Syntactic Conditions for Antichain Property in Consistency Restoring Prolog

Vu Phan

May 27, 2018

Syntactic Conditions for Antichain Property in Consistency Restoring Prolog

Vu Phan
Rice University

submitted 15 April 2018; revised 27 May 2018; accepted 1 January 2003

Abstract

We study syntactic conditions which guarantee when a CR-Prolog (Consistency Restoring Prolog) program has antichain property: no answer set is a proper subset of another. A notable such condition is that the program's dependency graph being acyclic and having no directed path from one cr-rule head literal to another.

KEYWORDS: logic programming, answer set, dependency graph, proof of literal

1 Introduction

A-Prolog (Answer Set Prolog) is a programming language for knowledge representation and reasoning (Gelfond and Lifschitz 1988). An A-Prolog program comprises rules which determine the sets of beliefs that a logical agent can hold. A-Prolog relies on the stable model semantics of logic programs with negation.

A-Prolog has been applied to solve problems in various fields (Erdem et al. 2016). For instance, a logic program was used to guide multiple robots to collaboratively tidy up a house. Also, a tourism application suggested trips based on user preferences.

CR-Prolog (Consistency Restoring Prolog) extends A-Prolog with cr-rules (Balduccini and Gelfond 2003). Cr-rules apply only when regular rules alone would result in contradiction. Cr-rules are meant to represent rare exceptions.

CR-Prolog has also been utilized in several applications. For instance, CR-Prolog enables the space shuttle decision support system USA-Smart to find the most reasonable plans, even in the unlikely case of critical failures (Balduccini 2004). Another application of CR-Prolog is a formal encoding of negotiation, which is a multi-agent planning problem with incomplete information and dynamic goals (Son and Sakama 2009). Also, CR-Prolog is used as the back-end of the high-level domain representation of an architecture for knowledge representation and reasoning in robotics (Zhang et al. 2014). Yet one more application of CR-Prolog is the AIA architecture for intentional agents who observe and response to changing environments (Blount et al. 2014).

The first CR-Prolog inference engine is CR-MODELS, which was introduced in Balduccini (2007). Its efficiency is sufficient for medium-size programs, including an application developed for NASA. The second CR-Prolog implementation is SPARC, introduced in Balai et al. (2013). It implements a type system for the language using sort definitions.

In this paper, we investigate the antichain property that a logic program might have:

no answer set is a proper subset of another. Intuitively, a program is a specification for answer sets, which contain literals corresponding to beliefs to be held by an intelligent agent (Gelfond and Kahl 2014, pages 32-33). The formation of these answer sets adheres to some guidelines, including the rationality principle which tells reasoners to believe nothing they are not forced to believe. According to this principle, the antichain property is desirable: no logic program Π should have a chain of answer sets $S_1 \subsetneq S_2 \subsetneq \ldots$. If holding just the beliefs in S_1 suffices to satisfy the specification Π, then a reasoner should believe nothing in $S_2 \setminus S_1$.

All A-Prolog programs have antichain property, but some CR-Prolog programs do not. We look at syntactic conditions guaranteeing that a CR-Prolog program has this desired semantic property. A notable such condition – the primary achievement of this paper – is when the program's dependency graph is acyclic and has no directed path from one cr-rule head literal to another (Theorem 3.3.12). We will revisit a few known results in Section 2 and prove some new results in Section 3.

2 Preliminaries

The complete specifications of A-Prolog and CR-Prolog can be found in Gelfond and Kahl (2014, Sections 2.1 & 2.2 & 5.5). We also borrow some definitions from Ben-Eliyahu and Dechter (1994, Sections 1 & 2). In this paper, we only consider finite ground CR-Prolog programs whose abductive supports are minimal wrt (with respect to) cardinality.

2.1 Syntax

An **atom** a represents a boolean value. A **literal** is either an atom a or its **classical-negation** $\neg a$ (also called *strong negation*). An **extended literal** is either a literal l or its **default-negation** `not` l (also called *negation as failure*). Literal l **appears positive** in extended literal l and **appears negative** in extended literal `not` l.

A **regular rule** has the form:

$$l_1 \texttt{ or } \ldots \texttt{ or } l_k \longleftarrow l_{k+1}, \ldots, l_m, \texttt{ not } l_{m+1}, \ldots, \texttt{ not } l_n. \qquad (r)$$

Each l_i above is a literal. We assume $1 \leq k \leq m \leq n$.[1] When $k = m = n$, we call r a **fact**.

The **head** of a rule is the set of literals (disjuncts) before the arrow $\longleftarrow$. For instance, $\texttt{head}(r) = \{l_1, \ldots, l_k\}$. If R is a set of rules, $\texttt{head}(R) = \bigcup_{r \in R} \texttt{head}(r)$. If $k = 1$, r is **nondisjunctive**. A set R of rules is nondisjunctive if so are all rules in R.

The **body** of a rule comprises the extended literals after $\longleftarrow$ (**premises** of the rule). The **positive body** of a rule is the set of literals that appear positive in the body of the rule. For instance, $\texttt{body}_+(r) = \{l_{k+1}, \ldots, l_m\}$. If $m = n$, the rule is **default-negation-free**. A set R of rules is default-negation-free if so are all rules in R.

Similar to a regular rule, a **cr-rule** (consistency restoring rule)[2] has the form:

$$l_0 \stackrel{+}{\longleftarrow} l_1, \ldots, l_m, \texttt{ not } l_{m+1}, \ldots, \texttt{ not } l_n.$$

[1] Sometimes, $k = 0$ is allowed, and r becomes a *constraint*. But constraints can be equivalently translated to rules with $k > 0$. So this paper ignores constraints for simplicity.

[2] Cr-rules apply only when it would be inconsistent otherwise (more details in the following semantics subsection).

We call l_0 a **cr-literal**.

An **A-Prolog program** is a finite set of regular rules.

