Syntactic Conditions for Antichain Property in Consistency Restoring Prolog

Vu Phan
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 \( \Pi \) should have a chain of answer sets \( S_1 \subset S_2 \subset \ldots \). If holding just the beliefs in \( S_1 \) suffices to satisfy the specification \( \Pi \), 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 \( \text{not } l \) (also called negation as failure). Literal \( l \) appears positive in extended literal \( l \) and appears negative in extended literal \( \text{not } l \).

A regular rule has the form:

\[
\begin{align*}
  l_1 \text{ or } \ldots \text{ or } l_k & \leftarrow l_{k+1}, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n. \\
\end{align*}
\]

Each \( l_i \) above is a literal. We assume \( 1 \leq k \leq m \leq n \). When \( k = m = n \), we call \( r \) a fact.

The head of a rule is the set of literals (disjuncts) before the arrow \( \leftarrow \). For instance, \( \text{head}(r) = \{l_1, \ldots, l_k\} \). If \( R \) is a set of rules, \( \text{head}(R) = \bigcup_{r \in R} \text{head}(r) \). 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 \( \leftarrow \) (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, \( \text{body}_+ (r) = \{l_{k+1}, \ldots, l_m\} \). 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) has the form:

\[
\begin{align*}
  l_0 & \leftarrow l_1, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n. \\
\end{align*}
\]

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 $\Pi$ is a finite set of regular rules and cr-rules. The regular subprogram $\Pi_{reg}$ comprises the regular rules in $\Pi$. The cr-subprogram $\Pi_{cr}$ comprises the cr-rules in $\Pi$.

The application $\alpha(r)$ of a cr-rule $r$ is the regular rule obtained from $r$ by replacing $\leftarrow^+$ with $\leftarrow$. 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 $\text{not} \ l$ if $l \notin S$
3. a regular rule head $l_1 \lor \ldots \lor l_k$ if some $l_i \in S$
4. a regular rule body $l_{k+1}, \ldots, l_m$, $\text{not} \ l_{m+1}, \ldots, \text{not} \ l_n$ if $S$ satisfies all extended literals $l_{k+1}, \ldots, \text{not} \ l_n$ (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 $\Pi$ if $S$ satisfies every rule in $\Pi$

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

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

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

We say $S$ is an answer set of $\Pi$ if $S$ is an answer set of $\Pi^S$. When $\Pi$ has some answer set, we call $\Pi$ consistent.

Next, we define answer sets of a CR-Prolog program $\Pi$.

- First, let $R \subseteq \Pi_{cr}$ (meaning $R$ is a subset of cr-rules in $\Pi$). Then $R$ is an abductive support of $\Pi$ if:
— the A-Prolog program $\Pi^{\text{reg}} \cup \alpha(R)$ is consistent, and
— $R$ is minimal wrt cardinality: no $R' \subseteq \Pi^{\text{cr}}$ exists where $|R'| < |R|$ such that $\Pi^{\text{reg}} \cup \alpha(R')$ is consistent

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

Example 2.2.2 (Answer Sets of a CR-Prolog Program)
We encode a hypothetical complexity result using the solver SPARC

\begin{verbatim}
3 https://github.com/iensen/sparc

3 We thank the third referee for pointing out that this result can also be obtained from Lifschitz et al. (2001, Lemmas 1 & 2 & 3).

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).

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)\textsuperscript{3}.
Lemma 2.3.1 (Reduct Inclusion)
Let \( \Pi \) 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 \( \Pi^{S_2} \) has an arbitrary default-negation-free rule \( r \):
\[
l_1 \lor \ldots \lor l_k \leftarrow l_{k+1}, \ldots, l_m.
\]
The corresponding rule in \( \Pi \) is:
\[
l_1 \lor \ldots \lor l_k \leftarrow l_{k+1}, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n.
\]
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 \( \Pi^{S_1} \).

Proposition 2.3.2 (Antichain Property of A-Prolog Programs)
Let \( \Pi \) be an A-Prolog program and \( S_1 \subseteq S_2 \) be answer sets of \( \Pi \). 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 \( \Pi_1 \) and \( \Pi_2 \). By Lemma 2.3.1, \( \Pi_2 \subseteq \Pi_1 \). Then because \( S_1 \) satisfies \( \Pi_1 \), we know \( S_1 \) also satisfies \( \Pi_2 \). Now, being an answer set, \( S_2 \) minimally satisfies \( \Pi_2 \). 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 \( G_{\Pi} \) of a CR-Prolog program \( \Pi \): every vertex is a literal in \( \Pi \), and a directed edge to vertex \( l_1 \) from vertex \( l_2 \) exists ifff \( \Pi \) has some rule \( r \) where literals \( l_1 \in \text{head}(r) \) and \( l_2 \in \text{body}_+(r) \). We say \( \Pi \) is acyclic if \( G_{\Pi} \) contains no directed cycle.

