Journal of Logical and Algebraic Methods in Programming 85 (2016) 316–331

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

Incremental reasoning on monadic second-order logics with logic programming

Gulay Unel

Department of Information Technologies, Işık University, Şile, İstanbul 34980, Turkey

ARTICLE INFO

Article history:
Received 25 March 2015
Received in revised form 9 November 2015
Accepted 20 November 2015
Available online 2 December 2015

Keywords:
Stream
Reasoning
Incremental
Logic

ABSTRACT

Data streams occur widely in various real world applications. The research on streaming data mainly focuses on the data management, query evaluation and optimization on data, but the work on incremental reasoning procedures for streaming knowledge bases is very limited. Typically reasoning services on large knowledge bases are very expensive, and need to be applied continuously when the data is received as a stream. Hence new techniques for optimizing this continuous process is needed for developing efficient reasoners on streaming data. In this paper, we describe a solution to an incremental reasoning problem on an expressive logic, namely monadic second-order logic, and point out further research directions in this area.

1. Introduction

Reasoning is typically applied to static knowledge bases and often ignored for rapidly changing data. However, there is a clear need for the design and implementation of incremental reasoning methods for dynamic knowledge bases motivated by the increase in the use of sensor data, streaming web and multimedia data, rapidly changing medical and financial data, etc. In addition to the need on reasoning over streaming data, there are also applications where complex reasoning tasks need to be continuously applied such as verifying ontologies using ontology editors, service discovery and matchmaking on web service frameworks where services register and deregister rapidly, and logical learning from fluctuating data.

The research on streaming data mainly focuses on the data management and query processing [1,2], however the work on reasoning procedures which involves deducing new information from what is known is very limited. Data streams are widely available, however their usage is mainly restricted to retrieval and search, instead of tasks such as decision making and deriving conclusions from them which can be used in many applications. The information searched and uploaded to Web which has a very dynamic nature can be formalized with the advances on Semantic Web which in turn results in the ability of applying reasoning methods over this formalization. For instance, the spread of a health disease can be detected by reasoning over the streaming keywords and/or information uploaded to the Web.

The reasoning techniques on streaming knowledge bases need to support incrementality to be able to use the results of the previous computations in the active reasoning step performed after the arrival of new data. In this paper, we provide an incremental reasoning method for monadic second-order logics. Monadic second-order logics provide means to specify regular properties of systems in a succinct way. In addition, these logics are decidable by the virtue of the connection to automata theory [3,4]. However, only recently tools based on these ideas—in particular the MONA system [5]—have been developed and shown to be efficient enough for practical applications [6].

E-mail address: gulay.unel@isikun.edu.tr.

http://dx.doi.org/10.1016/j.jlamp.2015.11.002
2352-2208/

However, for reasoning in large theories consisting of relatively simple constraints, such as theories capturing UML class diagrams or database schemata, the MONA system runs into a serious state-space explosion problem—the size of the automaton capturing the (language of) models for a given formula quickly exceeded the space available in most computers. The problem can be traced to the *automata product* operation that is used to translate conjunctions in the original formulæ rather than to the theoretically more problematic projection/determinization operations needed to handle quantifier alternations.

A technique that combats this problem is provided [7] which is based on techniques developed for query evaluation in deductive databases, in particular on the *Magic Set transformation* [40] and the *SLG resolution*, a top–down resolution-based approach augmented with memoization [8,9]. The work establishes the connection between the automata-based decision procedures for WS1S (and, analogously, for WS2S) and query evaluation in Complex-value Datalog (Datalog^{cv}) [10]. The approach is based on representing automata using nested relations and on defining the necessary automata-theoretic operations using Datalog^{cv} programs. This reduces the satisfiability of a WSnS formula to posing a closed Datalog^{cv} goal over a Datalog^{cv} program representing implicitly the automaton transformation of the WSnS formula. This observation combined with powerful program execution techniques developed for deductive databases, such as the Magic Set rewriting and SLG resolution, in many cases limit the explored state space to elements needed to show non-emptiness of the automaton and, in turn, satisfiability of the corresponding formula.

This approach can be extended to a stream reasoning solution on monadic second-order logics where the problem can be defined as follows. Given an ordered set of formulas $\{\varphi_1, \varphi_2, \ldots, \varphi_n\}$ determine if the conjunction of the formulas at each step is satisfiable without generating the whole automaton representing the conjunction. This problem is solved by generating a set of logic programs representing the conjunction of formulas at each step where the query representing the satisfiability of a formula is evaluated using incremental evaluation techniques provided for logic programs [11].

This method can be used for reasoning on a series of formulas on large theories, such as those corresponding to rapidly changing constraints encoded with expressive logics where the set of constraints can be mapped to a series of monadic second-order logic formulas. Consider a series of constraints $\{\varphi_1, \varphi_2, \ldots, \varphi_n\}$ for a dynamic database that should be satisfied as conjunctions which can be encoded using monadic second-order logic. The satisfiability of each constraint, $\varphi_1, \varphi_1 \wedge \varphi_2, \ldots, \varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_n$ can be computed using the incremental reasoning method we propose instead of evaluating each one of them separately. Our method generates a series of Datalog^{cv} programs for the given series of formulas so that the satisfiability of each formula is reduced to the emptiness problem of the corresponding automaton and evaluates Datalog^{cv} programs using efficient evaluation techniques with incremental reasoning.

The paper is structured as follows. In Section 2 we formally introduce the weak monadic second-order logic and the connection to finite automata. We also define Datalog^{cv} programs, state their computational properties, and briefly discuss techniques used for program evaluation. Section 3 shows how Datalog^{cv} programs can be used to implicitly represent a finite automaton and to implement automata-theoretic operations on such a representation for the WS1S logic, and provides a method for incremental reasoning on monadic second-order logics. Section 4 shows the experimental evaluation of this method. Related work is discussed in Section 5. Finally, the conclusions and future research directions are given in Section 6.

2. Background and definitions

In this section we provide definitions needed for the technical development in the rest of the paper.

2.1. Logics

First, we define the syntax and semantics of the second-order logic of one successors.

Definition 1. Let $\mathsf{Var} = \{x, y, z, \ldots\}$ be a (countably infinite) set of variable names. Formulas of second-order logics are defined as follows.

- the expressions $s_i(x, y)$, $x \subseteq y$ for x, y second-order variables are atomic formulas (standing intuitively for the successor relations where $i \in \{0\}$ for WS1S, and the subset relation), and
- given formulas φ and ϕ and a variable x, the expressions $\varphi \wedge \phi$, $\neg\varphi$, and $\exists x : \varphi$ are also formulas.

As variables for individuals (first-order variables) can be simulated using second-order variables bound to singleton sets; a property expressible in WS1S, we allow writing $x \in y$ for $x \subseteq y$ whenever we know that x is a singleton. We also use the standard abbreviations $\varphi \vee \psi$ for $\neg(\neg\varphi \wedge \neg\psi)$, $\varphi \rightarrow \psi$ for $\neg\varphi \vee \psi$, and $\forall x : \varphi$ for $\neg\exists x : \neg\varphi$.

The semantics of WS1S is defined w.r.t. the set of natural numbers (successors of 0); second-order variables are interpreted as finite sets of natural numbers in WS1S. The interpretation of the atomic formula $s_0(x, y)$ is fixed to relating singleton sets $\{n\}$ and $\{n + 1\}$, $n \in \mathbf{N}$.[1]

[1] The atomic formula $s(x, y)$ is often written as $y = s(x)$ in the literature, emphasizing its nature as a *successor* function.