Syntactic Conditions for Antichain Property in Consistency-Restoring Prolog

Vu H. N. Phan
Rice University (2017–2022)
Texas Tech University (2014–2017)
https://vuphan314.github.io/

2018-07-18 (University of Oxford, England)
Workshop on Answer Set Programming and Other Computing Paradigms
affiliated with International Conference on Logic Programming
part of Federated Logic Conference

Contents

Section 1

Introduction

Introduction

Logic-programming languages:

1. Answer-Set Prolog (A-Prolog): standard
   {[1] Gelfond and Lifschitz 1988 "The Stable Model Semantics for Logic Programming"}
2. Consistency-Restoring Prolog (CR-Prolog): extension with CR-rules (for rare exceptions)
   {[2] Balduccini and Gelfond 2003 "Logic Programs with Consistency-Restoring Rules"}

Motivation

Informal semantics:

1. Program: a specification for answer sets

$$a \texttt{ or } b.$$

2. Answer set: a set of beliefs

$$S_1 = \{a\}$$
$$S_2 = \{b\}$$

3. Rationality principle: fewer beliefs are better

$$S_0 = \{a, b\} \qquad \text{(irrational)}$$

4. Antichain property: no answer set is a proper subset of another

Main Contribution

Syntactic conditions guaranteeing antichain property:

1. CR-independence
   (dependency graph has no path from one CR-rule head literal to another)
2. acyclicity
   (dependency graph has no cycle)

Section 2

Preliminaries

Preliminaries

Syntax & semantics:

1. A-Prolog (Answer-Set Prolog)
2. CR-Prolog (Consistency-Restoring Prolog)

{[3] Gelfond and Kahl 2014 *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: the Answer Set Programming Approach*}

A-Prolog Syntax

1. **Atom**:

   a (*a* is believed to be true)

2. **Literal**:
   1. atom: a
   2. **classical-negation**:

      $\neg a$ (*a* is believed to be false)

3. **Extended literal**:
   1. literal: $a, \neg a$
   2. **default-negation**:

      $\texttt{not } a$ (*a* is not believed to be true)

      $\texttt{not } \neg a$ (*a* is not believed to be false)

A-Prolog Syntax

1. **Rule**:

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

   1. **Rule Head**: the set of literals before $\longleftarrow$
   2. **Rule Body**: the set of extended literals after $\longleftarrow$

2. **Program**: a set of rules

A-Prolog Semantics

1. **Context**: a subset of literals in a program

$$\{a, \neg b\}$$

2. **Complementary literals**:

$$a$$

$$\neg a$$

3. **Consistent context**: no complementary literals
4. *Convention:* contexts are consistent (from now on)

A-Prolog Semantics

Context $\{a, c, e\}$ **satisfies**:

1. literal: a
2. extended literal: $\texttt{not}\ d$
3. rule head: $a\ \texttt{or}\ b$
4. rule body: $c,\ \texttt{not}\ d$
5. rules:

$$a\ \texttt{or}\ b \longleftarrow c,\ \texttt{not}\ d. \tag{1}$$

$$b \longleftarrow \texttt{not}\ e. \tag{2}$$

6. program: $\{(1), (2)\}$

A-Prolog Semantics

1. Program Π:

$$\begin{aligned} b &\longleftarrow . && (r_1) \\ a &\longleftarrow \texttt{not } b. && (r_2) \\ c &\longleftarrow \texttt{not } d. && (r_3) \end{aligned}$$

2. Context $S = \{b, c\}$
3. **Reduct** Π^S: default-negation-free program

$$\begin{aligned} b &\longleftarrow . && (r_1) \\ c &\longleftarrow . && (r_3') \end{aligned}$$

4. Context S:
   1. satisfies reduct Π^S
   2. has no proper subset that satisfies reduct Π^S
5. Context S is an **answer set** of program Π

A-Prolog Semantics

1. **Consistent program**: having an answer set
2. *Example:* inconsistent program

$$a \longleftarrow . \tag{1}$$

$$\neg a \longleftarrow \texttt{not}\ b,\ \texttt{not}\ c. \tag{2}$$

CR-Prolog Syntax

1. A-Prolog regular rule:

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

2. CR-Prolog **CR-rule** (consistency-restoring rule):

$$l_0 \xleftarrow{+} l_1, \dots, l_m, \texttt{ not } l_{m+1}, \dots, \texttt{ not } l_n.$$