A **CR-Prolog program** Π is a finite set of regular rules and cr-rules. The **regular subprogram** Π^{reg} comprises the regular rules in Π. The **cr-subprogram** Π^{cr} comprises the cr-rules in Π.

The **application** $\alpha(r)$ of a cr-rule r is the regular rule obtained from r by replacing $\overset{+}{\longleftarrow}$ with $\longleftarrow$. If R is a set of cr-rules, $\alpha(R) = \{\alpha(r) : r \in R\}$.

2.2 Semantics

We now look into the formal definitions of answer sets and the antichain property. But first, a **context** is a subset of literals in a CR-Prolog program. Two literals are **complementary** if one is the classical-negation of the other. A context is **consistent** if it contains no pair of complementary literals.

Convention 2.2.1 (*Consistent Contexts*)
For simplicity, this paper assumes all contexts (mentioned in results) are consistent.

Now, a context S **satisfies**:

1. a literal l if $l \in S$
2. an extended literal `not` l if $l \notin S$
3. a regular rule head l_1 `or` ... `or` l_k if some $l_i \in S$
4. a regular rule body $l_{k+1}, \dots, l_m,$ `not` $l_{m+1}, \dots,$ `not` l_n if S satisfies all extended literals $l_{k+1}, \dots,$ `not` l_n (we say this rule **fires** wrt S in case of satisfaction)
5. a regular rule r if S satisfies the head of r whenever S satisfies the body of r
6. an A-Prolog program Π if S satisfies every rule in Π

Also, a literal l is **supported** by a regular rule r wrt a context S if r fires wrt S and `head`$(r) \cap S = \{l\}$.

Next, whether a context S is an answer set of an A-Prolog program Π is defined in two steps.

- Case Π is default-negation-free. Then S is an **answer set** of Π if: S satisfies Π, and S is minimal wrt set inclusion (no proper subset of S satisfies Π).
- Case Π is general. The **reduct** Π^S is the default-negation-free program obtained from Π by:
  - — removing all rules containing `not` l where literal $l \in S$ (since these rules do not fire wrt S), then
  - — from each remaining rule: deleting every extended literal containing `not` l (as $l \notin S$ now, so `not` l is satisfied and can be dropped from the premises of the rule)

  We say S is an **answer set** of Π if S is an answer set of Π^S. When Π has some answer set, we call Π **consistent**.

Next, we define answer sets of a CR-Prolog program Π.

- First, let $R \subseteq \Pi^{cr}$ (meaning R is a subset of cr-rules in Π). Then R is an **abductive support** of Π if:

— the A-Prolog program $\Pi^{reg} \cup \alpha(R)$ is consistent, and
— R is minimal wrt cardinality: no $R' \subseteq \Pi^{cr}$ exists where $|R'| < |R|$ such that $\Pi^{reg} \cup \alpha(R')$ is consistent

- Then a context S is an **answer set** of Π if S is an answer set of $\Pi^{reg} \cup \alpha(R)$ for some abductive support R of Π.

Example 2.2.2 (*Answer Sets of a CR-Prolog Program*)
We encode a hypothetical complexity result using the solver SPARC[3] (Balai et al. 2013):

```
p_eq_np :- sat_p.            % P = NP if SAT is in P              (regular rule)
p_eq_np :- knapsack_p.       % P = NP if Knapsack is in P
-p_eq_np :- not surprise.    % P != NP unless there is a surprise
surprise :+.                 % a surprise is unlikely             (cr-rule)
sat_p | knapsack_p.          % SAT or Knapsack is in P            (hypothetically)
```

SPARC returns exactly two answer sets (the cr-rule must apply to make the program consistent):

```
SPARC V2.52
program translated
{knapsack_p, p_eq_np, surprise}

{sat_p, p_eq_np, surprise}
```

We continue with a few more definitions. CR-Prolog programs Π_1 and Π_2 are **equivalent** when S is an answer set of Π_1 iff S is an answer set of Π_2 (for every context S). Finally, a CR-Prolog program Π has **antichain property** if: for all answer sets S_1 and S_2 of Π, we have $S_1 \subseteq S_2 \Rightarrow S_1 = S_2$. Some CR-Prolog programs do not have this property.

Example 2.2.3 (*A CR-Prolog Program Without Antichain Property*)
Consider the following program Π:

$$a \longleftarrow .$$

$$\neg a \longleftarrow \texttt{not}\ b,\ \texttt{not}\ c.$$

$$b \longleftarrow c. \quad (r_0)$$

$$b \stackrel{+}{\longleftarrow} . \quad (r_1)$$

$$c \stackrel{+}{\longleftarrow} . \quad (r_2)$$

Observe Π has an answer set chain $S_1 = \{a, b\} \subsetneq \{a, b, c\} = S_2$. (The corresponding abductive supports are $R_1 = \{r_1\}$ and $R_2 = \{r_2\}$.) Intuitively, the answer set chain is induced by the "dependence" of cr-literal b (from rule r_1) on cr-literal c (from rule r_2) in rule r_0 ("$b \longleftarrow c.$"). We will show that cr-independence guarantees antichain property, at least for acyclic programs such as Π, in Theorem 3.3.12. The terms *cr-independence* and *acyclicity* will be formally defined in Subsection 3.1.

2.3 Antichain A-Prolog

Every A-Prolog program is known to have antichain property; but for completeness, we will still provide a direct proof by Gelfond (2016).[4]

[3] `https://github.com/iensen/sparc`
[4] We thank the third referee for pointing out that this result can also be obtained from Lifschitz et al. (2001, Lemmas 1 & 2 & 3).

Lemma 2.3.1 (*Reduct Inclusion*)
Let Π be an A-Prolog program and S_1 & S_2 be contexts. If $S_1 \subseteq S_2$, then $\Pi^{S_2} \subseteq \Pi^{S_1}$.

