Chapter IX
A Logic Programming Perspective on Rules

Leon Sterling
University of Melbourne, Australia

Kuldar Taveter
Tallinn University of Technology, Estonia

ABSTRACT

Logic programming emerged from the realization that expressing knowledge in an appropriate clausal form in logic was akin to programming. The basic construct of a logic program can be viewed as a rule. This chapter will review rules from a logic programming perspective with an eye to developments within modern rule languages. It mentions rule interpreters, hybrid computing, interaction with the Web, and agents. An extended example is given concerning rule-based modelling and simulation of traffic at airports.

1. BACKGROUND

Rules have a long history in mathematics and computing, from inference rules such as modus ponens in logic, rewrite rules in grammar, to rules as norms or guidelines. More recently, rules have been seen as a key part of computing in applications such as expert systems and electronic commerce. There is a natural tendency to reinvent the wheel. To try to minimise reinvention in the case of rules, it is worth being aware of some of the developments that rules have gone through over the past forty years. It is in that spirit that this chapter is being written.

The discussion is focussed on how rules are used in logic programming, a computing paradigm that is arguably the most rule-based. The development of logic programming occurred in the intersection of automated theorem proving, artificial intelligence and programming languages. The most successful logic programming language has been Prolog (Clocksin & Mellish,

1981), (Sterling & Shapiro, 1994). We claim that experience with using Prolog has influenced current thinking about rules.

The roots of logic programming lie in the resolution rule of inference developed by Alan Robinson (Robinson, 1965). Robinson's research aimed to improve the behaviour of automated theorem provers by developing, in Robinson's words, a "machine-oriented rule of inference". A key component of Robinson's approach was two-way matching of logical terms, which became known as unification.

Resolution was regarded as a promising approach for achieving artificial intelligence. Situations could be expressed with logical formulae, problems expressed as a theorem to be proved from the logic describing the situation, and resolution used as the mechanism for machine reasoning. Unfortunately, resolution did not live up to the hype that it could be a universal mechanism for intelligence. Many artificial intelligence researchers abandoned logic-based rule approaches.

Several researchers, instead of abandoning resolution, tried to understand when resolution worked well and when it did not. One idea to improve the performance of resolution was to restrict the form of logical axioms to be used in theorem proving. The most successful restriction was using Horn clauses. Logic programming emerged in the early 1970s from the confluence of the work by Bob Kowalski working on restrictions to resolution-based theorem provers and the work by Alain Colmerauer on using grammar rules in logic for parsing sentences in natural language. The history is described by each of the main protagonists in (Kowalski, 1988) and (Colmerauer & Roussel, 2000) and separately by Jacques Cohen (Cohen, 1988).

The programming in logic programming came from the observation that the process of restricting how logic was expressed was akin to programming. The relevance for this book is that structuring logic is effectively structuring rules and the way that rules are expressed significantly affects their computational efficacy. The programming aspect of logic programming is not directly pursued in this chapter, but the reader is referred to (Kowalski, 1979) and (Sterling & Shapiro, 1994).

In the 1980s there was an explosion of interest in using artificial intelligence in practical applications. Expert systems were a key technology and were often rule-based. Expert system shells were developed and commercialised. The shells allowed developers to write their own rules. The interpreter contained within the shell was essentially either backward chaining from goals to facts or forward chaining from data to conclusions.

More recently rules have been studied as an entity in their own right. As the Internet has transformed the computing landscape, rules have been integrated at various levels. Three examples are as application development constructs for electronic commerce applications, for describing Web content, and for facilitating search. Taveter & Wagner (2001) identified the three most basic types of rules as *integrity constraints* (also called constraint rules or integrity rules), *derivation rules*, and *reaction rules* (also called stimulus-response rules, action rules, event-condition-action rules, or automation rules).

This chapter is organised as follows. Section 2 gives the basic conception of logic programming rules, as from the first section of (Sterling & Shapiro, 1994). Logic programming rules most typically are viewed as derivation rules. The execution mechanism of Prolog is backward chaining, applying rules to the goal that needs to be achieved. Section 3 looks at using rules to achieve alternative control mechanisms such as forward chaining. The notion of a rule interpreter is introduced. Section 4 looks at how constraints and complex objects have been integrated into logic programming systems. A major trigger for the current interest in business rules has been the Web and the possibilities of electronic commerce. Section 5 looks at one attempt to integrate logic programming rules with the Web. Section