Notes on Answer Set Programming

1 Traditional Programs and their Answer Sets

The definition of an answer set was originally proposed as a semantics for Prolog programs with negation, and later extended to more general logic programs. We begin by considering the case of “traditional” programs, which are essentially Prolog programs without variables.

1.1 Syntax

A traditional rule is an expression of the form

\[ A_0 \leftarrow A_1, \ldots, A_m, \text{not } A_{m+1}, \ldots, \text{not } A_n \]  \hspace{1cm} (1)

where \( n \geq m \geq 0 \) and \( A_0, \ldots, A_n \) are propositional atoms. The atom \( A_0 \) is called the head of the rule, and the list \( A_1, \ldots, A_m, \text{not } A_{m+1}, \ldots, \text{not } A_n \) is its body. If the body is empty \( (n = 0) \) then the rule is called a fact and identified with its head \( A_0 \).

A traditional program is a finite set of traditional rules. For instance,

\[ p \leftarrow p, q \]  \hspace{1cm} (2)

and

\[ p \leftarrow \text{not } q \\
q \leftarrow \text{not } r \]  \hspace{1cm} (3)

are traditional programs.

A traditional rule (1) is positive if \( m = n \), that is to say, if it has the form

\[ A_0 \leftarrow A_1, \ldots, A_m. \]  \hspace{1cm} (4)

A traditional program is positive if each of its rules is positive. For instance, program (2) is positive, and (3) is not.
1.2 The Answer Set of a Positive Program

We will first define the concept of an answer set for positive traditional programs. We need auxiliary definitions. A set \( X \) of atoms satisfies a positive traditional rule (4) if \( A_0 \in X \) whenever \( \{A_1, \ldots, A_m\} \subseteq X \). For instance, any positive traditional rule (4) is satisfied by its head \( A_0 \).

**Problem 1.** (5pt) Given a set \( X \) of atoms and a positive traditional rule (4) \( \exists X \) satisfies rule (4) ?

\[
\begin{array}{|c|c|}
\hline
\text{Does } X \text{ satisfies rule (4) ?} & \text{A}_0 \in X \text{ and } \{A_1, \ldots, A_m\} \subseteq X \\
\hline
\text{A}_0 \in X \text{ and } \{A_1, \ldots, A_m\} \not\subseteq X \\
\hline
\text{A}_0 \not\in X \text{ and } \{A_1, \ldots, A_m\} \subseteq X \\
\hline
\text{A}_0 \not\in X \text{ and } \{A_1, \ldots, A_m\} \not\subseteq X \\
\hline
\end{array}
\]

A set \( X \) of atoms satisfies a positive traditional program \( \Pi \) if \( X \) satisfies every rule (4) in \( \Pi \). For instance, any positive traditional program is satisfied by the set of the heads \( A_0 \) of all its rules (4).

**Problem 2.** (5pt) \( \exists X \) satisfies program (2) ?

\[
\begin{array}{|c|c|}
\hline
\text{Does } X \text{ satisfies program (2) ?} & X \\
\hline
\emptyset \\
\{p\} \\
\{q\} \\
\{r\} \\
\{p,q\} \\
\{p,r\} \\
\{q,r\} \\
\{p,q,r\} \\
\hline
\end{array}
\]

**Problem 3.** (10pt) For any positive traditional program \( \Pi \), the intersection of all sets satisfying \( \Pi \) satisfies \( \Pi \) also.

The assertion in Problem 3 allows us to talk about the smallest set of atoms that satisfies \( \Pi \) [van Emden and Kowalski1976]. This set is called the answer set of \( \Pi \).

For instance, the sets of atoms satisfying program (2) are

\( \{p\}, \{p,r\}, \{p,q,r\} \),

and its answer set is \( \{p\} \).
Problem 4. (10pt) If $X$ is an answer set of a positive traditional program $\Pi$, then every element of $X$ is the head of one of the rules of $\Pi$.

Intuitively, we can think of (4) as a rule for generating atoms: once you have generated $A_1, \ldots, A_m$, you are allowed to generate $A_0$. The answer set is the set of all atoms that can be generated by applying rules of the program in any order. For instance, the first rule of (2) allows us to include $p$ in the answer set. The second rule says that we can add $r$ to the answer set if we have already included $p$ and $q$. Given these two rules only, we can generate no atoms besides $p$. If we extend program (2) by adding the rule $q \leftarrow p$ then the answer set will become $\{p, q, r\}$.