   - **CR-literal**: l_0

3. **CR-Prolog program**: a set of regular rules & CR-rules

CR-Prolog Semantics

1. CR-Prolog program Π:

$$a \longleftarrow . \tag{1}$$

$$\neg a \longleftarrow \texttt{not } b, \texttt{ not } c. \tag{2}$$

$$b \xleftarrow{+} . \tag{3}$$

$$c \xleftarrow{+} . \tag{4}$$

2. An **abductive support**: $R_1 = \{(3)\}$
   1. Π_{R_1}: A-Prolog program under R_1-application

$$a \longleftarrow . \tag{1}$$

$$\neg a \longleftarrow \texttt{not } b, \texttt{ not } c. \tag{2}$$

$$b \longleftarrow . \tag{3'}$$

   2. Context $S_1 = \{a, b\}$: an answer set of Π_{R_1}, so an **answer set** of Π
3. Another abductive support: $R_2 = \{(4)\}$; corresponding answer set: $S_2 = \{a, c\}$

Section 3

Results

Results

1. Antichain property: desirable semantic feature
2. Dependency graphs: syntactic abstractions of programs
3. Main antichain guarantee: CR-independence & acyclicity

Antichain Property

Program has **antichain property** if: no answer set is a proper subset of another

1. All A-Prolog programs have antichain property
2. Some CR-Prolog programs do not have antichain property

Antichain Property

Example: CR-Prolog program without antichain property

$$a \longleftarrow . \tag{1}$$

$$\neg a \longleftarrow \ \texttt{not}\ b,\ \texttt{not}\ c. \tag{2}$$

$$b \overset{+}{\longleftarrow} . \tag{3}$$

$$c \overset{+}{\longleftarrow} . \tag{4}$$

$$b \longleftarrow c. \tag{5}$$

1. Abductive supports: $R_1 = \{(3)\}$ & $R_2 = \{(4)\}$
2. Answer set chain: $S_1 = \{a, b\} \subsetneq \{a, b, c\} = S_2$
3. In (5): "dependence" of CR-literal b from (3) on CR-literal c from (4)

Dependency Graphs

1. CR-Prolog program:

$$a \ \mathtt{or} \ b \longleftarrow c, d, \ \mathtt{not} \ e. \qquad (1)$$

$$x \stackrel{+}{\longleftarrow} y. \qquad (2)$$

2. **Dependency graph**:
   1. Vertices: a, b, c, d, e, x, y
   2. Directed edges: from positive body to head of each rule

$$c \mapsto a$$

$$c \mapsto b$$

$$d \mapsto a$$

$$d \mapsto b$$

$$y \mapsto x$$

Dependency Graphs: CR-Independence

1. CR-Prolog program:

$$a \longleftarrow x. \tag{1}$$

$$x \overset{+}{\longleftarrow} b. \tag{2}$$

2. Literal *a* **depends** on literal *b* if: dependency graph has path from *b* to *a*
3. Program is **CR-independent** if: no CR-literal depends on another

Example: CR-dependent program

$$a \longleftarrow . \tag{1}$$

$$\neg a \longleftarrow \texttt{not}\ b,\ \texttt{not}\ c. \tag{2}$$

$$b \stackrel{+}{\longleftarrow} . \tag{3}$$

$$c \stackrel{+}{\longleftarrow} . \tag{4}$$

$$b \longleftarrow c. \tag{5}$$

Answer set chain: $S_1 = \{a, b\} \subsetneq \{a, b, c\} = S_2$

Dependency Graphs: Acyclicity

1. Cycle in dependency graph:

$$a \longleftarrow b. \tag{1}$$
$$b \longleftarrow a. \tag{2}$$

2. CR-Prolog program is **acyclic** if: dependency graph contains no cycle
3. *Remark:* context S is answer set of acyclic A-Prolog program Π if:
    1. S satisfies Π
    2. for each literal $l \in S$, some rule $r \in \Pi$ exists where:
        1. S satisfies $\texttt{body}\,(r)$
        2. $\texttt{head}\,(r) \cap S = \{l\}$

{[4] Ben-Eliyahu and Dechter 1994 "Propositional Semantics for Disjunctive Logic Programs"}

Main Antichain Guarantee

Theorem

A CR-Prolog program Π has antichain property (no answer set is a proper subset of another) if Π is:

1. *CR-independent (no path from a CR-literal to another), and*
2. *acyclic (no cycle).*

Section 4

Conclusion

Summary

1. CR-Prolog: A-Prolog extended with consistency-restoring rules
2. Desirable antichain property: no answer set is a proper subset of another
3. Sufficient syntactic conditions:
    1. CR-independence
    2. acyclicity