Encoding Design Ontologies in Answer Set Prolog

Glen Hunt
Computer Science
Arizona State University
glenrhunt@asu.edu

Mahmoud Dinar
Mechanical Engineering
Arizona State University
mdinar@asu.edu

Give an overview of the paper, and discuss the order in which we are going to cover things. Essentially, we want to cover some background and motivation for encoding such ontologies, where the function-behavior-structure idea comes from, give a brief introduction to ASP, and then show a motivating example for the development of such an encoding.

Introduction

Normally, this section would talk about the design ontology we are developing, and the computational tool we are thinking about for the long term, but because this paper is for internal use at the moment, we will exclude this section until a later date.

Background and Motivation

Answer Set Prolog was introducted in 1988 by Vladmir Lifschitz and Michael Gelfond[1]. It is rooted in research on the semantics of negation as failure in "classical" Prolog, and in work on non-monotonic logics. Informally, the semantics of Answer Set Prolog (hereafter ASP) may be stated as follows:**[CITATION-NEEDED]**

- A program, Π, may be viewed as a specification for sets of beliefs of a rational reasoner associated with Π.
- Beliefs are represented by consistent sets of literals, called *answer sets*, which must satisfy the rules of Π and the Rationality Principle, which states: "Believe nothing you are not forced to believe."[1]

To illustrate the applicability of non-monotonic logic, consider the following example:

> *On the first of December, John and Bob met in Paris. Immediately after the meeting, John took a plane from Paris to Baghdad.*

Suppose we then ask: where should one look for John and Bob on December 2nd? In order to obtain this answer, we need to know (among other things) that normally, John traveling to Baghdad changes the whereabouts of John, but not Bob.

Now consider the following addendum to our knowledge:

> *On the way, the plane stopped in Rome, where John was arrested.*

This new information would cause us to withdraw our previous conclusion, which illustrates the non-monotonic nature of this sort of reasoning.

A Motivating Example

To illustrate the applicability of our approach to the representation of design ontologies, we have selected a simple: designing a coffee grinder. This machine should be designed to grind coffee beans into a powder for use in a coffee machine. To describe the design of this machine, we discuss function composition, and the behavior/structure relationship with one function which might be included in the design. Additionally, we give ASP representations of both the function decomposition and the sample behavior/structure relationship.

Function Decomposition

In the design literature, *function* describes what a device does or is designed to do.**[CITATION-NEEDED]** Functions are an abstract relation between an input and an output for a system, and may be grouped into *function structures*. In a function structure, high-level functions can be decomposed into lower-level functions that, when considered together, satisfy the higher-level function. Each decomposition may be

[1]By literals, we mean ground literals from first- and second-order logic. A set of literals is consistent if there are no contradictions among literals in the set; for example, $\{p, q\}$ is consistent, whereas $\{p, \neg p, q\}$ is inconsistent.

unique, as a particular decomposition represents a particular approach for achieving the top-level function.

A sample function decomposition for the design of a coffee grinder is given in Figure 1. This decomposition illustrates how a top-level function, Make Coffee Grind, is decomposed into several other functions, each of which is required to satisfy Make Coffee Grind. This kind of function hierarchy is useful in the design process, because it gives the designer ideas about a partial ordering of functions, and an estimation of their importance in realizing a complete design.

Listing 1 gives the ASP encoding of part of the function decomposition given in Figure 1.

```
%% top-level function
function(make_coffee_grind).

%% first level functions
function(input_beans).
function(input_on_off_signal).
function(input_power).
function(hold).
function(rotate_shaft).
function(provide_coffee).
function(chop_beans).
function(permit_cleaning).

function(accept_beans).
function(indicate_amount).

%% relations
requires(make_coffee_grind, input_beans).
requires(make_coffee_grind, input_on_off_signal).
requires(make_coffee_grind, input_power).
requires(make_coffee_grind, hold).
requires(make_coffee_grind, rotate_shaft).
requires(make_coffee_grind, provide_coffee).
requires(make_coffee_grind, chop_beans).
requires(make_coffee_grind, permit_cleaning).

requires(input_beans, accept_beans).
requires(input_beans, indicate_amount).

%% rules
requires(X, Y) :-
  function(X),
  function(Y),
  function(Z),
  requires(X, Z),
  requires(Z, Y).
```

Listing 1: Partial Function Decomposition Example

Function-Behavior-Structure

For the next part of this example, we have decided to focus only on function, behavior, and structure. In this example, structure has been decomposed into the following parts:

- **Component**: describes some physical part of the system (design?) in an abstract way; and
- **Parameter**: states some measurable property of a component of the system/design.