1.3 Answer Sets of a Program with Negation

To extend the definition of an answer set to arbitrary traditional programs, we will introduce one more auxiliary definition. The reduct $\Pi^X$ of a traditional program $\Pi$ relative to a set $X$ of atoms is the set of rules (4) for all rules (1) in $\Pi$ such that $A_{m+1}, \ldots, A_n \notin X$. Thus $\Pi^X$ is a positive traditional program.

Problem 5. (5pt) Let $\Pi$ be (3),

\begin{align*}
X & | & \text{What is } \Pi^X \text{?} & | & \text{Explanation} \\
\emptyset & | & p \leftarrow & | & p \leftarrow \text{not-}\ q \\
& | & q \leftarrow & | & q \leftarrow \text{not-}\ r \\
\{p\} & | & p \leftarrow & | & p \leftarrow \text{not-}\ q \\
& | & q \leftarrow & | & q \leftarrow \text{not-}\ r \\
\{q\} & | & q \leftarrow & | & p \leftarrow \text{not-}\ q \\
& | & & | & q \leftarrow \text{not-}\ r \\
\{r\} & | & & & \\
\{p\ q\} & | & & & \\
\{p\ r\} & | & & & \\
\{q\ r\} & | & & & \\
\{p\ q\ r\} & | & & & \\
\end{align*}

We say that $X$ is an answer set of $\Pi$ if $X$ is the answer set of $\Pi^X$ (that is, the smallest set of atoms satisfying $\Pi^X$).
Problem 6. (5pt)

<table>
<thead>
<tr>
<th>$X$</th>
<th>Is $X$ an answer set of program (3)?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td></td>
</tr>
<tr>
<td>${p}$</td>
<td></td>
</tr>
<tr>
<td>${q}$</td>
<td></td>
</tr>
<tr>
<td>${r}$</td>
<td></td>
</tr>
<tr>
<td>${p, q}$</td>
<td></td>
</tr>
<tr>
<td>${p, r}$</td>
<td></td>
</tr>
<tr>
<td>${q, r}$</td>
<td></td>
</tr>
<tr>
<td>${p, q, r}$</td>
<td></td>
</tr>
</tbody>
</table>

The definition of an answer set was proposed in [Gelfond and Lifschitz1988], where answer sets were called “stable models.” Its idea came from early work on nonmonotonic reasoning [McDermott and Doyle1980] and [Reiter1980], which was related to logic programming in [Gelfond1987].

If $\Pi$ is positive then, for any $X$, $\Pi^X = \Pi$. It follows that the new definition of an answer set is a generalization of the definition from Section 1.2: for any positive traditional program $\Pi$, $X$ is the smallest set of atoms satisfying $\Pi^X$ iff $X$ is the smallest set of atoms satisfying $\Pi$.

Intuitively, rule (1) allows us to generate $A_0$ as soon as we generated the atoms $A_1, \ldots, A_m$ provided that none of the atoms $A_{m+1}, \ldots, A_n$ can be generated using the rules of the program. There is a vicious circle in this sentence: to decide whether a rule of $\Pi$ can be used to generate a new atom, we need to know which atoms can be generated using the rules of $\Pi$. The definition of an answer set overcomes this difficulty by employing a “fixpoint construction.” Take a set $X$ that you suspect may be exactly the set of atoms that can be generated using the rules of $\Pi$. Under this assumption, $\Pi$ has the same meaning as the positive program $\Pi^X$. Consider the answer set of $\Pi^X$, as defined in Section 1.2. If this set is exactly identical to the set $X$ that you started with then $X$ was a “good guess”; it is indeed an answer set of $\Pi$.

In Problem 6 we established that $\{q\}$ is the only answer set of program (3): we checked each of the subsets of $\{p, q, r\}$ to establish this fact. The following general properties of answer sets of traditional programs allow one to establish such conclusions by verifying fewer sets of atoms.