Proof
Assume Π^{S_2} has an arbitrary default-negation-free rule r:

$$l_1 \texttt{ or } \ldots \texttt{ or } l_k \longleftarrow l_{k+1}, \ldots, l_m.$$

The corresponding rule in Π is:

$$l_1 \texttt{ or } \ldots \texttt{ or } l_k \longleftarrow l_{k+1}, \ldots, l_m, \texttt{ not } l_{m+1}, \ldots, \texttt{ not } l_n.$$

For each i in $\{m+1, \ldots, n\}$, we know $l_i \notin S_2$, so $l_i \notin S_1$ (as $S_1 \subseteq S_2$). Therefore, r is also a rule in Π^{S_1}. □

Proposition 2.3.2 (*Antichain Property of A-Prolog Programs*)
Let Π be an A-Prolog program and $S_1 \subseteq S_2$ be answer sets of Π. Then $S_1 = S_2$.

Proof
Let the reducts $\Pi_1 = \Pi^{S_1}$ and $\Pi_2 = \Pi^{S_2}$. Notice S_1 and S_2 are respectively answer sets of Π_1 and Π_2. By Lemma 2.3.1, $\Pi_2 \subseteq \Pi_1$. Then because S_1 satisfies Π_1, we know S_1 also satisfies Π_2. Now, being an answer set, S_2 minimally satisfies Π_2. So $S_2 \subseteq S_1$. Since $S_1 \subseteq S_2$ (hypothesis), we have $S_1 = S_2$. □

3 Results

We will proceed with the main contributions of this paper. Let us start by reviewing some concepts involving dependency graphs of logic programs (Ben-Eliyahu and Dechter 1994).

3.1 Dependency Graphs

In the **dependency graph** G_Π of a CR-Prolog program Π: every vertex is a literal in Π, and a directed edge to vertex l_1 from vertex l_2 exists iff Π has some rule r where literals $l_1 \in \texttt{head}(r)$ and $l_2 \in \texttt{body}_+(r)$. We say Π is **acyclic** if G_Π contains no directed cycle.

Remark 3.1.1 (*Answer Set of Acyclic A-Prolog Program*)
Let Π be an acyclicA-Prolog program and S be a context. Then S is an answer set of Π iff: S satisfies Π, and every literal in S is supported by a rule in Π wrt S (Ben-Eliyahu and Dechter 1994, Theorem 2.7, page 58).

Now, a **head-cycle** in the dependency graph G_Π of a CR-Prolog program Π is a directed cycle C containing vertices $l_1 \neq l_2$ such that there is a rule $r \in \Pi$ where literals $l_1, l_2 \in \texttt{head}(r)$ (Ben-Eliyahu and Dechter 1994, page 56). We say Π is **head-cycle-free** if G_Π contains no head-cycle. The class of head-cycle-free programs has several convenient properties that we will make use of later.

Also, literal l_1 **depends** on literal l_2 in a CR-Prolog program Π if the dependency graph G_Π has a directed path to l_1 from l_2. The following definition formalizes an important syntactic indicator of antichain property.

Definition 3.1.2 (*CR-Independence*)
A CR-Prolog program Π is called **cr-independent** if l_1 does not depend on l_2 for all cr-literals l_1 and l_2 in Π.

3.2 Abductive Supports

We continue with some technical lemmas related to abductive supports in CR-Prolog. Surprisingly, some of the following formal proofs are quite involved for their intuitive claims.

Lemma 3.2.1 (*Satisfying Context Intersection*)
Let Π be a nondisjunctive default-negation-free A-Prolog program. If contexts S_1 and S_2 satisfy Π, then context $S_0 = S_1 \cap S_2$ also satisfies Π.

Proof
Let r be a rule in Π. If r does not fire wrt S_0, then r is vacuously satisfied by S_0. Assume r fires wrt S_0. Then r also fires wrt the supersets S_1 and S_2 (as $r \in \Pi$ is default-negation-free). So S_1 and S_2 satisfy $\texttt{head}\,(r) = \{l\}$ for some literal l (recall $r \in \Pi$ is nondisjunctive). Thus $l \in S_1$ and $l \in S_2$. Hence $l \in S_0$. Therefore S_0 satisfies r. □

The following result was obtained by Gelfond (2016).

Lemma 3.2.2 (*Same-Head Rule Removal & Answer Set*)
Let Π be a nondisjunctive default-negation-free A-Prolog program. Assume Π has rules $r_1 \neq r_2$ such that $\texttt{head}\,(r_1) = \texttt{head}\,(r_2)$. Let $\Pi_0 = \Pi \setminus \{r_1, r_2\}$, $\Pi_1 = \Pi_0 \cup \{r_1\}$, and $\Pi_2 = \Pi_0 \cup \{r_2\}$. If S is an answer set of Π, then S is also an answer set of either Π_1 or Π_2.

Proof
To the contrary, assume S is an answer set of neither Π_1 nor Π_2. Still, S satisfies both Π_1 and Π_2 (as S satisfies their superset Π). So there exist two proper subsets of S, say S_1 and S_2, which respectively satisfy Π_1 and Π_2 (the programs are default-negation-free).

1. Case 1 of 2: either r_1 fires wrt S_1, or r_2 fires wrt S_2. Without loss of generality, assume the former. Then S_1 satisfies $\texttt{head}\,(r_1) = \texttt{head}\,(r_2)$. So S_1 also satisfies both the rule r_2 and the program $\Pi = \Pi_1 \cup \{r_2\}$. As an answer set, S minimally satisfies Π (default-negation-free). But $S_1 \subsetneq S$, contradiction.
2. Case 2 of 2: neither r_1 fires wrt S_1, nor r_2 fires wrt S_2. So neither r_1 nor r_2 fires wrt $S_0 = S_1 \cap S_2$ (the rules are default-negation-free). Then S_0 vacuously satisfies r_1 and r_2. Notice S_1 and S_2 satisfy Π_0 (subset of Π_1 and Π_2), then S_0 satisfies Π_0 too (by Lemma 3.2.1). Therefore, S_0 satisfies $\Pi = \Pi_0 \cup \{r_1, r_2\}$. But S is an answer set of Π, and $S_0 \subsetneq S$, contradiction.

