3 Declarative Update Policies for Nonmonotonic Knowledge Bases

Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits
Technische Universität Wien, Institut für Informationssysteme 184/3, Favoritenstraße 9-11, A-1040 Vienna, Austria
e-mail: [eiter,giuliana,michael,tompits]@kr.tuwien.ac.at

Abstract. Updating databases, and in particular relational databases, is a central issue that has been well studied in the database field for many years, and solutions have been incorporated into commercial database systems. As for more advanced databases such as logical databases and in particular, for nonmonotonic knowledge bases, work on this problem is more recent. Various approaches for incorporating a single or a sequence of changes into a nonmonotonic knowledge base have been proposed. However, most of these approaches were concerned with how a change should be accomplished at a technical level but did not address the issue of strategic update behavior, i.e. which change should be issued in reaction to new information. In this chapter, we present a generic framework for declarative specification of update policies, that addresses this issue. In this framework, the update behavior of an agent maintaining its knowledge base is described in a rule-based language, whose statements describe change actions that are dependent on the information received, the current set of beliefs, and possible other change actions. In a layered approach, update policies are evaluated and compiled to update semantics for sequences of nonmonotonic logic programs using generic components. The framework can be instantiated to obtain different update mechanisms and thus, provides flexible support for the development of adaptive reasoning agents.

3.1 Introduction

Since the 1980s, the problem of updating logical data and knowledge bases has been studied by many researchers, and a number of different approaches have been developed (see [21,40] and [19] for accounts of early methods). In these investigations, a knowledge base is a collection of formulas or rules, and an update consists of another set of such objects. The problem then is how to incorporate the update in the given knowledge base.

For instance, given a knowledge base, $K$, containing some statement $\text{best\_buy}(\text{shop}_1)$, and assuming that there is a constraint expressing that $K$ should include only one atom of form $\text{best\_buy}(X)$, an update of $K$ with formula $\text{best\_buy}(\text{shop}_2)$ forces the deletion of $\text{best\_buy}(\text{shop}_1)$. However, in general, a completely automated approach to updating knowledge bases is rather complex, and sometimes there is not enough information to perform an update automatically. View updates are a case in point. As an illustration, suppose that a database contains
Given no further information, the atom \( r(a) \) can be inferred. But if \( r(a) \) should be deleted from the database, the question is, should \( p(a) \) be removed, or \( r(a) \), or even both? Methods to handle this problem, like, for example, [25] (see also [8,21]), involve principles used in artificial intelligence which would discard removing both \( r(a) \) and \( p(b) \) and leave the other options. However, this may be undesired, and the most appropriate removal option may depend on the particular application and on user-specific preferences, which are not respected by an automated general-purpose update procedure.

To avoid the difficulties of a completely automated approach, update policies allow users to specify how updates should be performed. Here, one is interested in update specifications, that is, in formalisms for describing the way in which a knowledge base has to be adjusted in reaction to incoming information in a dynamic and flexible way, based on change primitives. A realization of ideas in this direction is active databases [39], in which the dynamics of a database is specified by means of event-condition-action (ECA) rules that are triggered by events. However, most of these methods are ad hoc, lacking a clearly defined declarative semantics, and, moreover, complex conditions can, in general, not be expressed. Update policies, on the other hand, should allow the specification of update requests in a designated policy language, providing suitable flexibility by taking into account information from the environment and the current knowledge base. We note that similar approaches have also been developed in the area of network management and security [32,13,12,9].

Besides updating classical, monotonic knowledge bases, the issue of updating nonmonotonic knowledge bases has received growing interest in recent years, in particular, where the knowledge base is represented in terms of (nonmonotonic) logic programs (see [1,2,14,18,28,36,42]). For nonmonotonic knowledge bases of the latter kind, the LUPS language of dynamic updates [3,4] has been proposed as a general and flexible language for expressing different kinds of operations for changing the knowledge base, such as adding or removing a clause from it. In LUPS, a sequence \((U_1,\ldots,U_n)\) of sets \(U_i\) of update statements describes how a so-called dynamic logic program \((P_1,\ldots,P_n)\), i.e. a sequence of logic programs \(P_i\) expressing subsequent updates [2], is formed step-by-step. An important feature that is missing in LUPS, but is indispensable for the use of nonmonotonic knowledge bases to model the behavior of dynamic agents, is the possibility of modeling flexible reaction to events of the environment, making update actions depend on the content of rules or facts which are, in a discrete way, communicated by the environment, for example, by some cooperating agent or by some sensing device.