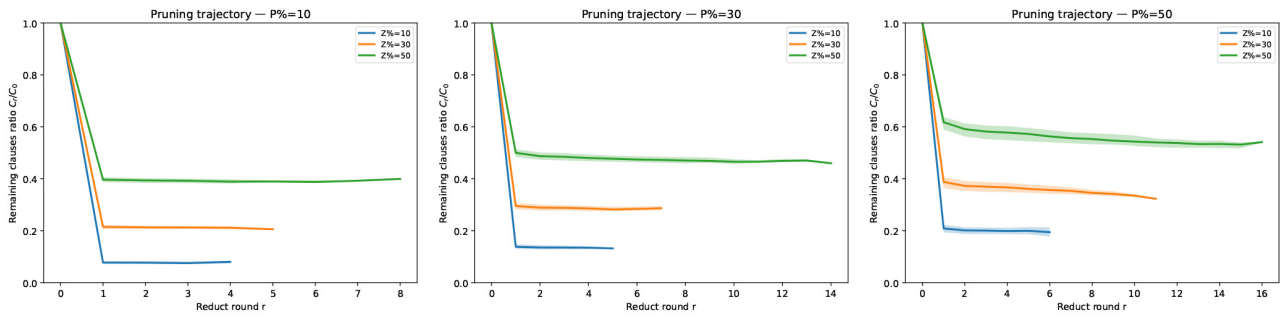

FIGURE 1
Per-round pruning trajectories by $|P|/|\mathcal{A}|$. Each panel fixes $|P|/|\mathcal{A}|$ and shows the median remaining-clauses ratio C_r/C_0 across reduct rounds, with curves stratified by $|Z|/|\mathcal{A}| \in \{0.1, 0.3, 0.5\}$. Shaded bands indicate the interquartile range (IQR, 25th–75th percentile) across instances at each round; solid curves show the median. At round r, statistics aggregate only instances that reached at least r rounds.

while there is no much difference between `circ` and `circ-reduct`.

- There is no big difference between `circ` and `circ-reduct` in terms of the number of solved instances or the average CPU time per instance. Both `circ` and `circ-reduct` solved exactly the same number of satisfiable instances for all benchmarks excluding *Johnson*, for which `circ-reduct` solved slightly less number of instances than that of `circ` in at most 2. For the benchmark `crypto` the overall average CPU

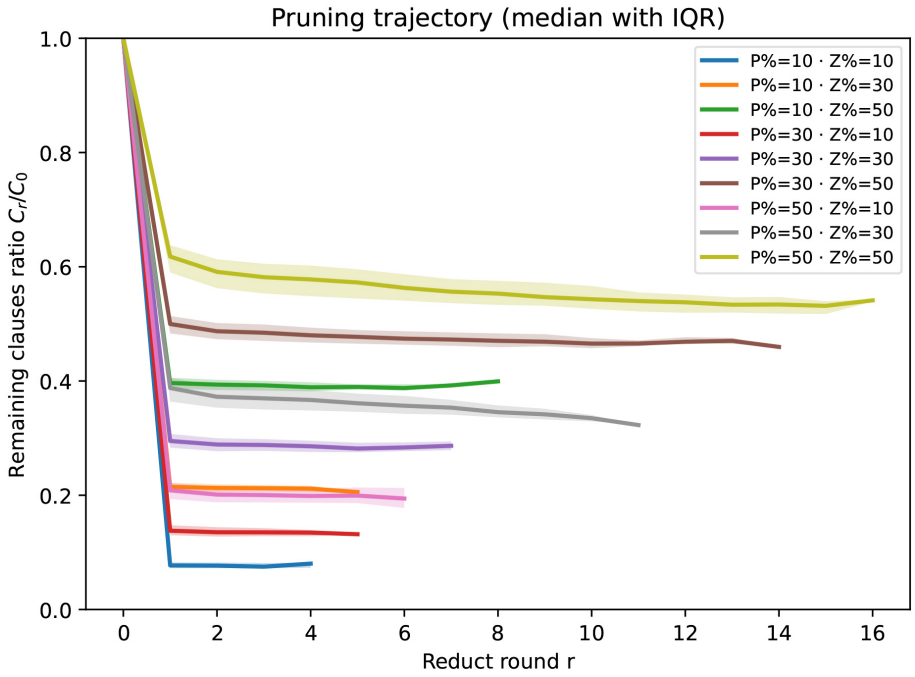

FIGURE 2
Overall pruning trajectories across ($|P|/|\mathcal{A}|$, $|Z|/|\mathcal{A}|$). Median remaining-clauses ratio C_r/C_0 with IQR bands (25th–75th percentile) across instances; lower curves indicate stronger pruning. Curves flatten more slowly as $|P|/|\mathcal{A}|$ or $|Z|/|\mathcal{A}|$ increases, indicating more reduct rounds before convergence.

time of `circ` (resp. `circ-reduct`) is 840.37 (resp. 803.90), and for the benchmark `Giraldez` it is 63.77 (resp. 59.92). For the benchmark `Callatz`, the overall average CPU time of `circ` (resp. `circ-reduct`) is 1417 (resp. 1,437).