□

Lemma 3.2.3 (*CR-Literal Determining CR-Rule*)
Let Π be a nondisjunctive CR-Prolog program with some abductive support R. For all cr-rules r_1 and r_2 in R: if $\texttt{head}\,(r_1) = \texttt{head}\,(r_2)$, then $r_1 = r_2$.

Proof

By way of contradiction, assume there exist cr-rules $r_1 \neq r_2$ in R where $\texttt{head}(r_1) = \texttt{head}(r_2)$. Let: $R_1 = R \setminus \{r_2\}$ & $R_2 = R \setminus \{r_1\}$ be sets of cr-rules; $\Pi_1 = \Pi^{reg} \cup \alpha(R_1)$ & $\Pi_2 = \Pi^{reg} \cup \alpha(R_2)$ be A-Prolog programs; S be an answer set of $\Pi^{reg} \cup \alpha(R)$; and $\Pi_a = (\Pi_1)^S$ & $\Pi_b = (\Pi_2)^S$ be (default-negation-free) reducts. By Lemma 3.2.2, S is an answer set of either Π_a or Π_b. Without loss of generality, assume the former. Then S is an answer set of Π_1. So R_1 is another abductive support of Π. But $|R_1| < |R|$ (recall $R_1 = R \setminus \{r_2\}$), violating the minimality of abductive support R. □

Lemma 3.2.4 (*CR-Literal only Supported by CR-Rule Application*)

Let Π be an acyclic CR-Prolog program having an answer set S with a corresponding abductive support R. Let cr-rule $r \in R$ where $\texttt{head}(r) = \{l\}$ for some literal l. Then $\alpha(r)$ is the only rule in $\Pi_R = \Pi^{reg} \cup \alpha(R)$ which supports l wrt S.

Proof

By way of contradiction, assume l is also supported by a rule $r' \neq \alpha(r)$ in Π_R. Let $R' = R \setminus \{r\}$ and $\Pi' = \Pi^{reg} \cup \alpha(R')$. We will prove S is an answer set of Π':

1. First, S satisfies $\Pi' \subseteq \Pi_R$.
2. Next, let l' be an arbitrary literal in S; we shall show l' is supported wrt S by some rule in Π'. Recall that S is an answer set of Π_R. Applying Remark 3.1.1 to Π_R, we deduce that l' is supported wrt S by some rule r_0 in Π_R.

   2.1. Case 1 of 2: $r_0 = \alpha(r)$. Recall $\texttt{head}(r) = \{l\}$. Then $l = l'$. Notice r' also supports $l' = l$ wrt S, and $r' \in \Pi'$.

   2.2. Case 2 of 2: $r_0 \neq \alpha(r)$. Then $r_0 \in \Pi'$ by construction.

In both cases, l' is supported by some rule in Π' wrt S.

Now, applying Remark 3.1.1 to Π', we deduce that S is an answer set of Π'. So Π' is consistent, and R' is an abductive support of Π. But $|R'| < |R|$, contradicting the minimality of abductive support R. □

Sometime, only the head of a rule matters semantically (but not its body), and we can turn it into a fact for syntactic simplicity.

Definition 3.2.5 (*Factified Rule*)

For a regular rule r, let $\texttt{fact}(r)$ denote the **factified rule** obtained from r by dropping the body of r. If R is a set of rules, define $\texttt{fact}(R) = \{\texttt{fact}(r) : r \in R\}$.

Lemma 3.2.6 (*Factified Abductive Support Application & Answer Set*)

Let Π be a CR-Prolog program with some answer set S and a corresponding abductive support R. Then S is also an answer set of the A-Prolog program $\Pi' = \Pi^{reg} \cup \texttt{fact}(\alpha(R))$.

Proof
We prove S is a minimal context which satisfies the reduct $(\Pi')^S$:

1. Let A-Prolog program $\Pi_R = \Pi^{reg} \cup \alpha(R)$. Recall S is an answer set of Π_R and thus satisfies the reduct $(\Pi_R)^S = (\Pi^{reg})^S \cup (\alpha(R))^S$. Since R is an abductive support for answer set S, we know $\mathtt{head}(R) \subseteq S$. Notice $\mathtt{head}(\mathtt{fact}(\alpha(R))) = \mathtt{head}(\alpha(R)) = \mathtt{head}(R)$. Then S satisfies $(\Pi')^S = (\Pi^{reg})^S \cup \mathtt{fact}(\alpha(R))$.
2. Assume some context $S' \subseteq S$ also satisfies $(\Pi')^S$. Since $\mathtt{fact}(\alpha(R))$ contains only facts, we know $\mathtt{head}(\alpha(R)) = \mathtt{head}(\mathtt{fact}(\alpha(R))) \subseteq S'$. Then S' satisfies $(\Pi_R)^S$. Recall S minimally satisfies $(\Pi_R)^S$, as S is an answer set of Π. So $S \subseteq S'$. Therefore $S' = S$.

□

Lemma 3.2.7 (*Same-Head Abductive Supports & Answer Set Inclusion/Equality*)
Let Π be a CR-Prolog program with answer sets $S_1 \subseteq S_2$ and corresponding abductive supports R_1 & R_2. If $\mathtt{head}(R_1) = \mathtt{head}(R_2)$, then $S_1 = S_2$.

Proof
By Lemma 3.2.6, S_1 and S_2 are respectively answer sets of $\Pi^{reg} \cup \mathtt{fact}(\alpha(R_1))$ and $\Pi^{reg} \cup \mathtt{fact}(\alpha(R_2))$, which are the same A-Prolog program because $\mathtt{head}(R_1) = \mathtt{head}(R_2)$. By Proposition 2.3.2, since $S_1 \subseteq S_2$, we have $S_1 = S_2$. □

3.3 Antichain Sufficient Condition: Acyclicity & CR-Independence

Next, we explore some concepts related to proofs of literals, which were introduced in Ben-Eliyahu and Dechter (1994). Then we will be ready to prove the primary result of the paper: Theorem 3.3.12.