Behavior, in this ontology and others, describes a relationship between parameters (measurable properties of design components) and function.

One representation of a design for the coffee grinder problem (considering only the spin shaft function) is given in Figure 2. In this representation, we consider two behaviors, power and speed. The speed in revolutions-per-minute, r, of a wound electric motor is given by:

$$r = \frac{120F}{p}$$

where F is the frequency of the AC power supply, and p is the number of poles in the stator winding. The power in horsepower, ρ, of an electric motor is given by:

$$\rho = \frac{r \cdot \tau}{5252}$$

where r is the speed of the motor in revolutions-per-minute, and τ is the torque in foot pounds.

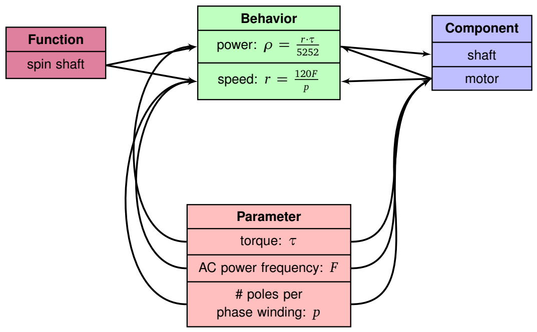
Function
spin shaft
Behavior
power: $\rho = \frac{r \cdot \tau}{5252}$
speed: $r = \frac{120F}{p}$
Component
shaft
motor
Parameter
torque: τ
AC power frequency: F
# poles per phase winding: p

Figure 2: Function-Behavior-Structure diagram for the Coffee Grinder

Conclusion

Here is where the text for the conclusion should go. Basically, this part should reinforce what we have already said, and summarize the basic points (that ASP is a natural choice for the encoding of design ontologies, and that our example illustrates this well).

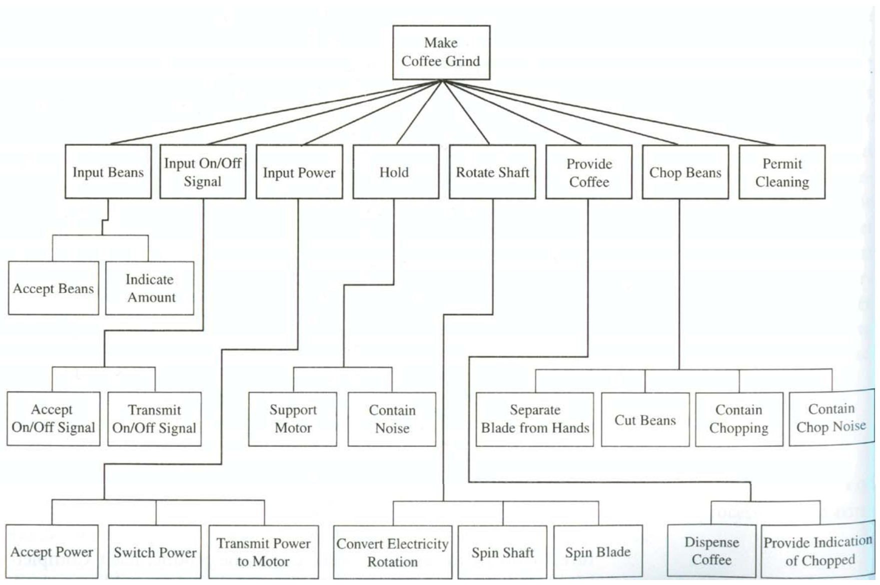

Figure 1: Function Decomposition for the Coffee Grinder

References

[1] M. Gelfond and V. Lifschitz, “The stable model semantics for logic programming,” in *The International Conference on Logic Programming*, pp. 1070 – 1080, 1988.