- `circ` and `circ-reduct` solved more instances than `aspino` and `circ2dlp` for the benchmarks *Johnson, Giraldez* and *grieu*. For the benchmark *Giraldez*, both `circ` and `circ-reduct` solve 18 instances, while `aspino` solved two instances, see Table 7. For the benchmark *Johnson*, `circ` and `circ-reduct` solved 5–7 instances, while `circ2dlp` solved up to two instances, see Table 6. There are similar results for the benchmark *grieu*, see Table 8.
- Both `circ` and `circ-reduct` have usually less average CPU time than `aspino` and `circ2dlp` for the solved instances of the benchmarks *Johnson, Giraldez* and *grieu*, even though `circ` and `circ-reduct` solved more instances.
- It seems that for all mentioned computing methods the ratios of minimizing and varying predicates over the signature do not have a distinguishing effect on the computing efficiency.

For enumerating circumscription models, we have implemented the algorithm `circ-enum` calling `circ` or `circ-reduct` and evaluated it against all the above benchmarks. In this case, the CPU limit is set up to 30 min. The experimental results do not show a remarkable difference between `circ-enum` and `aspino`, while they outperform `circ2dlp`. For instance, `circ2dlp` enumerates all the 60 models for the diagnosis of circuit c17, `aspino` enumerates 0 models for the diagnosis of circuit c6288. For the random CNFs, `aspino` solved a larger number of instances than that of `circ-enum`. For the benchmark *grieu*, `circ-enum` computed more instances than `aspino`. We do not report these experimental results in detail for simplicity. Interested readers may refer to the experiments in the above github link.

6 Related work

Since the introduction of circumscription, the challenge of efficiently computing circumscription models has become a significant focus of research.

Currently, methods for computing circumscription are categorized into two classes: translating circumscriptions into logic programs (under stable model semantics) or leveraging SAT solvers.

6.1 Translating circumscription to logic programming

Early research concentrated on simplifying circumscription. Lifschitz (1985) proposed eliminating mutable predicates in circumscription in 1985, but this method introduced existential quantifiers. In 1988, Yuan and Wang extended this approach to specific cases, but both methods suffered from exponential growth (Yuan and Wang, 1988). Later, in 1992, Cadoli introduced a technique for eliminating variable predicates in circumscription (Cadoli et al., 1992); however, this method focused on inferring formulas from circumscription rather than transforming it. In

TABLE 4 The number of solved satisfiable instances and average CPU time in seconds for *Collatz*.

$\|P\|/\|\mathcal{A}\|$	$\|Z\|/\|\mathcal{A}\|$	# Solved satisfiable instances		Avg. CPU time (s)	
		Circ	Circ-reduct	Circ	Circ-reduct
0.1	0.1	7	7	1,549.74	1,645.93
	0.3	8	8	1,376.15	1,396.52
	0.5	8	8	1,346.88	1,368.55
0.3	0.1	8	8	1,418.38	1,484.58
	0.3	8	8	1,342.50	1,421.92
	0.5	8	8	1,453.62	1,437.54
0.5	0.1	8	8	1,500.67	1,445.96
	0.3	8	8	1,362.92	1,305.51
	0.5	8	8	1,405.37	1,429.43

TABLE 5 The number of solved satisfiable instances, average CPU time in seconds for *crypto*.

$\|P\|/\|\mathcal{A}\|$	$\|Z\|/\|\mathcal{A}\|$	# Solved satisfiable instances		Avg. CPU time (s)	
		Circ	Circ-reduct	Circ	Circ-reduct
0.1	0.1	10	10	864.08	790.95
	0.3	10	10	848.10	837.96
	0.5	10	10	827.59	820.00
0.3	0.1	10	10	821.66	756.67
	0.3	10	10	862.14	824.34
	0.5	10	10	831.38	804.78
0.5	0.1	10	10	847.98	793.09
	0.3	10	10	822.27	806.67
	0.5	10	10	838.17	800.66

TABLE 6 The number of solved satisfiable instances and average CPU time in seconds for *Johnson*.