Definition 3.3.1 (*Proof of Literal*)
Let Π be an A-Prolog program, S be a context, and l be a literal. A **proof** of l wrt S in Π is a nonempty sequence $p = \langle r_1, \ldots, r_n \rangle$ of rules in Π such that:

1. the head of each rule r_i has a literal supported by r_i wrt S; call this sole literal $\mathtt{h}_S(r_i)$
2. $l = \mathtt{h}_S(r_n)$
3. $\mathtt{body}_+(r_1) = \varnothing$
4. for every rule r_i, each literal in $\mathtt{body}_+(r_i)$ is $\mathtt{h}_S(r_j)$ for some $j < i$

In this definition, there is a caveat on criterion (3.). Details follow.

Note 3.3.2 (*Non-Fact as First Rule in Proof of Literal*)
In the original definition of proofs of literals, the first rule r_1 must be a fact (Ben-Eliyahu and Dechter 1994, page 57). However, that seems to be too strong. For instance, consider a head-cycle-free A-Prolog program Π containing a sole rule:

$$l \longleftarrow \mathtt{not}\ b. \qquad (r_1)$$

The only answer set is $S = \{l\}$. Now, every literal in an answer set of a head-cycle-free program has a proof (Ben-Eliyahu and Dechter 1994, Lemma B.5, page 83). So l has a proof wrt S in Π. The only candidate for such a proof is $p = \langle r_1 \rangle$. But r_1 is not a fact, so there is no proof of l according to the original definition, contradiction. In the adjusted Definition 3.3.1, p is a proof of l, since $\mathtt{body}_+(r_1) = \varnothing$. Additionally, all original results in Ben-Eliyahu and Dechter (1994) seem to still hold under this adjusted definition.

We continue with proofs of literals. For a proof $p = \langle r_1, \ldots, r_n \rangle$, let $\mathtt{h}_S(p)$ denote $\{\mathtt{h}_S(r) : r \in p\}$ and $\mathtt{body}_+(p)$ denote $\{\mathtt{body}_+(r) : r \in p\}$. Also, let $P(l, S, \Pi)$ denote the **set of all proofs** of a literal l wrt a context S in an A-Prolog program Π. A proof $p \in P(l, S, \Pi)$ is called a **minimal proof** if p is shortest: there is no proof $p' \in P(l, S, \Pi)$ where $|p'| < |p|$.

Convention 3.3.3 (*Distinct Rules in Proof of Literal*)
Let proof $p = \langle r_1, \ldots, r_n \rangle \in P(l, S, \Pi)$. As usual, each r_i is a rule, l is a literal, S is a context, and Π is an A-Prolog program. This paper assumes that the rules in p are pairwise distinct. Indeed, if there were rules $r_i = r_j$ where $i < j$, then $p' = \langle r_1, \ldots, r_{j-1}, r_{j+1}, \ldots, r_n \rangle \in P(l, S, \Pi)$ would readily be a shorter proof, and r_j would be obviously redundant.

Lemma 3.3.4 (*Proofs of Literals in Answer Set*)
If Π is a head-cycle-free A-Prolog program with an answer set S, then each literal in S has a proof wrt S in Π.

Proof
This lemma follows immediately from Ben-Eliyahu and Dechter (1994, Theorem 2.3, page 57). □

Intuitively, given an answer set S of an A-Prolog program, there may be an order on the literals of S that indicates which literal can be proven before another. The following concepts formalize this intuition.

The **rank** of a literal $l \in S$ wrt an answer set S in a head-cycle-free A-Prolog Π is the postive integer $\mathtt{rank}(l, S, \Pi) = \min\{|p| : p \in P(l, S, \Pi)\}$, which is the length of a minimal proof. Note that $\mathtt{rank}(l, S, \Pi)$ is well-defined, since proofs p of l wrt S in Π exist due to Lemma 3.3.4.

The **ranking function** wrt an answer set S in a head-cycle-free A-Prolog program Π is a function $\mathtt{f} : S \to \mathbb{Z}^+$ where $\mathtt{f}(l) = \mathtt{rank}(l, S, \Pi)$ for each literal $l \in S$. Note that $\mathtt{f}(l)$ is well-defined, as so is $\mathtt{rank}(l, S, \Pi)$.

Now, we introduce a normal proof of a literal. A proof can be "normal" in the sense that every literal a to be derived (from the head of a rule in the proof) has higher rank than each of its premise literals b (from the positive body of the same rule). Intuitively, a will be derived after b. The following definition is inspired by Ben-Eliyahu and Dechter (1994, Theorem 2.8, page 59).

Definition 3.3.5 (*Normal Proof of Literal*)
Let: Π be a head-cycle-free A-Prolog program with an answer set S; $\mathtt{f}$ be the ranking function wrt S in Π; and p be a proof of a literal $l \in S$ wrt S in Π. We say p is a **normal proof** if: for each rule $r \in p$ and each literal $l' \in \mathtt{body}_+(r)$, we have $\mathtt{f}(\mathtt{h}_S(r)) > \mathtt{f}(l')$.

The following desirable property of normal proofs will be needed later.

Remark 3.3.6 (*Normal Subproofs within Normal Proofs*)
Let: Π be a head-cycle-free A-Prolog program with an answer set S; $\mathtt{f}$ be the ranking function wrt S in Π; l be a literal in S; $p = \langle r_1, \ldots, r_n \rangle$ be a normal proof in $P(l, S, \Pi)$; and r_i be a rule in p. Then $p_i = \langle r_1, \ldots, r_i \rangle$ is a normal proof of $\mathtt{h}_S(r_i)$ wrt S in Π. We say p_i is a **subproof** within p.