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

Now, a head-cycle in the dependency graph \( G_{\Pi} \) of a CR-Prolog program \( \Pi \) is a directed cycle \( C \) containing vertices \( l_1 \neq l_2 \) such that there is a rule \( r \in \Pi \) where literals \( l_1, l_2 \in \text{head}(r) \) (Ben-Eliyahu and Dechter 1994, page 56). We say \( \Pi \) is head-cycle-free if \( G_{\Pi} \) 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 \( \Pi \) if the dependency graph \( G_{\Pi} \) 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 $\Pi$ is called **cr-independent** if $l_1$ does not depend on $l_2$ for all cr-literals $l_1$ and $l_2$ in $\Pi$.

### 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 $\Pi$ be a nondisjunctive default-negation-free A-Prolog program. If contexts $S_1$ and $S_2$ satisfy $\Pi$, then context $S_0 = S_1 \cap S_2$ also satisfies $\Pi$.

**Proof**
Let $r$ be a rule in $\Pi$. 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 $\text{head}(r) = \{l\}$ 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 $r$. \qed

The following result was obtained by Gelfond (2016).

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

**Proof**
To the contrary, assume $S$ is an answer set of neither $\Pi_1$ nor $\Pi_2$. Still, $S$ satisfies both $\Pi_1$ and $\Pi_2$ (as $S$ satisfies their superset $\Pi$). So there exist two proper subsets of $S$, say $S_1$ and $S_2$, which respectively satisfy $\Pi_1$ and $\Pi_2$ (the programs are default-negation-free). But $S_1 \not\subseteq S$, contradiction.

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 $\text{head}(r_1) = \text{head}(r_2)$. 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 $\Pi$ (default-negation-free). But $S_1 \not\subseteq S$, 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 $\Pi_0$ (subset of $\Pi_1$ and $\Pi_2$), then $S_0$ satisfies $\Pi_0$ 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 $\Pi$, and $S_0 \not\subseteq S$, contradiction. \qed

**Lemma 3.2.3 (CR-Literal Determining CR-Rule)**
Let $\Pi$ be a nondisjunctive CR-Prolog program with some abductive support $R$. For all cr-rules $r_1$ and $r_2$ in $R$: if $\text{head}(r_1) = \text{head}(r_2)$, then $r_1 = r_2$. 
Proof
By way of contradiction, assume there exist cr-rules \( r_1 \neq r_2 \) in \( R \) where \( \text{head}(r_1) = \text{head}(r_2) \). Let: \( R_1 = R \setminus \{ r_2 \} \) & \( R_2 = R \setminus \{ r_1 \} \) be sets of cr-rules; \( \Pi_1 = \Pi^{\text{reg}} \cup \alpha(R_1) \) & \( \Pi_2 = \Pi^{\text{reg}} \cup \alpha(R_2) \) be A-Prolog programs; \( S \) be an answer set of \( \Pi^{\text{reg}} \cup \alpha(R) \); 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 \( \Pi_a \) or \( \Pi_b \). Without loss of generality, assume the former. Then \( S \) is an answer set of \( \Pi_1 \). So \( R_1 \) is another abductive support of \( \Pi \). 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 \( \Pi \) be an acyclic CR-Prolog program having an answer set \( S \) with a corresponding abductive support \( R \). Let cr-rule \( r \in R \) where \( \text{head}(r) = \{ l \} \) for some literal \( l \). Then \( \alpha(r) \) is the only rule in \( \Pi R = \Pi^{\text{reg}} \cup \alpha(R) \) which supports \( l \) wrt \( S \).

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

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 \( \Pi' \). Recall that \( S \) is an answer set of \( \Pi R \). Applying Remark 3.1.1 to \( \Pi R \), we deduce that \( l' \) is supported wrt \( S \) by some rule \( r_0 \) in \( \Pi R \).

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

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 \( \Pi' \) wrt \( S \).