$\|P\|/\|\mathcal{A}\|$	$\|Z\|/\|\mathcal{A}\|$	# Solved satisfiable instances				Avg. CPU time (s)			
		Circ	Circ-reduct	Aspino	Circ2dlp	Circ	Circ-reduct	Aspino	Circ2dlp
0.1	0.1	6	6	2	0	1,874.11	1,836.31	630.27	–
	0.3	6	5	2	1	1,817.85	891.59	667.37	4,794.84
	0.5	7	6	2	2	2,462.56	1,984.04	661.55	5,804.20
0.3	0.1	7	5	4	1	2,497.59	900.14	3,264.90	4,181.70
	0.3	6	5	3	1	1,857.05	875.71	1,993.10	4,541.34
	0.5	7	5	4	1	2,523.02	889.17	3,325.13	4,911.33
0.5	0.1	7	5	3	1	2,431.79	866.36	3,107.61	3,986.31
	0.3	6	5	3	1	1,865.32	922.71	3,080.66	5,184.98
	0.5	7	6	3	1	2,588.24	1,916.91	2,783.58	4,909.26

–: timeout.

1989, Kleer and Konolige developed a method for eliminating fixing predicates in circumscription (de Kleer and Konolige, 1989).

In 1995, Sakama and Inoue proposed translating circumscription into a general disjunctive logic program (Sakama and Inoue, 1995), where the stable models correspond to circumscription models, but their method required the computation of characteristic clauses, leading to potential clause explosion. In 2008, Oikarinen and Janhunen addressed prioritized circumscription in the propositional case by translating it into a disjunctive logic program (Oikarinen and Janhunen, 2008),

TABLE 7 The number of solved satisfiable instances and average CPU time in seconds for *Giraldez*.

$\|P\|/\|\mathcal{A}\|$	$\|Z\|/\|\mathcal{A}\|$	# Solved satisfiable instances				Avg. CPU time (s)			
		Circ	Circ-reduct	Aspino	Circ2dlp	Circ	Circ-reduct	Aspino	Circ2dlp
0.1	0.1	18	18	2	4	63.03	60.27	153.66	6,650.75
	0.3	18	18	2	11	58.11	60.07	118.12	5,203.65
	0.5	18	18	2	11	62.31	58.64	131.91	5,245.27
0.3	0.1	18	18	2	11	63.91	57.99	161.29	4,470.13
	0.3	18	18	2	11	66.90	59.33	160.48	5,100.03
	0.5	18	18	2	12	65.62	57.83	154.51	4,171.95
0.5	0.1	18	18	2	11	64.95	59.07	139.26	4,099.05
	0.3	18	18	2	11	63.88	57.39	109.29	4,645.46
	0.5	18	18	2	12	65.28	59.68	149.95	4,124.79

TABLE 8 The number of solved satisfiable instances and the average CPU time in seconds for *grieu*.

$\|P\|/\|\mathcal{A}\|$	$\|Z\|/\|\mathcal{A}\|$	# Solved satisfiable instances				Avg. CPU time (s)			
		Circ	Circ-reduct	Aspino	Circ2dlp	Circ	Circ-reduct	Aspino	Circ2dlp
0.1	0.1	10	10	6	5	753.31	853.19	894.55	1,185.86
	0.3	10	10	7	5	754.60	870.55	1,758.11	853.23
	0.5	10	10	6	5	712.04	878.16	936.14	851.03
0.3	0.1	10	10	5	5	765.83	824.98	581.69	850.02
	0.3	10	10	5	5	752.25	870.59	584.62	743.07
	0.5	10	10	5	5	720.84	873.77	625.78	739.57
0.5	0.1	10	10	5	5	731.53	860.21	1,713.07	634.19
	0.3	10	10	5	5	707.74	904.30	1,693.95	821.94
	0.5	10	10	4	5	735.66	959.91	2,147.24	607.86

and implementing the tool `circ2dlp`. Similarly, in 2011, Gebser et al. developed `metasp` (Gebser et al., 2011), a general implementation technique using meta-programming, which reuses existing ASP systems to capture various forms of qualitative preferences among answer sets. This approach also enables the computation of circumscription in the propositional case without fixing predicates.

In 2014, Wan et al. developed `cfo2lp`, which translates first-order circumscription into a logic program, outperforming both `circ2dlp` and `metasp` on circuit diagnosis problem (Wan et al., 2014).

