Chapter 4

Answer Set Programming

Using Answer Set Programming (ASP) makes it possible to describe a computational problem as a logic program whose answer sets correspond to the solutions of the given problem. In this chapter we define the syntax of an Answer Set program and the related notations that will be used in this work. We also present an answer set approach used to restore consistency called CR-Programs.

In situations where we have a set of desiderata that we would like to satisfy and not all of them can be simultaneously satisfied, the use of preferences to resolve those indeterminate situations results very natural and effective. Then applying preferences among desiderata we may obtain an appropriate solution. Currently, there are a large number of approaches dealing with preferences [12]. In this chapter we studied Logic Programs with Ordered Disjunction [7]. The intuition behind Logic Programs with Ordered Disjunction is to express default knowledge with knowledge about preferences.

4.1 Answer Set Programming framework

We want to stress the fact that in our approach, *a program is interpreted as a propositional theory* and that the only negation used is *default negation*. For readers not familiar with this approach, we recommend [38, 33] for further reading. Hence, we will restrict our discussion to propositional programs.

We shall use *the language of propositional logic* in the usual way, using:

- propositional symbols: $p, q, \ldots,$
- propositional connectives: $\wedge, \vee, \rightarrow, \bot$, and
- auxiliary symbols: $(,)$.

A *well formed propositional formula* is inductively defined as follows: A propositional symbol is a well formed propositional formula. If f and g are well formed propositional formulas, then so are $\neg f$, $f \wedge g$, $f \vee g$, $f \rightarrow g$. A well formed propositional formula can be just called $formula$.

An *atom* is a propositional symbol.

Given any well formed propositional formula f, we assume that $\neg f$ is just an abbreviation of $f \rightarrow \bot$ and $\top$ is an abbreviation of $\bot \rightarrow \bot$. We remark that the only negation used in this work is *default negation* and it is represented by the symbol $\neg$. It is worth mentioning that we always can handle the other negation called *classical* or even *strong* negation, denoted by $-$, by transforming the atoms with classical negation [15]. Each atoms with classical negation, $-a$, that occurs in a formula is replaced by a new atom, a°, and the rule $\neg(a \wedge a^{\circ})$ is added. Rule $\neg(a \wedge a^{\circ})$ can also be written as $(a \wedge a^{\circ}) \rightarrow \bot$ based on the assumption that $\neg f$ is just an abbreviation of $f \rightarrow \bot$.

Example 4.1. Let P be the following program:

$$\begin{array}{rcl} buyBanana & \leftarrow & \neg - eatBanana. \\ -eatBanana & \leftarrow & bananaIsRotten. \end{array}$$

Then, transforming the atoms with classical negation the program P is the following:

$$\begin{aligned} buyBanana &\leftarrow \neg eatBanana^{\circ}. \\ eatBanana^{\circ} &\leftarrow bananaIsRotten. \\ &\leftarrow eatBanana^{\circ}, eatBanana. \end{aligned}$$

□

A *signature* $\mathcal{L}$ is a finite set of atoms. The *signature of a program* P, denoted as $\mathcal{L}_P$, is the set of atoms that occur in P.

Example 4.2. Let P_1 be the following program:

$$\begin{aligned} p &\leftarrow \neg q. \\ q&. \end{aligned}$$

Then $\mathcal{L}_{P_1} = \{p, q\}$. □

A *literal* is either an atom p (a positive literal) or the negation of an atom $\neg p$ (a negative literal).

A *negated literal* is the negation sign $\neg$ followed by any literal, i.e. $\neg p$ or $\neg\neg p$.

In particular, $f \to \bot$ is called *constraint* and it is also denoted as $\leftarrow f$.

A *regular theory* or *logic program* is just a finite set of clauses, it can be called just *theory* or *program* where no ambiguity arises.

4.2 Answer sets in terms of Intuitionistic Logic

In some definitions we use Heyting's *intuitionistic* logic [45], which will be denoted by the subscript I.