Now, applying Remark 3.1.1 to \( \Pi' \), we deduce that \( S \) is an answer set of \( \Pi' \). So \( \Pi' \) is consistent, and \( R' \) is an abductive support of \( \Pi \). 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 \( \text{fact}(r) \) denote the factified rule obtained from \( r \) by dropping the body of \( r \). If \( R \) is a set of rules, define \( \text{fact}(R) = \{ \text{fact}(r) : r \in R \} \).

Lemma 3.2.6 (Factified Abductive Support Application & Answer Set)
Let \( \Pi \) 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 \( \Pi' = \Pi^{\text{reg}} \cup \text{fact}(\alpha(R)) \).
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 \( \Pi_R \) 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 \( \text{head}(R) \subseteq S \). Notice \( \text{head}(\text{fact}(\alpha(R))) = \text{head}(\alpha(R)) \). Then \( S \) satisfies \((\Pi')^S = (\Pi^{reg})^S \cup \text{fact}(\alpha(R))\).

2. Assume some context \( S' \subseteq S \) also satisfies \((\Pi')^S\). Since \( \text{fact}(\alpha(R)) \) contains only facts, we know \( \text{head}(\alpha(R)) = \text{head}(\text{fact}(\alpha(R))) \subseteq S' \). Then \( S' \) satisfies \((\Pi_R)^S\). Recall \( S \) minimally satisfies \((\Pi_R)^S\), as \( S \) is an answer set of \( \Pi \). So \( S \subseteq S' \). Therefore \( S' = S \).

\( \Box \)

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

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

\( \Box \)

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 \( \Pi \) be an A-Prolog program, \( S \) be a context, and \( l \) be a literal. A proof of \( l \) wrt \( S \) in \( \Pi \) is a nonempty sequence \( p = \langle r_1, \ldots, r_n \rangle \) of rules in \( \Pi \) such that:

1. the head of each rule \( r_i \) has a literal supported by \( r_i \) wrt \( S \); call this sole literal \( h_S(r_i) \)
2. \( l = h_S(r_n) \)
3. \( \text{body}_+(r_1) = \emptyset \)
4. for every rule \( r_i \), each literal in \( \text{body}_+(r_i) \) is \( h_S(r_j) \) 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 \( \Pi \) containing a sole rule:

\[ l \leftarrow \text{not } b. \]

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 \( \Pi \). 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 \( \text{body}_+(r_1) = \emptyset \). 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 \( h_S (p) \) denote \( \{ h_S (r) : r \in p \} \) and \( \text{body}_+ (p) \) denote \( \{ \text{body}_+ (r) : r \in p \} \). 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 \( \Pi \). 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 \( \Pi \) 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 \( p' = \langle r_1, \ldots, r_i, r_{i+1}, \ldots, r_n \rangle \in P (l, S, \Pi) \) would readily be a shorter proof, and \( r_j \) would be obviously redundant.

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

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

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

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: \( \Pi \) be a head-cycle-free A-Prolog program with an answer set \( S \); \( f \) be the ranking function wrt \( S \) in \( \Pi \); and \( p \) be a proof of a literal \( l \in S \) wrt \( S \) in \( \Pi \). We say \( p \) is a normal proof if: for each rule \( r \in p \) and each literal \( l' \in \text{body}_+ (r) \), we have \( f (h_S (r)) > f (l') \).

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

**Remark 3.3.6 (Normal Subproofs within Normal Proofs)**
Let: \( \Pi \) be a head-cycle-free A-Prolog program with an answer set \( S \); \( f \) be the ranking function wrt \( S \) in \( \Pi \); \( l \) 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 \( h_S (r_i) \) wrt \( S \) in \( \Pi \). 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:

\[
\begin{align*}
  a &\leftarrow b, c. \\
  b &\leftarrow c1x. \\
  c &\leftarrow c1x. \\
  c1x &\leftarrow c1y. \\
  c1y &\leftarrow . \\
  c &\leftarrow . \\
  c2 &\leftarrow .
\end{align*}
\]

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 \(\langle 5 \rangle, \langle 4 \rangle, \langle 3 \rangle, \langle 1 \rangle, \langle 6 \rangle \rangle\) and \(\langle 5 \rangle, \langle 4 \rangle, \langle 2 \rangle, \langle 3 \rangle, \langle 1 \rangle, \langle 6 \rangle \rangle\). Within both of these proofs, the only subproof of \(c\) is \(\langle 5 \rangle, \langle 4 \rangle, \langle 3 \rangle\), which is nonminimal. (The minimal proof of \(c\) wrt \(S\) in Π is \(\langle 7 \rangle, \langle 6 \rangle \rangle\).)