6.2 Translating circumscription to SAT

Inspired by loop formulas in computational logic programs (Lee and Lifschitz, 2003; Lin and Zhao, 2004), (Lee and Lin, 2006) proposed using loop formulas and completions to compute circumscription in the propositional case. However, the number of loops can grow exponentially, and the method requires eliminating existential quantifiers, making it potentially exponentially complex. In 2017, Alviano introduced an approach for enumerating propositional circumscription models using unsatisfiable core analysis, with the SAT solver functioning as a search engine (Alviano, 2017). This method also requires the introduction of auxiliary variables during computation.

In the propositional case, when the fixing predicate set is empty, the circumscription model is equivalent to the P-minimal model. Giunchiglia and Maratea (2006) proposed a tool in 2006 based on the Davis-Logemann-Loveland (DLL) algorithm to solve SAT-related optimization problems, including P-minimal models. In 2009, Koshimura introduced a SAT-based algorithm for computing P-minimal models (Koshimura et al., 2009). However, a common limitation of these approaches is their inability to handle circumscription containing fixing predicates.

A notable SAT-based approach relevant to our work is the counterexample-guided abstraction refinement (CEGAR) framework for propositional circumscription proposed by Janota et al. (2010). While their primary focus is on solving the entailment problem through iterative abstraction refinement, their CEGAR-based algorithm shares conceptual similarities with our constraint refinement process in Algorithms 1, 4. Specifically, both approaches employ counterexamples to guide the incremental refinement of

TABLE 9 Comparison of methods for computing circumscription models.

Method	Methodology	Strengths and limitations
`circ2dlp` (Oikarinen and Janhunen, 2008)	ASP translation	Needs auxiliary vars.
`aspino` (Alviano, 2017)	SAT + unsat core analysis	Solver-dependent; only support for cardinality constraints
Loop formulas (Lee and Lin, 2006)	SAT + loop formulas	Exponential in number of loop formula
`circ`	SAT-based	No blowup; no aux./refresh vars.
`circ-reduct`	Minimal reduct + SAT-based	No blowup; no aux./refresh vars.
CEGAR (Janota et al., 2010)	Abstraction refinement	Computes vars. false in all models

constraints. In the context of circumscription, CEGAR excels at computing the set of variables assigned false in all models of a circumscribed formula.

6.3 Summary

Table 9 provides a comprehensive summary of the primary methodologies for computing circumscription models, delineating their core strategies and inherent limitations. Due to the absence of reported complexity analyses for certain methods, complexity metrics are excluded from the comparison. Furthermore, as the evaluation is restricted to comparisons with `aspino` and `circ2dlp`, performance metrics are not included. In general, the `circ`/`circ-reduct` approach demonstrates superior performance compared to `aspino`, while the `circ` and `circ-reduct` methods exhibit equivalent performance. Moreover, our proposed approach scales to significantly larger input instances than previously reported methods, highlighting its strong scalability.

7 Conclusion and future work

In this study, we introduced the notion of minimal reduct for circumscription and established a new characterization theorem for circumscription based on minimal reduct. We proposed and implemented two algorithms for computing models of circumscription: `circ`, which operates directly on propositional clause theories, and `circ-reduct`, which computes with the help of minimal reduct. We further presented the algorithm `circ-enum` to enumerate circumscription models.

We have evaluated our algorithms `circ`, `circ-reduct` and `circ-enum` on the benchmark model-based circuit diagnosis, random CNFs formulas, and several industrial benchmarks from SAT competitions. The results show that by employing the efficient SAT solver, both `circ` and `circ-reduct` are effective and significantly outperform `circ2dlp` and the solver based on unsatisfiable core analysis `aspino`. The former translates circumscriptions into disjunctive logic programs under stable model semantics. This study provides new directions for computing circumscription models.

The following issues deserve our further investigation:

- To optimize the model enumeration algorithm `circ-enum` for circumscription, we aim to draw inspiration from recent advancements in answer set programming (ASP) enumeration techniques (Calimeri et al., 2019; Comploi-Taupe et al., 2023). We plan to incorporate conflict-driven clause learning, backtracking, and heuristic search techniques. Additionally, given the connection between circumscription completion and loop formulas (Lee and Lin, 2006), we will explore how loop formulas can be fully utilized in enumerating circumscription models.
- Given a model of a circumscription, how to establish a justification for the atoms in the model. Similar ideas have been designed from answer set programming with the help of minimal reduct (Wang et al., 2023).
- Note that prioritized circumscription (Lifschitz, 1985) extends parallel circumscription by introducing priorities among minimizing predicates, which allows for the representation of priority-related applications. Given the existence of efficient computational methods for prioritized circumscription (Wakaki and Satoh, 1995, 1997; Chen, 1999; Oikarinen and Janhunen, 2008), we will consider how the notion of minimal reduct can be extended to prioritized circumscription. In particular, since prioritized circumscription can be reduced to the conjunction of multiple parallel circumscriptions, a natural preliminary direction is to apply the proposed minimal reduct approach sequentially to each component. Similarly, nested circumscription, which allows circumscription formulas themselves to appear as components, poses additional challenges. We leave the exploration of extending minimal reduct to nested circumscription as an interesting direction for future work.
- An interesting direction for future work is to extend our algorithms to parallel and distributed frameworks. Currently designed for sequential execution, the algorithms could benefit from parallelization, particularly during minimal reduct computation and model enumeration, which would significantly enhance scalability and efficiency for large-scale instances.

Data availability statement

The datasets presented in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found in the article/Supplementary material.

Author contributions

ZX: Software, Writing – original draft, Methodology, Validation. YW: Supervision, Funding acquisition, Conceptualization, Writing – review & editing, Project

administration. LY: Writing – review & editing, Validation. RF: Writing – review & editing.

Funding

The author(s) declare that financial support was received for the research and/or publication of this article. This work was supported by the National Natural Science Foundation of China under Grants 62376066 and 61976065.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Generative AI statement

The author(s) declare that no Gen AI was used in the creation of this manuscript.

Any alternative text (alt text) provided alongside figures in this article has been generated by Frontiers with the support of artificial intelligence and reasonable efforts have been made to ensure accuracy, including review by the authors wherever possible. If you identify any issues, please contact us.

Publisher's note

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/frai.2025.1614894/full#supplementary-material

References

Alviano, M. (2017). Model enumeration in propositional circumscription via unsatisfiable core analysis. *Theory Pract. Log. Program.* 17, 708–725. doi: 10.1017/S1471068417000278

Alviano, M. (2019). Argumentation reasoning via circumscription with pyglaf. *Fundam. Inform.* 167, 1–30. doi: 10.3233/FI-2019-1808

Amendola, G., Ricca, F., and Truszczynski, M. (2020). New models for generating hard random boolean formulas and disjunctive logic programs. *Artif. Intell.* 279:103185. doi: 10.1016/j.artint.2019.103185

Angiulli, F., Ben-Eliyahu-Zohary, R., Fassetti, F., and Palopoli, L. (2022). Graph-based construction of minimal models. *Artif. Intell.* 313:103754. doi: 10.1016/j.artint.2022.103754

Ben-Eliyahu, R., and Dechter, R. (1993). "On computing minimal models," in *Proceedings of the 11th National Conference on Artificial Intelligence*, eds. R. Fikes, W. G. Lehnert (Washington, DC: AAAI Press/The MIT Press), 2–8.

Cadoli, M., Eiter, T., and Gottlob, G. (1992). An efficient method for eliminating varying predicates from a circumscription. *Artif. Intell.* 54, 397–410. doi: 10.1016/0004-3702(92)90051-X

Cadoli, M., and Lenzerini, M. (1994). The complexity of propositional closed world reasoning and circumscription. *J. Comput. Syst. Sci.* 48, 255–310. doi: 10.1016/S0022-0000(05)80004-2

Cai, S., and Zhang, X. (2022). "Deep cooperation of CDCL and local search for SAT (extended abstract)," in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, ed. L. D. Raedt, 5274–5278. doi: 10.24963/ijcai.2022/734

Cai, S., Zhang, X., Fleury, M., and Biere, A. (2022). Better decision heuristics in CDCL through local search and target phases. *J. Artif. Intell. Res.* 74, 1515–1563. doi: 10.1613/jair.1.13666

Calimeri, F., Perri, S., and Zangari, J. (2019). Optimizing answer set computation via heuristic-based decomposition. *Theory Pract. Log. Program.* 19, 603–628. doi: 10.1017/S1471068419000036

Callewaert, B., Vandevelde, S., and Vennekens, J. (2025). VERUS-LM: a versatile framework for combining LLMs with Symbolic reasoning. CoRR abs/2501.14540. doi: 10.48550/arXiv.2501.14540

Chen, J. (1999). Embedding prioritized circumscription in disjunctive logic programs. *J. Exp. Theor. Artif. Intell.* 11, 553–563. doi: 10.1080/095281399146427

