Incremental Tabling for Query-Driven Propagation of Logic Program Updates

Ari Saptawijaya* and Luís Moniz Pereira

Centro de Inteligência Artificial (CENTRIA), Departamento de Informática
Faculdade de Ciências e Tecnologia, Univ. Nova de Lisboa, 2829-516 Caparica, Portugal
ar.saptawijaya@campus.fct.unl.pt, lmp@fct.unl.pt

Abstract. We foster a novel implementation technique for logic program updates, which exploits incremental tabling in logic programming – using XSB Prolog to that effect. Propagation of updates of fluents is controlled by initially keeping any fluent updates pending in the database. And, on the initiative of queries, making active just those updates up to the timestamp of an actual query, by performing incremental assertions of the pending ones. These assertions, in turn, automatically trigger system-implemented incremental bottom-up tabling of other fluents (or their negated complements), with respect to a predefined overall upper time limit, in order to avoid runaway iteration. The frame problem can then be dealt with by inspecting a table for the latest time a fluent is known to be assuredly true, i.e., the latest time it is not supervened by its negated complement, relative to the given query time. To do so, we adopt the dual program transformation for defining and helping propagate, also incrementally and bottom-up, the negated complement of a fluent, in order to establish whether a fluent is still true at some time point, or rather if its complement is. The use of incremental tabling in this approach affords us a form of controlled, but automatic, system level truth-maintenance, up to some actual query time. Consequently, propagation of update side-effects need not employ top-down recursion or bottom-up iteration through a logically defined frame axiom, but can be dealt with by the mechanics of the underlying world. Our approach thus reconciles high-level top-down deliberative reasoning about a query, with autonomous low-level bottom-up world reactivity to ongoing updates, and it might be adopted elsewhere for reasoning in logic.

Keywords: logic program updates, updates propagation, incremental tabling, dual program transformation, XSB Prolog.

1 Introduction

The tabled logic programming paradigm, i.e., logic programming (LP) with tabling mechanisms, is supported by a number of Prolog systems, to different extent. Tabling affords solutions reuse, rather than recomputing them, by keeping in tables subgoals and their answers obtained by query evaluation. Incremental tabling, available in XSB Prolog [23], is an advanced recent tabling feature that ensures the consistency of answers in a table with all dynamic clauses on which the table depends. It does so by incrementally

* Affiliated with Fakultas Ilmu Komputer at Universitas Indonesia, Depok, Indonesia.
maintaining the table, rather than by recomputing answers in the table from scratch to keep it updated. The applications of incremental tabling in LP have been demonstrated in pointer analyses of C programs in the context of incremental program analyses [18], data flow analyses [19], static analyses [6], incremental validation of XML documents and push down model checking [17]. This range of applications suggests that incremental tabling lends itself to dynamic environments and evolving systems, including notably logic program updates, as we proceed to show.

In [20], an approach to logic program updates, termed EVOLP/R, theoretically based on Evolving Logic Programs (EVOLP) [1], is proposed. It simplifies EVOLP by restricting updates to fluents only. Rule updates are nevertheless achieved by attaching to each rule, in its body, a name fluent that uniquely identifies that rule (cf. [16]). Updating such a rule name fluent, via its assertion or retraction, permits time-activation or deactivation of the corresponding rule, respectively. Its implementation preliminarily exploits incremental tabling, plus another tabling feature: answer subsumption [22]. Incremental tabling of fluents is employed to automatically maintain the consistency of program states due to assertion and retraction of fluents, whether obtained as updated facts or concluded by rules. On the other hand, answer subsumption of fluents allows to address the frame problem, by automatically keeping track of the latest assertion or retraction of fluents with respect to a given query time. The combined use of incremental tabling and answer subsumption is realized in the tabled predicate \texttt{fluent(F, Ht, Qt)}: given query time \texttt{Qt}, it looks for dynamic definitions of fluent \texttt{F}, and returns \texttt{Ht}, the latest time fluent \texttt{F} is true. Predicate \texttt{fluent/3} depends on dynamic fluent definitions of \texttt{F}, and this dependency indicates that \texttt{fluent/3} is tabled incrementally, to avoid abolishing the table each time a Prolog assertion is made and then recomputing from scratch. Moreover, since \texttt{fluent/3} aims at returning only the latest time \texttt{F} is true (with respect to a given \texttt{Qt}), \texttt{fluent/3} can be tabled using answer subsumption on its second argument. While answer subsumption is shown useful in this approach to avoid recursing through the frame axiom by allowing direct access to the latest time when a fluent is true, it requires \texttt{fluent/3} to have query time \texttt{Qt} as its argument. Consequently, it may hinder reuse of tabled answers of \texttt{fluent/3} by similar goals which differ only in their query time. In truth, the state of a fluent in time depends solely on the changes made to the world, and not on whether that world is being queried. For instance, suppose \texttt{fluent(a, 2, 4)} is already tabled, and fluent \texttt{a} is inertially true till it is supervened by its negated complement, say at time $T = 7$. When a new goal \texttt{fluent(a, Ht, 5)} is posed, it cannot reuse the tabled answer \texttt{fluent(a, 2, 4)}, as they differ in their query time. Instead, \texttt{fluent(a, Ht, 5)} unnecessarily recomputes the same solution $Ht = 2$ (recall that fluent \texttt{a} is only retracted at $T = 7$), and subsequently tables \texttt{fluent(a, 2, 5)} as a new answer. A similar situation occurs when \texttt{fluent(a, Ht, 6)} is queried, where \texttt{fluent(a, 2, 6)} is eventually added into the table. This is clearly superfluous, as existing tabled answers could actually be reused and such redundancies avoided, if the tabled answers are independent of query time. However, in XSB answer subsumption on argument \texttt{Ht} cannot be made to ignore argument \texttt{Qt}, by its very design.

In this paper we address the aforementioned issue by fostering further incremental tabling, but leaving out the problematic use of the answer subsumption feature by reconceptualizing the issue at hand. The main idea, which was not captured in [20], is