Now, the following long technical lemma basically says: if \(S_1 \nsubseteq S_2\) are answer sets of A-Prolog programs \(\Pi_1\) and \(\Pi_2\), 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: \(\Pi_1 \& \Pi_2\) be head-cycle-free A-Prolog programs with corresponding answer sets \(S_1 \subseteq S_2\); \(l\) be a literal in \(S_2 \setminus S_1\); and \(p = (r_1, \ldots, r_n)\) 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 \(f\) be the ranking function wrt \(S_2\) in \(\Pi_2\). We employ induction on \(f(l)\).

- **Base step:** \(f(l) = \min \{f(l_0) : l_0 \in S_2 \setminus S_1\}\).
  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 \(l' \in \text{body}_+(r_n)\), we have \(f(l') < f(l) = \min \{f(l_0) : l_0 \in S_2 \setminus S_1\}\). So \(l' \in S_1 \cap S_2\).
  4. Then \(r_n\) fires wrt \(S_1\) (recall: \(r_n\) fires wrt \(S_2\), and \(S_1 \nsubseteq S_2\)).
  5. As \(S_1\) is an answer set of \(\Pi_1\), we know \(S_1\) satisfies \(\text{head}(r_n)\).
  6. Let \(l'\) be a literal in \(\text{head}(r_n)\).
    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 \(\Pi_2\)), so \(l' \notin S_1\).

In both cases, \(l' \notin S_1\). So \(S_1\) does not satisfy \(\text{head}(r_n)\), contradiction.

- **Inductive step:** \(f(l) \leq \max \{f(l_0) : l_0 \in S_2 \setminus S_1\}\).
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 \( \not\exists (l') \prec \not\exists (l) \), 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 \( l' \in \text{body}_+ (r_n) \) where \( l' \in S_2 \setminus S_1 \).

2.1.1. Notice \( \not\exists (l') \prec \not\exists (l) = \not\exists (h_{S_2} (r_n)) \).

2.1.2. Choose some positive integer \( m < n \) where \( h_{S_2} (r_n) = l' \).

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 \( l' \in \text{body}_+ (r_n) \), 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 \( \Pi_1 \), we know \( S_1 \) satisfies \( \text{head} (r_n) \).

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

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 \( \Pi_2 \)), so \( l' \notin S_1 \).

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

\[ \square \]

**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 \( \Pi \). If \( p \) is a normal proof (or more specifically, a minimal proof), then \( l \) depends on \( h_S (r_i) \) 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 \( \Pi \) be a nondisjunctive acyclic CR-Prolog program. If \( \Pi \) has answer sets \( S_1 \subseteq S_2 \), then there exist literals \( l_1 \) and \( l_2 \) in \( \text{head} (\Pi^c) \) 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 \( \text{head} (R_1) \neq \text{head} (R_2) \).

2. Construct two sets of facts: \( R'_1 = \text{fact} (\alpha (R_1)) \) and \( R'_2 = \text{fact} (\alpha (R_2)) \).

3. Introduce A-Prolog programs \( \Pi_1 = \Pi^c \cup R'_1 \) and \( \Pi_2 = \Pi^c \cup R'_2 \). By Lemma 3.2.6, \( S_1 \) and \( S_2 \) are respectively answer sets of \( \Pi_1 \) and \( \Pi_2 \).

We follow these steps:
1. Note that Π₁ and Π₂ are nondisjunctive. By the contrapositive of Lemma 3.2.3, the cr-literals in R₁ are pairwise distinct. So are the cr-literals in R₂. Then |R'₁| = |R₁| and |R'₂| = |R₂|.