For a given set of atoms M and a program P, we will write $P \vdash_{\mathrm{I}} M$ to abbreviate $P \vdash_{\mathrm{I}} a$ for all $a \in M$.

For a given set of atoms M and a program P, we will write $P \Vdash_{\mathrm{I}} M$ to denote that:

1. $P \vdash_{\mathrm{I}} M$, and
2. P is consistent w.r.t. logic I (i.e. there is no formula A such that $P \vdash_{\mathrm{I}} A$ and $P \vdash_{\mathrm{I}} \neg A$).

Sometimes we will write $\vdash$ instead of $\vdash_{\mathrm{I}}$ when no ambiguity arises.

Now we define answer sets (or stable models) of logic programs. The stable model semantics was first defined in terms of the so called *Gelfond-Lifschitz reduction* [14] and it is usually studied in the context of syntax dependent transformations on programs. We follow an alternative approach started by Pearce [38] and also studied by Osorio et.al. [33]. This approach characterizes the answer sets of a propositional theory in terms of intuitionistic logic and it is presented in the following theorem. The notation is based on [33].

Theorem 4.1. *Let P be any theory and M a set of atoms. M is an answer set for P iff $P \cup \neg(\mathcal{L}_P \setminus M) \cup \neg\neg M \Vdash_{\mathrm{I}} M$.*

For instance, if we consider the program $P_1 = \{\, p \leftarrow \neg q.\ \ q.\,\}$ from Example 4.2 then we can verify that $M = \{q\}$ is an answer set of P_1 and that $P_1 \cup \{\neg p\} \cup \{\neg\neg q\} \Vdash_{\mathrm{I}} \{q\}$.

4.3 Additional Notation

We present a list of additional notations that will be used in the rest of this work.

- The formula $f \leftarrow g$ is just another way of writing $g \rightarrow f$.
- The formula $(g \leftarrow f) \wedge (f \leftarrow g)$ can be abbreviated as $g \leftrightarrow f$.
- Given a set of formulas F, we define $\neg F = \{\neg f \mid f \in F\}$. For instance, if $F = \{\neg p, q\}$ then $\neg F = \{\neg\neg p, \neg q\}$.
- Following the traditional notation of logic programming, we may use *not* instead of $\neg$ and we also may use a, b instead of $a \wedge b$.

- Given a set of formulas F, we write $literals(F)$ to denote the set of literals in F.
- Given a set of literals L, $pos(L)$ denotes the set of positive literals in L and $neg(L)$ denotes the set of negative literals in L.
- Given a program P. The set of *theorems of* P, denoted as $th(P)$, can be defined as follows: $th(P) = \{\alpha \mid \alpha \text{ is a formula over } \mathcal{L}_P \text{ and } P \vdash \alpha\}$. For instance, if $P = \{a \leftarrow b,\ b\}$ then $th(P) = \{b, a, a \wedge a, a \wedge b, a \rightarrow a, a \rightarrow (b \rightarrow a), \ldots\}$.
- For a given set of sets of formulas R, we write $K(R)$ to denote the set of formulas that belong to every set in R, i.e., $K(R) = \bigcap_{S \in R} S$. For instance, if $R = \{\ \{b, a, a \wedge a\}, \{a \wedge b, b\}\ \}$ then $K(R) = \{b\}$.
- A finite set of formulas P is *consistent* [27] if there is no formula A such that both A and $\neg A$ are theorems of P.
- A finite set of formulas P is *complete* [27] if, for any formula A of P, either $P \vdash A$ or $P \vdash \neg A$.
- A finite set of formulas P' is an *extension* of a finite set of formulas P [27] if every theorem of P is a theorem of P'.
- Given two sets X and Y, we write $X \subset Y$ to denote that $X \subseteq Y$ and $X \neq Y$.

4.4 Programs with predicate symbols and variables

As usual in answer set programming, we take for granted that a program with predicate symbols is only an abbreviation of the ground program, denoted as $ground(P)$. Hence, we are going to present the definition of predicate symbols, ground program and some other definitions needed.

