**CHAPTER 4**

**ANSWER-SET PROGRAMMING**

**4.1 Overview**

The system that has been designed uses an ASP-based approach to represent knowledge from natural language text. So, a basic understanding of answer-set programming is required to understand the remainder of the thesis. This chapter introduces the answer-set programming paradigm and further elaborates on some of the important definitions, concepts and patterns used in answer-set programming. At the end of this chapter, we will go over some of the systems that are developed to run ASP programs.

**4.1 What is Answer-Set Programming (ASP)**

Answer-Set Programming is a declarative problem-solving paradigm that uses both non-monotonic reasoning and logic programming. It is widely used in automatically solving problems relating to representation and reasoning tasks such as modeling reasoning agents, non-monotonic inferences, common sense reasoning, modeling preferences and priorities and many more. An answer set program is a collection of statements that describe the objects of a domain and model relations between them. The semantics of an ASP Program defines a set of possible beliefs that an agent has associated with the program. This set of beliefs is called as an answer-set. The basic constituents of an ASP program are the rules, facts and constraints that describe the problem. Such a program is then passed onto an answer-set solver, which generates answer-sets to the program, that are used to obtain solutions to the problem.

**4.2 Syntax**

In this subsection, we introduce the syntax of an ASP program.

**4.2.1 Atom**

The most basic constituent of the ASP program is an atom. An atomic statement or an atom, is an expression of the form p(t1,…,tn) where p is a predicate symbol of arity n and t1…tn are n terms belonging to the predicate p. Here n >= 0 and the terms ti can be integers or strings of letters, numbers, or underscore that either begin with an underscore or a lower-case letter. If in an atomic statement n = 0, then the brackets are omitted. As an example, ‘parent(mary, alice)’ and ‘alice’ are both atoms, whereas ‘parent(mary, girl(alice))’ is not an atom.

**4.2.2 Literal**

A literal is an atom of the form p(t1,…,tn) or its negation -p(t1,…,tn). Here, -p(t1,…,tn) is referred to as a negative literal. It means that p(t1,…,tn) is false. An atom is called as a ground literal if every term ti in the atom is ground. For example ‘parent(X, Y)’ is a literal where as ‘parent(mary, alice)’ is called as a ground literal.

**4.2.3 Rule or Clause**

An ASP Program consists of a collection of rules of the form

1. l0
2. li <- li+1, …,lm,not lm+1,…,not ln

Here, the symbol ‘not’ is a logical connective and is called as a default negation or negation as failure. Its semantic is discussed later in the chapter. An ASP rule is divided into two parts viz. head and a body. A head is a literal on the left side of the rule and a body is a set of literals on the right side of the rule. The head or the body in a rule can be empty. A rule with an empty head is called as a constraint whereas a rule with an empty body is called as a fact.

**4.3 Semantics**

Using the earlier mentioned syntax, we create an ASP program as a collection of rules, facts, and constraints. In this section, we shall discuss about the meaning of these rules and how they are interpreted while reasoning using these rules. The following are a few semantic patterns commonly used in answer-set programs.

**4.3.1 Modelling Implication**

As we saw earlier, every rule (excluding facts) in ASP has two parts separated by the consequence operator “:-“. In such a rule the head of the rule is said to succeed only if every literal in the body of the rule succeeds. As an example, consider the rule

p :- q, r.

we can read this kind of rule as “if q and r succeed then p succeeds”. Such kind of a pattern is commonly used in ASP programs to show implications.

**4.3.2 Classical Negation**

Classical Negation is a pattern in which we use negative literals, to show the fact that the literal under consideration has been proved to be false. As an example, consider the following rule.

-p(a) :- q(a)

The above rule states that if q(a) is shown to succeed then p(a) is false or -p(a) is true. Classical negation is one of the ways to represent negations in ASP programs.

**4.3.3 Epistemic Disjunction**

We model epistemic disjunctions in ASP when we need to model the semantics for the statement, “Either p(a) succeeds or q(a) succeeds”. Epistemic disjunction is different from exclusive or, where both p(a) and q(a) might succeed at the same time. Thus, to model epistemic disjunction we can make use of even loops in the following manner.

p(a) :- not q(a).

q(a) :- not p(a).

If we solve the above ASP program using an answer-set solver we will get two answer sets {p(a)} and {q(a)}, i.e. either p(a) succeeds or q(a) succeeds.

**4.3.4 Constraints**

Constraints are applied in places where we know for a fact that certain rules are always false and should not be part of the answer-set. As an example, if we know that it is impossible for p(a) to succeed then we can model this constraint as follows

:- p(a).

The above rule states that p(a) is always false. Here we see that a constraint limits the sets of beliefs that an agent has but does not help to derive new information.

**4.3.5 Default Negation or Negation As Failure (NAF)**

Default Negation, also called as Negation As Failure is used to make conclusions based on the absence of information. This type of negation is used to conclude about default rules and assume defaults to be true in case of absence of enough information. As an example, consider the following example where we state that if we are not able to prove that q(a) succeeds then p(a) succeeds.

p(a) :- not q(a).