Comploi-Taupe, R., Friedrich, G., Schekotihin, K., and Weinzierl, A. (2023). Domain-specific heuristics in answer set programming: a declarative non-monotonic approach. *J. Artif. Intell. Res.* 76, 59–114. doi: 10.1613/jair.1.14091

de Kleer, J., and Konolige, K. (1989). Eliminating the fixed predicates from a circumscription. *Artif. Intell.* 39, 391–398. doi: 10.1016/0004-3702(89)90018-0

Doherty, P., Lukaszewicz, W., and Szalas, A. (1997). Computing circumscription revisited: a reduction algorithm. *J. Autom. Reason.* 18, 297–336. doi: 10.1023/A:1005722130532

Dvořák, W., Järvisalo, M., Wallner, J. P., and Woltran, S. (2015). "Cegartix v0.4: a sat-based counter-example guided argumentation reasoning tool," in *International Competition on Computational Models of Argumentation (ICCMA)*.

Eiter, T., and Gottlob, G. (1993). Propositional circumscription and extended closed-world reasoning are IIp2-complete. *Theor. Comput. Sci.* 114, 231–245. doi: 10.1016/0304-3975(93)90073-3

Friedrich, G., Stumptner, M., and Wotawa, F. (1999). Model-based diagnosis of hardware designs. *Artif. Intell.* 111, 3–39. doi: 10.1016/S0004-3702(99)00034-X

Gebser, M., Kaminski, R., and Schaub, T. (2011). Complex optimization in answer set programming. *Theory Pract. Log. Program.* 11, 821–839. doi: 10.1017/S1471068411000329

Gelfond, M., and Lifschitz, V. (1988). "The stable model semantics for logic programming," in *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes)*, eds. R. A. Kowalski, and K. A. Bowen (Seattle, WA: MIT Press), 1070–1080.

Giunchiglia, E., and Maratea, M. (2006). "OPTSAT: a tool for solving SAT related optimization problems," in *Logics in Artificial Intelligence, 10th European Conference, JELIA 2006, volume 4160 of Lecture Notes in Computer Science*, eds. M. Fisher, W. van der Hoek, B. Konev, and A. Lisitsa (Liverpool: Springer), 485–489. doi: 10.1007/11853886_43

Hu, X., Jiang, Y., Pedrycz, W., Deng, Z., Gao, J., Tang, Y., et al. (2025). Automated cluster elimination guided by high-density points. *IEEE Trans. Cybern.* 55, 1717–1730. doi: 10.1109/TCYB.2025.3537108

Janhunen, T., and Oikarinen, E. (2004). "Capturing parallel circumscription with disjunctive logic programs," in *Logics in Artificial Intelligence, 9th European Conference, volume 3229 of Lecture Notes in Computer Science*, eds. J. J. Alferes, and J. A. Leite (Lisbon: Springer), 134–146. doi: 10.1007/978-3-540-30227-8_14

Janota, M., Grigore, R., and Marques-Silva, J. (2010). "Counterexample guided abstraction refinement algorithm for propositional circumscription," in *Logics in Artificial Intelligence - 12th European Conference, JELIA, volume 6341 of Lecture Notes in Computer Science*, eds. T. Janhunen, and I. Niemelä (Helsinki: Springer), 195–207. doi: 10.1007/978-3-642-15675-5_18

Koshimura, M., Nabeshima, H., Fujita, H., and Hasegawa, R. (2009). "Minimal model generation with respect to an atom set," in *Proceedings of the 7th International Workshop on First-Order Theorem Proving, FTP 2009, Volume 556 of CEUR Workshop Proceedings, Oslo, Norway*, eds. N. Peltier, and V. Sofronie-Stokkermans (Aachen: CEUR-WS).

Lee, J., and Lifschitz, V. (2003). "Loop formulas for disjunctive logic programs," in *ICLP, Volume 2916 of Lecture Notes in Computer Science*, ed. C. Palamidessi (Cham: Springer), 451–465. doi: 10.1007/978-3-540-24599-5_31

Lee, J., and Lin, F. (2006). Loop formulas for circumscription. *Artif. Intell.* 170, 160–185. doi: 10.1016/j.artint.2005.09.003

Lifschitz, V. (1985). "Computing circumscription," in *Proceedings of the 9th International Joint Conference on Artificial Intelligence, Volume 1*, ed. A. K. Joshi (Los Angeles, CA: Morgan Kaufmann), 121–127.

Lifschitz, V. (1986). "Pointwise circumscription: preliminary report," in *Proceedings of the 5th National Conference on Artificial Intelligence, Volume 1*, ed. T. Kehler (Philadelphia, PA: Morgan Kaufmann), 406–410.

