How Flexible Is Answer Set Programming?  
An Experiment in Formalizing Commonsense in ASP

Marcello Balduccini
Intelligent Systems, OCTO
Eastman Kodak Company
Rochester, NY 14650-2102 USA
marcello.balduccini@gmail.com

Abstract. This paper describes an exercise in the formalization of commonsense with Answer Set Programming aimed at finding the answer to an interesting riddle, whose solution is not obvious to many people. Solving the riddle requires a considerable amount of commonsense knowledge and sophisticated knowledge representation and reasoning techniques, including planning and adversarial reasoning. Most importantly, the riddle is difficult enough to make it unclear, at first analysis, whether and how Answer Set Programming or other formalisms can be used to solve it.

1 Introduction

This paper describes an exercise in the formalization of commonsense [12] with Answer Set Programming (ASP) [3], aimed at solving the riddle:

“A long, long time ago, two cowboys were fighting to marry the daughter of the OK Corral rancher. The rancher, who liked neither of these two men to become his future son-in-law, came up with a clever plan. A horse race would determine who would be allowed his daughter’s hand. Both cowboys had to travel from Kansas City to the OK Corral, and the one whose horse arrived LAST would be proclaimed the winner.

The two cowboys, realizing that this could become a very lengthy expedition, finally decided to consult the Wise Mountain Man. They explained to him the situation, upon which the Wise Mountain Man raised his cane and spoke four wise words. Relieved, the two cowboys left his cabin: They were ready for the contest!

Which four wise words did the Wise Mountain Man speak?”

This riddle is interesting because it is easy to understand, but not trivial, and the solution is not obvious to many people. The story can be simplified in various ways without losing the key points. The story is also entirely based upon commonsense knowledge. The amount of knowledge that needs to be encoded is not large, which simplifies the encoding; on the other hand, as we will see in the remainder of this paper, properly dealing with the riddle requires various sophisticated capabilities, including modeling direct and indirect effects of actions, encoding triggers, planning, dealing with defaults and their exceptions, and concepts from multi-agent systems such as adversarial reasoning. The riddle is difficult enough to make it unclear, at first analysis, whether and how ASP or other formalisms can be used to formalize the story and underlying reasoning.
In the course of this paper we will discuss how the effects of the actions involved in the story can be formalized and how to address the main issues of determining that “this could be a lengthy expedition” and of answering the final question.

We begin with a brief introduction on ASP. Next, we show how the knowledge about the riddle is encoded and how reasoning techniques can be used to solve the riddle. Finally, we draw conclusions.

2 Background

ASP \cite{3} is a programming paradigm based on language A-Prolog \cite{4} and its extensions \cite{5,6,7}. In this paper we use the extension of A-Prolog called CR-Prolog \cite{5}, which allows, among other things, simplified handling of exceptions, rare events. To save space, we describe only the fragment of CR-Prolog that will be used in this paper.

Let $\Sigma$ be a signature containing constant, function, and predicate symbols. Terms and atoms are formed as usual. A literal is either an atom $a$ or its strong (also called classical or epistemic) negation $\neg a$.

A regular rule (rule, for short) is a statement of the form:

$$h_1 \lor \ldots \lor h_k \leftarrow l_1, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n$$

where $h_i$’s and $l_i$’s are literals and not is the so-called default negation. The intuitive meaning of a rule is that a reasoner, who believes $\{l_1, \ldots, l_m\}$ and has no reason to believe $\{l_{m+1}, \ldots, l_n\}$, must believe one of $h_i$’s.

A consistency restoring rule (cr-rule) is a statement of the form:

$$h_1 \lor \ldots \lor h_k \leadsto l_1, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n$$

where $h_i$’s and $l_i$’s are as before. The informal meaning of a cr-rule is that a reasoner, who believes $\{l_1, \ldots, l_m\}$ and has no reason to believe $\{l_{m+1}, \ldots, l_n\}$, may believe one of $h_i$’s, but only if strictly necessary, that is, only if no consistent set of beliefs can be formed otherwise.

A program is a pair $\langle \Sigma, \Pi \rangle$, where $\Sigma$ is a signature and $\Pi$ is a set of rules and cr-rules over $\Sigma$. Often we denote programs by just the second element of the pair, and let the signature be defined implicitly.

Given a CR-Prolog program $\Pi$, we denote the set of its regular rules by $\Pi^r$ and the set of its cr-rules by $\Pi^{cr}$. By $\alpha(r)$ we denote the regular rule obtained from cr-rule $r$ by replacing the symbol $\leadsto$ with $\leftarrow$. Given a set of cr-rules $R$, $\alpha(R)$ denotes the set obtained by applying $\alpha$ to each cr-rule in $R$. The semantics of a CR-Prolog program is defined in two steps.

**Definition 1.** Given a CR-Prolog program $\Pi$, a minimal (with respect to set-theoretic inclusion) set $R$ of cr-rules of $\Pi$, such that $\Pi^r \cup \alpha(R)$ is consistent is called an abductive support of $\Pi$.

**Definition 2.** Given a CR-Prolog program $\Pi$, a set of literals $A$ is an answer set of $\Pi$ if it is an answer set of the program $\Pi^r \cup \alpha(R)$ for some abductive support $R$ of $\Pi$.

\footnote{We also allow the use of SMODELS style choice rules, but omit their formal definition to save space.}