So in the above rule we assumed that p(a) has succeeded based on the absence of information about q(a). NAF is an important tool to model defaults in ASP programs. Negation as Failure assumes closed-world assumption (CWA), in which we assume, what is not currently know to be true, as false.

**4.4 Default Reasoning**

Default Reasoning or Representing Defaults is one of the advantages of using ASP. The concept of closed-world assumption discussed earlier is an example of default reasoning where we default the value of the literal to fail in the absence of the literal in the answer set. Default reasoning is very useful in modelling human reasoning as we can draw conclusions even in the absence of information by defaulting to the default rule. Default reasoning thus plays an important role in common sense reasoning and understanding. In case of ASP, a default d stated as “Normally elements of class C have property P” is represented as the following rule

p(X) :- c(X),

not ab(d(X)),

not -p(X).

Here, ab(d(X)) can be read as “X is abnormal with respect to the default assumption d” and not -p(X) can be read as “We can’t successfully prove that p(X) is false” or “p(X) may be true”.

Default reasoning uses two kinds of exceptions viz Strong exceptions and weak exceptions. Weak exception makes the default inapplicable and stop the agent from making a default conclusion. For example, in the above-mentioned default rule we can apply a weak exception e(X) by adding the following rule to the program

ab(d(X)) :- not -e(X).

The exception states that X may not be applicable to d if e(X) may be true. Similarly, Strong Exceptions refute the defaults conclusion by allowing the agent to derive the opposite to be true. This can be demonstrated by adding the following rule to the program

-p(X) :- e(X).

The above rule states that p(X) is false if e(X) succeeds, which allows us to defeat d’s conclusion that normally class C elements have the property P.

**4.5 Present systems SASP, CLASP**

TBD

**References:**

1. Knowledge Representation, Reasoning, and the Design of Intelligent Agents (The Answer-Set Programming Approach), *Micheal Gelfond, Yulia Kahl*
2. Dissertation – Goal Directed Answer Set Programming, *Kyle Marple*
3. Answer-Set Programming – A Primer, *Thomas Eiter, Giovambattista Ianni,and Thomas Krennwallner*

**CHAPTER 5**

**SYSTEM ARCHITECTURE**

**5.1 Overview**

This chapter mainly focuses on the various parts of the system and how they interact with each other. It also describes the various sub-components and modules used in generating the knowledge base and goes through the various steps required to answer queries with the help of the generated ASP program.

**5.2 System Architecture**

The system is composed of two main components or sub systems viz. the Knowledge Generation System and the Query Generation System. Both these systems function independent of each other. The architecture comprises of a common resource framework that is shared by both these systems. This chapter will describe all these components in detail in the rest of the chapter.

**5.3 Components of the System**



As illustrated in the figure, the Knowledge Generation System, the Query Generation System and the Common Resource Framework are the three components of the architecture. The Common Resource Framework consists of Natural Language Processing tools such as Stanford Core NLP Tools, WordNet API as well as modules for preprocessing incoming text. The Knowledge Generation System is mainly responsible for extracting knowledge from a natural language text. For extracting the knowledge from text, this component uses Stanford NLP tools like the POS Tagger, Stanford Dependency Parser, and the Stanford NER Tagger to gain more information about the input text. Apart from these resources it also taps into the vast information that is provided by WordNet and tries to extract information from the same. As currently there are a very few digital resources about verbs in the NLP domain, this component provides a flexible way to add custom information about verbs that would be reusable in many scenarios. Thus, the Knowledge Generation System takes in the natural language passage as input and produces rules in the form of three chunks of information, which can be aggregated together to form an ASP program representing all the extractable knowledge from the source text.

To help answer questions posed in Natural Language, the Query Generation System is used to automatically generate a set of queries that can be used to find solutions from the answer-sets generated by the ASP program. To ask queries to the ASP program we need to provide both the queries as well as the ASP program to an Answer-Set Solver like SaSP or Clasp. The Query Generation System generates multiple queries for a question and arranges them in the order of significance, keeping the more constraint queries before the less constraint ones. Hence, the kind of query that would lead to an answer is also a rough metric as to the quality of the answer. Now let’s dive deep into the various components in the architecture and talk about its sub modules and their interactions.

**5.4 Common Resources Framework**

The Common Resources Framework consists of the following modules as illustrated in the diagram.



**5.4.1 Text Preprocessing Module**

-- (98% automatic)

-- Combining concepts

-- Coreference resolution

-- Missing terms/ assumed words for correct parsing

**5.4.2 Stanford NLP Core Tools**

-- Provides information about natural language text

**5.4.3 WordNet Interface**

-- Extracts concepts and their relations from WordNet

**5.5 Knowledge Generation System**



What is the main motive of the knowledge generation system to convert and represent. It is divided into multiple sub components

**5.5.1 Default Knowledge Base**

-- Knowledge about verbs (reusable)

**5.5.2 Knowledge Extracted from Text**

-- Uses rules mentioned in the later chapters

-- Automatic conversion in a set of rules and facts

**5.5.3 WordNet to Ontology Rules**

-- Hypernym rules

-- Meronym rules (not implemented)

-- Disambiguation rules

**5.6 Query Generation System**



Generates Queries with the help of the Text Preprocessing and the Stanford Core NLP Tools