Now, every minimal proof is a normal proof. But the next example justifies the need for normal proofs by showing that the "subproof transformation" does not preserve minimality (as it does normality in the previous remark).

Example 3.3.7 (*A Nonminimal Subproof within a Minimal Proof*)
Consider this acyclic A-Prolog program Π:

$$a \longleftarrow b, c. \tag{1}$$
$$b \longleftarrow c1x. \tag{2}$$
$$c \longleftarrow c1x. \tag{3}$$
$$c1x \longleftarrow c1y. \tag{4}$$
$$c1y \longleftarrow . \tag{5}$$
$$c \longleftarrow c2. \tag{6}$$
$$c2 \longleftarrow . \tag{7}$$

The sole answer set of Π is $S = \{a, b, c, c1x, c1y, c2\}$. The only minimal proofs of literal a wrt S in Π are the two sequences of rules $\langle (5), (4), (3), (2), (1) \rangle$ and $\langle (5), (4), (2), (3), (1) \rangle$. Within both of these proofs, the only subproof of c is $\langle (5), (4), (3) \rangle$, which is nonminimal. (The minimal proof of c wrt S in Π is $\langle (7), (6) \rangle$.)

Now, the following long technical lemma basically says: if $S_1 \subsetneq S_2$ are answer sets of A-Prolog programs Π_1 and Π_2, then the proofs of literals in $S_2 \setminus S_1$ contain rules in $\Pi_2 \setminus \Pi_1$.

Lemma 3.3.8 (*Answer Set Difference Literal Proven using Program Difference Rule*)
Let: Π_1 & Π_2 be head-cycle-free A-Prolog programs with corresponding answer sets $S_1 \subsetneq S_2$; l be a literal in $S_2 \setminus S_1$; and $p = \langle r_1, \ldots, r_n \rangle$ be a normal proof in $P(l, S_2, \Pi_2)$. Then there exists a rule $r \in p$ such that $r \in \Pi_2 \setminus \Pi_1$.

Proof
Let $\mathtt{f}$ be the ranking function wrt S_2 in Π_2. We employ induction on $\mathtt{f}(l)$.

- Base step: $\mathtt{f}(l) = \min\{\mathtt{f}(l_0) : l_0 \in S_2 \setminus S_1\}$.
  1. To the contrary, assume: for every rule $r \in p$, we have $r \in \Pi_1 \cap \Pi_2$.
  2. Then $r_n \in \Pi_1$.
  3. Since p is a normal proof of l, for each literal $l' \in \mathtt{body}_+(r_n)$, we have $\mathtt{f}(l') < \mathtt{f}(l) = \min\{\mathtt{f}(l_0) : l_0 \in S_2 \setminus S_1\}$. So $l' \in S_1 \cap S_2$.
  4. Then r_n fires wrt S_1 (recall: r_n fires wrt S_2, and $S_1 \subsetneq S_2$).
  5. As S_1 is an answer set of Π_1, we know S_1 satisfies $\mathtt{head}(r_n)$.
  6. Let l' be a literal in $\mathtt{head}(r_n)$.
     - 6.1. Case 1 of 2: $l' = l$. We have already assumed $l \in S_2 \setminus S_1$.
     - 6.2. Case 2 of 2: $l' \neq l$. We have $l' \notin S_2$ (as only l is supported by r_n wrt S_2 in Π_2), so $l' \notin S_1$.

     In both cases, $l' \notin S_1$. So S_1 does not satisfy $\mathtt{head}(r_n)$, contradiction.
- Inductive step: $\mathtt{f}(l) \leq \max\{\mathtt{f}(l_0) : l_0 \in S_2 \setminus S_1\}$.

1. Induction hypothesis: for each literal $l' \in S_2 \setminus S_1$, let p' be a normal proof in $P(l', S_2, \Pi_2)$; if $\mathtt{f}(l') < \mathtt{f}(l)$, then there exists a rule $r \in p'$ such that $r \in \Pi_2 \setminus \Pi_1$.
2. To the contrary, assume: for every rule $r \in p$, we have $r \in \Pi_1 \cap \Pi_2$.

2.1. Case 1 of 2: there exists a literal $l' \in \mathtt{body}_+(r_n)$ where $l' \in S_2 \setminus S_1$.

2.1.1. Notice $\mathtt{f}(l') < \mathtt{f}(l) = \mathtt{f}(\mathtt{h}_{S_2}(r_n))$.

2.1.2. Choose some positive integer $m < n$ where $\mathtt{h}_{S_2}(r_m) = l'$.

2.1.3. As $p = \langle r_1, \ldots, r_n \rangle$ is a normal proof in $P(l, S_2, \Pi_2)$, we know $p' = \langle r_1, \ldots, r_m \rangle$ is a normal subproof in $P(l', S_2, \Pi_2)$, by Remark 3.3.6.

2.1.4. By the induction hypothesis, p' contains some rule $r' \in \Pi_2 \setminus \Pi_1$.

2.1.5. So p also contains r'.

2.1.6. But we assumed $r \in \Pi_1 \cap \Pi_2$ for every rule $r \in p$, contradiction.

2.2. Case 2 of 2: for every literal $l' \in \mathtt{body}_+(r_n)$, we have $l' \in S_1 \cap S_2$.

2.2.1. Then the rule r_n fires wrt S_1.

2.2.2. By our assumption, $r_n \in \Pi_1$.

2.2.3. As S_1 is an answer set of Π_1, we know S_1 satisfies $\mathtt{head}(r_n)$.

2.2.4. Let l' be a literal in $\mathtt{head}(r_n)$.

2.2.4.1. Subcase 1 of 2: $l' = l$. We have already assumed $l \in S_2 \setminus S_1$.

2.2.4.2. Subcase 2 of 2: $l' \neq l$. We know $l' \notin S_2$ (as only l is supported by r_n wrt S_2 in Π_2), so $l' \notin S_1$.