In a program we also can find an atom of the form $p(a_1, \ldots, a_n)$ where p is an n-ary predicate symbol and $a_1, \ldots, a_n$ $(n \geq 0)$ are terms. A *term* may be either a variable or a constant. An atom with no variables is called a *ground atom*. Then a *ground literal* is either a ground atom or its negation. The *Herbrand Universe* of a program is the set of constants in the program. The *Herbrand Base* of a program is the set of all ground atoms that can be constructed using the predicates in the program and the Herbrand Universe.

Example 4.3. Let P_2 be the following program:

$$\begin{array}{lcl} alive(X) & \leftarrow & \neg death(X). \\ death(X) & \leftarrow & \neg alive(X). \\ object(a). & & object(b). \end{array}$$

Then the Herbrand Universe is $\{a, b\}$, and the Herbrand Base is $\{alive(a), alive(b), death(a), death(b)\}$. $\mathcal{L}_{P_2} = \{p, q\}$. $\square$

An instance of an atom, a literal or a formula is constructed by replacing all variables in it by ground terms. The *grounding* of a program, also called the *ground program*, is the set of all ground instances of the formulas of the program that may be constructed using terms in the Herbrand Universe of the program.

Example 4.4. The grounding of the program P_2, $ground(P_2)$, from Example 4.3 is :

$$\begin{array}{lcl} alive(a) & \leftarrow & \neg death(a). \\ alive(b) & \leftarrow & \neg death(b). \\ death(a) & \leftarrow & \neg alive(a). \\ death(b) & \leftarrow & \neg alive(b). \\ object(a). & & object(b) \end{array}$$

We recall that in fact P_2 is the abbreviation of the ground program $ground(P_2)$ $\square$

Another example is the following:

Example 4.5. Let P_3 be a program with two predicate symbols, $s(X)$ and $p(X)$, as follows:

$$\begin{array}{lcl} p(X) & \leftarrow & s(X). \\ s(a). & & \\ s(b). & & \end{array}$$

Then, P_3 is only an abbreviation of the ground program $ground(P_3)$, such that $ground(P_3)$ is the following program:

$$\begin{array}{lcl} p(a) & \leftarrow & s(a). \\ p(b) & \leftarrow & s(b). \\ s(a). & & \\ s(b). & & \end{array}$$

then $\mathcal{L}_{P_3} = \mathcal{L}_{ground(P_3)} = \{s(a), s(b), p(a), p(b)\}$. □

4.5 Adding sorts to Answer Set Programming framework

In some cases, we need to model some problems using predicate symbols with variables where these variables must be only instantiated using a subset of the Herbrand Universe of the program. This kind of programs are called *sorted programs with predicate symbols* [4]. For instance, we will use *sorted programs with predicate symbols* when we study the encoding of a planning problem in Section 5.2.

Then, to define a *sorted programs with predicate symbols* it is necessary to specify a nonempty set I whose members are called *sorts*, and a*sort specification* for constants and variables appearing in a predicate [4]. The sort specification assigns each predicate symbols, variable and constant to a sort in I. Each n-ary predicate symbol is assigned an n-tuple $(s_i, \ldots, s_n)$, where for each i, $1 \leq i \ \leq n$, $s_i \ \in I$ [4]. In addition it is stipulated that there must be at least one constant symbol of each sort in I. The atoms, and literals are defined as before except that they must respect the sort specifications [4].

Example 4.6. Let P_3 be the following sorted program with predicate symbols.

$$\begin{aligned} p(X,Y) &\leftarrow r(X), \neg q(X,Y). \\ r(a). \\ q(a,0). \end{aligned}$$

The set of sorts I is {*letter, number*}. The sort specifications are as follows:
sort(a) = letter, sort(0) = sort(1) = sort(2) = number; and
sort(p) = (letter, number), sort(q) = (letter, number);
sort(r) = letter.