Problem 7. (10pt) If $X$ is an answer set of a traditional program $\Pi$ then every element of $X$ is the head of one of the rules of $\Pi$. 
Problem 8 (%1). (20pt) (a) For any traditional program \( \Pi \) and any sets \( X, Y \) of atoms, if \( X \subseteq Y \) then \( \Pi^Y \subseteq \Pi^X \). (b) If \( X \) is an answer set for a traditional program \( \Pi \) then no proper subset of \( X \) can be an answer set of \( \Pi \).

In application to program (3), the assertion of Problem 7 tells us that its answer sets do not contain \( r \), so that we only need to check \( \emptyset \) and \( \{p, q\} \). By the assertion of Problem 7(b), \( \emptyset \) cannot be an answer set because it is a proper subset of the answer set \( \{q\} \), and \( \{p, q\} \) cannot be an answer set because the answer set \( \{q\} \) is its proper subset. Consequently, \( \{q\} \) is the only answer set of (3).

Program (3) has a unique answer set. On the other hand, the program

\[
\begin{align*}
p & \leftarrow \text{not } q \\
q & \leftarrow \text{not } p
\end{align*}
\]
(5)

has two answer sets: \( \{p\} \) and \( \{q\} \). The one-rule program

\[
r \leftarrow \text{not } r
\]
(6)

has no answer sets.

Problem 9. (5pt) Find all answer sets of the following program, which extends (5) by two additional rules:

\[
\begin{align*}
p & \leftarrow \text{not } q \\
q & \leftarrow \text{not } p \\
r & \leftarrow p \\
r & \leftarrow q.
\end{align*}
\]

Problem 10. (5pt) Find all answer sets of the following combination of programs (5) and (6):

\[
\begin{align*}
p & \leftarrow \text{not } q \\
q & \leftarrow \text{not } p \\
r & \leftarrow \text{not } r \\
r & \leftarrow p.
\end{align*}
\]

1.4 Prolog vs. Answer Set Programming

The definition of an answer set for traditional programs can be viewed as a possible definition of a “correct” answer to a query in Prolog.

\(^{1}\)Problems marked by * are for extra credit
Let II be a Prolog program without variables (or the set of ground rules obtained from a Prolog program with variables by replacing each rule by all its ground instances). If II is a traditional program in the sense of Section 1.1, and it has a unique answer set \( X \), then the correct answer to a ground query \( A \) is *yes* or *no* depending on whether \( A \) belongs to \( X \) or not. For instance, given program (3), a Prolog system is supposed to answer *yes* to query \( q \) and *no* to queries \( p \) and \( r \).

From this perspective, a program with several answer sets, such as (5), is “bad”—it does not provide an unambiguous specification for the behavior of a Prolog system. Programs without answer sets, such as (6), are “bad” also.

In answer set programming, on the other hand, programs without a unique answer set are quite useful. The idea of ASP is to represent the search problem we are interested in as the problem of finding an answer set of some logic program, and then find a solution using an answer set solver—a system for generating answer sets. ASP programs with several answer sets correspond to search problems that have several solutions. ASP programs without answer sets correspond to problems that have no solutions.

The concept of an answer set is only one of several definitions of the semantics of Prolog with negation proposed in the literature. Some of the others are program completion [Clark1978], iterated fixpoints [Apt et al.1988] and well-founded models [Van Gelder et al.1991].
2 Answer Set Programming

Answer set programming (ASP) [Marek and Truszczyński1999, Niemelä1999] is a declarative programming formalism based on the answer set semantics of logic programs [Gelfond and Lifschitz1988, Gelfond and Lifschitz1991]. The idea of ASP is to represent a given computational problem by a program whose answer sets correspond to solutions, and then use an answer set solver to generate answer sets for this program.

In this course we will use the answer set system clingo\(^2\) that incorporates answer set solver clasp\(^2\) [Gebser et al.2007] with its front-end ( grounder) gringo\(^2\) (user guide [Gebser et al.2010]). System clingo is currently one of the most widely used answer set solvers.

A common methodology to solve a problem in ASP is to design GENERATE, DEFINE, and TEST [Lifschitz2002] parts of a program. The GENERATE part defines a large collection of answer sets that could be seen as potential solutions. The TEST part consists of rules that eliminate the answer sets of the GENERATE part that do not correspond to solutions. The DEFINE section expresses additional concepts and connects the GENERATE and TEST parts.