Lin, F., and Zhao, Y. (2004). ASSAT: computing answer sets of a logic program by SAT solvers. *Artif. Intell.* 157, 115–137. doi: 10.1016/j.artint.2004.04.004

McCarthy, J. (1980). Circumscription - a form of non-monotonic reasoning. *Artif. Intell.* 13, 27–39. doi: 10.1016/0004-3702(80)90011-9

McCarthy, J. (1986). Applications of circumscription to formalizing common-sense knowledge. *Artif. Intell.* 28, 89–116. doi: 10.1016/0004-3702(86)90032-9

Metodi, A., Stern, R., Kalech, M., and Codish, M. (2014). A novel sat-based approach to model based diagnosis. *J. Artif. Intell. Res.* 51, 377–411. doi: 10.1613/jair.4503

Niskanen, A., and Järvisalo, M. (2020). "μ-toksia: an efficient abstract argumentation reasoner," in *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020*, eds. D. Calvanese, E. Erdem, and M. Thielscher (Rhodes), 800–804. doi: 10.24963/kr.2020/82

Ogunniye, G., and Kökciyan, N. (2023). A survey on understanding and representing privacy requirements in the internet-of-things. *J. Artif. Intell. Res.* 76, 163–192. doi: 10.1613/jair.1.14000

Oikarinen, E., and Janhunen, T. (2005). "circ2dlp - translating circumscription into disjunctive logic programming," in *Logic Programming and Nonmonotonic Reasoning, 8th International Conference, Volume 3662 of Lecture Notes in Computer Science*, eds. C. Baral, G. Greco, N. Leone, and G. Terracina (Diamante: Springer), 405–409. doi: 10.1007/11546207_36

Oikarinen, E., and Janhunen, T. (2008). "Implementing prioritized circumscription by computing disjunctive stable models," in *Artificial Intelligence: Methodology, Systems, and Applications, 13th International Conference, Volume 5253 of Lecture Notes in Computer Science*, eds. D. Dochev, M. Pistore, and P. Traverso (Varna: Springer), 167–180. doi: 10.1007/978-3-540-85776-1_15

Olausson, T., Gu, A., Lipkin, B., Zhang, C., Solar-Lezama, A., Tenenbaum, J., et al. (2023). "LINC: a neurosymbolic approach for logical reasoning by combining language models with first-order logic provers," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, eds. H. Bouamor, J. Pino, and K. Bali (Association for Computational Linguistics), 5153–5176. doi: 10.18653/v1/2023.emnlp-main.313

Pan, L., Albalak, A., Wang, X., and Wang, W. (2023). "Logic-LM: empowering large language models with symbolic solvers for faithful logical reasoning," *Findings of the Association for Computational Linguistics: EMNLP 2023*, eds. H. Bouamor, J. Pino, and K. Bali (Singapore: Association for Computational Linguistics), 3806–3824. doi: 10.18653/v1/2023.findings-emnlp.248

Przymusinski, T. C. (1989). An algorithm to compute circumscription. *Artif. Intell.* 38, 49–73. doi: 10.1016/0004-3702(89)90067-2

Rajasekharan, A., Zeng, Y., Padalkar, P., and Gupta, G. (2023). "Reliable natural language understanding with large language models and answer set programming," in *Proceedings 39th International Conference on Logic Programming, ICLP, Volume 385 of EPTCS*, eds. E. Pontelli, S. Costantini, C. Dodaro, S. A. Gaggl, R. Calegari, A. S. d'Avila Garcez, et al. (London: Imperial College London), 274–287. doi: 10.4204/EPTCS.385.27

Reiter, R. (1987). A theory of diagnosis from first principles. *Artif. Intell.* 32, 57–95. doi: 10.1016/0004-3702(87)90062-2

Reiter, R. (1988). "Chapter 12–Nonmonotonic reasoning," in *Exploring Artificial Intelligence*, eds. H. E. Shrobe, and the American Association for Artificial Intelligence (Burlington, MA: Morgan Kaufmann), 439–481. doi: 10.1016/B978-0-934613-67-5.50016-2

Ryu, H., Kim, G., Lee, H. S., and Yang, E. (2025). "Divide and translate: compositional first-order logic translation and verification for complex logical reasoning," in *The Thirteenth International Conference on Learning Representations* (Singapore: ICLR).