The ground program $ground(P_3)$ is

$$\begin{aligned} p(a,\ 0) &\leftarrow r(a), \neg q(a,\ 0). \\ p(a,\ 1) &\leftarrow r(a), \neg q(a,\ 1). \\ p(a,\ 2) &\leftarrow r(a), \neg q(a,\ 2). \\ r(a). \\ q(a,\ 0). \end{aligned}$$

The above program has the unique answer set $\{r(a),\ q(a,\ 0),\ p(a,\ 1),\ p(a,\ 2)\}$. □

4.6 Answer Set Solvers and systems used in this work

Currently, there are several answer set solvers. In this work we have used DLV[1] and SMODELS[2] . In particular for modeling planning problems there exist a frontend

[1] http://www.dbai.tuwien.ac.at/proj/dlv/
[2] http://www.tcs.hut.fi/Software/smodels/

to the DLV system called DLVK[3] . Also we have used PSMODELS, a modification of SMODELS, that can be used to compute preferred answer sets under the ordered disjunction semantics.

In particular, in this work we used mainly SMODELS for several reasons:

—SMODELS compared with DLV allows nested predicates in a program which results very useful when we want to express planning problems (see Section 5.2). We recall that a program with predicate symbols is only an abbreviation of the ground program.

—In spite of DLVK needs less time to generate the plans than SMODELS, the second one results to be more flexible. Using SMODELS we could test consistency restoring rules in planning problems (see Chapter 8), which it is not possible using DLVK. Additionally, using SMODELS is possible to change in an easy way the independent part definition of planning problems (see Subsection 5.2.2). For instance, we could change the definition of concurrency depending of the domain description of a planning problem.

—In this work we studied the ordered disjunction semantics (see Section 4.8) and we proposed an extension of it. SMODELS has a modification called PSMODELS that compute the preferred answer sets under the ordered disjunction semantics. Then once we had the planning problem modeled using SMODELS we easily extended these programs with ordered disjunction or extended ordered disjunction to compute the preferred answer sets (see Chapter 6).

In this work we also worked with The Logics Workbench(LWB) [4] that offers the possibility to work in a user-friendly way in among other logics in Intuitionistic Logic.

[3] http://www.dbai.tuwien.ac.at/proj/dlv/K/
[4] http://www.lwb.unibe.ch/about/index.html

4.7 CR-Programs and Abductive Logic Programs

Consistency Restoring Rules (CR-rules) are rules that are added to standard disjunctive logic programs, but they are only applied when the standard rules in the program lead to inconsistency. A CR-rule is written $\alpha \overset{+}{\leftarrow} \beta$, which is intuitively read as: *If the program is inconsistent, then assume* $\alpha \leftarrow \beta$*. Otherwise, ignore the rule.* We recall the following example presented in [3], since it illustrates clearly the use of CR-rules. In this example r_1 denotes the name of the CR-rule.

$$a \leftarrow \neg b.$$

$$\neg\, a.$$

$$r_1 : b \overset{+}{\leftarrow} .$$

The first two rules are *regular rules* and the third rule is a CR-rule. This kind of programs will be called CR-programs. We can see that the program without the CR-rule is inconsistent. However, if the CR-rule is used then consistency is restored, thus the answer set $\{\neg a, b\}$ is obtained.

The semantics of CR-programs are defined in terms of *minimal generalized answer sets.* In [3] the authors give a translation of CR-programs into abductive logic programs, which they call *hard reduct*, and the semantics are defined as the minimal generalized answer sets of the hard reduct of the program.

The following definitions are found in [3]:

Definition 4.1. [3] An *abductive logic program* is a pair $\langle P, A\rangle$ where P is an arbitrary program and A a set of atoms, called *abducibles.*

Example 4.7. Let P_1 be the program from Example 4 introduced in [3]:

$$p \leftarrow \neg q.$$
$$r \leftarrow \neg s.$$
$$q \leftarrow t.$$
$$s \leftarrow t.$$
$$\leftarrow p, r.$$

Then, we can define $\langle P_1, A\rangle$, an *abductive logic program*, where the set of *abducibles* is $A = \{q, s, t, p\}$. □

Definition 4.2. [3] Let $\langle P, A\rangle$ be an abductive logic program and $\Delta \subseteq A$. We say that $M(\Delta)$ is a *generalized answer set* of P w.r.t. A iff $M(\Delta)$ is an answer set of $P \cup \Delta$.