In addition to Prolog-like rules such as rule (1), GRINGO also accepts rules of other kinds — “choice rules” and “constraints”. For example, rule

\[
\{p, q, r\}.
\]

is a choice rule. Answer sets of this one-rule program are arbitrary subsets of the atoms \(p, q, r\). Choice rules are typically the main members of the GENERATE part of the program. Constraints often form the TEST section of a program. Syntactically, a constraint is the rule with an empty head. It encodes the conditions on the answer sets that have to be met. For instance, the constraint

\[
\leftarrow p, \text{not } q.
\]

eliminates the answer sets of a program that include \(p\) and do not include \(q\).

System GRINGO allows the user to specify large programs in a compact way, using rules with schematic variables and other abbreviations. A detailed description of its input language can be found in the online manual [Gebser et al.2010]. Grounder GRINGO takes a program “with abbreviations” as an input and produces its propositional counterpart that is then processed by CLASP. The system CLINGO can be used as a shortcut for invoking both of these systems at once.

\(^2\)http://potassco.sourceforge.net/.
3 Example: N-Queens

The goal is to place $n$ queens on an $n \times n$ chessboard so that no two queens would be placed on the same row, column, and diagonal. A solution can be described by a set of atoms of the form $q(i,j)$ $(1 \leq i, j \leq n)$; including $q(i,j)$ in the set indicates that there is a queen at position $(i,j)$. A solution is a set $X$ satisfying the following conditions:

1. the cardinality of $X$ is $n$,
2. $X$ does not contain a pair of different atoms of the form $q(i,j), q(i',j)$ (two queens on the same row),
3. $X$ does not contain a pair of different atoms of the form $q(i,j), q(i,j')$ (two queens on the same column),
4. $X$ does not contain a pair of different atoms of the form $q(i,j), q(i',j')$ with $|i' - i| = |j' - j|$ (two queens on the same diagonal).

Here is the representation of this program in the input language of CLINGO/GRINGO:

```
number(1..n).
#domain number(I).
#domain number(I1).
#domain number(J).
#domain number(J1).

%Condition 1 and 2
1\{q(K,J): number(K)\}1.

%Condition 3
:-q(I,J), q(I,J1), J<J1.

%Condition 4
:-q(I,J), q(I1,J1), J<J1, #abs(I1-I)==J1-J.
```

The command line

```
clingo -c n=4 queens.gr
```

instructs the answer set system CLINGO to find a single solution for 4-queens problem. Alternatively, the command line

```
8
```
gringo -c n=4 queens.gr > queens.4.grounded

instructs the grounder GRINGO to ground 4-queens problem; the ground problem (ready for processing with clasp) is stored in file queens.4.grounded.

The command line

gringo -t -c n=4 queens.gr

will produce human-readable grounded 4-queens problem.

The command line

clasp < queens.4.grounded

will instruct the answer set solver clasp to look for an answer set of program queens.4.grounded.

The command line

clingo -c n=8 queens.gr 0

instructs CLINGO to find all solutions for 8-queens problem. An extract from the output of the last command line follows

... 
Answer: 92
number(1) number(2) number(3) number(4)
number(5) number(6) number(7) number(8)
q(5,8) q(7,7) q(2,6) q(6,5) q(3,4) q(1,3) q(8,2) q(4,1)
SATISFIABLE

This ninety second solution found by the solver encodes the following valid configuration of queens on the board

1 2 3 4 5 6 7 8
1 Q
2 Q
3 Q
4Q
5 Q
6 Q
7 Q
8 Q

Problem 11. (10pt) Use CLINGO to find all solutions to the 8 queens problem that (a) have a queen at (1,1); (b) have no queens in the 4 × 4 square in the middle of the board.
Problem 12. (10pt) Report times used by gringo, clasp individually to process a program to find a single solution for 60, 100, 140, 180, 220 queens problem. Also report sizes of ground programs produced by gringo.

Acknowledgments

Parts of these notes follow the lecture notes on Answer Sets; and Methodology of Answer Set Programming; course Answer set programming: CS395T, Spring 2005 by Vladimir Lifschitz. The introduction to answer set programming followed the lines of [Lierler and Schüller2012].

References


http://www.cs.utexas.edu/~vl/teaching/asp.html