In both subcases, $l' \notin S_1$. Then S_1 does not satisfy $\mathtt{head}(r_n)$, contradiction.

□

Remark 3.3.9 (*Normal/Minimal Proof of Literal & Dependence of Proven Literal*)
Let proof $p = \langle r_1, \ldots, r_n \rangle \in P(l, S, \Pi)$ for some literal l in an answer set S of an A-Prolog program Π. If p is a normal proof (or more specifically, a minimal proof), then l depends on $\mathtt{h}_S(r_i)$ for all $i < n$.

The following lemma asserts (equivalently) that cr-independence implies antichain property in certain cases.

Lemma 3.3.10 (*Answer Set Chain Implying CR-Dependence*)
Let Π be a nondisjunctive acyclic CR-Prolog program. If Π has answer sets $S_1 \subsetneq S_2$, then there exist literals l_1 and l_2 in $\mathtt{head}(\Pi^{cr})$ such that l_1 depends on l_2.

Proof
Some notations first:

1. By the contrapositive of Lemma 3.2.7, there exist abductive supports R_1 and R_2 (respectively corresponding to S_1 and S_2) where $\mathtt{head}(R_1) \neq \mathtt{head}(R_2)$.
2. Construct two sets of facts: $R'_1 = \mathtt{fact}(\alpha(R_1))$ and $R'_2 = \mathtt{fact}(\alpha(R_2))$.
3. Introduce A-Prolog programs $\Pi_1 = \Pi^{reg} \cup R'_1$ and $\Pi_2 = \Pi^{reg} \cup R'_2$. By Lemma 3.2.6, S_1 and S_2 are respectively answer sets of Π_1 and Π_2.

We follow these steps:

1. Note that Π_1 and Π_2 are nondisjunctive. By the contrapositive of Lemma 3.2.3, the cr-literals in R_1 are pairwise distinct. So are the cr-literals in R_2. Then $|R_1'| = |R_1|$ and $|R_2'| = |R_2|$.
2. Notice $|R_1| = |R_2| > 0$. Then $|R_1'| = |R_2'| > 0$.
3. Observe $|\texttt{head}\,(R_1')| = |R_1'|$ and $|\texttt{head}\,(R_2')| = |R_2'|$. Thus $|\texttt{head}\,(R_1')| = |\texttt{head}\,(R_2')| > 0$.
4. Recall $\texttt{head}\,(R_1') = \texttt{head}\,(R_1) \neq \texttt{head}\,(R_2) = \texttt{head}\,(R_2')$. Then $\texttt{head}\,(R_1') \setminus \texttt{head}\,(R_2') \neq \varnothing$.
5. Select some literal $l_1 \in \texttt{head}\,(R_1') \setminus \texttt{head}\,(R_2')$. Let r_1 be the fact "$l_1 \longleftarrow .$" in $R_1' \setminus R_2'$.
6. Since S_1 is an answer set of Π_1, we must have $l_1 \in S_1$. Recall $S_1 \subsetneq S_2$. Then $l_1 \in S_2$.
7. As S_2 is an answer set of Π_2, there exists a rule $r \in \Pi_2$ which supports l_1 wrt S_2. Note that $\texttt{body}_+\,(r) \subseteq S_2$.

   7.1. Case 1 of 2: there exists a literal $l \in \texttt{body}_+\,(r)$ where $l \in S_2 \setminus S_1$.

   7.1.1. Let p be a minimal proof in $P\,(l, S_2, \Pi_2)$.

   7.1.2. By Lemma 3.3.8, there exists a rule $r_2 \in p$ where $r_2 \in \Pi_2 \setminus \Pi_1$.

   7.1.3. Then $r_2 \in R_2' \setminus R_1' \subseteq \alpha\,(\Pi^{cr})$. Let literal $l_2 = \texttt{h}_{S_2}\,(r_2) \in \texttt{head}\,(\Pi^{cr})$.

   7.1.4. As p is a minimal proof, l depends on l_2, by Remark 3.3.9.

   7.1.5. Recall l_1 depends on l in r. By transitivity, l_1 depends on l_2.

   7.2. Case 2 of 2: $\texttt{body}_+\,(r) \subseteq S_1 \subsetneq S_2$. We show that this case is impossible.

   7.2.1. Subcase 1 of 2: $r \in \Pi_1 \cap \Pi_2$.

   7.2.1.1. Recall r supports l_1 wrt S_2. Since $\texttt{body}_+\,(r) \subseteq S_1 \subsetneq S_2$, we know r also supports l_1 wrt S_1.

   7.2.1.2. Applying Lemma 3.2.4 to Π_1, we have $r = r_1$.

   7.2.1.3. However, $r \in \Pi_2$ whereas $r_1 \in \Pi_1 \setminus \Pi_2$, contradiction.

   7.2.2. Subcase 2 of 2: $r \in \Pi_2 \setminus \Pi_1$.

   7.2.2.1. So $r \in R_2' \setminus R_1'$. Then r is the fact "$l_1 \longleftarrow .$", which is exactly r_1.

   7.2.2.2. However, we selected r_1 from $R_1' \setminus R_2'$ while $r \in R_2'$, contradiction.

□

Lemma 3.3.11 (*Equivalent Nondisjunctive Program*)
For every acyclic cr-independent CR-Prolog program Π, there is a nondisjunctive acyclic cr-independent program Π' equivalent to Π.