Example 4.8. Let $\langle P_1, A\rangle$ be the abductive logic program from Example 4.7 where P_1 is the following program

$$p \leftarrow \neg q.$$
$$r \leftarrow \neg s.$$
$$q \leftarrow t.$$
$$s \leftarrow t.$$
$$\leftarrow p, r.$$

and the set of *abducibles* is $A = \{q, s, t, p\}$. In Table 4.1 we present different *generalized answer sets* of P_1 with their corresponding subset of abducibles denoted by Δ. □

Definition 4.3. [3] Let $M(\Delta_1)$ and $M(\Delta_2)$ be two generalized answer sets of P w.r.t. A. We define $M(\Delta_1) < M(\Delta_2)$ iff $\Delta_1 \subset \Delta_2$ (set inclusion order).

Example 4.9. If we consider the different generalized answer sets of program P_1 from the abductive logic program of Example 4.7 showed in Table 4.1 then, we can verify that if $M(\{q\}) = \{q, r\}$ and $M(\{t, q\}) = \{t, q, s\}$ then $M(\{q\}) < M(\{t, q\})$ since

Δ	Generalized answer set
$\{q\}$	$\{q, r\}$
$\{s\}$	$\{s, p\}$
$\{s, p\}$	$\{s, p\}$
$\{t\}$	$\{t, q, s\}$
$\{t, q\}$	$\{t, q, s\}$
$\{t, q, s\}$	$\{t, q, s\}$

Table 4.1: Generalized answer sets of P_1 with different $\Delta \subseteq A$.

$\{q\} \subset \{t, q\}$; or that if $M(\{t, q\}) = \{t, q, r\}$ and $M(\{t, q, s\}) = \{t, q, s\}$ then $M(\{t, q\}) < M(\{t, q, r\})$ since $\{t, q\} \subset \{t, q, r\}$. Moreover, we can say nothing about $M(\{q\}) = \{q, r\}$ and $M(\{s\}) = \{s, p\}$ then since $\{q\}$ is not comparable with $\{s\}$ w.r.t. set inclusion and viceversa. $\square$

Definition 4.4. Let P be a program and let A be a set of atoms. $M(\Delta)$ is a *minimal generalized answer set* of P w.r.t. A and the ordering $<$ iff $M(\Delta)$ is a generalized answer set of P and it is minimal w.r.t. set inclusion order.

Example 4.10. If we consider the different generalized answer sets of program P_1 from the abductive logic program of Example 4.7 showed in Table 4.1 then, we can verify that the minimal generalized answer sets of P_1 are $\{q, r\}$, $\{s, p\}$ and $\{t, q, s\}$. $\square$

Finally, the following example illustrates how the semantics of CR-programs are defined in terms of *minimal generalized answer sets.* As we mentioned, in [3] the authors give a translation of CR-programs into abductive logic programs, which they call *hard reduct*, and the semantics are defined as the minimal generalized answer sets of the hard reduct of the program.

Example 4.11. Let P_1 be the program from Example 4 introduced in [3] and also the program presented in the previous Example 4.7:

$$\begin{aligned} & p \leftarrow \neg q. \\ & r \leftarrow \neg s. \\ & q \leftarrow t. \\ & s \leftarrow t. \\ & \leftarrow p, r. \end{aligned}$$

We can see that program P_1 does not have answer sets. However, in order to restore consistency, we can define a CR-Program adding some CR-rules. For instance, let us define the CR-Program $P_1' = P1 \cup CR_1$ where $CR_1 = \{r_1 : q \overset{+}{\leftarrow}, r_2 : s \overset{+}{\leftarrow}, r_3 : t \overset{+}{\leftarrow}\}$

$$\begin{aligned} & p \leftarrow \neg q. \\ & r \leftarrow \neg s. \\ & q \leftarrow t. \\ & s \leftarrow t. \\ & \leftarrow p, r. \\ & r_1 : q \overset{+}{\leftarrow} . \\ & r_2 : s \overset{+}{\leftarrow} . \\ & r_3 : t \overset{+}{\leftarrow} . \end{aligned}$$