2. Notice |R₁| = |R₂| > 0. Then |R'₁| = |R'₂| > 0.

3. Observe |head (R'₁)| = |R'₁| and |head (R'₂)| = |R'₂|. Thus |head (R'₁)| = |head (R'₂)| > 0.

4. Recall head (R'₁) = head (R₁) ≠ head (R₂) = head (R'₂). Then head (R'₁) \ head (R'₂) ≠ ∅.

5. Select some literal l₁ ∈ head (R'₁) \ head (R'₂). Let r₁ be the fact “l₁ ← .” in R'₁ \ R'₂.

6. Since S₁ is an answer set of Π₁, we must have l₁ ∈ S₁. Recall S₁ ⊆ S₂. Then l₁ ∈ S₂.

7. As S₂ is an answer set of Π₂, there exists a rule r ∈ Π₂ which supports l₁ wrt S₂. Note that body⁺ (r) ⊆ S₂.

7.1. Case 1 of 2: there exists a literal l ∈ body⁺ (r) where l ∈ S₂ \ S₁.

7.1.1. Let p be a minimal proof in P (l, S₂, Π₂).

7.1.2. By Lemma 3.3.8, there exists a rule r₂ ∈ p where r₂ ∈ Π₂ \ Π₁.

7.1.3. Then r₂ ∈ R₂ \ R₁ ⊆ Π (Π²⁺). Let literal l₂ = h₂ (r₂) ∈ head (Π²⁺).

7.1.4. As p is a minimal proof, l depends on l₂, by Remark 3.3.9.

7.1.5. Recall l₁ depends on l in r. By transitivity, l₁ depends on l₂.

7.2. Case 2 of 2: body⁺ (r) ⊆ S₁ ⊆ S₂. We show that this case is impossible.

7.2.1. Subcase 1 of 2: r ∈ Π₁ \ Π₂.

7.2.1.1. Recall r supports l₁ wrt S₂. Since body⁺ (r) ⊆ S₁ ⊆ S₂, we know r also supports l₁ wrt S₁.

7.2.1.2. Applying Lemma 3.2.4 to Π₁, we have r = r₁.

7.2.1.3. However, r ∈ Π₂ whereas r₁ ∈ Π₁ \ Π₂, contradiction.

7.2.2. Subcase 2 of 2: r ∈ Π₂ \ Π₁.

7.2.2.1. So r ∈ R₂ \ R₁. Then r is the fact “l₁ ← .”, which is exactly r₁.

7.2.2.2. However, we selected r₁ from R₁ \ R₂ while r ∈ R₂, contradiction.

\[ \begin{array}{l}
\text{Lemma 3.3.11 (Equivalent Nondisjunctive Program)} \\
\text{For every acyclic cr-independent CR-Prolog program Π, there is a nondisjunctive acyclic cr-independent program Π' equivalent to Π.}
\end{array} \]

\textbf{Proof}

We will construct such a program Π'. Recall Π = Π⁺⁺ ∪ Π⁺⁻, the union of its regular subprogram and cr-subprogram. Assume Π⁺⁺ has an arbitrary rule:

\[ l₁ \text{ or } \ldots \text{ or } l_k ← l_{k+1}, \ldots, l_m, \text{ not } l_{m+1}, \ldots, \text{ not } l_n. \]  

(r)

We first build the nondisjunctive regular subprogram Π₀ of Π'. For each such rule r ∈ Π⁺⁺, add the following set of k rules to Π₀:

\[ l₁ ← l_{k+1}, \ldots, l_m, \text{ not } l_{m+1}, \ldots, \text{ not } l_n, \text{ not } l_2, \text{ not } l_3, \ldots, \text{ not } l_k. \]

\[ l₂ ← l_{k+1}, \ldots, l_m, \text{ not } l_{m+1}, \ldots, \text{ not } l_n, \text{ not } l_1, \text{ not } l_3, \ldots, \text{ not } l_k. \]

\[ \vdots \]

\[ l_k ← l_{k+1}, \ldots, l_m, \text{ not } l_{m+1}, \ldots, \text{ not } l_n, \text{ not } l_1, \text{ not } l_2, \ldots, \text{ not } l_{k-1}. \]

(R)

Then let Π' = Π₀ ∪ Π⁺⁻.
1. Firstly, $\Pi'$ is nondisjunctive, as so are its subprograms $\Pi_0$ and $\Pi^{cr}$ (every cr-rule head has exactly one literal).

2. Next, we show $\Pi'$ is acyclic and cr-independent. The only syntactic difference between $\Pi$ and $\Pi'$ is that $\Pi$ has arbitrary regular rules $r$ whereas $\Pi'$ 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 $\Pi$ is acyclic and cr-independent, so is $\Pi'$.

3. Now, we prove the equivalence between $\Pi'$ and $\Pi$. Since $\Pi^{reg}$ is acyclic, it is equivalent to $\Pi_0$, 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 $\Pi$ is acyclic and cr-independent, then $\Pi$ has antichain property.

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