Proof
We will construct such a program Π'. Recall $\Pi = \Pi^{reg} \cup \Pi^{cr}$, the union of its regular subprogram and cr-subprogram. Assume Π^{reg} has an arbitrary rule:

$$l_1 \texttt{ or } \ldots \texttt{ or } l_k \longleftarrow l_{k+1}, \ldots, l_m, \texttt{ not } l_{m+1}, \ldots, \texttt{ not } l_n. \qquad (r)$$

We first build the nondisjunctive regular subprogram Π_0 of Π'. For each such rule $r \in \Pi^{reg}$, add the following set of k rules to Π_0:

$$\left.\begin{array}{l} l_1 \longleftarrow l_{k+1}, \ldots, l_m, \texttt{ not } l_{m+1}, \ldots, \texttt{ not } l_n, \texttt{ not } l_2, \texttt{ not } l_3, \ldots, \texttt{ not } l_k. \\ l_2 \longleftarrow l_{k+1}, \ldots, l_m, \texttt{ not } l_{m+1}, \ldots, \texttt{ not } l_n, \texttt{ not } l_1, \texttt{ not } l_3, \ldots, \texttt{ not } l_k. \\ \vdots \\ l_k \longleftarrow l_{k+1}, \ldots, l_m, \texttt{ not } l_{m+1}, \ldots, \texttt{ not } l_n, \texttt{ not } l_1, \texttt{ not } l_2, \ldots, \texttt{ not } l_{k-1}. \end{array}\right\} \qquad (R)$$

Then let $\Pi' = \Pi_0 \cup \Pi^{cr}$.

1. Firstly, Π' is nondisjunctive, as so are its subprograms Π_0 and Π^{cr} (every cr-rule head has exactly one literal).
2. Next, we show Π' is acyclic and cr-independent. The only syntactic difference between Π and Π' is that Π has arbitrary regular rules r whereas Π' has corresponding collections R of k rules. But r induces the same $k \cdot (m-k)$ directed edges as R does. So the dependency graphs $G_{\Pi} = G_{\Pi'}$. Then because Π is acyclic and cr-independent, so is Π'.
3. Now, we prove the equivalence between Π' and Π. Since Π^{reg} is acyclic, it is equivalent to Π_0, by Ben-Eliyahu and Dechter (1994, Theorem 4.17, page 73). Therefore $\Pi = \Pi^{reg} \cup \Pi^{cr}$ and $\Pi' = \Pi_0 \cup \Pi^{cr}$ are also equivalent.

□

At last, we are ready to prove the main result of this paper.

Theorem 3.3.12 (*Antichain Property of Acyclic CR-Independent CR-Prolog Programs*)
If a CR-Prolog program Π is acyclic and cr-independent, then Π has antichain property.

Proof
By Lemma 3.3.11, there exists a nondisjunctive acyclic cr-independent program Π' equivalent to Π. Now, Π' has antichain property, by the contrapositive of Lemma 3.3.10. Therefore, the equivalent original program Π has antichain property too. □

4 Conclusion

We have found a reasonably weak syntactic condition which guarantees that a CR-Prolog program has antichain property: acyclicity and cr-independence. We think most natural logic programs are acyclic and cr-independent. In order to induce cycles, a program would need to have circular reasoning in some sense, which is not very helpful for practical tasks. Being cr-dependent is uncommon as well. Given that cr-rules only apply in catastrophic situations (when the program would be inconsistent otherwise), a natural program would rarely specify that a cr-literal should also be derivable indirectly from another cr-literal via a longer path.

The future goal is to find weaker sufficient conditions to extend the class of CR-Prolog programs known to have antichain property. We thank the fourth referee for the suggestion to relax Theorem 3.3.12 by: either dropping acyclicity from the premises, or weakening it into head-cycle-freedom. So far, we have found no cyclic (with or without head-cycles) cr-independent program that has an answer set chain. Maybe cr-independence alone is sufficient for antichain property. This is a promising future research direction.

5 Acknowledgment

We sincerely appreciate the significant guidance of Prof. Michael Gelfond at Texas Tech University. This paper includes his proofs of Proposition 2.3.2 and Lemma 3.2.2. We also thank Dr. Evgenii Balai for the informative discussions. Finally, we are indebted to the four referees for their constructive feedback.

References

Balai, E., Gelfond, M., and Zhang, Y. 2013. SPARC: sorted ASP with consistency restoring rules. *arXiv preprint arXiv:1301.1386*.

Balduccini, M. 2004. USA-SMART: improving the quality of plans in answer set planning. In *International Symposium on Practical Aspects of Declarative Languages*. Springer, 135–147.

Balduccini, M. 2007. CR-MODELS: an inference engine for CR-Prolog. In *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer, 18–30.

Balduccini, M. and Gelfond, M. 2003. Logic Programs with Consistency-Restoring Rules. In *International Symposium on Logical Formalization of Commonsense Reasoning, AAAI 2003 Spring Symposium Series*. Vol. 102.

Ben-Eliyahu, R. and Dechter, R. 1994. Propositional Semantics for Disjunctive Logic Programs. *Annals of Mathematics and Artificial intelligence 12,* 1, 53–87.

Blount, J., Gelfond, M., and Balduccini, M. 2014. Towards a Theory of Intentional Agents. In *Knowledge Representation and Reasoning in Robotics. AAAI Spring Symp. Series*. 10–17.

Erdem, E., Gelfond, M., and Leone, N. 2016. Applications of Answer Set Programming. *AI Magazine 37,* 3, 53–68.

Gelfond, M. 2016. Personal Communication.

Gelfond, M. and Kahl, Y. 2014. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: the answer-set programming approach*. Cambridge University Press.

Gelfond, M. and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In *ICLP/SLP*. Vol. 88. 1070–1080.

Lifschitz, V., Pearce, D., and Valverde, A. 2001. Strongly equivalent logic programs. *ACM Transactions on Computational Logic (TOCL) 2,* 4, 526–541.

Son, T. C. and Sakama, C. 2009. Negotiation using Logic Programming with Consistency Restoring Rules. In *IJCAI*. Vol. 9. 930–935.

Zhang, S., Sridharan, M., Gelfond, M., and Wyatt, J. 2014. Towards an Architecture for Knowledge Representation and Reasoning in Robotics. In *International Conference on Social Robotics*. Springer, 400–410.