As we mentioned, the semantics of P_1' is defined in terms of *minimal generalized answer sets*. Then, translating P_1' into an *abductive logic program* we obtain the following: $\langle P_1, A \rangle$ where the set of *abducibles* is $A = \{q, s, t\}$. As we presented in Example 4.10 we can verify that the minimal generalized answer sets of P_1 are $\{q, r\}$, $\{s, p\}$ and $\{t, q, s\}$. □

4.8 Logic Programs with Ordered Disjunction

In [7], Brewka introduced Logic Programs with Ordered Disjunction, abbreviated as LPODs. In order to define standard ordered disjunction programs, the connective $\times$ is added, called *ordered disjunction*, to the set of connectives of propositional formulas presented in Section 4.6. The intuition behind LPODs is to express default knowledge with knowledge about preferences in a simple and elegant way [9]. As we have mentioned in Section 4.6, in this work we will consider only one type of negation,default negation, but this does not affect the validity of the results as given in [7].

Example 4.12. Let us consider the following situation: A person should travel from his/her home to school, and this person prefers to travel by bus to travel by bicycle and prefers to travel by bicycle to walk. This person also should consider that part of the path from his/her home to school can become blocked by snow in Winter. Then we can model this situation considering the following ordered disjunction rule:

$$\begin{array}{lcl} travelBus \times travelBicycle \times walk & \leftarrow & winter, \neg pathBlocked \\ travelBicycle \times walk & \leftarrow & \neg winter \end{array}$$

□

Formally, an ordered disjunction program is a set of rules of the form [5] :

$$C_1 \times \ldots \times C_n \leftarrow A_1, \ldots, A_m, \neg B_1, \ldots, \neg B_k \tag{4.1}$$

where C_i, A_j and B_l are all ground atoms. $C_1 \ldots C_n$ are usually called the choices of a rule and their intuitive reading is as follows: If the body is true and C_1 is possible, then C_1; if C_i is not possible, then C_2; $\ldots$; if none of $C_1, \ldots, C_{n-1}$ is possible then C_n. Some special cases can be pointed out, namely $n = 0$ (constraints), $n = 1$ (extended programs), $m = k = 0$ (facts).

[5] Recall that $\neg$ represents default negation.

The semantics of standard ordered disjunction programs will be presented in Section 6 where an extension of standard ordered disjunction programs is presented. In Section 6 we present the semantics of extended ordered disjunction programs and we show that standard ordered disjunction programs are particular cases of extended ordered disjunction programs. In this work, we will call the ordered disjunction programs defined by Brewka [7] as *standard ordered disjunction programs* in order to make a difference when we present the extended version of these programs.

Currently, there is a modification of SMODELS [6] that can be used to compute preferred answer sets under the ordered disjunction semantics called PSMODELS[7] [9].

4.9 Conclusion

In this chapter we presented a synthesis about Answer set programming framework and its related notation that we will use in this work. We also presented a brief overview about CR-Programs and Abductive logic programs. We studied that the semantics of a CR-program is defined in terms of the *minimal generalized answer sets* of a particular abductive logic program.

We are interested in CR-Programs since they are used as a possible solution to find alternative evacuation problems, as we propose in Chapter 8. Additionally, in Chapter 6 we propose a characterization of minimal generalized answer sets in terms of ordered disjunction programs. We also presented a brief overview of Logic Programs with Ordered Disjunction introduced in [7]. In Contribution Part of this work we present how to extended Ordered Disjunction programs and we show some of their possible applications to real problems.

[6] http://www.tcs.hut.fi/Software/smodels/
[7] http://www.tcs.hut.fi/Software/smodels/priority/