Sakama, C., and Inoue, K. (1995). "Embedding circumscriptive theories in general disjunctive programs," in *Logic Programming and Nonmonotonic Reasoning, Third International Conference, Volume 928 of Lecture Notes in Computer Science*, eds. V. W. Marek, and A. Nerode (Lexington, KY: Springer), 344–357. doi: 10.1007/3-540-59487-6_25

Sierra-Santibáñez, J. (2000). "Declarative formalization of strategies for action selection: applications to planning," in *Logics in Artificial Intelligence, European Workshop, JELIA 2000, volume 1919 of Lecture Notes in Computer Science*, eds. M. Ojeda-Aciego, I. P. de Guzmán, G. Brewka, and L. M. Pereira (Malaga: Springer), 133–147. doi: 10.1007/3-540-40006-0_10

Stern, R. T., Kalech, M., Feldman, A., and Provan, G. M. (2012). "Exploring the duality in conflict-directed model-based diagnosis," in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, eds. J. Hoffmann, and B. Selman (Toronto, ON: AAAI Press), 828–834. doi: 10.1609/aaai.v26i1.8231

Wakaki, T., and Satoh, K. (1995). "Computing prioritized circumscription by logic programming," in *Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming*, ed. L. Sterling (Tokyo: MIT Press), 283–297. doi: 10.7551/mitpress/4298.003.0034

Wakaki, T., and Satoh, K. (1997). "Compiling prioritized circumscription into extended logic programs," in *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, Volume 2* (Nagoya: Morgan Kaufmann), 182–189.

Wan, H., Xiao, Z., Yuan, Z., Zhang, H., and Zhang, Y. (2014). "Computing general first-order parallel and prioritized circumscription," in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, eds. C. E. Brodley, and P. Stone (Québec City: AAAI Press), 1105–1111. doi: 10.1609/aaai.v28i1.8860

Wang, Y., Eiter, T., Zhang, Y., and Lin, F. (2023). Witnesses for answer sets of logic programs. *ACM Trans. Comput. Logic* 24, 1–46. doi: 10.1145/3568955

Wang, Y., Wang, K., Wang, Z., and Zhuang, Z. (2015). "Knowledge forgetting in circumscription: a preliminary report," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, Volume 29*, eds. B. Bonet, and S. Koenig (Frisco, TX: AAAI Press), 1649–1655. doi: 10.1609/aaai.v29i1.9419

Wotawa, F., and Kaufmann, D. (2022). Model-based reasoning using answer set programming. *Appl. Intell.* 52, 16993–17011. doi: 10.1007/s10489-022-03272-2

Yang, Z., Ishay, A., and Lee, J. (2023). "Coupling large language models with logic programming for robust and general reasoning from text," in *Findings of the Association for Computational Linguistics: ACL 2023*, eds. A. Rogers, J. Boyd-Graber, and N. Okazaki (Toronto, ON: Association for Computational Linguistics), 5186–5219. doi: 10.18653/v1/2023.findings-acl.321

Ye, X., Chen, Q., Dillig, I., and Durrett, G. (2023). "SatLM: satisfiability-aided language models using declarative prompting," in *Advances in Neural Information Processing Systems*, eds. A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Curran Associates, Inc.), 4554845580. Available online at: https://proceedings.neurips.cc/paper_files/paper/2023/file/8e9c7d4a48bdac81a58f983a64aaf42b-Paper-Conference.pdf

Yuan, L., and Wang, C. H. (1988). "On reducing parallel circumscription," in *Proceedings of the 7th National Conference on Artificial Intelligence*, eds. H. E. Shrobe, T. M. Mitchell, and R. G. Smith (Paul, MN: AAAI Press/The MIT Press), 450–454.

Zeng, Y., Rajasekharan, A., Basu, K., Wang, H., Arias, J., Gupta, G., et al. (2024). A reliable common-sense reasoning socialbot built using LLMs and goal-directed ASP. *Theory Pract. Log. Program.* 24, 606–627. doi: 10.1017/S147106842400022X

Zhang, H., Zhang, Y., Ying, M., and Zhou, Y. (2011). "Translating first-order theories into logic programs," in *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, ed. T. Walsh (Barcelona: IJCAI/AAAI), 1126–1131.

Zhang, J., Fan, R., Tao, H., Jiang, J., and Hou, C. (2023). Constrained clustering with weak label prior. *Front. Comput. Sci.* 18:183338. doi: 10.1007/s11704-023-3355-7

Zhang, L., Wang, Y., Xie, Z., and Feng, R. (2021). Computing propositional minimal models: minisat-based approaches. *J. Comput. Res. Dev.* 58, 2515–2523. doi: 10.7544/issn1000-1239.2021.20200370